Evolutionäre Methoden für die Steuerung und Regelung von Mikrorobotern

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
von der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe

genehmigte

D i s s e r t a t i o n

von
Dipl.-Ing. Michael Thiel
aus Berlin

Tag der mündlichen Prüfung: 8. Februar 2007
Erster Gutachter: Prof. Dr.-Ing. H. Wörn
Zweiter Gutachter: Prof. Dr. rer.nat. U. Brinkschulte

Inhaltsverzeichnis

Kapitel 1

Einleitung

Mikrosystemtechnik allgemein und Mikrorobotik im speziellen gelten als Schlüsseltechnologien der Zukunft. Die Entwicklung in diesem Bereich geht nicht nur in Richtung fortschreitender Miniaturisierung von mobilen Mikrorobotern, sondern auch hin zu einer immer exakteren Positionierung der Roboter und deren Manipulatoren bis zum Subnanometerbereich.

Eine andere Zielrichtung in der Mikrorobotik ist die Bildung von insektenähnlichen Schwärmen aus Mikrorobotern, die - mit sozialer Intelligenz ausgestattet - komplexe Aufgaben gemeinsam lösen können. Ein Schlagwort, das in diesem Zusammenhang häufig fällt, ist das der evolutionären Strategien. Die Evolution steuert die Entwicklung von organischen Lebewesen und hat in Jahrmillionen eine Form von Strukturiertheit und Intelligenz hervorgebracht, die uns immer wieder erstaunt. Diese Intelligenz äußert sich aber nicht nur in komplexen Verhaltensmustern, sondern hat auch für die Ausbildung elementarer Fähigkeiten, wie die Entwicklung und dem zielgerichtem Einsatz von Muskeln zur Bewegungssteuerung gesorgt. Eine naheliegende Idee ist es nun, dieselbe Grundidee - nämlich die Steuerung eines Optimierungsprozesses durch evolutionäre Abläufe - auch für die Steuerung und Regelung von Mikrorobotern zu nutzen.

1.1 Problemstellung

Der Einsatz von evolutionären Methoden in Bezug auf Roboterintelligenz kann nun auf vielfältigste Weise geschehen, Kapitel 2 liefert einen Überblick über Forschungsaktivitäten auf diesem Gebiet. Ein durchgängiger Entwurf für alle Aufgaben der Robotik ist zwar wünschenswert, jedoch nicht realistisch. Ein komplettes Konzept für die Verhaltenssteuerung beinhaltet eine Reihe von Einzelkomponenten, beginnend mit Bewegungssteuerung und -regelung hin zum Erlernen von komplexen Verhaltensmustern. Während bei Makrorobotern die Bewegungs-

regelung im Prinzip eine gut beherrschte Aufgabe darstellt, ist die Regelung von Mikrorobotern häufig noch problematisch. Dies hat seine Ursache vor allem in Effekten, die spezifisch für den Einsatz der Mikroroboter sind:

- Die Genauigkeitsanforderungen sind wesentlich höher.
- Die von den Robotern aufgebrachten Kräfte sind so gering, dass Umwelteinflüsse das Bewegungsverhalten signifikant verändern können. Selbst leichte Verschmutzungen der Arbeitsfläche oder der Einfluss einer Versorgungs- oder Signalleitung sind hier zu berücksichtigen.
- Die Genauigkeit der Positionsmessung liegt häufig in der gleichen Größenordnung wie die Anforderung an die Positionierung. Leichtes Messrauschen hat schon gravierende Auswirkungen auf die Stabilität der Regelung.

Dies alles motiviert die Suche nach einem zuverlässigen und hochgenauen Regelungssystem, das zugleich flexibel genug sein sollte, auf jede Art von mobilen Mikroroboter angewandt zu werden. Im Rahmen dieser Arbeit soll deshalb der Versuch unternommen werden, diese beiden Gebiete - Bewegungssteuerung von Mikrorobotern einerseits und evolutionäre Methoden andererseits - zu kombinieren und zu untersuchen, wie sich diese Kombination auch im Vergleich mit anderen Verfahren bewährt. Konkret ergibt sich für diese Arbeit folgende Aufgabenstellung:

- Entwicklung, Implementierung und Test eines evolutionären Algorithmus, der in der Lage ist, für die Steuerung und Regelung von Mikrorobotern eingesetzt zu werden. Dabei ist zu untersuchen, inwieweit bestehende Ansätze verbessert und weiterentwickelt werden können.
- Integration dieses Algorithmus in das bestehende Software-Konzept bzw. - sofern erforderlich - Erweiterung desselben.
- Einsatztest der evolutionären Methoden für die Steuerung und Regelung von Mikrorobotern. Zum Zwecke einer vergleichenden Bewertung sollten hier auch andere Verfahren untersucht werden.

Auf dieser Zielsetzung basierend, ist die Arbeit entstanden. Sie gliedert sich in insgesamt acht Einzelkapitel mit folgendem Aufbau:

Kapitel 2 liefert einen Überblick über Forschungsaktivitäten in der Mikrorobotik mit besonderer Berücksichtigung von Regelungsansätzen.

Kapitel 3 stellt den technischen Rahmen, in dem sich diese Arbeit bewegt, näher vor. Hier werden auch die zur Untersuchung herangezogen Mikroroboter detailliert vorgestellt.

Kapitel 4 erläutert das Regelungskonzept wie auch seine Umsetzung und die Integration in die bestehende Software-Umgebung.

Kapitel 5 beschreibt die evolutionäre Methode, die verwendet wurde. Dies beginnt mit einem Überblick über bestehende Ansätze und deren Probleme und endet mit einer Beschreibung des eigenen Ansatzes.

Kapitel 6 zeigt auf, wie die Methode zur Steuerung von Mikrorobotern genutzt werden kann. Um eine Bewertung zu ermöglichen, werden hier auch parallel andere Verfahren getestet. Insbesondere wird ein analytisches Modell der Roboterbewegung, das auf physikalisch-technischen Grundsätzen beruht, entwickelt und getestet.

Kapitel 7 untersucht den Einsatz der Methode für die Regelung von Mikrorobotern und bewertet sie im Vergleich mit anderen Verfahren.

Kapitel 8 beschließt die Arbeit mit einer Zusammenfassung und zeigt auf, welche künftigen Entwicklungen auf diesem Gebiet erwartet werden können.

Kapitel 2

Stand der Forschung

2.1 Mikroroboter

Die Handhabung von kleinsten Objekten bis hin zu einzelnen biologischen Zellen ist eine technische Herausforderung, die dazu geführt hat, dass in den letzten Jahren das Interesse an der Mikrorobotik stark gestiegen ist. Mobile Mikroroboter haben gegenüber stationären Mikromanipulationseinheiten einen eindeutigen Flexibilitätsvorteil und erweitern den Einsatz der Mikrosystemtechnik auf Anwendungsbereiche wie etwa:

- Mikromontage: Zusammenbau oder Reparatur sehr kleiner Komponenten. Ein mögliches Szenario ist die Entwicklung sehr großer Gruppen von Mikrorobotern, die kooperativ komplexe Montageaufgaben wahrnehmen und die - wegen ihrer geringen Größe und großen Flexibilität - dann kostengünstiger sind als eine klassische Montagestraße.

- Medizin: Handhabung einzelner Zellen oder auch als Werkzeug für die Chirurgie. Eine Vision, die auf Richard P. Feynman zurückgeht, ist die Idee, Nanoroboter im menschlichen Blutkreislauf Reinigungsarbeiten durchführen zu lassen oder Medikamente zielgenau absetzen zu lassen.)

- Inspektion von für Menschen unzugänglichen oder gefährlichen Umgebungen.

- Nachbildung von Schwarm-Verhalten - meist anhand von Beispielen aus der Natur, z.B. sozialen Insekten. Mikroroboter sind hier wegen ihres geringeren Platzbedarfs und Kosten größeren Robotern überlegen.

Am Institut für Prozessrechentechnik, Automation und Robotik der Universität Karlsruhe (IPR) wurde in den vergangenen Jahren eine Reihe von mobilen Mi-

krorobotern entwickelt, mit denen sich Mikrohandhabungsaufgaben unterschiedlicher Art durchführen lassen. Im Rahmen des von der EU geförderten MINIMAN-Projekts wurden Roboter mit einer Größe von 80x80x50 mm entwickelt, deren Bewegung auf dem piezoelektrischen Antriebsprinzip beruht [87].
Auf dem gleichen Prinzip beruhen auch deren Nachfolger, die MiCRoN-Roboter. Gleichfalls von der EU gefördert, lag der Fokus dieses Projekts auf der weiteren Miniaturisierung und der Verbesserung der Autonomie der Roboter. Da die hier vorgestellte Arbeit auf diesen beiden Roboterbaureihen beruht, wird im Kapitel 3 eine etwas detailliertere Darstellung von Aufbau und Ansteuerung erfolgen.

Ein aktuelles Projekt des IPR, das sich mit Mikrorobotern beschäftigt ist I-SWARM. Es basiert auf einem Schwarm von bis zu 1000 Mikrorobotern, die mit einem gewissen Maß an *Onboard*-Intelligenz ausgerüstet, sich ähnlich wie soziale Insekten selbst organisieren sollen. Sie sollen über eine Infrarot-Schnittstelle mit den anderen Robotern lokal kommunizieren können. Die benötigte Energie sollen sie aus einer Solarzelle beziehen. Eine zentrale Regelung ist hier jedoch ebensowenig vorgesehen wie eine präzise Bahnregelung.

Selbstverständlich werden auch anderswo mobile Mikroroboter entwickelt. Ein bewährter Vertreter seiner Art ist der Khepera-Roboter, der seit seiner Entstehung 1993 ständig weiterentwickelt wurde [57]. Seine Größe beträgt 55 mm im Durchmesser und 30 mm in der Höhe. Er verfügt über acht Infrarot-Näherungssensoren und kann zentral von einem PC oder direkt über die *Onboard*-Elektronik gesteuert werden.

Im Gegensatz zu den meisten anderen Mikrorobotern, die als Prototypen teuer entwickelt wurden, ist der 20x20x20 mm große Mikroroboter Alice [15] sehr einfach aufgebaut und deshalb recht kostengünstig. Sein niedriger Energieverbrauch lässt einen (batteriegebunden) autonomen Betrieb von bis zu 10 Stunden zu. Die Kommunikation kann unter anderem über eine Infrarot-Schnittstelle realisiert werden, die Umgebung kann über vier Distanz-Sensoren erkundet werden.

Ein Konzept, das neben der Miniaturisierung auch auf der präzisen Bewegung im Nanometer-Bereich basiert, ist der Nano-Walker [53]. Er ist 30x30x20 mm groß und basiert auf einem piezoelektrischen Antrieb mit Schrittweiten im Nanometer-Bereich (30 nm/Schritt). Die Roboter bewegen sich auf einem induktiven Feld und nehmen somit drahtlos die benötigte Energie auf (*Power Floor*). Nachteilig beim NanoWalker ist jedoch, dass sein Energieverbrauch so hoch ist, dass er gekühlt werden muss, so dass gewisse Anwendungsszenarien - etwa die Handhabung von lebenden Zellen - erschwert oder sogar unmöglich werden. Eine Weiterentwicklung des Roboters ist jedoch im Gange.

Zahlreiche weitere Mikroroboter, die sich in Bewegungsart, Autonomie, Größe und anderen Kriterien unterscheiden, wurden und werden entwickelt. Das Bewegungsprinzip von Mikrorobotern basiert normalerweise entweder auf dem Nutzen von Trägheitskräften (*Stick-Slip*-Effekt) oder einem *Walking*-Mechanismus.

Ein Mikroroboter, der im Gegensatz zu anderen Rollreibungskräfte und nicht Gleitreibungskräfte für die Fortbewegung nutzt wird in [78] beschrieben. Er ist verdrahtet und besitzt eine Bewegungsauflösung von ca. 20 Nanometern.

Donald [21] berichtet vom derzeit kleinsten Mikroroboter, der unverdrahtet ist und sich eigenständig fortbewegen kann. Er ist nur 60x250 Mikrometer groß und legt 10 Nanometer pro Schritt zurück. Er bewegt sich über ein Gitter aus Elektroden, die neben der Energieversorgung auch gleichzeitig seine Bewegungssteuerung übernehmen.

Ein mögliches zukünftiges Einsatzgebiet von Mikrorobotern ist wie erwähnt als Chirurgiewerkzeug in der Medizintechnik. Neben der weiteren Miniaturisierung ist hier vor allem die Energieversorgung ein kritischer Punkt. Ein Projekt, das in diese Richtung zielt, ist `MR-Sub` von der École Polytechnique Montreal [54]. Die Idee ist, das Magnetfeld, wie es in der Medizintechnik bei Tomographie-Untersuchungen von Menschen ohnehin schon benutzt wird, auch zur Energieversorgung von Mikrorobotern zu nutzen.

Eine Fortführung dieser Idee stellt der von Brad Nelson [59] entwickelte Mikroroboter dar. Er ist mit 200µm klein genug, um in die menschliche Blutbahn injiziert zu werden, und nutzt ebenfalls die Energie aus einem Magnetfeld. Dieser Mikroroboter konnte schon erfolgreich durch ein wässriges Medium manövriert werden.

2.2 Regelung von Mikrorobotern

Eine Grundvoraussetzung für die Nutzung von mobilen Robotern ist die Bewegungssteuerung zur Navigation und Positionierung. Dabei sind zwei grundsätzliche Aufgabentypen bei Mikrorobotern zu unterscheiden:

1. Planungsaufgaben, wie etwa Kollisionsvermeidung, Finden von Objekten oder Landmarken. Bei Roboterschwärmen kommen auch kollektive Aufgaben hinzu, wie etwa Gruppen- und Musterbildung oder kollektive Erkennung (*collective perception*).

2. Für eine praktische Anwendung von mobilen Mikrorobotern ist die hochgenaue Bewegung in der Mikro-Welt und präzise Handhabung von Mikro-Objekten entscheidend.

Ein Schwerpunkt der Forschung im Gebiet der Regelung beschäftigt sich mit dem ersten Themenkomplex. Gelegentlich werden dabei evolutionäre Methoden eingesetzt, um Regelungsstrategien zu entwickeln. Beispiele dafür sind etwa [62], der mit genetischen Methoden ein Programm erstellt, mit dessen Hilfe er ausgehend von aktuellen Sensordaten die Auswirkungen von Aktorbefehlen vorausschätzt.

Damit können dann geeignete Signale für Aufgaben wie Kollisionsvermeidung in einer (zunächst) unbekannten Umgebung oder Entlangfahren an einer Wand gefunden werden.

Pollack [65] berichtet von Experimenten, bei denen ein evolutionärer Lernalgorithmus *onboard* auf realen Robotern durchgeführt wird. Diese entwickeln neuronale Netze, die sie per IR-Kommunikation auch an benachbarte Roboter versenden können wodurch eine Art Selektion stattfindet. Fitte Roboter sind dann in der Lage, Gegenstände zu schieben oder andere Roboter zu verfolgen.

Diese Arbeit hat ihren Schwerpunkt im zweiten Themenkomplex, also der hochgenauen Geschwindigkeitsregelung von Mikrorobotern mit dem Ziel, eine vorgegebene Bahn möglichst exakt nachzufahren. Dies ist für die Handhabung von Mikroobjekten eine eminent wichtige Aufgabe. Es gibt nun viele Forschungsaktivitäten, die sich mit der Geschwindigkeitsregelung von Robotern allgemein beschäftigen, die sich aber nur eingeschränkt auf die Mikro-Welt übertragen lassen. Wenige Arbeiten existieren jedoch, die sich mit den speziellen Problemen der Regelung von Mikrorobotern beschäftigen.

Die Dissertation von Santa [68] kann als Vorgänger der hier vorliegenden Arbeit gesehen werden. Er hat einen PID-Regler, einen Fuzzy-Regler und einen neuronalen Regler für die Regelung von Mikrorobotern - unter anderem einer frühen Version des Miniman - miteinander verglichen. Dabei hat er eine strikte Trennung von translatorischen und rotatorischen Bewegungen vorgenommen sowie die Regelung auf einen Aktor beschränkt. Die maximale Bahnabweichung betrug für den verwendeten Miniman-Roboter ± 0.3mm bzw. $\pm 0.5°$.

In [32] wird *Reinforcement-Learning* genutzt, um einen Khepera-Roboter in einer zunächst unbekannten Umgebung zu einem Ziel zu navigieren. Angaben zur Genauigkeit des Verfahrens wurden hier jedoch nicht gemacht.

Chhabra [17] kombiniert einen Potentialfeld-Ansatz für die Bahnplanung mit einem PID-Regler für die eigentliche Navigation. Ergebnisse liegen jedoch nur für Simulationen vor. Ebenfalls auf einem Potentialfeld basiert der Bahnplanungsalgorithmus in [50]. Der Algorithmus beinhaltet keine Geschwindigkeitsregelung im eigentlichen Sinne, sondern garantiert Kollisionsvermeidung und Konvergenz in Bezug auf eine Zielposition. Er wurde an einem Miniman- und MiCRoN-Roboter in einem Testlauf erfolgreich eingesetzt.

In [74] wird ein Fuzzy-Regler für die *Onboard*-Regelung eines mobilen Mikroroboters vorgestellt. Resultate von Experimenten sind hierzu jedoch nicht publiziert.

An der Universität Oldenburg finden verschiedene Forschungsaktivitäten zum Thema Mikrorobotik statt. In [38] wird ein mehrstufiger Aufbau zur Regelung von Mikrorobotern vorgeschlagen, der auf selbstorganisierenden *Feature-Maps* (Kohonen-Netze) basiert. Diese Karten werden mit einer neuen Methode namens SOLIM gebildet, die wie ein Fuzzy Interpolator funktioniert, der mit dem Koho-

nen Lernalgorithmus trainiert wird. Die Methode hat in Simulationen gute Ergebnisse gezeigt.

Als Resumee des bisher gesagten lässt sich konstatieren, dass zwar die Mikrorobotik ein Gebiet von großem Interesse ist, die hochgenaue Geschwindigkeits- und Positionsregelung der mobilen Mikroroboter ein noch wenig erforschtes Teilgebiet ist. Die vorhandenen Ansätze leiden unter einer oder mehrerer der folgenden Schwächen:

- Es liegen nur Simulationsergebnisse vor.
- Die Experimente werden nur mit einem einzigen Mikroroboter durchgeführt. Eine allgemeingültige Schlussfolgerung fällt daher schwer.
- Zu wenige Testläufe wurden durchgeführt, um eine statistisch sichere Aussage treffen zu können.
- Es werden Ansätze aus der Regelung für Makroroboter übernommen, ohne auf die Besonderheiten der Mikrorobotik einzugehen.
- Die Bahnabweichung bzw. Regeldifferenz ist zu hoch für hochgenaue Anwendungen.
- Es findet kein bewertender Vergleich mehrerer Methoden statt.

In der vorliegenden Arbeit soll nun der Versuch unternommen werden, eine Reihe unterschiedlicher Regelungsstrategien an realen Mikrorobotern zu testen und eine Regelgüte zu erreichen, die es erlaubt, die Roboter in praktischen Anwendungen einzusetzen, die eine hochgenaue Positionierung erfordern.

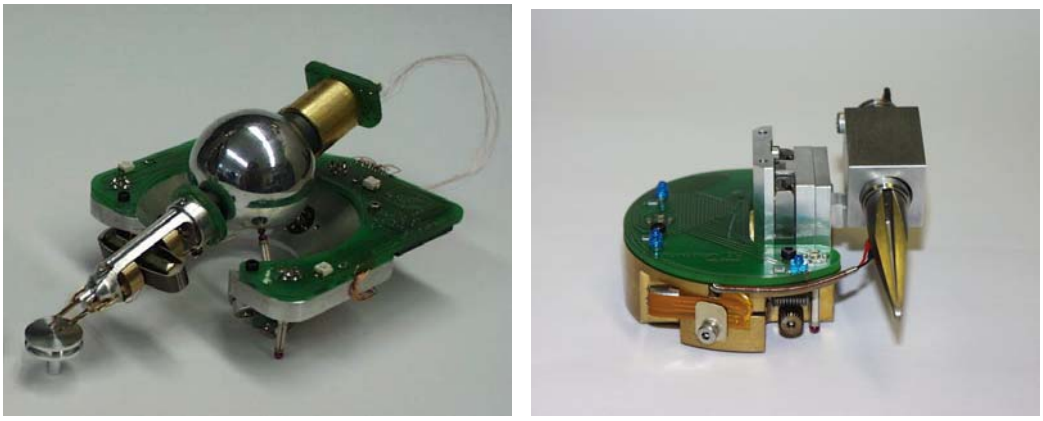
Kapitel 3

Systemumgebung

3.1 Mikroroboter

Die im Rahmen dieser Arbeit eingesetzten Mikroroboter besitzen jeweils drei Freiheitsgrade, sind vollständig holonom, können also durch gleichzeitiges Ansteuern aller Beine beliebig auf der Arbeitsfläche bewegt werden.

Sie sind alle mit Piezoelementen ausgestattet, nutzen den inversen piezo-elektrischen Effekt und bewegen sich auf der Basis des *Stick-Slip*-Effekts. Durch Anlegen einer Spannung an den Piezokeramiken verändert sich das atomare Kristallgitter und es kommt zu einem Durchbiegen der Piezoelemente. Die Piezoelemente bilden sozusagen die Beine des Roboters.



(a) Miniman III-2 (b) Miniman IV

Abbildung 3.1: Verschiedene realisierte Modelle des Miniman-Projekts

Jeder Schritt besteht aus einer langsamen Steigerung der Spannung und einem plötzlichen Umpolen (Sägezahnsignal). Während der langsamen Spannungser-

Roboter	Geschwindigkeit		Schrittweite	Auflösung
	Translation [mm/s]	Rotation [mrad/s]	[nm]	[nm]
Miniman III	3.5	20	4000	20
Miniman IV	4	20	4000	20
MiCRoN 0	0.5	30	125	10

Tabelle 3.1: Charakteristische Größen der Mikroroboter

höhung biegen sich die Beine durch und es verschiebt sich der Massenschwerpunkt, jedoch wegen der Haftreibung nicht die Kontaktposition der Beine auf der Arbeitsfläche (*Stick*-Phase). Während des Umpolens werden dann die Beine durch die Massenträgheitskräfte in Richtung des Massenschwerpunkts beschleunigt (*Slip*-Phase). Dieser Vorgang kann mit einer Frequenz von bis zu einigen kHz wiederholt werden. Eine detaillierte mathematische Beschreibung und Analyse des Bewegungsprinzips wird in Kapitel 6.2 erfolgen. Neben diesem Bewegungsmodus können die Roboter auch in einem sogenannten *Scan*-Modus betrieben werden. Dabei wird auf die *Slip*-Phase verzichtet und die Position des Roboters lediglich durch langsames Verändern der Spannung und darauf resultierendem Verbiegen der Beine leicht verändert. Die erreichte Bewegungsauflösung - also die minimal mögliche Positionsauslenkung - ist in diesem Fall im Nanometer-Bereich angesiedelt. Ungefähre maximale Geschwindigkeiten sowie Schrittweiten und Auflösungen der verschiedenen Roboter zeigt Tabelle 3.1.

3.1.1 Miniman-Roboter

Im Rahmen des EU-Projekts Miniman entstanden in den Jahren 1998 bis 2002 eine Reihe unterschiedlicher Mikroroboter, deren grundsätzlicher Aufbau und Bewegungsprinzip sich aber ähnelt [87]. Abbildung 3.1 zeigt zwei erfolgreiche Modelle, die auch im Rahmen dieser Arbeit näher untersucht wurden.

Der Miniman-Roboter besteht aus einer mobilen Plattform und einem Manipulator. Die Plattform bewegt sich mit Hilfe von drei Piezobeinen, bei denen der *Stick-Slip*-Effekt genutzt wird. Das äußere Röhrchen des Piezobeins ist in vier orthogonal zueinander angeordnete Segmente aufgeteilt, an deren Elektroden unabhängig voneinander eine Spannung angelegt werden kann. Dabei sind die angelegten Spannungen an gegenüberliegenden Elektroden immer gegenläufig, um den Durchbiegungseffekt zu verstärken (vgl. Abb. 3.2). Auf diese Weise können die oben näher beschriebenen Modi (*Stick-Slip* und *Scan*) realisiert werden.

Die Miniman-Roboter werden über ein hybrides Parallel-Multiprozessor-System (PMS) [87] angesteuert, das die Spannungsverläufe für alle Elektroden mit der gewünschten Frequenz generiert (im Folgenden kurz *Hardware-Controller*

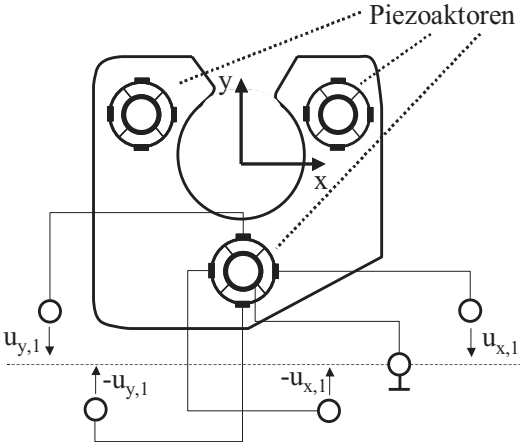


Abbildung 3.2: (Schematische) Anordnung der Aktoren und Elektroden im Querschnitt der Miniman III-Plattform

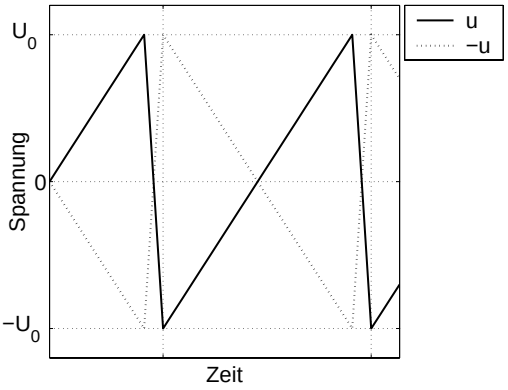


Abbildung 3.3: Spannungsverlauf an den gegenüberliegenden Elektroden beim *Stick-Slip*-Modus

genannt). Dieses ist über Ethernet mit dem Rechner verbunden, von dem es die Steuersignale erhält.

Für die Ansteuerung von Piezoaktoren sind grundsätzlich zwei Parameter von Bedeutung: Die Spannungsamplitude U_0 und die Frequenz f. Beim Miniman wurde die Vereinbarung getroffen, alle Aktoren mit der gleichen Frequenz anzusteuern. Die Spannungsamplitude eines der Aktoren beträgt $U_0 = U_{0,max} = \pm 150V$, die der anderen Aktoren sind hingegen geringer und variabel. Damit ergeben sich drei unabhängige Größen, nämlich die Frequenz und die Spannungsamplituden zweier der drei Aktoren. Die Spannungsamplitude jedes Aktors verfügt ihrerseits entsprechend der Wirkrichtungen der Elektrodenpaare über jeweils zwei Komponenten ($U_{0,x}$ und $U_{0,y}$, vgl. Abb. 3.2). Um eine etwas benutzerfreundlichere Bedienung zu ermöglichen, enthält der Stellbefehl u eines Aktors zwei andere Parameter, nämlich Ansteuerwinkel θ und -betrag u_a, die vom Hardware-Controller in das Steuersignal u_H überführt wird:

$$u\{u_a, \theta\} \rightarrow u_H\{U_{0,x}, U_{0,y}, f\} \tag{3.1}$$

Dabei werden folgende Beziehungen benutzt:

$$\begin{aligned} U_{0,x,i} &= \frac{u_{a,i}}{u_{a,max}} U_{0,max} \cos\theta_i \\ U_{0,y,i} &= \frac{u_{a,i}}{u_{a,max}} U_{0,max} \sin\theta_i \\ f &= g_f(u_{a,max}) \end{aligned} \tag{3.2}$$

Die Funktion g_f ist fest vorgegeben und in Abbildung 6.1 näher dargestellt. Der

Ansteuerwinkel θ entspricht dabei der Richtung, in die die vom Piezoaktor aufgebrachte Kraft wirkt (vgl. Abb. 3.4). Dies ist nicht zwangsläufig identisch mit der effektiven Bewegungsrichtung des Roboters, da einerseits noch andere Aktoren Einfluss haben, andererseits Störungen die Bewegung verändern können. Die Größenordnung des Ansteuerbetrags u_a ist bei den Miniman- Robotern auf 1 Byte beschränkt, es ist hier also maximal ein Wert von 255 zulässig.

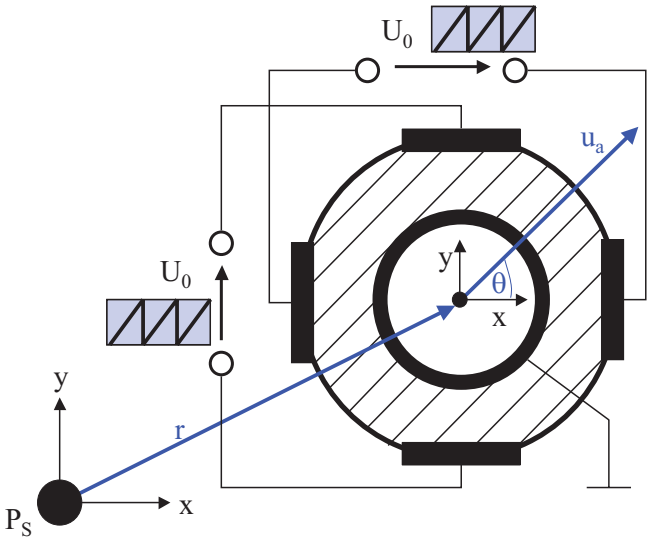


Abbildung 3.4: Steuersignal und Stellbefehl für einen Aktor des Miniman

Die Miniman III-Roboter können mit verschiedenen Werkzeugen insbesondere Mikrogreifern ausgestattet werden. Das Werkzeug ist in eine Kugel integriert, die mit den gleichen Prinzipien angesteuert und bewegt wird wie die Plattform.

Auch der Miniman IV besitzt den Vorteil der Austauschbarkeit der Werkzeuge, besitzt aber ferner die Besonderheit, auch eine Kamera tragen zu können. Damit kann die Oberfläche aus einer sehr geringen Entfernung beobachtet werden, etwa um Mikroobjekte zu lokalisieren.

3.1.2 MiCRoN-Roboter

Die MiCRoN-Roboter wurden im Rahmen eines gleichnamigen EU-Projekts in einem europaweiten Konsortium in den Jahren 2002 bis 2005 entwickelt [26]. Dabei sollte eine kleine Gruppe von Mikrorobotern hergestellt werden, die bei einer Größe von ca 1 cm^3, eine Bewegungsauflösung bis unter den Mikrometer-Bereich haben sollten.

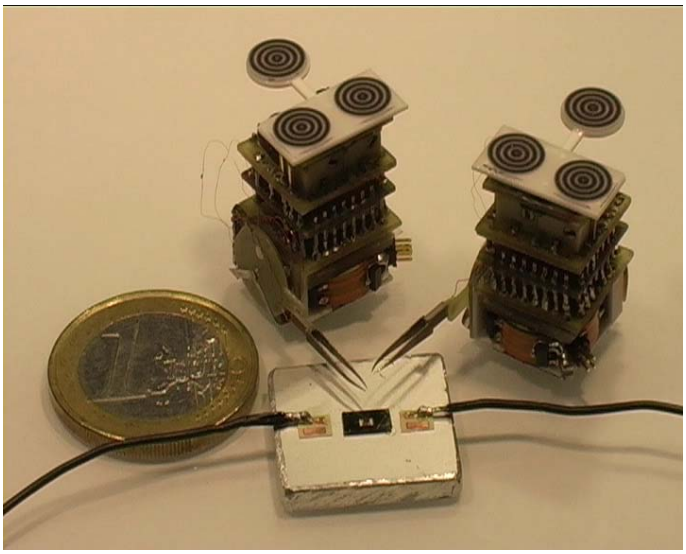


Abbildung 3.5: Zwei MiCRoN-Roboter

Eine der wesentlichen Herausforderungen des Projekts war, dass die Roboter - im Vergleich etwa zum Miniman-Robot - über eine weitgehende Autonomie verfügen sollten. Das bedeutet konkret, dass sowohl Energie- als auch Signalübertragung kabellos erfolgen sollte. Die Steuersignale werden über eine IR-Schnittstelle mit einer Reichweite von ca. 80 cm und einer Rate von bis zu 4 MBit/s auf den Roboter übertragen. Die Umwandlung der Steuersignale in einen von den Piezoaktoren benötigten Spannungsverlauf erfolgt über die Onboard-Elektronik. Die benötigte Energie wird von einem induktiven Feld, das von einer speziellen Grundplatte erzeugt wird (*Power Floor*) berührungs- und drahtlos auf den Roboter übertragen. Der Energieverbrauch der Roboter beträgt ca. 400 mW.

Der für den Rahmen dieser Arbeit vielleicht interessanteste Aspekt des Projekts ist das Fehlen von Versorgungs- oder Signalleitungen, welche im Normalfall einen beträchtlichen Einfluss auf das Bewegungsverhalten des Roboters haben. Selbst dünne Kabel üben in Anbetracht der geringen Kräfte, die vom Aktorsystem aufgebracht werden, eine nicht zu unterschätzende Zugkraft auf den Roboter aus.

Leider ist das ursprünglich avisierte Ziel in Bezug auf die Autonomie nicht vollständig erreicht worden. Da sich die von den Robotern benötigte Energiemenge als zu hoch herausgestellt hat, wurde es unmöglich, mehr als einen der Roboter gleichzeitig ohne Kabel zu betreiben.

Auch die MiCRoN-Roboter nutzen den *Stick-Slip*-Effekt, besitzen aber - anders als die Miniman-Roboter - nicht drei Piezobeine, sondern sechs Piezoprofile.

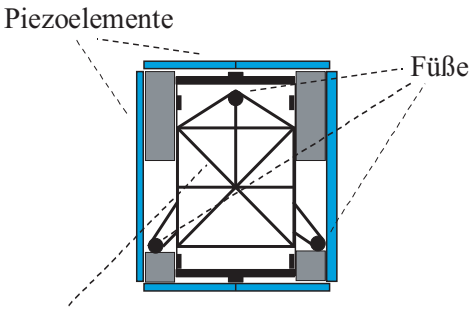


Abbildung 3.6: Aufbau des MiCRoN-Roboters (Blick von unten)

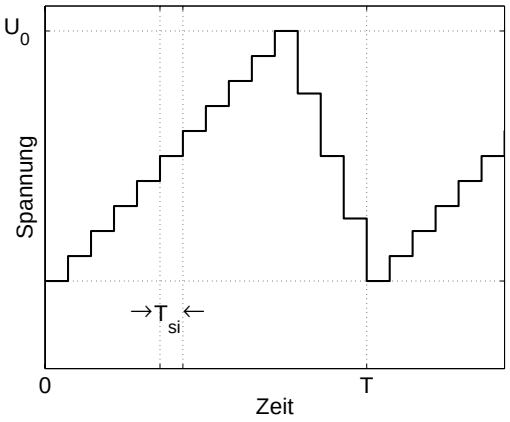


Abbildung 3.7: Schematische Darstellung des Spannungsverlauf beim MiCRoN

Mit diesen sechs Aktoren kann eine beliebige Kombination aus translatorischen und rotatorischen Bewegungen realisiert werden. Die Piezoprofile sind seitlich an einer flexiblen Struktur befestigt, die oben am Roboter und unten an drei Füßen angebracht ist. Durch Biegen der Piezoprofile werden Kräfte auf die Struktur übertragen, die mit Hilfe der Massenträgheitskräfte zu einer Roboterbewegung führen (vgl. Abb. 3.6).

Ein weiterer Unterschied zu den Miniman-Robotern ist die Tatsache, dass der Spannungsverlauf für die Aktoren *onboard* erzeugt wird, ein externer Hardware-Controller also nicht vonnöten ist. Der Stellbefehl u enthält beim MiCRoN drei Komponenten pro Aktor:

- Spannungsamplitude U_0
- Taktzeit T
- Zeitabstand zwischen zwei Spannungsstufen T_{si}

Daraus wird von der Elektronik der gewünschte Spannungsverlauf generiert. Obwohl also auch U_0 und T_{si} für Regelungszwecke variiert werden könnten, hat es sich - ähnlich wie beim Miniman - als völlig ausreichend herausgestellt, nur die Taktzeit T bzw. die Frequenz zu verändern.

3.2 Sensorik

Für die zielgerichtete, geregelte Bewegung der Roboter ist eine kontinuierliche und zuverlässige Positionsbestimmung unerlässlich. Wegen der Größe der Robo-

ter fehlt eine interne Sensorik. Stattdessen existieren zwei einsatzfähige externe Verfahren, die auf Bildverarbeitung beruhen:

- Die Miniman-Roboter verfügen sowohl auf der Plattform als auch auf dem Manipulator über Leuchtdioden, die zentral aktiviert (und deaktiviert) werden können. Mit Hilfe eines Kamerabildes können die Positionen der Dioden erkannt werden. Dies ist auf zweierlei Weise möglich:

  – Die Leuchtdioden bleiben permanent eingeschaltet, der Bildhintergrund sollte dann möglichst einheitlich und dunkler sein als die Dioden. Bei einer geeigneten Blendeneinstellung an der Kamera heben sich die Leuchtdioden als hellste Punkte im Bild ab und können mit Standardmethoden erkannt werden.

  – Bei alternierendem Ein- und Ausschalten der LEDs werden Bilder aufgenommen. Durch Differenzbildung zweier Bilder können die Positionen der Leuchtdioden bestimmt werden.

  Diese Verfahren sind recht schnell, jedoch nicht besonders genau und stabil. Als weitere Nachteile haben sich herausgestellt:

  – Ein dunkler Hintergrund ist je nach Anwendung nicht immer zu gewährleisten. Ferner wirken sich Störeinflüsse in der Helligkeit sofort deutlich aus und führen zu Fehlmessungen.

  – Das Verfahren erfordert in seiner zweiten Variante zwei Bildaufnahmen pro Positionsbestimmung, was - bei sich gleichzeitig bewegenden Robotern - zu Verzerrungen führt.

  Für nähere Details zu der Vorgehensweise sei auf [70] verwiesen.

- Ein neueres Verfahren basiert auf der Erkennung von Markierungen (sogenannten Moiré-Marken) auf den Robotern mit Hilfe eines speziellen Algorithmus [25]. Diese sind so klein und leicht, dass auch bestehende Roboter damit nachgerüstet werden können. Das Verfahren liefert eine Genauigkeit von $5\mu m$ (vgl. Tabelle 3.2) und ist damit deutlich besser als das LED-basierte Verfahren.

Mit diesen Verfahren sind grundsätzlich zwei Betriebsarten möglich: Bei der *Erkennung* wird das gesamte Kamerabild untersucht und alle Markierungen oder Leuchtpunkte einem Roboter zugeordnet. Dieses Verfahren ist recht zeitaufwändig (größenordnungsmäßig im Bereich von ca. 5 bis 10 Sekunden für eine komplette Bildauswertung), so dass eine Echtzeitfähigkeit hier nicht gegeben ist. Dagegen ist *Tracking* deutlich schneller: Hier wird nur ein begrenzter Bildbereich

Verfahren	Taktzeit		Genauigkeit
	Erkennung	Tracking	
LED	3 s	50 ms	100 µm
Moiré	5-10 s	15-60 ms	5 µm

Tabelle 3.2: Charakteristische Größen der Sensorik

(ROI[1]) untersucht. Die Lage des Bildbereichs ist vom Ergebnis der letzten Messung im Allgemeinen mit hinreichender Genauigkeit bekannt, wenn die Messzeitpunkte nicht zu weit auseinander liegen bzw. die Robotergeschwindigkeit nicht zu hoch ist.

Tabelle 3.2 liefert einen Überblick über charakteristische Größen der beiden Verfahren. Es ist zu beachten, dass sich die Taktzeiten im Falle des *Tracking* auf eine Markierung bzw. Leuchtdiode bezieht. Für eine komplette Erkennung der Roboterplattform sind jedoch mindestens drei Markierungen erforderlich, so dass sich die Taktzeit entsprechend erhöht. Die MiCRoN-Roboter sind nicht mit Leuchtdioden ausgerüstet, so dass sich hier die Frage nach dem geeigneten Verfahren gar nicht erst stellt, es ist aber auch für die Miniman-Roboter klar, dass das Moiré-basierte Verfahren schneller, genauer und weniger störanfällig ist.

Auch die Manipulatorerkennung kann im Prinzip mit diesem Verfahren durchgeführt werden. Dies kann aber nur bei bestimmten Manipulatorwinkeln zuverlässig funktionieren, da ansonsten bei einer Draufsicht die Marken verzerrt erkannt werden. Für dieses Problem existiert noch keine zuverlässige Lösung.

Die Sensordaten beinhalten lediglich die Positionen von Markierungen auf dem Roboter und noch nicht die gesuchte Position und Orientierung des Roboters insgesamt. Deshalb müssen nun die Daten pro Roboter zu einem Gesamtmodell fusioniert werden. Zu diesem Zweck wird ein Kalman-Filter benutzt. Neben der Fusionierung bietet er noch den Vorteil, dass ungenaue oder verrauschte Messdaten leicht herausgefiltert bzw. geglättet werden können. Desweiteren besteht die Möglichkeit der Prädiktion, d.h. auch zukünftige Roboterpositionen können vorausgeschätzt werden. Auch die Ausfälle einzelner Sensoren können unter Umständen kompensiert werden.

Der eingesetzte Kalman-Filter ist ein sogenannter *Unscented Kalman Filter*, der nach der *SCAAT*-Methode benutzt wird [88] [70]. Damit können jederzeit - und nicht erst nach Vorliegen aller Messungen - neue Prädiktionen abgegeben werden.

[1] ROI: **R**egion **o**f **i**nterest

Kapitel 4

Konzept und Umsetzung

4.1 Prinzip

Allgemeine Ziele von Regelungen sind es, die vom Benutzer vorgegebenen Sollwerte möglichst schnell und stabil nachzuführen (Führungsverhalten) und das System robust gegenüber Störeinflüssen zu machen (Störungsverhalten). Bei der Auslegung eines geeigneten Reglers für mobile Mikroroboter stößt man jedoch auf einige zusätzliche Aspekte, die zu berücksichtigen sind:

- Die Regelung muss wegen der Anwendungsgebiete der Mikroroboter besonders präzise sein.

- Es ist eine Reihe von unterschiedlichen Robotern zu regeln, die sich in ihrem Bewegungsverhalten aber auch der Art der Ansteuerung (Hardware und Parameter) teilweise erheblich voneinander unterscheiden.

- Die physikalischen Vorgänge im Mikroroboter sind zwar grundsätzlich bestimmbar, jedoch ist eine genaue analytische Beschreibung für jeden einzelnen Roboter aufwändig. Materialkonstanten, die das Verhalten des Roboters bestimmen, können sich zudem mit der Zeit ändern. Ein modellfreier Ansatz ist für die Regelungsstruktur deshalb vorzuziehen.

- Sich ändernde Umgebungsgrößen haben einen entscheidenden Einfluss auf das Bewegungsverhalten des Roboters. Hier sind vor allem die Oberflächenbeschaffenheit und eventuelle Verschmutzungen zu bedenken.

- Obwohl in der Theorie piezoelektrisch betriebene Aktoren ein über weite Teile des Arbeitsbereiches lineares Verhalten zeigen, hat sich in Experimenten herausgestellt, dass einige Mikroroboter ein ausgeprägtes nichtlineares Verhalten haben. Im hohen Geschwindigkeitsbereich tendiert das

Bewegungsmuster von Piezo-Aktoren aus technischen Gründen zu einem chaotischen Verhalten [13].

- Sensorinformationen liegen vor, sind aber mit einer gewissen Ungenauigkeit behaftet. Auch ist naturgemäß - abhängig von der aktuellen Belastung des Prozessors - eine begrenzte Taktzeit einzuhalten. Diese liegt bei ca. 200 ms (vgl. Abschnitt 3.2). Die genaue Bewegungsdynamik des Roboters ist damit natürlich nicht zu erfassen. Ein robuster Regler sollte in der Lage sein, das System trotzdem stabil zu halten.

- Wegen der vergleichsweise langsamen Bewegungen der Mikroroboter wirken sich Ungenauigkeiten in der Positionsmessung selbst von wenigen Mikrometern stark auf das Ergebnis der Geschwindigkeitsbestimmung aus. Man erhält ein stark verrauschtes Messergebnis. Ein Regler muss also sehr robust damit umgehen können.

Als Schlußfolgerung aus den genannten Problemen ergeben sich die Anforderungen an das Regelsystem. Das System sollte also

- flexibel
- adaptiv
- nichtlinear
- und robust

sein. Zunächst soll deshalb eine Struktur entworfen werden, die eine leichte Austauschbarkeit der Regler ermöglicht.

Das Regelungssystem steht im Spannungsfeld zwischen den Vorgaben des Benutzers bzw. einer Planungsinstanz einerseits (w) und den Rückgaben des Meßsystems andererseits (y, vgl. Abb. 4.1). Aus diesen Informationen muss ein Stellbefehl (u) generiert werden, der vom Roboter bzw. einem vorgeschalteten Hardwarecontroller umgesetzt werden kann. Im allgemeinen Anwendungsfall ist die Führungsgröße des Systems eine Zielposition des Roboters. In diesem Fall ist eine Lageregelung erforderlich. Üblicherweise wird diese in einer Kaskadenstruktur realisiert, bei der der Ausgang des Lagereglers als Führungsgröße des untergeordneten Geschwindigkeitsreglers verwendet wird (vgl. Abb. 4.1). Es sind aber auch Anwendungsszenarien denkbar, bei denen lediglich eine Geschwindigkeitsregelung erforderlich ist. Im Rahmen dieser Arbeit liegt der Fokus auf der Entwicklung des Geschwindigkeitsreglers, der im folgenden näher untersucht wird. Bei einer erfolgreichen Geschwindigkeitsregelung ist es dann im allgemeinen ausreichend, für den übergeordneten Positionsregler (sofern erforderlich) einen sehr einfachen Ansatz - beispielsweise einen P-Regler - zu wählen.

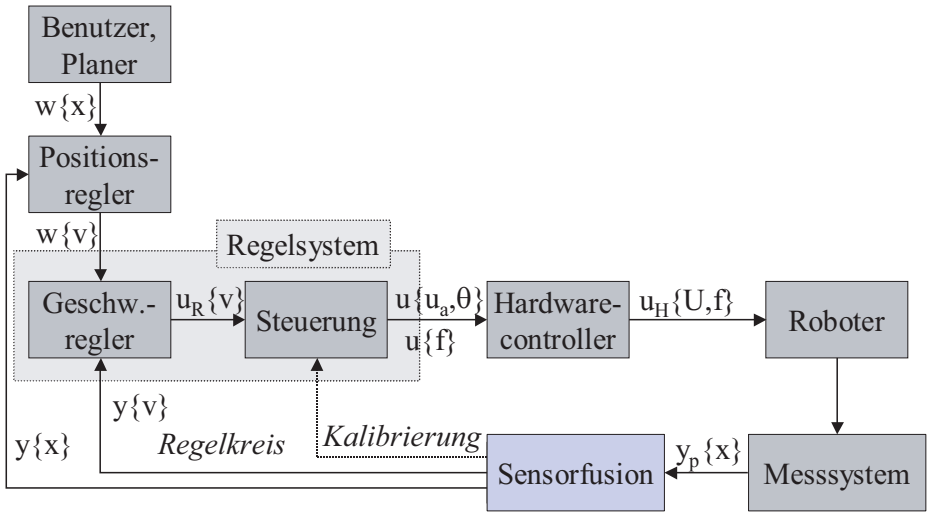


Abbildung 4.1: Überblick über das Regelsystem

Um diese Vorgabe zu realisieren, wird ein zweistufiger Systemaufbau vorgeschlagen: Jedem speziellen Roboter ist ein eigenes Steuerungsmodell (Treiber) zugeordnet, der die Ausgänge des eigentlichen Reglers geeignet umsetzt. Die Informationen zur Steuerung erhält der Treiber in einem Kalibriervorgang, der für jeden Roboter einmalig oder bei Bedarf - etwa bei signifikanter Änderung des Betriebsverhaltens durch Materialermüdung - auch mehrfach durchgeführt wird. Auf diese Weise werden Probleme, die durch zwei geschlossene Regelkreise hervorgerufen werden, vermieden, zugleich kann aber dem unterschiedlichen Bewegungsverhalten der Roboter Rechnung getragen werden. Die Aufgabe des Treibers ist jedoch nicht nur der Ausgleich unterschiedlicher Bewegungsmuster, sondern auch ein weitestgehender Ausgleich von Nichtlinearitäten mit dem Ziel, einem Regler eine möglichst einheitliche Regelstrecke zur Verfügung zu stellen. Der Regler als übergeordnete Instanz schließt dann den Regelkreis anhand der Positions- bzw. Geschwindigkeitsrückmeldung des Messsystems.

In dieser Arbeit sollen verschiedene Möglichkeiten untersucht werden, einen geeigneten Steuerungsalgorithmus zu finden und einen Regler für Mikroroboter zu entwerfen. Ein besonderer Schwerpunkt liegt dabei auf evolutionären Methoden, deren Vielseitigkeit und Fähigkeit, auch stark nichtlineares Verhalten zu glätten sie als besonders geeignet für diese Aufgabe erscheinen lassen. Um einen Bewertung durchführen zu können, wurde ein Vergleich mit anderen Verfahren durchgeführt. Hierbei wurde insbesondere der Einsatz von neuronalen Netzen untersucht.

Wegen des Fehlens einer zuverlässigen Sensorik für Manipulatorbewegungen (vgl. Abschnitt 3.2) kann nur die Regelung der Roboterplattform näher untersucht werden. Die hierfür entwickelten Konzepte und Algorithmen lassen sich jedoch ohne weiteres auch auf die Geschwindigkeitsregelung in anderen Freiheitsgraden übertragen.

Als Hilfsmittel zur Umsetzung kamen verschiedene Werkzeuge zum Einsatz - darunter die Algebra-Systeme Matlab und Maple - ein Großteil der Anwendungen wurde jedoch mit der Programmiersprache C++ direkt erstellt.

4.2 Software-Architektur

Zu Beginn dieser Arbeit bestand bereits eine Software-Struktur, die es ermöglichte, die Miniman-Roboter-Hardware anzusteuern, Positionsdaten von Robotermarkierungen zu erfassen und Sensordaten zu fusionieren. Was fehlte, war eine flexible und übersichtliche Gesamtstruktur, mit dessen Hilfe neue Module leicht integriert werden können und die einzelne Software-Teile voneinander kapselt. Deswegen wurde im Rahmen dieser Arbeit ein Re-Design der Software vorgenommen. Komponenten, die unter anderem integriert werden mussten, sind:

- Positionsmessung und Sensordatenfusion (bereits vorhanden, siehe [70] bzw. [25])
- Ansteuerung der Miniman- und MiCRoN-Roboterhardware (größtenteil bereits vorhanden [87] bzw. parallel zu dieser Arbeit entstanden)
- Steuerungsalgorithmen für Roboter (Kapitel 6)
- Regelungsalgorithmen für Roboter (Kapitel 7)

Um dies alles integrieren zu können, wurde eine zentrale Klasse *MWorldModel* geschaffen, die eine Verwaltung und Zugriff auf alle erstellten Software-Objekte ermöglicht. Darüber hinaus entstanden Klassen, die eine komfortable Schnittstelle zu den bereits existierenden Modulen zur Verfügung stellen.

- Die Klasse *MeasureDriver* erlaubt die Steuerung, Überwachung und Positionsabfrage des Sensorsystems.
- Die Klasse *MinimanRobotMover* erlaubt den Zugriff auf die Miniman-Hardware. Zusätzlich wurde, um auch eine Ansteuerung des MiCRoN-Roboters zu ermöglichen, die Klasse *MicronRobotMover* geschaffen. Der Gebrauch von *SimulationRobotMover* erlaubt Tests in einer virtuellen Roboterumgebung. Bei entsprechender Parametrierung des zu dem Roboter gehörigen

MeasureDriver ist auch die Simulation von Positionsrückmeldungen möglich. Um eine leichte Austauschbarkeit zu gewährleisten, wurde die abstrakte Klasse *MIRobotMover* geschaffen, so dass nach außen alle Roboter-Hardware-Klassen eine einheitliche Schnittstelle besitzen, die sich im Wesentlichen nur in der Anzahl der Ansteuersignale und ihrer Gültigkeitsgrenzen unterscheidet.

- *MIRobotDriver* ist eine abstrakte Klasse, die den Zugriff auf den gewählten Steueralgorithmus ermöglicht. Eine detaillierte Betrachtung dieses Themas wird in Kapitel 6 erfolgen. Gleiches gilt für die Regelungsalgorithmen, die von der Basisklasse *MIControlAlgorithm* ableiten und näher in Abschnitt 7 beschrieben werden.

- Jede Instanz von *MIRobot* sollte einem (simulierten oder realen) Roboter der aktuellen Arbeitsumgebung entsprechen. Jede *MIRobot*-Instanz sollte genau eine Instanz des *MIRobotMover* besitzen, aber bis zu zwei *MIRobot-Driver*-Instanzen, je eine für Plattform und Manipulator. Auf diese Weise können für Plattform und Manipulator getrennte Steueralgorithmen zur Anwendung kommen.

- Die aktuelle Konfiguration des Roboters ebenso wie Angaben zu seiner Geometrie und andere allgemeine Daten werden in der Klasse *MIRobot-Model* gespeichert. Diese bestand im Wesentlichen schon vor Beginn dieser Arbeit.

Abbildung 4.2 zeigt den stark schematisierten Aufbau des Software-Systems. Tatsächlich ist die Aufteilung sehr viel detaillierter (mehr als 100 Software-Klassen sind involviert).

Die Software-Objekte werden vom Weltmodell (*MWorldModel*) dynamisch - d.h. je nach Benutzereingabe - erzeugt oder bei Bedarf auch zerstört. Damit sind auch Änderungen der Konfiguration während der Laufzeit möglich. Die zur Erstellung der Objekte benötigten Informationen werden aus einer Konfigurationsdatei mit Hilfe der Klasse *MicronFactory* ausgelesen.

Da eine Reihe von unterschiedlichen Aufgaben gleichzeitig erledigt werden muss, zugleich aber die Taktzeit nicht zu sehr steigen sollte, hat sich eine auf mehrere Rechner verteilte Anwendung als notwendig herausgestellt: Robotererkennung, Objekterkennung und - im Falle des MiCRoN - auch das Senden des *Reset*-Signals laufen auf externen Rechnern ab und werden über Ethernet angesprochen.

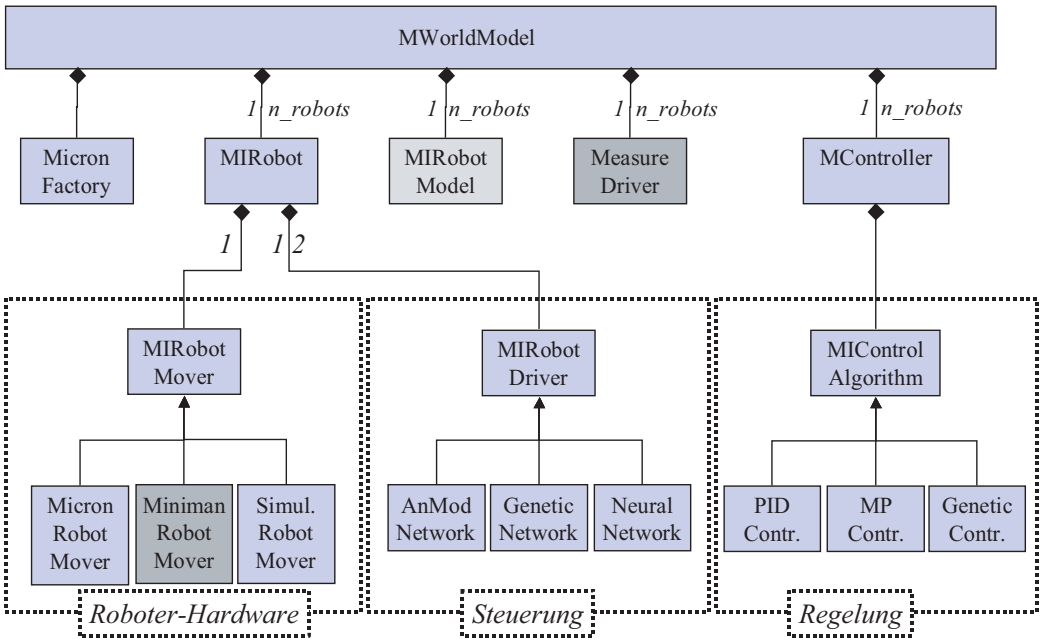


Abbildung 4.2: Vereinfachtes Kollaborationsdiagramm des Gesamtsystems

4.3 Benutzerschnittstelle

Um eine leichte Bedienung und Überwachung der Roboter zu ermöglichen, wurde eine graphische Benutzerschnittstelle auf MDI-Basis geschaffen. Abbildung 4.3 zeigt einen *Screenshot* der Oberfläche mit einem Miniman IV und zwei MiCRoN-Robotern.

Für jeden Roboter wird ein eigenes Fenster erstellt, über das alle Konfigurations- und Überwachungsaufgaben erledigt werden können. Die Roboter können über die Oberfläche in verschiedenen Betriebsarten betrieben werden:

- Aktor-Modus: Die Steuersignale werden direkt vom Benutzer vorgegeben (keine Regelung).
- Manueller (gesteuerter) Betrieb: Der Benutzer gibt die Richtung und Geschwindigkeiten vor - entweder über Eingabe an der Oberfläche oder über eine 6D-Maus. Eine Regelung findet auch hier nicht statt.
- Geregelter Betrieb: Erneut werden Geschwindigkeiten und Richtung vorgegeben, diesmal wird die Positionsrückmeldung zur Regelung benutzt.
- Zielmodus: Eine Zielposition und -orientierung sowie die zum Erreichen zur Verfügung stehende Zeit werden vorgegeben (Regelung).

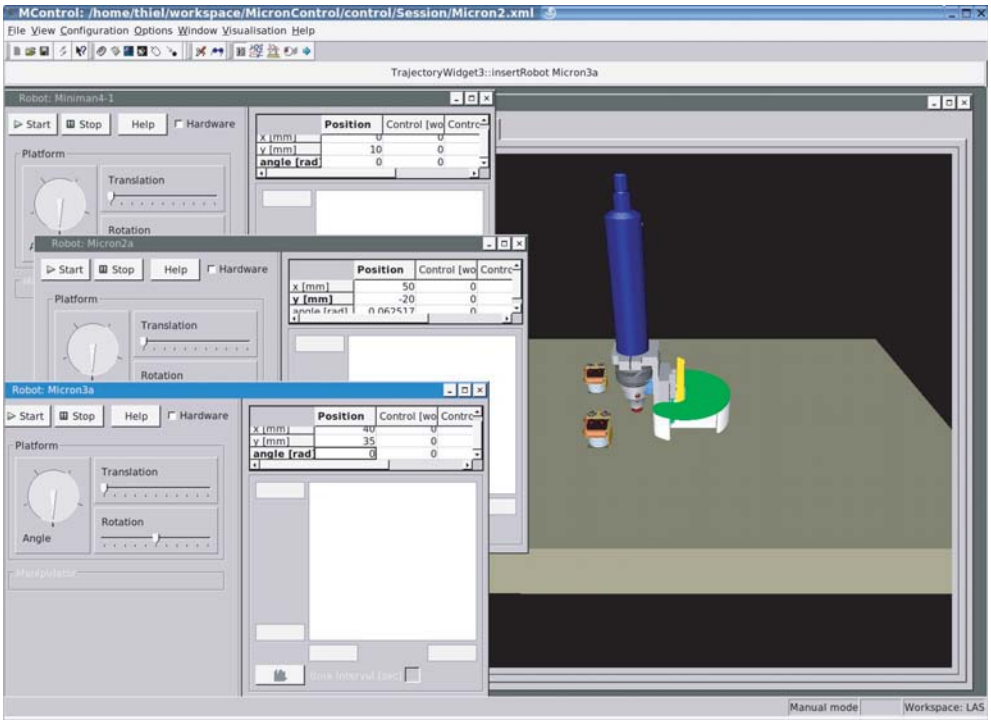


Abbildung 4.3: Screenshot des GUI

- Inspektionsmodus: Ein Bereich der Arbeitsfläche wird vom Benutzer ausgewählt. Dieser Bereich wird in ein Gitter aufgeteilt. Mit Hilfe einer Regelung wird der Roboter über diesen Bereich bewegt. Sinn ist dabei das Scannen von Oberflächen, etwa zum Finden kleiner Objekte mit Hilfe des kamerabestückten Miniman IV.

Eine Überwachung der Arbeitsfläche durch den Benutzer ist einerseits über ein oder mehrere eingeblendete Kamerabilder möglich, aber auch mit Hilfe einer symbolischen 3D-Darstellung der Szenerie. Selbstverständlich können über die Oberfläche auch angeschlossene Geräte - etwa die Kommunikation mit externen Rechnern zur Positionserkennung - konfiguriert und überwacht werden.

Kapitel 5

Methode

5.1 Grundlagen

5.1.1 Evolutionäre Strategien

Die Natur inspiriert schon seit langen Zeiten die technologische Entwicklung und menschliche Erfindungen. Einer der großen (wenn auch nicht unumstrittenen) Träume der Menschheit ist es, eine Kopie seiner selbst zu erstellen. In diesem Zusammenhang kam die Idee auf, Computerprogramme die »denken« und selbstständig Lösungen finden können, zu entwickeln. Anders als bei deterministischen Ansätzen basieren evolutionäre Strategien auf mehreren konkurrierenden Lösungen, die sich in einem iterativen Prozess einem Ziel nähern. Dabei kann das Optimum durchaus unterschiedliche Formen annehmen, häufig werden nicht komplette Computerprogramme, sondern mathematische Gleichungen oder logische Ausdrücke gesucht. Vorteile dieser Ansätze gegenüber klassischen, deterministischen Optimiermethoden sind:

- Die Tendenz bei ungünstigem Verlauf zu lokalen Optima zu konvergieren, ist geringer. Die Verfahren kommen gut mit Problemen mit vielen lokalen Optima zurecht.
- Startwerte sind von untergeordneter Bedeutung
- Die gleiche Methode kann mit geringfügigen Anpassungen für unterschiedliche Optimieraufgaben eingesetzt werden
- Eine Bearbeitung von diskreten Problemen ist problemlos möglich
- Die Berechnung von Gradienten ist nicht notwendig, so dass die Verfahren auch gut bei diskontinuierlichen und nichtlinearen Problemen eingesetzt werden können.

- Der Algorithmus kann sehr leicht parallelisiert werden
- Eine Normierung von Ein- und Ausgabegrößen, wie es beispielsweise bei neuronalen Netzen oft notwendig ist, entfällt.
- Das Verfahren ist sehr gut für multimodale Optimierungen geeignet.

Neben diesen Vorteilen gibt es jedoch auch einige Nachteile:

- Das Verfahren ist langsam
- Zahlreiche Steuerparameter sind einzustellen
- Das Verfahren hat keine Kenntnis von einem globalen Optimum, weshalb eine Prüfung auf Optimalität oft schwerfällt.

Der Begriff evolutionäre Strategien stellt einen Oberbegriff für eine Reihe unterschiedlicher Techniken und Methoden dar, welche die biologische Evolution als Vorbild nehmen. Der praktische Hintergrund, auf dem die Methoden sich entwickelt haben, war in der Regel bestimmend für die Namensgebung. Leider ist in der Literatur die Benennung der Methoden nicht immer einheitlich:

- Numerische Evolution
  - Genetische Algorithmen, *Genetic algorithms* (GA)
  - Evolutionsstrategische Algorithmen, *Evolutionary strategies* (ES)
- Strukturelle Evolution
  - Genetische Programmierung, *Genetic programming* (GP)
  - Grammatische Evolution, *Grammar Evolution* (GE)
  - Evolutionäres Programmieren, *Evolutionary programming* (EP)
- Klassifizierungssysteme, *Classifier systems* (CS)

Darüber hinaus gibt es noch zahlreiche andere Ansätze, die sich aber von den hier aufgeführten Methoden nur marginal unterscheiden. Gemeinsam ist all diesen Methoden die automatische Generierung von Parametersätzen oder ganzen Computerprogrammen zur Lösung eines speziellen Problems. Ausgehend von einer Startpopulation, die im Allgemeinen nach dem Zufallsprinzip zusammengestellt wird, entwickeln sich die Lösungen durch die Anwendung genetischer Operatoren. Diese genetischen Operatoren sind im Allgemeinen Selektion, Kreuzung und Mutation. Der gesamte Prozeß wird über eine Fitnessfunktion gesteuert, die quantitativ angibt, wie gut oder schlecht eine Lösung ist. Bessere Individuen haben

dann größere Chancen sich zu reproduzieren. Während die Fitness der Individuen mit fortschreitender Generation zunimmt, besteht eine Tendenz, dass die genotypische und phänotypische Diversität[1] abnimmt.

Die Anwendungsgebiete von evolutionären Strategien sind breit gefächert und umfassen so unterschiedliche Gebiete wie Softwareerstellung, Auslegung von elektronischen Schaltkreisen, Parameteroptimierung, Molekularbiologie, Reglerentwurf und andere [49].

Während bei einigen der Methoden (*Numerische Evolution*) lediglich die Parameter eines Problems gesucht werden, werden bei den anderen Methoden (*Strukturelle Evolution*) zusätzlich auch die Struktur, also Gleichungen oder ganze Computerprogramme, bestimmt. Darunter fallen insbesondere die genetische Programmierung und die grammatische Evolution. Bei GP wird jede Lösung durch einen Satz an eigenständigen Softwareobjekten repräsentiert, die gewissermaßen die Gene des Individuums darstellen. Diese Gene stehen in einem definierten Abhängigkeitsverhältnis zueinander und bilden somit einen Lösungsbaum. Bei dem grammatikbasierten Ansatz (GE) wird die Lösung über einen Binärstring dargestellt. Mit Hilfe einer Grammatik wird dieser String in eine mathematisch auswertbare Gleichung oder ein Programm überführt.

Die Evolutionsstrategischen Algorithmen (ES) wurden von Rechenberg [66] begründet und später von Schwefel [71] weiterentwickelt. Ähnlich wie bei GA geht es hier darum, Parameter zu optimieren. ES unterscheidet sich von den anderen Methoden unter anderem dadurch, dass auf Kreuzung verzichtet wird und die Steuerung der Mutation in der Regel adaptiv erfolgt.

Evolutionäres Programmieren (EP) wurde von Lawrence Fogel [31] begründet und dann von seinem Sohn David [30] weiterentwickelt. Ziel dabei ist eine *Finite State Machine* (FSM) zu entwickeln, die eine Serie von Eingangssignalen in eine Reihe von Ausgangssignalen transformiert. Die Transformation basiert auf einem Satz an Zuständen und Zustandsübergangsregeln.

Im weiteren Verlauf der Arbeit wird ein Beschränkung auf drei dieser Methoden erfolgen - nämlich GA, GP und GE - wobei aber die folgenden Ausführungen fast unverändert auch für andere evolutionäre Methoden gilt. Zunächst einige Worte zum grundsätzlichen Ablauf bei evolutionären Strategien:

Ablauf

Der Berechnungsablauf lässt sich in folgenden Schritten zusammenfassen (vgl. auch Abbildung 5.1):

[1] Unter der *genotypischen* Diversität wird die strukturelle Verschiedenheit mehrerer Lösungen verstanden. Dagegen bezeichnet die *phänotypische* Diversität die Verteilung der Fitness-Werte innerhalb der Population.

1. Zunächst wird eine initiale Population einer vorher gewählten Anzahl an Lösungen (Individuen) in zufälliger Weise zusammengestellt. Dieses ist die Startgeneration (0. Iteration).

2. Für jede Lösung wird eine Bewertung ihrer Fähigkeit durchgeführt, die gegebene Aufgabe zu lösen. Diese Bewertung wird *Fitness* genannt.

3. Auf der Basis einer Abbruchbedingung kann nun entschieden werden, ob weitere Iterationen durchgeführt werden. Als Abbruchbedingung denkbar sind die Anzahl der Generationen, die Rechenzeit, die Fitness oder auch der aktuelle Fehler der besten Lösung.

4. Im Rahmen einer Selektion können nun einzelne Lösungen anhand ihres Fitness-Wertes aus der Population entfernt werden. Dies entspricht dem Darwinschen Prinzip des *Survival of the Fittest*.

5. Bei der Kreuzung (Rekombination) werden Teile von Lösungen untereinander ausgetauscht. Bei einigen genetischen Verfahren wird auf diesen Schritt verzichtet.

6. Die Mutation von Teilen führt zu einer zufälligen Veränderung der Lösung.

7. Es wird eine erneute Fitness-Berechnung durchgeführt.

8. Bei einigen Methoden wird nun die beste Elter-Lösung in die nächste Generation kopiert (*Elitismus*). Dadurch ist sichergestellt, dass es keine Verschlechterung im Ablauf der Evolution gibt. Dieser Ansatz ist jedoch nicht unumstritten und wird noch sehr kontrovers diskutiert.

9. Der Prozess wird solange bei Punkt 3 fortgesetzt, bis die Abbruchbedingung erfüllt ist.

Eine wichtige Fragestellung im Zusammenhang mit evolutionären Strategien ist die nach der Fitness einer Lösung. Die Fitness entscheidet über die Wahrscheinlichkeit der Reproduktion. Üblicherweise wird dafür ein mittlerer Fehler $E_m(k)$ des Modellausgangs $y_m(k)$ des k-ten Individuums gewählt. Aus Gründen der Anschaulichkeit wird in dieser Arbeit dafür stets der mittlere absolute Fehler (MAE) benutzt:

$$E_m(k) = \frac{1}{n_d} \sum_{i=1}^{n_d} |y_{s,i} - y_{m,i}(k)| \tag{5.1}$$

mit n_d als Anzahl der Datensätze und y_s dem gemessenen Ausgang. Aus dem Modellfehler kann dann die Fitness Φ berechnet werden. Da im allgemeinen

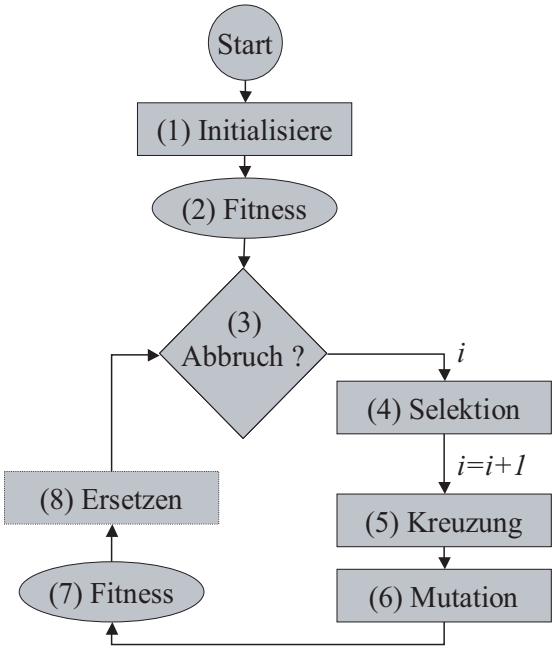


Abbildung 5.1: Grundsätzlicher Ablauf bei evolutionären Strategien

Sprachgebrauch eine steigende Fitness eine stärkere Überlebenswahrscheinlichkeit impliziert, ist mathematisch gesehen die Fitness der Kehrwert des mittleren Modellfehlers E_m.

$$\Phi(k) = \frac{1}{1 + E_m(k)} \tag{5.2}$$

Die Addition einer Konstanten im Nenner des Terms stellt sicher, dass auch Lösungen, deren Fehler E_m sich Null nähert, noch numerisch sicher behandelt werden können. Häufig fließen auch Nebenbedingungen oder Strafterme in die Fitnessberechnung ein.

Ein verbreitetes Phänomen, auf das man im evolutionären Ablauf stößt, ist die Tatsache, dass Teile des Konfigurationsraums von einer Lösung bereits recht gut approximiert werden, andere Teile aber deutlich höhere Fehler als bei anderen Lösungen produzieren. In der Summe schneidet eine solche Lösung dann im Allgemeinen schlecht ab und wird oft aus der Population eliminiert, wobei dann auch die guten Lösungsteile verloren gehen. Selbstverständlich lässt sich nicht immer eine Korrelation von Teilbäumen der Lösung zu Bereichen des Konfigurationsraums herstellen, dennoch lohnt es sich, diesen Lösungen eine größere Überlebenswahrscheinlichkeit zu geben. Darauf basiert die Idee der *partiellen Fitness*, dazu später mehr.

Genetische Algorithmen

Das Konzept der genetischen Algorithmen (GA) wurden von John Holland bereits 1975 entworfen [39] und von ihm selbst [40] und anderen weiterentwickelt. Die grundlegenden Prinzipien sind trotz mancher Abwandlungen des Algorithmus noch bis heute gültig.

Beim klassischen GA wird jede Lösung (Individuum) durch einen Binärstring (Chromosom[2], Genom) repräsentiert. Der String setzt sich aus mehreren Genen, d.h. Teilabschnitten des Strings zusammen. Jedes Gen entspricht einem Parameter des Optimierproblems. Da die Gene und damit der Binärstring eine beliebige Länge annehmen können, ist es möglich, auf dieser Basis nicht nur diskrete Zustände, sondern auch ganzzahlige und reelle Werte oder eine Kombination aus mehreren Datentypen abzubilden. Häufig werden von vornherein reelle Zahlen in einem Vektor zu einem Chromosom zusammengestellt, so dass das Umrechnen von binär zu reell entfällt.

GA sind sehr flexibel und kann auf eine Vielzahl unterschiedlicher Optimierungsprobleme angewendet werden, darunter sind die Bestimmung von Gewichten für ein neuronales Netz [1], die Ermittlung von geeigneten Regeln für einen Fuzzy Logik Regler [44], die Überprüfung der Fehlerfreiheit eines Kommunikationsprotokoll [2] aber auch NP-harte Probleme wie das *Traveling salesman problem* [14].

Genetische Programmierung

John R. Koza [48] erweiterte Hollands Konzept auf eine beliebige Datenstruktur. Dies bietet den Vorteil, dass nun keine Vorkenntnisse über die Struktur einer möglichen Lösung des Problems existieren muss und dass ferner jedes beliebige Problem - sofern es sich nur mathematisch oder logisch formulieren lässt - gelöst werden kann.

Eine Lösung im Sinne der GP besteht aus einer Menge von gültigen Funktionen und Terminalen. Bei den Funktionen können nicht nur arithmetische, trigonometrische oder exponentielle (also im engeren Sinne mathematische) Funktionen, sondern auch Elemente aus der benutzten Computersprache eingesetzt werden, z.B. Verzweigungen, Schleifen etc. Die Terminale bilden Konstanten und die Eingangsgrößen des Problems. Der Funktionsvorrat sollte so gewählt werden, dass eine mathematische Approximation der Lösung damit möglich ist. Bei komplett unbekannten Suchräumen empfiehlt sich eine großzügige Auslegung des Funktionenmenge, es ist jedoch zu bedenken, dass mit der Größe des Funktionsvorrats

[2]Die Gleichsetzung von Individuum, Genom und Chromosom mag verwirrend sein, da in der Natur alle Eukaryoten eine Chromosomenzahl größer 1 aufweisen. Diese Bezeichnung hat sich jedoch im Kontext von evolutionären Strategien eingebürgert.

auch die Komplexität der Suche und damit die Dauer bis zum Erreichen einer optimalen Lösung ansteigt.

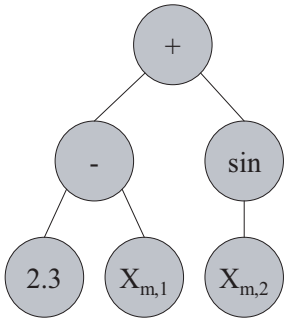


Abbildung 5.2: Beispiel für einen (sehr einfachen) Syntaxbaum in GP. Die Gleichung entspricht $f(x) = (2.3 - x_{m,1}) + \sin x_{m,2}$ mit $x_{m,1}, x_{m,2}$ Eingangsgrößen des Problems.

Koza benutzte LISP als Programmiersprache für die Implementierung seiner Methode. LISP ist durch seine listenorientierte Struktur sehr gut geeignet, mathematische Ausdrücke wiederzugeben. Objektorientierte Programmiersprachen wie C++ sind aber ebenso in der Lage, Programmbäume abzubilden und können somit zur genetischen Programmierung genutzt werden. Es ergeben sich dabei sogenannte Syntaxbäume (vgl. Abb. 5.2), die rekursiv - beginnend beim Wurzelknoten - abgearbeitet werden. Durch die richtige Syntax wird gewährleistet, dass eine geschlossene Lösung erzielt wird (*closure*).

Die Initialisierung der Population am Anfang der Evolution stellt bei der genetischen Programmierung eine besondere Aufgabe dar, da ja die Größe und Struktur der Programme auch zu Beginn sehr unterschiedlich sein können. Es gibt für die Initialisierung eine Reihe unterschiedlicher Ansätze [48]:

- *Grow*: Hier ist nur die maximale Baumtiefe vorgegeben. Baumknoten werden zufällig aus dem Funktionsvorrat ausgewählt. Wird ein Terminal ausgewählt, endet der Ast hier, andernfalls wird die Erstellung mit einer dem Operator angepassten Anzahl an Kindknoten fortgesetzt.

- *Full*: Es werden solange Nicht-Terminale ausgewählt, bis die maximal zulässige Baumtiefe erreicht ist. Auf diese Weise ist garantiert, dass alle Äste die maximale Größe erreichen.

- *Half-and-half*: Dieser Ansatz stellt eine Kombination der beiden anderen dar. Ein Teil der Population wird nach dem *Grow*-, ein anderer Teil nach dem *Full*-Prinzip erstellt.

- *Ramped Half-and-half* funktioniert ähnlich wie *Half-and-half*, nur dass die maximale Tiefe nicht für alle Individuen gleich ist. Startend bei einer Mindesttiefe wird die zulässige Tiefe im Verlauf des Erstellprozesses allmählich erhöht. Auf diese Weise wird eine größere Diversität der Startpopulation erzielt.

GP ist ein äußerst vielseitiges Verfahren, mit dem sich im Prinzip Lösungen für beliebige Probleme finden lassen (sofern vorhanden). Erfolgreiche Anwendungen sind u.a. aus folgenden Bereichen dokumentiert [86]:

- Systemmodellierung
- Steuerung und Regelung
- Optimierung
- Entwurf
- Signalverarbeitung

Es gibt zahlreiche Varianten zum baumbasierten GP, beispielsweise das *Compiling Genetic Programming System* (CGPS) [61] [7] oder Graphen-Modelle wie PADO [79].

Grammatische Evolution

Die Grammatische Evolution benutzt das gleiche Prinzip wie GP und wird deshalb oft als Grammatik-basierte Variante von GP angesehen. Bei der GE wird das Programm in einem Binärstring codiert, anschließend wird GA zur Optimierung genutzt [84] [67]. Eine Grammatik (üblicherweise in der *Backus-Naur*-Form (BNF)) wird herangezogen, um den resultierenden Binärstring in ein Programm zu übersetzen, mit dessen Hilfe dann die Programmausgabe und daraus die Fitness berechnet werden kann. Die BNF Grammatik besteht aus den Nicht-Terminalen N, den Terminalen T und einer Anzahl von geeigneten Produktionsregeln, um aus einem Binärcode (dem Genotyp) ein geeignetes Programm bzw. einen mathematischen Ausdruck zu erstellen[3]. Die genaue Umsetzung dieses Prinzip variiert leicht, das im Rahmen dieser Arbeit benutzte Schema sei im Folgenden an einem einfachen Beispiel kurz erläutert: Für ein Problem aus dem Bereich der symbolischen Regression mit zwei Eingangsgrößen ($x_{m,1}$ und $x_{m,2}$) sind folgende Operatoren denkbar:

- Terminale: $+, -$, sin, cos, Konstanten , $x_{m,1}, x_{m,2}$

[3]Der Begriff Terminale und Nicht-Terminale wird in diesem Kontext anders benutzt als bei der Genetischen Programmierung

(a) Produktionsregeln

Regel	Argumente	Auswahl
<expr>	<op1> <expr>	R(0)
	<expr> <op2> <expr>	R(1)
	<const>	R(2)
	<var>	R(3)

(b) Terminale

Regel	Arg.	Auswahl
<op1>	sin	A(0)
	cos	A(1)
<op2>	+	B(0)
	-	B(1)
<const>	Konst.	C(0)
<var>	$x_{m,1}$	D(0)
	$x_{m,2}$	D(1)

(c) Beispiel:$(23 - x_{m,1}) + \sin(x_{m,2})$

Binär	Ganzzahl	Modulo	Auswahl	Operator
01000001	65	1	R(1)	
00001000	8	0	B(0)	$+$
11010001	209	1	R(1)	
10000101	133	1	B(1)	$-$
00000001	1	1	R(2)	
00010111	23	-	C(0)	23
10100111	167	3	R(3)	
00010010	18	0	D(0)	$x_{m,1}$
10111000	184	0	R(0)	
00000100	4	0	A(0)	sin
00001011	11	3	R(3)	
01001011	75	1	D(1)	$x_{m,2}$

Tabelle 5.1: Aufbau eines GE-Systems für symbolische Regression

- Nicht-Terminale: expr, op1, op2

Mit Hilfe der in den Tabellen 5.1(a) und (b) genannten Produktionsregeln und Terminale kann dann der Programmausgang bestimmt werden. Dabei wird mit einer Regel begonnen und die Argumente der Regel solange durch Terminale und weitere Regeln substituiert, bis der Ausdruck nur noch Terminale enthält. Die Ersetzung erfolgt durch die sequentielle Abarbeitung des Binärstrings (im Allgemeinen byteweise) und die Auswahl der Regel bzw. des Terminals über die Modulo-Funktion.

Als Beispiel möge hier wieder die Gleichung aus Abbildung 5.2 dienen ($(23 - x_{m,1}) + \sin(x_{m,2})$). In Tabelle 5.1(c) ist aufgeführt, wie der Binärstring aussehen müsste und wie er abgearbeitet wird. Für die Darstellung von Konstanten wäre grundsätzlich auch die Einführung von Datentypen größer als 1 Byte denkbar.

Man kann sich aber leicht vorstellen, dass dann bei Mutationen oder Kreuzungen leicht absurd hohe Zahlenwerte zustande kommen, welche die gesamte Lösung unbrauchbar machen. Beschränkt man sich auf Stringabschnitte von 1 Byte Länge lassen sich Werte größer 255 durch Multiplikation oder Addition mehrere Konstanten erzielen.

Da GE auf Binärstrings arbeitet, lassen sich bezüglich der genetischen Operationen die gleichen Regeln anwenden wie bei GA. Vorteile von GE sind darin zu sehen, dass hier - im Gegensatz zu GP - kein Augenmerk mehr auf die syntaktische Korrektheit des Codes zu legen ist. Bei GE ist die zugrundeliegende Grammatik schon so aufgebaut, dass das Programm immer ausführbar ist. Hier tritt jedoch das Problem auf, dass der Binärstring nicht eine ausreichende Länge hat, um die mathematische Gleichung abzuschließen. Man umgeht dieses Problem dadurch, dass man - bei Erreichen des Stringendes - den String rückwärts erneut abarbeitet.

5.1.2 Genetische Operatoren

Selektion

Üblicherweise wird die Selektion im Rahmen eines Turniers durchgeführt [6]. Dabei werden aus mehreren $(n_p \geqslant 2)$, zufällig aus der Gesamtpopulation selektierten Teilnehmern diejenigen $(n_w < n_p)$ Lösungen ausgewählt, die die besten Fitness-Werte besitzen. Die Sieger der Turniere werden dann in die nächste Generation kopiert. Über diese beiden Parameter (n_p, n_w) kann der Selektionsdruck eingestellt werden, üblich sind $n_p = 2$ und $n_w = 1$. Im Allgemeinen wird die Anzahl der Turniersieger aller Turniere in einer Generation identisch sein mit der Populationsgröße um eine gleichgroße Population aufrecht zu erhalten. Um zu verhindern, dass immer die selben Individuen an den Turnieren einer Generation teilnehmen, sollte die maximale Anzahl an Turnierteilnahmen pro Individuum und Generation beschränkt werden. Alternativ zur Turnierselektion sind auch gebräuchlich:

- Fitnessproportionale Selektion: Die zu reproduzierenden Lösungen werden gemäß einer Wahrscheinlichkeit p_s, die proportional zu Ihrer Fitness ist, ausgewählt [39]:

$$p_s(k) = \frac{\Phi_k}{\sum_{j=1}^{n_{pop}} \Phi_j} \tag{5.3}$$

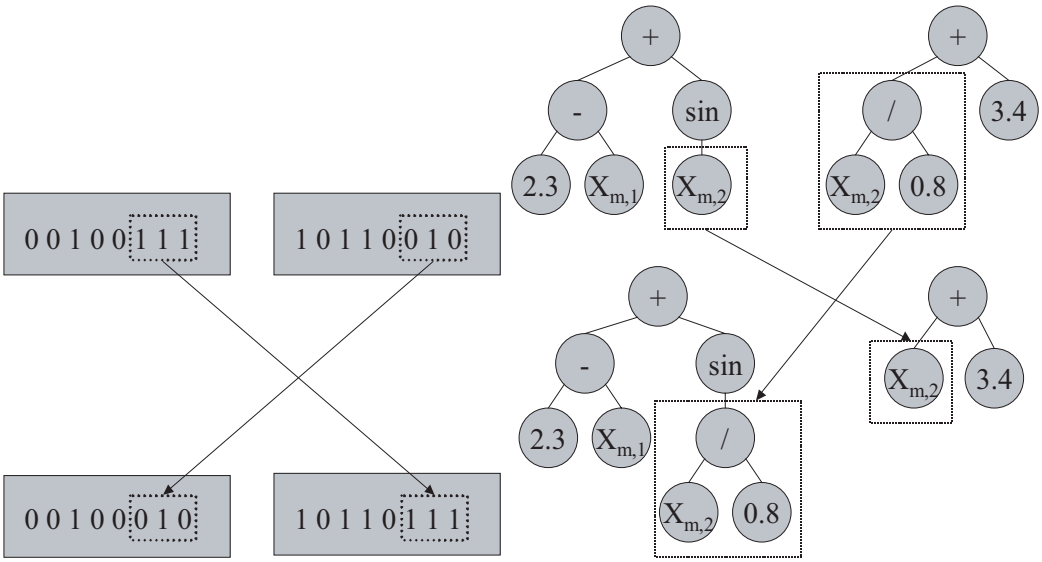
  mit Φ Fitness-Funktion, n_{pop}: Anzahl der Individuen in der Population.

- Rangbasierte Selektion: Die Lösungen werden gemäß ihrer Fitness in einer Rangliste sortiert. Die Wahrscheinlichkeit sich bei der Selektion durchzusetzen steigt dann mit dem Rang des Individuums [85].

Es gibt nun zahlreiche Varianten zu diesen Ansätzen und auch noch weitere Modelle wie etwa FUSS [41]. Für eine detailliertere Übersicht über Selektionsvarianten sei auf [3] verwiesen.

Kreuzung (Rekombination)

Bei der Kreuzung werden Teile zweier Lösungen getauscht. Die Wahrscheinlichkeit, mit der ein Individuum an einer Kreuzungsoperation teilnimmt, wird über die Kreuzungsrate p_c bestimmt. Es existiert eine Reihe unterschiedlicher Ansätze, wie die Kreuzungspunkte ausgewählt werden könnten, der üblichste ist der so genannte Einpunkt-*Crossover* (*Single-point Crossover* oder *One-point Crossover*). Dabei werden bei beiden Kreuzungspartnern an identischen Stellen Teilstringe oder -bäume ausgetauscht (vgl. Abb. 5.3(a)). Beim N-Punkt-*Crossover* wird nicht



(a) Einpunkt-Crossover eines Binärstrings (b) Kreuzung einer Baumstruktur

Abbildung 5.3: Beispiele für Kreuzungsoperationen

ein durchgängiger Teilstring ausgetauscht, sondern mehrere kleinere Teilabschnitte. Eine Variante dazu ist der *Uniform-Crossover* [75], bei dem für jedes Bit bzw. jeden Baumknoten ein (statischer) Wahrscheinlichkeitswert besteht, der bestimmt, ob es ausgetauscht wird oder nicht. Grundgedanke ist bei beiden Verfahren, dass die Elemente einer Lösung, die am meisten zur Fitness beitragen, nicht zusammenhängen, sondern über das Individuum verteilt sind.

Darüber hinaus gibt es noch zahlreiche weitere Kreuzungsoperatoren, wie etwa den *Shuffle Crossover* [16] und *Diagonal Crossover*, die in der Literatur hin-

reichend beschrieben sind. Wird die Lösung nicht binär, sondern direkt mit reellen Zahlen codiert, werden andere Ansätze benutzt [20].

Da sich bei Bäumen die Struktur im Allgemeinen voneinander unterscheidet und identische Stellen nicht existieren, ist bei GP ein Einpunkt-*Crossover* umständlich oder u.U. sogar unmöglich. Obwohl einige Autoren von Vorteilen dieser Kreuzungsvariante berichten [64], hat sich diese Vorgehensweise bei Baumstrukturen nicht durchgesetzt. Deswegen wird hier üblicherweise das in Abbildung 5.3(b) illustrierte Verfahren eingesetzt, bei dem beliebige Teiläste ausgetauscht werden. Kreuzungen bei GP sind insofern problematisch, als bei voll besetzter Baumtiefe beider Kreuzungspartner im Prinzip nur noch eine Kreuzung von Teilästen der gleichen Tiefe stattfindet[4].

Schaffer [69] hat herausgefunden, dass ein Zusammenhang zwischen Populationsgröße und geeigneter Kreuzungsrate besteht. Hohe Raten sind besser für kleine Populationen (vermutlich um die Diversität hoch zu halten), wohingegen bei großen Populationen kleinere Raten geeigneter sind. Andere Autoren [82] haben gezeigt, dass dies auch vom zu lösenden Problem abhängt.

Mutation

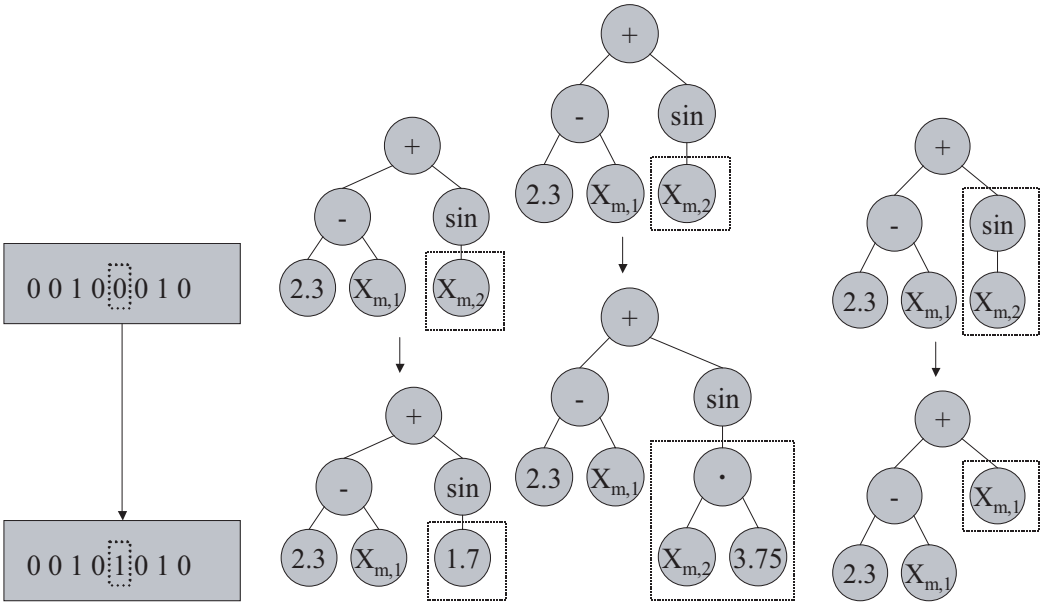
Der Mutationsoperator führt dazu, dass eine zufällige Änderung eines Zahlenwerts oder eines mathematischen Operators durchgeführt wird. Ist die Lösung binär codiert, bedeutet dies nichts anderes, als dass ein Bit seinen Zustand ändert ($0 \leftrightarrow 1$, vgl. Abb. 5.4). Obwohl dies als willkürliche Operation erscheinen mag, bleibt doch festzuhalten, dass es eine ausgeprägte Tendenz gibt, dass der neue Zahlenwert numerisch in der Nähe der alten Lösung liegt.

Die Bedeutung der Mutation liegt vor allem in der Vermeidung von Stagnation und Aufrechterhaltung einer gewissen Diversität, die eine genetische Entwicklung überhaupt erst möglich macht. Die richtige Wahl der Mutationsrate, also der Wahrscheinlichkeit mit der ein gegebenes Gen einer Mutation unterworfen wird, ist mitentscheidend für die Konvergenzgeschwindigkeit. Die Auswahl der geeigneten Mutationsrate hängt nicht nur vom Mutationsoperator, sondern auch vom zu lösenden Problem ab. Für kleine Populationsgrößen ist Mutation der Kreuzung als Evolutionsoperator überlegen [51].

Der tatsächlich eingesetzte Mutationsoperator ist - ähnlich wie im Falle der Kreuzung - abhängig von der Struktur der Lösungen. Es existieren in der Literatur eine Vielzahl von Ansätzen für die unterschiedlichsten Probleme.

Binärcode: Ist die Lösung binär codiert, wählt man häufig eine der folgenden Möglichkeiten:

[4] Dieser Fall tritt nur dann auf, wenn die maximale Baumtiefe begrenzt ist.



(a) Binärstring (b) Baumstruktur (c) *Grow* Mutation (d) *Shrink* Mutation

Abbildung 5.4: Beispiele für Mutationsoperationen

- Jedes Bit des Bitstrings wird mit der Mutationsrate p_m gekippt. Häufig wird $p_m = 1/n_b$ gewählt (n_b: Anzahl der Bits im String)
- Anzahl der Bits wird fest gewählt, Position(en) wird zufällig ausgewählt.

Im Allgemeinen wird der Hammingabstand gleich der Anzahl der Bits gesetzt, d.h. jedes der ausgewählten Bits wird tatsächlich gekippt (und nicht nur zufällig gesetzt oder zurückgesetzt).

Reelle Zahlen: Für der Mutation reeller Zahlen sind die Gauss- (GMO) bzw. Cauchy-Zufallszahl (CMO) basierten Ansätze zu nennen [8] [72], die besonders im Bereich der evolutionären Programmierung verbreitet sind.

$$\sigma_{i+1}(k) = \sigma_i(k) exp(\tau_1 N(0,1) + \tau_2 N_k(0,1)) \quad (5.4)$$

$$y_{m,i+1}(k) = y_{m,i}(k) + \sigma_{i+1}(k) N_k(0,1) \quad (5.5)$$

Wert (y_m) und Standardabweichung (σ) werden für jede Lösung $k = 1, .., n$ mitgespeichert. $N(0,1)$ ist eine normalverteilte Zufallszahl nach Gauss oder Cauchy. Im Gegensatz zu $N_k(0,1)$ gilt sie für die Gesamtpopulation. Die Faktoren τ_1 und τ_2 werden üblicherweise zu $1/(\sqrt{2n})$ bzw. $1/(\sqrt{2\sqrt{n}})$ gewählt. Andere Varianten werden u.a. von [55] und [60] vorgeschlagen.

Ein anderer grundlegender Ansatz benutzt ebenfalls eine normalverteilte Zufallszahl nach Gauss, macht die Berechnung der Mutationsbreite aber abhängig vom numerischen Wert des betroffenen Gens:

$$y_{m,i+1} = y_{m,i} + N(0, y_{m,i} \cdot r_m) \tag{5.6}$$

Die Standardabweichung der Gauss-Zufallszahl N wird auch vom Mutationsbereich r_m beeinflusst. Dies ist eine Variable, die im Laufe der Evolution angepasst werden kann.

Da es sich als äußerst schwierig herausgestellt hat, einen für alle Probleme geeigneten Mutationsoperator zu finden, wird häufig auch folgende (rein heuristische) Regel zur Anpassung der Mutationsschrittweite angewendet:

> Bei einer optimalen Mutation ist genau einer von fünf Mutanten besser als sein Elter. Ist keiner der Mutanten besser, sollte die Streuung gesenkt werden, ist es mehr als einer, sollte sie erhöht werden (um eine höhere Suchgeschwindigkeit zu erzielen).

Diese Regel wird auch als 1/5 Erfolgsregel bezeichnet [66].

Syntaxbäume: Die Mutation von syntaktischen Strukturen betrifft typischerweise nur die Genetische Programmierung. Die folgende Auflistung liefert einen Überblick über die gebräuchlichsten Mutationsoperatoren:

- *Cycle*: Der mathematische Operator eines zufällig ausgewählten Knotens wird verändert. Dabei ist zu beachten, dass die Anzahl der Argumente (Unterbäume) des neuen Knotens denen des alten Knotens entspricht.
- *Shrink*: Eine Funktion wird durch eine andere Funktion mit einer geringeren Anzahl an Argumenten ersetzt (oder durch ein Terminal). Dies ist ein wichtiger Mutationsoperator bei GP, um den Programmbaum und damit die Individuengröße zu verkleinern (vgl. Abb. 5.4(d)). Bei Verzicht auf *Shrink Mutation* kommt es durch Kreuzungen sehr schnell zum beständigen Anwachsen der Individuengröße bis hin zur maximal zulässigen Größe. *Shrink Mutation* kann dem entgegenwirken und damit die Wirksamkeit des Kreuzungsoperator deutlich erhöhen.
- *Sub Tree Mutation*: Ein zufällig ausgewählter Teilast wird um eine Ebene nach oben zu verschoben, ersetzt also seinen Elterknoten. Auch dies ist ein Operator, mit dem die Baumtiefe verringert wird.
- *Grow*: Ein Knoten des Programmbaumes wird durch ein zufällig erzeugten Ast ersetzt (dessen Wurzelknoten kein Terminal ist, vgl. Abb. 5.4(c)). Es ist zu beachten, dass die Obergrenze für die Baumtiefe nicht überschritten

Name	Funktion	Bereich
Binomial3	$f(x_m) = (1 + x_m)^3$	± 1
Quartic	$f(x_m) = x_m^4 + x_m^3 + x_m^2 + x_m$	± 1
Rosenbrock	$f(x_{m,i}\|_{i=1,2}) = \sum_{i=2}^{n} [100(x_{m,1}^2 - x_{m,i})^2 + (x_{m,i} - 1)^2]$	± 2
Rastrigin	$f(x_{m,i}\|_{i=1,3}) = 100 + \sum_{i=1}^{n} [x_{m,i}^2 - 10\cos(2\pi x_{m,i})]$	± 5

Tabelle 5.2: Verwendete Testfunktionen

wird. Es gibt wenige Anhaltspunkte, ab welchem Stadium der Evolution ein Einsatz dieses Operators sinnvoll ist.

- *Switch*: Die Position zweier Unterbäume eines Knotens (Argumente der Funktion) wird vertauscht, d.h. aus $2 - x$ würde dann beispielsweise $x - 2$.

5.1.3 Bewertung

Testfunktionen

Sowohl der grammatische als auch der baumbasierte Ansatz erfordern die Spezifikation einer Reihe von Parametern, die die Konvergenzgeschwindigkeit entscheidend beeinflussen. Um einschätzen zu können, wie stark die Konvergenz von diesen Parametern abhängt, werden die in Tabelle 5.1.3 aufgeführten Testfunktionen als Vergleich herangezogen. Da sich die vorliegende Arbeit Problemen aus dem Bereich der symbolischen Regression widmet, wurden die Testfunktionen entsprechend ausgewählt. Aus diesen Testfunktionen wird ein randomisierter Datensatz von 100 Stützpunkten erstellt, auf dem dann jeweils die Fitness-Berechnung durchgeführt wird.

Rechenaufwand

Um mehrere Varianten miteinander vergleichen zu können, ist ein Maß für die Wirksamkeit einer Methode oder eines Parameters notwendig. Dabei hat sich eingebürgert, als Rechenaufwand einer Population in einer Generation i einfach die Anzahl der Funktionsauswertungen gemäß Gleichung 5.7 zu benutzen:

$$E_{t,i} = n_{pop} \cdot n_{gene,i} \tag{5.7}$$

Obwohl die Funktionsauswertung in der Tat einen großen Anteil an der gesamten Rechenzeit einnimmt, ist dieser Ansatz hier dennoch ungeeignet. Er berücksichtigt zum Einen nicht Zeiten, die für die Durchführung von Mutation, Kreuzung

etc. benötigt wird, ist aber vor allem auch ungeeignet, um GE und GP miteinander zu vergleichen. Es wird deshalb im Folgenden einfach die effektive Rechenzeit als Vergleichsbasis herangezogen. Es muss bei der Durchführung von Versuchen selbstverständlich darauf geachtet werden, dass die Läufe unter gleichen Bedingungen ablaufen. Obwohl dieses Verfahren auch einige Fehlerquellen beinhaltet, ermöglicht es doch einen zuverlässigeren Vergleich.

Einflussgrößen

Zahlreiche Parameter beeinflussen den Erfolg einer Evolution entscheidend. Eine wesentliche Frage ist die der Diversität einer Population. Bei einer zu geringen Verschiedenheit der Lösungen ist ein günstiger Evolutionsverlauf unwahrscheinlich. Eine gleichförmige Population hat die Tendenz in lokalen Optima stecken zu bleiben. Dabei ist nicht nur die phänotypische Diversität, also die Fitness-Verteilung in der Population, sondern auch die genotypische Diversität, also die strukturellen Unterschiede der Lösungen zu betrachten.

Es gibt nun zahlreiche Einflussgrößen, die den Ablauf und die Geschwindigkeit des evolutionären Prozesses beeinflussen. Teilweise sind sie mit einzelnen genetischen Operatoren eng verknüpft und können über diese direkt beeinflusst werden, teilweise ergeben sie sich aber auch aus der Kombination mehrerer Parameter.

Funktionsmenge

Die Auswahl der Funktionen hat einen nicht zu unterschätzenden Einfluss auf Konvergenzgeschwindigkeit und -qualität. Diese Fragestellung taucht natürlich nur bei GP und GE auf, jedoch nicht bei GA. Es ist zwar einerseits darauf zu achten, dass mit den ausgewählten Funktionen das untersuchte mathematische Problem auch beschrieben werden kann, andererseits würde eine zu extensive Zusammenstellung auch den Suchraum unnötig vergrößern und die Berechnungsdauer erhöhen.

Populationsgröße

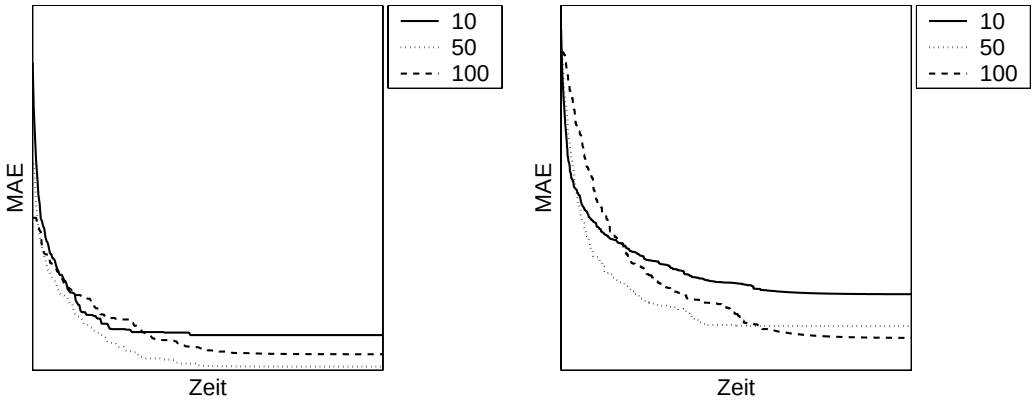
Kleine Populationen konvergieren schneller, neigen aber auch schnell zur Verringerung der Diversität. Wie jedoch aus Abbildung 5.6 ersichtlich, ist die optimale Populationsgröße auch vom untersuchten Problem abhängig.

Selektionsdruck

Über die Anpassung des Selektionsdrucks im Verlauf der Evolution kann die Diversität länger aufrecht erhalten werden. Ein zu hoher Selektionsdruck führt zu einer raschen Abnahme der Diversität und damit zur Gefahr, in einem lokalen Optimum stecken zu bleiben. Ein zu niedriger Selektionsdruck führt zu einer sehr niedrigen Konvergenzgeschwindigkeit (vgl. Abb. 5.6(a)).

Individuengröße

Ein wichtiges Thema ist auch die Begrenzung der Individuengröße. Beim Fehlen



(a) Quartic-Problem (b) Rosenbrock-Problem

Abbildung 5.5: Einfluss der Populationsgröße

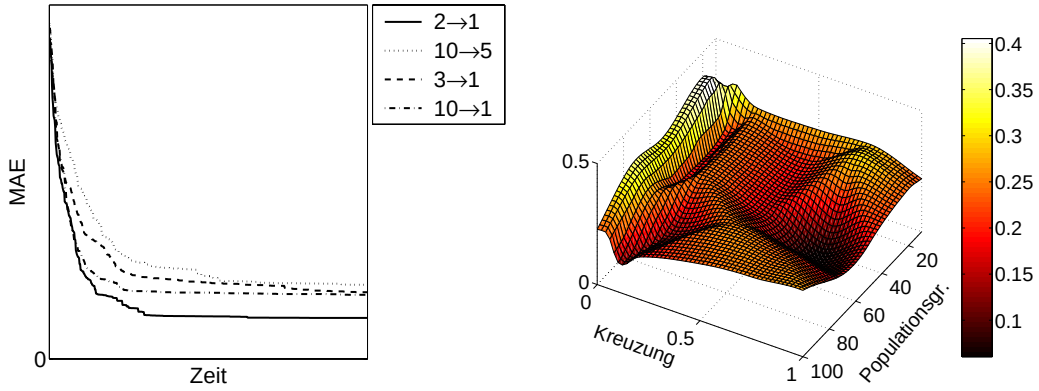
einer Begrenzung entstehen sehr leicht unfunktionale, völlig überdimensionierte Lösungen, die die Berechnungszeit stark in die Höhe treiben. Grund hierfür ist ein Phänomen was in der Literatur als *bloat* bezeichnet wird und durch die Kreuzungsoperation zustande kommt. Die maximale Programmlänge bei der Rekombination muss also begerenzt werden, was dazu führt, dass die Kreuzungspunkte bei beiden zu kreuzenden Bäumen häufig in der Nähe liegen, im Prinzip also nur noch eine lokale Suche durchgeführt wird [63].

Kreuzungsrate

Die Bedeutung der Kreuzungsrate ebenso wie die Wechelwirkung mit anderen Parametern ist bereits ausführlich diskutiert worden. In Abbildung 5.6(b) ist der Einfluss der Kreuzungsrate in Abhängigkeit der Populationsgröße beim Binomial3-Problem dargestellt. Es wird deutlich, dass eine Bestimmung der optimalen Kreuzungsraten kein triviales Problem ist.

Mutationsrate und -bereich

Eine Beurteilung der Auswirkungen von Änderungen der Mutationsparameter allein ist nur schwer, es wirken stets auch andere Parameter mit. Generelle Tendenz ist jedoch, dass sich eine geeignete Mutationsrate positiv auf die Diversität auswirkt, eine zu niedrige Rate häufig eine frühe Gleichförmigkeit der Lösungen zur Folge hat. Am Anfang sollte die Mutationsrate also recht hoch, später (bei der Feinabstimmung) kann sie durchaus sinken. Hohe Mutationsraten sind günstig in hoch dynamischen Umgebungen oder dann, wenn die Population sehr schlecht angepasst ist (wie dies am Anfang eines Optimierungsprozesses die Regel ist). Eine zu hohe Mutationsrate in einem fortgeschrittenen Stadium der Evolution stört hingegen den Verlauf der Optimierung. Es gibt also offensichtlich ein Optimum für die Mutationsrate, dieses ist aber stark von anderen Parametern abhängig und



(a) Einfluss des Selektionsdrucks beim Rastrigin-Problem

(b) Einfluss der Kreuzungsrate in Abhängigkeit von der Populationsgröße beim Binomial3-Problem

Abbildung 5.6: Einfluss verschiedener Parameter

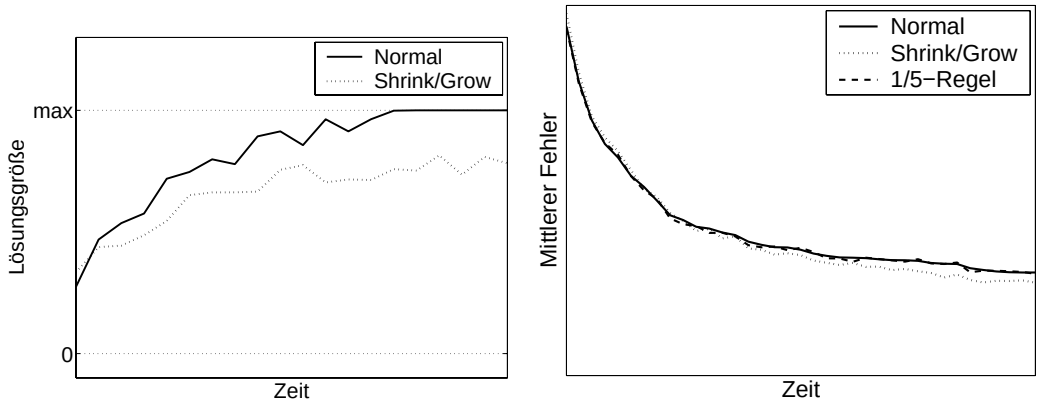
ändert sich im Verlauf der Evolution.

Eine Maßnahme, die Individuengröße zu begrenzen und die Diversität aufrecht zu erhalten, ist der *Shrink*-Mutationsoperator. Abbildung 5.7 zeigt die Wirksamkeit des Operators in Bezug auf die Baumtiefe (Abb. (a)) und seine positiven Auswirkungen auf das Konvergenzverhalten führt (Abb. (b)).

Parameteradaptierung

Aus den bisherigen Ausführungen wird offensichtlich, dass die Einstellung geeigneter Parameter wie Kreuzungsraten, Mutationswahrscheinlichkeiten, Populationsgrößen und anderen zugleich schwierig wie auch entscheidend für den Erfolg der Evolution ist. Vielfach wird das Anpassen der Parameter durch Ausprobieren oder anhand einfacher heuristischer Regeln durchgeführt. Das ist oft zeitaufwändig, fehleranfällig und passt nicht zum Konzept evolutionärer Strategien. Ein Schema zu finden, wie diese Parameter automatisch zu finden sind, ist deshalb ein dringendes Anliegen. Daher ist es nicht verwunderlich, dass zahlreiche Untersuchungen zu diesem Thema existieren. Die Ansätze können nach [37] unterschieden werden:

- Deterministisch: Nur anhand einer vor dem Beginn der Evolution festgelegten Regel werden Parameter während des Ablaufs verändert. Vorläufige Ergebnisse der Evolution fließen nicht in die Parameteränderung ein.
- Adaptiv: Die Parameter werden anhand von Rückmeldungen/Ergebnissen der Evolution selbst verändert. Ein Beispiel dafür ist die bereits erwähnte 1/5 Erfolgsregel zur Anpassung der Mutationsrate.



(a) Wirkung der *Shrink*- und *Grow*-Mutation auf die mittlere Lösungsgröße

(b) Auswirkung verschiedener Mutationsoperatoren

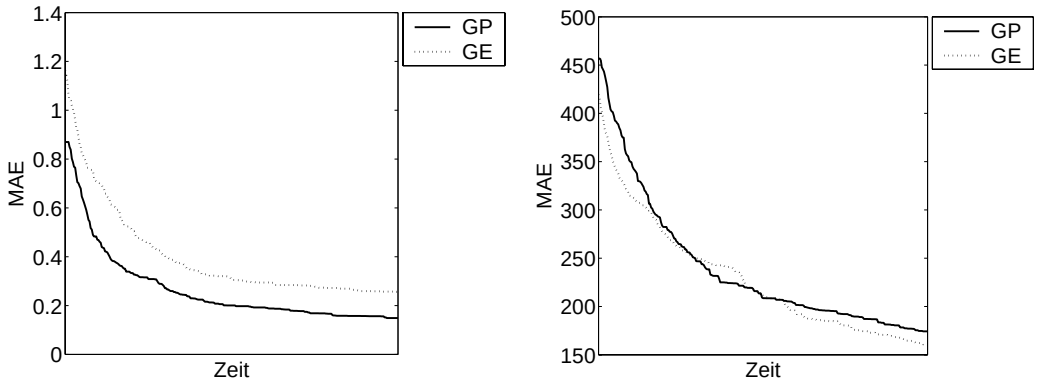
Abbildung 5.7: Einfluss verschiedener Mutationsoperatoren (Rosenbrock-Problem, Mittelwert aus 50 Läufen)

- Selbst-Adaptiv: Hier werden die Parameter direkt auf dem Chromosom codiert und dann den genetischen Operationen unterzogen. Ein Beispiel dafür ist die Co-Evolution [82], ein anderes HAEA [35]. Diese Ansätze beziehen sich jedoch nur auf Mutations- und Kreuzungsrate.

Eine gute Übersicht über das Thema findet sich in [23]. Die in dieser Arbeit vorgeschlagene Methode ist in Abschnitt 5.2 erläutert.

Vergleich von GP und GE

Genetische Programmierung und Grammatische Evolution sind zwei Verfahren, die sich vor allem durch die Art der Genrepräsentation unterscheiden. Daraus ergeben sich jedoch auch weitreichende Folgen für einige Operationen, etwa für die Durchführung der Mutation und Kreuzung. Die Frage steht deshalb im Raum, welches der beiden Verfahren für die vorliegende Arbeit geeigneter ist. Die beiden wesentlichen Beurteilungskriterien sind die Konvergenzgeschwindigkeit und die Qualität der Lösung. Wie sich schon aus einem einfachen Vergleich der beiden Ansätze für zwei Beispiel-Probleme zeigt (vgl. Abb. 5.8), ist eine eindeutige Aussage kaum zu treffen, vielmehr hängt die Auswahl offensichtlich auch vom untersuchten Problem ab. Die Frage, welches Verfahren für Anwendungen der Mikrorobotik geeigneter ist, muss deshalb an geeigneter Stelle in den folgenden Kapiteln beantwortet werden.



(a) Binomial3-Problem (b) Rosenbrock-Problem

Abbildung 5.8: Fehlerverlauf bei GP und GE im Vergleich

5.1.4 Parallelisierung

Die genetischen Operationen wie Reproduktion, Rekombination und Mutation verursachen einen relativ geringen Rechenaufwand und erfordern deshalb vergleichweise wenig Rechenzeit. Anders sieht es aus bei der Fitness-Berechnung, sie ist verantwortlich für nahezu 90% der Rechenzeit in einem Rechenzyklus. Eine Verringerung des Rechenaufwands kann aber durch eine effizientere Form

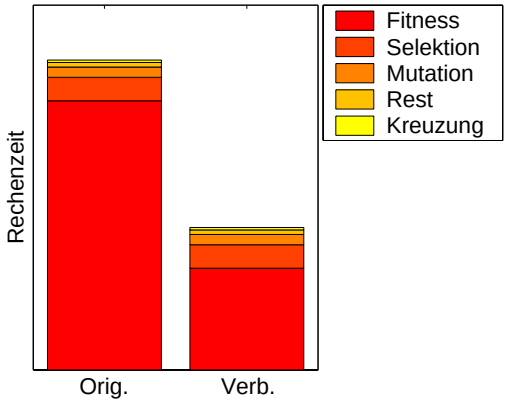


Abbildung 5.9: Zeitaufwand für die einzelnen genetischen Operationen

der Berechnung erzielt werden: Statt bei jeder Fitness-Berechnung den gesamten Baum durchzugehen, wird bei Änderungen (Kreuzung oder Mutation) nur der betroffene Teilast und die darüber liegenden Knoten neu berechnet. Die Rechenzeit

kann dadurch ca. halbiert werden[5] (vgl. Abb. 5.9).

Bei mathematisch einfachen Problemen ist die resultierende Rechenzeit bis zum Erreichen des Optimums im Allgemeinen vernachlässigbar. Bei komplexen Problemen kann jedoch der Rechenaufwand auch bei Nutzen von modernen Prozessoren im Bereich von mehreren Stunden für eine komplexe Optimierungsaufgabe liegen.

Glücklicherweise sind genetische Strategien gut geeignet für parallele Anwendungen, eine Eigenschaft die dazu genutzt werden kann, Rechenzeit zu sparen. Damit wird es dann auch möglich, größere Populationen zu bilden, mit denen hochkomplexe Optimierungsaufgaben gelöst werden können. Dieser Ansatz wird auch Parallele Genetische Algorithmen (PGA) bzw. Programmierung (PGP) genannt (im Gegensatz zur sequentiellen Abarbeitung). Neben der Beschleunigung des Ablaufs ergibt sich häufig auch eine bessere Qualität der Lösung, vor allem hervorgerufen durch die Tatsache, dass im evolutionären Verlauf länger eine hohe Diversität der Gesamtpopulation erhalten bleibt.

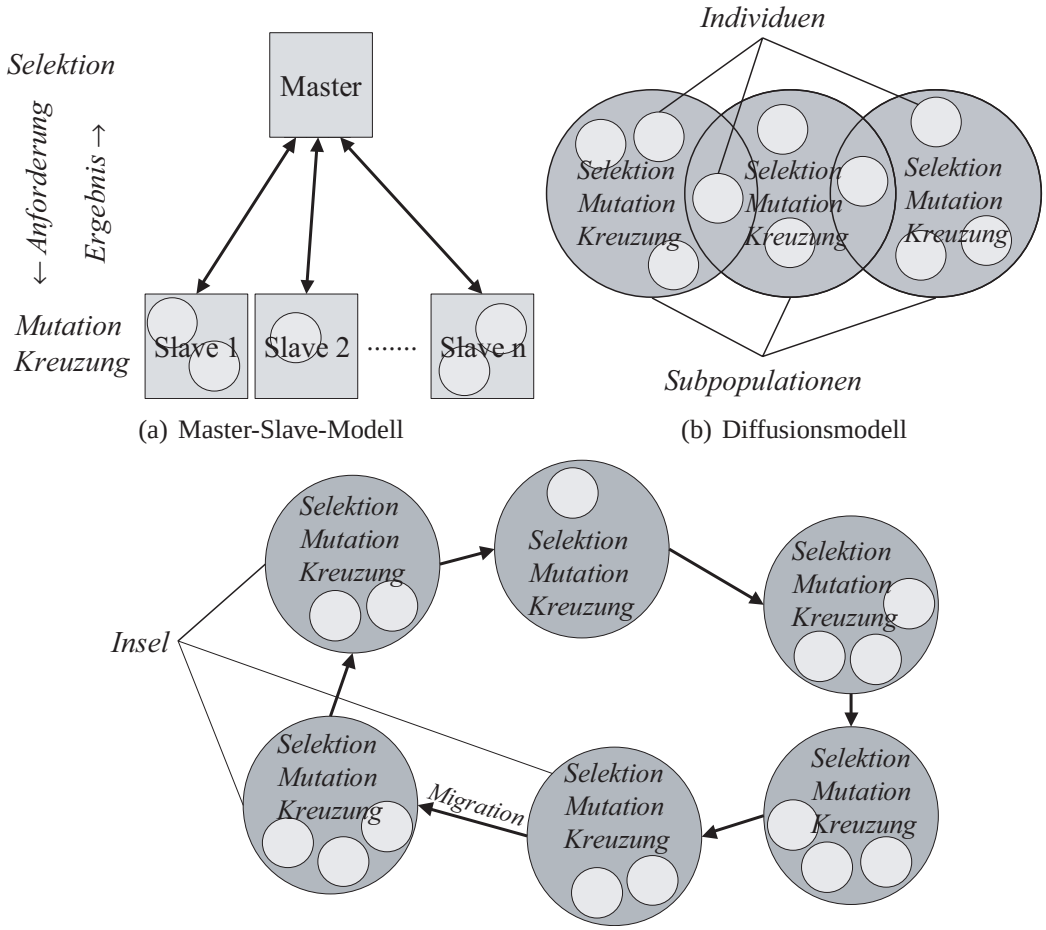
Erste Versuche genetische Algorithmen auf mehreren Rechnern parallel laufen zu lassen gehen zurück auf Bethke [11]. Weitere Untersuchungen an GA wurden unter anderem durchgeführt von Grefenstette [36], Tanese [77], Alba [3] und Golub [34]. Unter anderem Andre [4] und Eklund [24] erweiterten das Prinzip auf GP. Konfrst [46] liefert einen guten Überblick über die Forschungstätigkeiten der letzten Jahre in diesem Bereich. Es haben sich bei der Aufteilung der Anwendungen eine Reihe unterschiedlicher Ansätze herausgebildet, die sich aber alle einem der drei grundsätzlichen Modelle zuordnen lassen, die im Folgenden näher beschrieben werden.

Farming Modell

Beim *Farming* Modell (auch Master-Slave-Modell) existiert eine globale, panmiktische Population[6], jedoch wird die Bearbeitung der Individuen, d.h. die Fitness-Berechnung, Mutation usw. auf die zur Verfügung stehenden Prozessoren verteilt. Nur die Selektion wird zentral durchgeführt (vgl. Abb. 5.10(a)). Es ist offensichtlich, dass bei diesem Modell ein hoher Kommunikationsbedarf entsteht. Ferner ist hier das Problem der Synchronisierung stets präsent, d.h. bevor mit dem nächsten Arbeitsschritt begonnen wird, muss auf die Ergebnisse des langsamsten Rechners gewartet werden.

[5] Die genaue Quote hängt natürlich von vielen Faktoren ab, z.B. der Baumtiefe, der Populationsgröße, der Verteilung der mathematischen Operatoren u.a.

[6] *panmiktisch*=Mischung vorherrschend, d.h. die gesamte Population lebt unter den gleichen Bedingungen in einer gemeinsamen Gruppe



(a) Master-Slave-Modell

(b) Diffusionsmodell

(c) Ein Inselmodell in Ring-Topologie

Abbildung 5.10: Parallelisierungsmodelle

Diffusionsmodell

Das Diffusionsmodell (auch *Fine-grained Distributed Population* Modell) zeichnet sich durch eine andere Aufteilung der Individuen aus: Mehrere Individuen bilden eine Subpopulation, wobei jedoch einige Individuen mehreren Subpopulationen angehören (vgl. Abb. 5.10(b)). Die genetischen Operationen (also auch Selektion und Kreuzung) werden nur innerhalb dieser lokalen Umgebung durchgeführt. Dadurch, dass einige Lösungen mehreren Subpopulationen angehören, können gute Lösungen auch ohne explizite Migration durch die Gesamtpopulation propagiert werden.

Inselmodell

Beim Insel-Modell (auch *Coarse-grained Distributed Population* Modell) wird die Gesamtpopulation in mehrere Gruppen (Inseln, Deme) eingeteilt, die sich dann weitestgehend unabhängig voneinander mit Hilfe der bekannten genetischen Operatoren entwickeln. Zu einem definierten Zeitpunkt findet dann eine Migration einzelner Individuen zu anderen Inseln statt (vgl. Abb. 5.10(c)). Im Extremfall, nämlich im Falle einer Migration zu jeder Generation, nähert sich dieses Modell einer panmiktischen Population an. Der Kommunikationsbedarf ist bei diesem Modell nur gering. Ein wichtiges Thema ist hier jedoch die Migration. Bei der synchronen Migration findet die Migration mit einer vorher festgelegten Frequenz auf allen Inseln gleichzeitig statt. Denkbar ist aber natürlich auch eine asynchrone Migration, die nur bei Vorliegen bestimmter Kriterien stattfindet, etwa abhängig von der Fitness der Inselbewohner.

Allgemein

Generell lassen sich folgende Anforderungen an eine Verteilung aufstellen:

- Kommunikationsaufwand sollte gering sein, um den Zeitvorteil der Parallelisierung nicht zu egalisieren.
- Architektur sollte skalierbar sein, d.h. mit einer unterschiedlichen Anzahl an Rechnern arbeiten können
- Struktur sollte so flexibel sein, dass verschiedene Probleme mit unterschiedlichen Operatorsätzen bearbeitet werden können

Die Aufteilung der Population in mehrere Subpopulationen bietet im Vergleich mit einer einzigen panmiktischen Population auch Vorteile für die Aufrechterhaltung der Diversität [81] und damit häufig zu besseren Lösungen. Dies betrifft sowohl die genotypische als auch die phänotypische Vielfalt. Anders als bei anderen Ansätze zur Erhöhung der Diversität ist die Aufteilung der Population in einzelne Gruppen praktisch umsonst, d.h. fast ohne zusätzlichen Rechenaufwand zu bekommen.

5.2 Eigene Entwicklung

Existierende evolutionäre Strategien weisen eine Reihe von Nachteilen auf:

- Sie konvergieren oft langsamer als andere Suchstrategien

- Die Parametrierung hat beträchtlichen Einfluss auf die Konvergenz, ist aber von vielen Einflussgrößen abhängig und deshalb schwierig.

- Die Aufrechterhaltung der Diversität ist oft problematisch, aber entscheidend für den Erfolg des Verfahrens.

Eine mögliche Lösung für das erste und dritte Problem ist die Parallelisierung des Prozesses, für das zweite hingegen adaptive und selbstadaptive Verfahren zur Parameteranpassung.

Eine naheliegende Idee ist es nun, beide Dinge miteinander zu kombinieren, d.h. ein parallelisiertes Modell zu benutzen, dessen Parameter in einem selbstadaptiven Prozess optimiert werden. Miki et. al [56] haben für den Kontext von GA ein Verfahren namens PDGA (*Parallel Distributed Genetic Algorithms*) entwickelt, bei dem die Population in mehrere Subpopulationen aufgeteilt wird, die unterschiedliche - jedoch konstante - Mutations- und Kreuzungsraten haben.

Im Bereich der Genetischen Programmierung sind Überlegungen zur Parallelisierung neuer und weniger gut erforscht [80]. MGP (*Meta-Genetic Programming*) ist ein Ansatz [22] bei dem parallel zum eigentlichen Programmbaum ein Operatorbaum im evolutionären Prozess entwickelt wird. Dieser Operatorbaum dient dann dazu, denn Programmbaum zu verändern und ersetzt auf diese Weise den Kreuzungsoperator.

Da alle existierenden Verfahren nur einen Teil der Evolutionsparameter adaptieren (meist Kreuzungs- und/oder Mutationsrate), wird nun im Folgenden eine Methode entwickelt, die die Adaptierung aller relevanten Parameter in einem meta-evolutionären Prozess zum Gegenstand hat. Das Ziel dabei ist, die Parametrierung von evolutionären Abläufen einfacher zu gestalten, die Vorteile der Parallelisierung zu nutzen und bessere Lösungen zu erzielen.

5.2.1 Verfahren

Bei der entwickelten Methode sind nun nicht mehr nur die Lösungen selbst, sondern zusätzlich die Evolutionsparameter Gegenstand des Optimierungsprozesses. Auf die biologische Sicht übertragen heißt das, dass Inseln (Deme oder Subpopulationen) gebildet werden, deren genetische Weiterentwicklung unterschiedlich abläuft. Zwischen den Inseln findet eine Migration einzelner Individuen nach einem vorher definierten Schema statt. Dies entspricht dem Ablauf beim Insel-Modell. Zusätzlich wird aber nach einer bestimmten Anzahl von Perioden ein eigener (meta-)evolutionärer Prozess gestartet, der nun nicht die Entwicklung der Individuen, sondern deren Steuerparameter zum Ziel hat. Es findet nun also eine separate Selektion, Mutation und Kreuzung der Parameter statt (vgl. Abbildung 5.11 und Listing 5.1).

Folgende Evolutionsparameter sind Gegenstand dieses Optimierungsprozesses:

- Kreuzungsrate
- Mutationsrate
- Populationsgröße
- Individuengröße = Baumtiefe, Programmlänge
- Selektionsdruck
- Funktionsmenge

Selbstverständlich können problemlos weitere Parameter in diesen Prozess einbezogen werden. Für die Meta-Evolution wird ein genetischer Algorithmus benutzt, der mit reell codierten Zahlen arbeitet.

Der Start von Migration und Meta-Evolution wird über die Anzahl der abgelaufenen Perioden gesteuert. Eine Periode kann eine bestimmte Anzahl an Generationen sein. Dies wird aber der Unterschiedlichkeit des Ablaufs auf den einzelnen Inseln nicht gerecht. Da unterschiedliche Werte für die Evolutionsparameter auch unterschiedlichen Rechenaufwand zur Folge haben und ferner die Inseln unter Umständen auf unterschiedlichen Prozessoren bearbeitet werden, ist es notwendig, eine Nivellierung über die pro Periode zur Verfügung stehende Rechenkapazität und -zeit durchzuführen. Nur so ist eine Vergleichbarkeit der Ergebnisse zu erzielen.

Die Initialisierung der Evolutionsparameter wird auf Basis der vom Benutzer vorgegebenen Werte randomisiert durchgeführt. Dabei erhält eine der Inseln genau die vorgegebenen Parameterwerte, die andere zufällig abgewandelte.

Die Meta-Selektion benutzt als Steuergröße die Fitness der jeweilig besten Lösungen einer Insel. Es sind dabei zwei Herangehensweisen denkbar:

1. Im Rahmen der Selektion werden die gesamten Inseln einschließlich ihrer Individuen reproduziert.

2. Nur die Evolutionsparameter sind Gegenstand der Reproduktion.

Eine komplette Reproduktion bzw. Vermehrung der Inseln (Variante 1) ist jedoch mit einem Verlust an Diversität verbunden, was der Grundidee des Verfahrens widerspricht. Deshalb ist es sinnvoller, nur die Metaparameter zu optimieren, die Inselns selbst im Rahmen der Meta-Evolution aber unverändert zu lassen.

Ein Sonderfall in diesem Zusammenhang tritt bei der Veränderung der Funktionsmenge auf: Die vorhandenen Lösungen können unter Umständen unbrauchbar

```
i:=0
Initialisierung
Auswerten
for k=1 to n_ins do                          // Evolution = GP
    i:=0                                  // Generationenzähler initial.
    while ι(P(i,k)) ≠ true do             // Abbruchbedingung
        Selektion
        Kreuzung
        Mutation
        Auswerten
        Ersetzen
        i:=i+1                            // Generationenzähler
    end while
end for
j:=j+1                                 // Periodenzähler
if (j mod N_mig = 0) then              // Migration
    Migration
end if
if (j mod N_me = 0) then               // Meta-Evolution = GA
    Selektion der Evolutionsparameter
    Kreuzung der Evolutionsparameter
    Mutation der Evolutionsparameter
end if
```

Listing 5.1: Pseudocode der Meta-Evolution

werden (weil beispielsweise ein genutzter Operator nun nicht mehr zugelassen ist). In einem solchen Fall wird der entsprechende Operator - sofern möglich - durch einen anderen Operator mit der gleichen Anzahl Argumente ersetzt. Andernfalls wird die betroffene Lösung gelöscht und durch eine zufällig generierte ersetzt.

Eine Verminderung der Baumtiefe (bei GP) hat einen ähnlichen Effekt: Würden die Knoten ab einer bestimmten Baumtiefe einfach gelöscht, wäre die Geschlossenheit der Lösung unter Umständen nicht mehr gegeben. In diesem Fall werden die Nicht-Terminale der untersten Ebene durch zufällig ausgewählte Terminale ersetzt.

Bei einer Erhöhung der Populationsgröße wird die Population mit zufälligen Lösungen aufgefüllt. Bei einer Verringerung der Populationsgröße werden zufällig Lösungen gelöscht (mit Ausnahme der besten Lösung).

Durch die Parallelisierung als solche wird eine größere Diversität der Inseln untereinander erzielt. Um auch die Diversität innerhalb einer Insel zu erhöhen, geht in die Fitnessbestimmung neben dem Modellfehler auch ein Maß der Diver-

sität ein. Diese Idee basiert auf dem Prinzip des *fitness sharing*[33]. Dabei wird der Abstand zweier Lösungen voneinander berücksichtigt. Hier gehen nur rein statistische Größen wie die Baumtiefe, die Anzahl der Gene und das Auftreten der mathematischen Operatoren ein. Es wird dabei die Distanz von einem Individuum zu allen anderen Lösungen einer Population gemessen und anhand der mittleren Distanzen ein Ranking der Individuen erstellt (Rg_{div}). Diese Rangliste wird dann mit einer anderen Rangliste, die auf dem Modellfehler basiert (Rg_{err}), aufgrund einer einstellbaren Gewichtung (r_{div}) gemittelt und ergibt eine (rangbasierte) Fitness eines Individuums:

$$\begin{aligned} Rg_i &= r_{div} \cdot Rg_{div,i} + (1 - r_{div}) \cdot Rg_{err,i} \\ Rg_{div,i} &= Rg \left(0.25 \sum_{j=1}^{n_{pop}} |n_{lev,j} - n_{lev,i}| + 0.25 \sum_{j=1}^{n_{pop}} |n_{gene,j} - n_{gene,i}| + \right. \\ &\quad \left. + 0.5 \sum_{j=1}^{n_{pop}} \sum_{k=1}^{n_{oper}} |n_{op,k,j} - n_{op,k,i}| \right) \end{aligned} \tag{5.8}$$

Im Gegensatz zu anderen Methoden der Diversitätsmessung ist dieser Ansatz fast ohne zusätzlichen Rechenaufwand umsetzbar.

Eine andere Maßnahme zur Erhaltung der Diversität ist die der partiellen Fitness: Für jeden Trainingsdatensatz wird zunächst der Modellfehler bestimmt. In die Fitnessberechnung wird dann aber nur das beste Quartil dieser Modellfehler einfließen. Die damit verbundene Erwartung ist, dass Modelle, die in bestimmten Bereichen des Suchraums recht gut, in anderen aber sehr schlecht sind, bevorzugt werden. Sie könnten bewahrenswerte Elemente (Teilbäume) enthalten, die vorteilhaft in eine größere und bessere Lösung einfließen können.

5.2.2 Aufteilung

Ein Problem, das im Falle einer Auslagerung auf mehrere Rechner zu klären wäre, ist die Verteilung der Inseln auf die zur Verfügung stehende Hardware. Dabei muss zunächst die Rechenleistung der Prozessoren bestimmt werden. Eine Möglichkeit wäre die theoretische Berechnung der Geschwindigkeitsunterschiede aufgrund von (hardware-)technischen Daten der Rechner: Prozessortakt und Prozessoraufbau (Befehlssatz), Arbeitsspeicher, Speicherhierarchie, Grafikkarte (wenn Ausgaben zu machen sind), Festplatte, Betriebssystem etc. Diese Variante ist jedoch zu aufwändig und letztlich auch zu unsicher. Stattdessen wird vor dem eigentlichen Start der Evolution eine Kalibrationsphase eingeleitet, bei der ein identischer (also nicht zufallsgesteuerter) und für die Evolution typischer Ablauf auf allen Rechnern durchgeführt wird. Die Zeit, die diese Evolutionsläufe in Anspruch nehmen, werden dann gespeichert und dienen als Basis für die Verteilung

der Inseln (Rechenzeitäquivalent). Um eine Bewertung der Evolutionsparameter der Inseln vornehmen zu können, ist es wichtig, dass im evolutionären Ablauf jeder Insel das gleiche Rechenzeitäquivalent für eine Periode zur Verfügung steht.

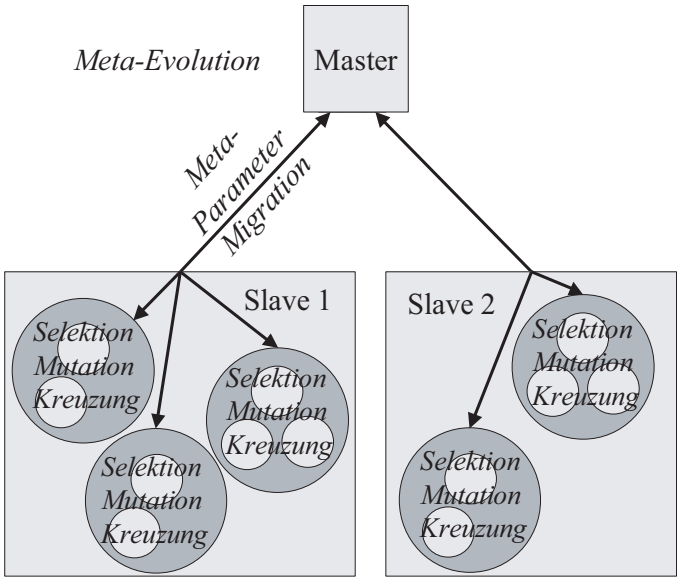


Abbildung 5.11: Aufteilung der Inseln bei der Meta-Evolution

Die Aufteilung kombiniert Aspekte des Master-Slave-Modells und des Insel-Modells (vgl. Abb. 5.11): Auf einem der Rechner läuft die Meta-Evolution ab und dieser koordiniert auch die Verteilung der Migranten. Ferner wird von hier aus die anfängliche Parametrierung der Inseln als auch die Veränderung der Parameter im meta-evolutionären Verlauf durchgeführt. Jeder der Inseln wird das gleiche Rechenzeitäquivalent zugeteilt.

Mehrere Inseln können selbstverständlich auf dem gleichen Rechner angesiedelt sein, d.h. Hard- und Software sind vollständig voneinander getrennt zu betrachten. Die Kommunikation zwischen den Inseln und dem Master-Rechner findet über Ethernet TCP/IP statt.

5.2.3 Implementierung

Um eine bequeme Bedienbarkeit zu gewährleisten, wurde eine Benutzeroberfläche mit Qt[7] erstellt, über die sich alle Parameter einstellen lassen (vgl. Abb. A.1 im Anhang). Parametersätze können auch gespeichert und später wieder geladen wer-

[7] Qt ist eine portable Klassenbibliothek der norwegischen Firma *Trolltech*

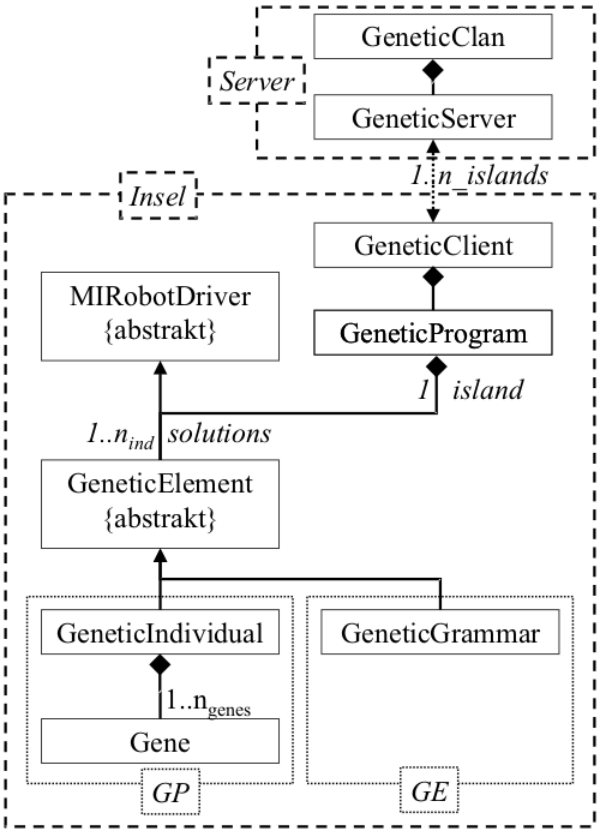


Abbildung 5.12: Kollaborationsdiagramm

den (XML-Format, siehe Listing A.1), wodurch sich sehr leicht längere Testreihen aufnehmen lassen.

Am Ende jeder Evolution werden die Ergebnisse zusammen mit den benutzten Parametern und dem Verlauf der Evolution in einer Ergebnisdatei (ebenfalls im XML-Format) gespeichert. Auch diese Datei kann später wieder geladen werden, um die Berechnung fortzusetzen. Es ist ferner möglich, ein Prototyp-Modell zu laden, das eine der Lösungen in der Startpopulation bildet. Die anderen Lösungen werden weiterhin zufällig zusammengestellt.

Der Code ist in C++ geschrieben. C++ lässt einen rekursiven Aufruf von Methoden zu, was eine Voraussetzung für die Umsetzung als genetisches Programm ist. Die Implementierung des Verfahrens ist jedoch so flexibel, dass gleichermaßen GP wie auch GE-Programme eingebaut sein können.

Bei der Baumstruktur werden die Verwandtschaftsverhältnisse in der Klasse *Gene* gespeichert (vgl. Abb. 5.12). Jedes Gen speichert neben den Verweisen

zu Kindgenen (Argumente der Funktion) auch eine unterschiedliche Anzahl von Zahlenwerten (Speicherwerte k_s). Diese werden zu Beginn der Evolution mit zufälligen Werten initialisiert und bleiben dann - je nach Bedeutung des Operators - entweder konstant oder verändern sich anhand der angegebenen Kriterien. Die Details zur Berechnungsweise der einzelnene Operatoren sind in Tabelle A.1 im Anhang aufgeführt.

Es können neben allgemein gebräuchlichen arithmetischen Operatoren auch Fallunterscheidungen und spezielle regelungstechnische Funktionen genutzt werden. In der Regelungstechnik werden häufig PT_n-Funktionen zur Streckenbeschreibung herangezogen. Die PT_1-Funktion ist wie folgt definiert:

$$f_{PT1} = y_m = K_R \left(1 - \exp\left(-\frac{t}{T_1}\right)\right) x_m \tag{5.9}$$

Dabei sind x_m als Argument und K_R, T_1 und t als Speicherwerte des Operators zu behandeln. Die Zeit t wird der eingestellten Taktzeit gemäß hochgezählt, jedoch bei jeder Veränderung von x_m zurückgesetzt.

Ein PT_2-Übertragungsverhalten ist durch folgende Differentialgleichung (DGL) charakterisiert:

$$\frac{1}{\omega_0^2}\ddot{y}_m + \frac{2D_r}{\omega_0}\dot{y}_m + y_m = K_R x_m \tag{5.10}$$

Eine Lösung dieser DGL liefert eine Funktion für f_{PT2}. In Abhängigkeit vom Dämpfungsfaktor D_R ergeben sich unterschiedliche Lösungen, die in der Literatur hinreichend dokumentiert sind, z.B. in [52]. Bei der PT_2-Funktion sind neben dem Argument x_m auch vier Speicherwerte zu berücksichtigen, nämlich die Eigenkreisfrequenz des ungedämpften Systems ω_0, der Dämpfungsfaktor D_R, die Verstärkung K_R und die Zeit t.

Eine Reihe von Funktionen erfordert die Kenntnis einer Taktrate Δt bzw. der aktuellen Zeit t. Details zur Bestimmung dieser Werte werden im Kapitel über die regelungstechnischen Anwendungen des Verfahrens (7.3.3) gegeben.

5.2.4 Vergleich

Um eine Bewertung des vorgestellten Modells zu ermöglichen, wird im Folgenden ein Vergleich von vier Modellen durchgeführt:

- Panmiktische Population (*GP-PM*)
- Insel-Modell ohne Migration (*GP-IM*)
- Insel-Modell mit Migration (*GP-IMM*)
- Insel-Modell mit Migration und Meta-Evolution (*GP-IMME*)

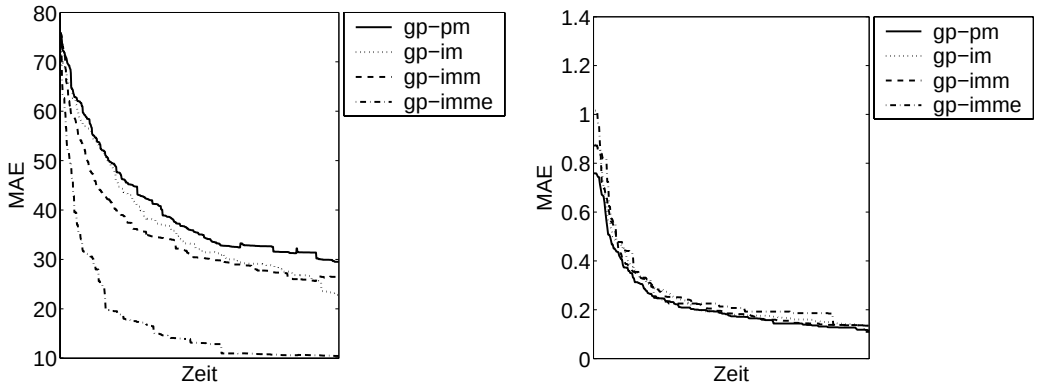
Testfunktion	Kürzel	Min. Fehler	Mittl. Fehler	Treffer-quote[8] [%]	Standard-abw.
Binomial3	GP-PM	0	0.092	10	0.048
	GP-IM	0	0.095	32	0.05
	GP-IMM	0	0.088	6	0.045
	GP-IMME	0	0.084	24	0.04
Quartic	GP-PM	0	0.016	32	0.019
	GP-IM	0	0.017	32	0.029
	GP-IMM	0	0.011	66	0.021
	GP-IMME	0	0.012	54	0.041
Rastrigin	GP-PM	8.8	29.5	0	16.3
	GP-IM	8.3	22.8	0	14.6
	GP-IMM	9	25.6	0	13.9
	GP-IMME	8.7	10.4	0	0.7
Rosenbrock	GP-PM	10.9	104.2	0	42.5
	GP-IM	24.1	152.7	0	56
	GP-IMM	16.3	89	0	37.6
	GP-IMME	0	82.9	2	31

Tabelle 5.3: Ergebnisse für verschiedene Aufteilungsschemata bei GP. Ergebnisse für jeweils 50 Durchläufe.

Die Analyse der Ergebnisse für die Testfunktionen zeigt zunächst einmal, dass je nach Komplexität der Testfunktion eine mehr oder weniger gute Approximation gelingt. Bei einfachen Funktionen - wie beim Quartic- und Binomial3-Problem - finden alle vier Varianten bei einer gewissen Prozentzahl der Versuche die gesuchte Funktion, wenn auch teilweise in einer mathematisch umständlichen Form (vgl. Tabelle 5.3). Bei etwas zerklüfteteren Funktionen - wie beim Rosenbrock- oder Rastrigin-Problem - gelingt diese vollständige Approximation jedoch fast nie. Das Insel-Modell (*GP-IMME*) zeigt sich den anderen Ansätzen im Allgemeinen überlegen, und zwar sowohl im mittleren Fehler als auch bei der Trefferquote. Dies wird umso deutlicher, je komplexer die Funktion ist. Erfreulicherweise ist hier auch die Streuung der Ergebnisse niedriger, so dass man davon ausgehen kann, dass auch eine geringere Anzahl von Testläufen eine verlässliche Lösung liefert.

Die Auswirkungen der Meta-Evolution sind deutlich: Ein Vergleich von *GP-IMME* zu *GP-IMM* zeigt ein Verbesserung der Ergebnisse um ca. 15% im Mittel. Um zu überprüfen, welcher Parameter sich hier am stärksten auswirkt, wurden verschiedene Meta-Evolutionsläufe mit jeweils nur einem der Metaparameter

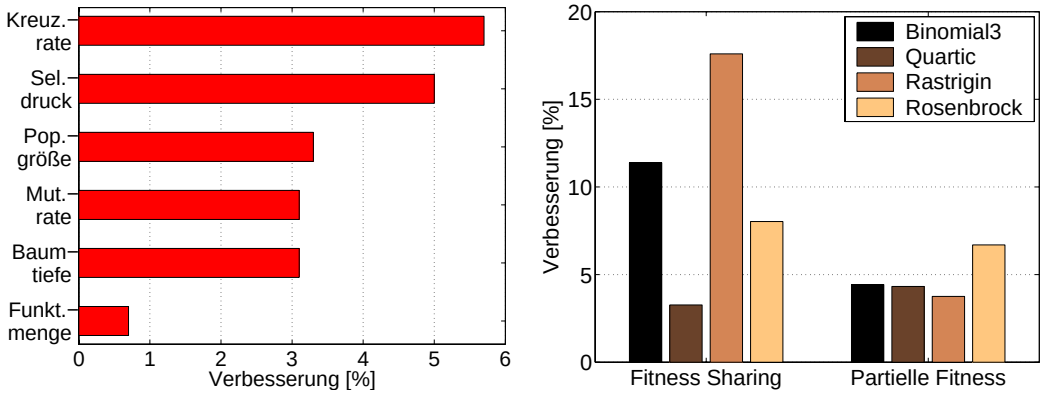
[8]Die Trefferquote gibt den Anteil der korrekten Lösungen an.



(a) Rastrigin-Problem (b) Binomial3-Problem

Abbildung 5.13: Vergleich der Modelle

durchgeführt. Die prozentualen Auswirkungen der Parameter unterscheiden sich in Abhängigkeit der Testfunktion leicht voneinander, als allgemeine Tendenz lässt sich aber der in Abb. 5.14(a) festgehaltene Trend feststellen. Selbstverständlich lassen sich die erzielten Verbesserungen nicht einfach addieren - tatsächlich ergibt die Summe der Werte mehr als die oben genannten 15% - denn die Wirkungen der Parameteränderungen können sich unter Umständen gegenseitig aufheben.



(a) Evolutionsparameter (b) Diversitätserhaltende Maßnahmen

Abbildung 5.14: Auswirkungen einzelner Parameter

Auffällig an den Ergebnissen sind die geringen positiven Auswirkungen der Meta-Evolution der Funktionsmenge. Es zeigt sich, dass es für die Evolution schwierig ist, aus der großen zur Verfügung stehenden Auswahl eine geeignete Gruppe zusammenzustellen. Die Tendenz geht dann schon am Beginn der Evolution zu sehr großen Gruppen, die aber den Suchraum enorm vergrößern und

im Endeffekt die Evolution verlangsamen. Auch wirkt bei der Veränderung der Funktionsmenge der negative Effekt, dass unter Umständen gute Lösungen unbrauchbar werden können.

Die Veränderung der Baumtiefe kann sich ebenfalls sowohl positiv - als problemspezifische Begrenzung der Individuengröße - als auch negativ - durch Zerstören von guten Lösungen - auswirken.

Ein sehr wichtiger Aspekt ist die Beeinflussung des Selektionsdrucks. Dies wird über die Anzahl der Turnierteilnehmer und -gewinner eingestellt. Anders als vielleicht zu erwarten war, zeigt sich hier im evolutionären Verlauf kein eindeutiger Trend hin zu niedrigerem Selektionsdruck (als diversitätserhaltende Maßnahme), sondern es ist vielmehr ein problemabhängiges Finden eines geeigneten Wertes zu beobachten, der dann über den weiteren Verlauf im Wesentlichen konstant bleibt.

Es wurden zwei verschiedene Möglichkeiten vorgestellt und implementiert, die Diversität der Population innerhalb einer Insel zu erhalten. Eine Analyse zeigt einen Vorteil des *Fitness sharing* gegenüber der partiellen Fitness (Abb. 5.14(b)). Der mittlere Fehler sinkt bei den vier untersuchten Testfunktionen hier um ca. 10% gegenüber 5% bei der partiellen Fitness. Bei einer Kombination beider Varianten kann der mittlere Modellfehler sogar um ca. 13% gesenkt werden.

Es zeigt sich somit, dass durch das meta-evolutionäre Insel-Modell mit zusätzlichen diversitätserhaltenden Maßnahmen eine erhebliche Verbesserung des Konvergenzverhaltens erzielt werden konnte. In den folgenden Kapiteln soll nun untersucht werden, wie dieses Verfahren für die Steuerung und Regelung von Mikrorobotern genutzt werden kann.

Kapitel 6

Steuerung

In Kapitel 4 wurde bereits verdeutlicht, dass das untersuchte Regelungskonzept auf einem zweistufigen Aufbau basiert, nämlich der Steuerung (offener Regelkreis) und der eigentlichen Regelung (geschlossener Regelkreis). Sinn und Zweck der Steuerungsebene ist

- eine Linearisierung des Roboterverhaltens zu erreichen, auch mit dem Hintergrund eine lineare Regelung einzusetzen.
- einen Ausgleich von Unterschieden im Bewegungsverhalten der Roboter zu erreichen.
- dem Regler eine vereinheitlichte, von der Hardware unabhängige Schnittstelle zur Verfügung zu stellen.
- ortsabhängige Parameter besser ausgleichen zu können.

Die Steuerung verfügt definitionsgemäß über keine Prozessrückmeldungen, so dass die Entwicklung des Steuerungsmodells vollständig vor dem eigentlichen Betrieb, also *offline*, in einem Kalibrierverfahren durchgeführt werden muss. Dies bedeutet konkret, dass vorab Messungen des Bewegungsverhaltens des Roboters durchgeführt werden, die den gesamten Konfigurationsraum des Roboters abdecken müssen.

Mathematisch gesehen ist die Steuerung ein invertiertes Modell des Roboterbewegung. Die Eingänge des Modells sind dabei die Reglerausgänge $\vec{u_R}$, die Ausgänge sind die Elemente der Stellgröße $\vec{u}$, die dann direkt zu Roboterhardware gesendet werden (vgl. Abb. 4.1). Die Anzahl der Elemente von $\vec{u_R}$ entspricht der Anzahl der Freiheitsgrade des Roboters. Da das Regelungssystem (vgl. Abschnitt 7) auf der Basis von Geschwindigkeiten arbeitet, hat auch $\vec{u_R}$ die Einheit einer Geschwindigkeit, im Allgemeinen Millimeter pro Sekunde. Um das Steuerungsmodell zu bestimmen, ist demzufolge eine Funktion $\vec{u} = f(\vec{u_R})$ mit dem

Roboter	Freiheits-grade	Unabh. Aktoren	Maximale Freq.[Hz]	Max. Ampl. [V]	Stell-befehl(u)
Miniman III-2	3	3	4000	150	$\vec{u}_a, \vec{\theta}$
Miniman IV	3	3	4000	150	$\vec{u}_a, \vec{\theta}$
MiCRoN 0	3	6	4000	30	$\vec{f}$

Tabelle 6.1: Konfiguration der Mikroroboter

Ziel gesucht, eine Annäherung der Regelgröße y - also der Ist-Geschwindigkeit des Roboters - an den Reglerausgang u_R zu erreichen. Das Problem lässt sich demzufolge auch formulieren als:

$$\vec{u} = f(\vec{y}) \tag{6.1}$$

Zusätzlich können als Modelleingänge weitere Größen eingesetzt werden. Ein Großteil der Mikroroboter ist nicht vollständig autonom, da die Energie- und Signalversorgung teilweise über Versorgungsleitungen erfolgt (vgl. Kapitel 3). Dieses Versorgungskabel hat - in Anbetracht der meist geringen Kräfte, die von der Aktorik aufgebracht werden - einen häufig beträchtlichen Einfluss auf das Bewegungsverhalten des Roboters. Unter der Annahme, dass der Einfluss des Kabels nur ortsabhängig, aber nicht zeitabhängig ist (also die Befestigung der Leitung nicht verändert wird), kann als zusätzlicher Eingang die aktuelle Position $\vec{x}$ des Roboters in das Modell einfließen:

$$\vec{u} = f(\vec{y}, \vec{x}) \tag{6.2}$$

Die Anzahl der Freiheitsgrade und insbesondere die Anzahl der Hardwaresignale u_H variiert natürlich von Roboter zu Roboter. Tabelle 6.1 liefert eine Übersicht über die im Rahmen dieser Arbeit benutzten Roboter.

Es stellt sich nun die Frage, welches Modell für den jeweiligen Roboter zu wählen ist und auf welche Weise die Modellparameter bestimmt werden sollten. Ziel dieser Arbeit ist es unter anderem, den Einsatz des in Kapitel 5 vorgestellten Ansatzes zu testen. Um eine Bewertung zu ermöglichen, werden eine Reihe weiterer Verfahren getestet und in diesem Kapitel vorgestellt. Dabei sind zwei grundsätzliche Herangehensweisen möglich:

- Das Modell wird aus der Analyse der physikalischen und technischen Gegebenheiten hergeleitet und in eine Form überführt, die einen Einsatz als Steuerungsmodell zulässt. Die Parameter des Modells sind entweder durch Kenntnis des Aufbaus der Roboter festgelegt oder werden in einem numerischen Optimierverfahren bestimmt. Für letzteres sind Messdaten des Roboterverhaltens erforderlich. Diese Vorgehensweise wird in Abschnitt 6.2 durchgeführt und analysiert.

- Das Modell wird in einem *Blackbox*-Verfahren bestimmt, d.h. ohne explizite Berücksichtigung von bekannten Robotereigenschaften. Auch hier sind die Messungen der Roboterbewegungen erforderlich. Für die Umsetzung dieser Vorgehensweise sind verschiedene Varianten denkbar. Im Rahmen dieser Arbeit soll neben einer evolutionär basierten Modellbestimmung (vgl. Abschnitt 6.4) auch der Einsatz von neuronalen Netzen näher untersucht werden (vgl. Abschnitt 6.3).

Neben dem für Steuerungszwecke benötigten inversen Modell wird auch ein Vorwärtsmodell für regelungstechnische Aufgaben benötigt. Ziel dabei ist es, herauszufinden, welcher Systemzustand sich bei einem vorgegebenen Stellbefehl einstellt, also eine Beziehung $\vec{y} = f(\vec{u})$ zu finden.

Bei den Untersuchungen der Roboter ist zu beachten, wie die Roboter bestückt sind, da sich durch den Manipulator der Schwerpunkt und das Gewicht des Systems verändert und damit auch das Bewegungsverhalten. Um nun die Komplexität der Untersuchungen nicht zu sehr ansteigen zu lassen, wird bei den folgenden Untersuchungen eine Beschränkung auf folgende Typen vorgenommen:

- Miniman III mit Manipulator
- Miniman IV ohne Manipulator
- MiCRoN 0

Der Sinn der Arbeit liegt ohnehin nicht in der Analyse einer speziellen Konfiguration, sondern im Finden einer allgemeingültigen Aussage.

Als Basis für die Bestimmung des Modellfehlers eines inversen Modells dient der mittlere relative Fehler eines Stellsignals:

$$e_m = \frac{1}{n_d n_3 n_s} \sum_{i=1}^{n_d} \sum_{j=1}^{n_3} \sum_{k=1}^{n_s} \frac{|u_{i,j,k} - y_{m,i,j,k}|}{u_{j,k}^{max}} \tag{6.3}$$

Mit n_d: Anzahl der Datensätze, n_3: Anzahl der Aktoren und n_s: Anzahl der Stellsignale des Aktors

Der Stellbefehl $\vec{u}$ setzt sich im Fall der Miniman-Roboter ($n_s = 2$) aus mehreren Signalen zusammen(u_a und θ). Da sich die Größenordnungen der Signale unter Umständen stark unterscheiden, ist es notwendig, eine Normierung über den Maximalwert eines jeden Stellsignals durchzuführen. Für das Vorwärtsmodell ergibt sich der Fehler entsprechend:

$$e_m = \frac{1}{n_d n_f} \sum_{i=1}^{n_d} \sum_{j=1}^{n_f} \frac{|y_{i,j} - y_{m,i,j}|}{y_j^{max}} \tag{6.4}$$

Auf diese Weise kann ein relativer Modellfehler e_m berechnet werden, der im Folgenden - sofern nicht anderweitig festgelegt - immer als Bewertungsgrundlage verwendet wird.

Bevor nun die Verfahren im Einzelnen vorgestellt und untersucht werden, müssen Messungen des Bewegungsverhaltens der Roboter durchgeführt werden.

6.1 Messungen

Die zur Ermittlung der Modellparameter benötigten Messwerte wurden vor dem eigentlichen Betrieb gewonnen. Es ging dabei darum, die bei einem Stellbefehl $\vec{u}$ erzielten Positions- und Orientierungsänderungen (=Geschwindigkeiten) $\vec{y}$ zu ermitteln. Wegen der mitunter recht hohen Streuung der Messwerte wurde die Wirkung eines Steuersignal über 50 Messzyklen aufgezeichnet, bevor das folgende Signal herausgegeben wurde. Die Streuung hat ihre Ursache einerseits in über den Arbeitsraum nicht einheitlichen Verhalten des Roboters, hervorgerufen etwa durch kleine Staubpartikel auf der Arbeitsfläche, zum Anderen aber auch in Schwankungen des Messsystems.

Wenn die Messwerte, die zu einem Stellbefehl gehören, den statistischen Anforderungen entsprachen, wurden die 50 gemessenen Werte gemittelt und zwischengespeichert. Als statistische Anforderung wurde definiert, dass alle Daten, die in einem Konfidenzintervall von 95% liegen, akzeptiert wurden, d.h. wenn der Wert im Bereich $\overline{x} \pm p_N(0.975) \cdot \frac{\sigma}{\sqrt{n_m}}$ liegt.

Wegen der möglichen Ortsabhängigkeit der Roboterbewegung, wurde die Arbeitsfläche in Quadrate mit einer Kantenlänge von 5x5cm aufgeteilt und der Messvorgang für jedes dieser Quadrate wiederholt. Das Gesamtergebnis ist dann der Mittelwert über alle Quadrate.

$$\overline{\vec{y_s}}(\vec{u}) = \frac{1}{50 \cdot n_q} \sum_{i=1}^{n_q} \sum_{j=1}^{50} \vec{y}_{s,i,j} \tag{6.5}$$

mit n_q als Anzahl der Quadrate und $\vec{y}_{s,i,j}$ als einzelne Geschwindigkeitsmessung des Roboters in einem Quadrat. Zur Weiterverarbeitung der Messwerte im Kalibrierprozess des Modells wurde die Datenmenge in eine Trainings- (70% der Daten) und eine Validierungsmenge (30% der Daten) aufgeteilt. Auf der letzteren wurde dann eine Vergleichsprüfung (*Cross-validation*) durchgeführt um eine gute Generalisierung des Modells zu überprüfen. Die in den folgenden Abschnitten präsentierten Ergebnisse der Modelloptimierung basieren auf der Validierungsgruppe.

6.1.1 Miniman-Roboter

Miniman III-2 Plattform

Für jeden Aktor wird aus dem Stellbefehl $u\{u_a, \theta\}$ (vgl. Gleichung 3.1) das Steuersignal $u_H\{U_{0,x}, U_{0,y}, f\}$ generiert (vgl. Abschnitt Abschnitt 3.1.1). Die erzeugte Frequenz ist nicht linear vom Stellbefehl $\vec{u} = \{u_1; u_2; u_3\}$ abhängig, sondern es ergibt sich der in Abbildung 6.1 aufgeführte Zusammenhang. Es ist zu beachten, dass bei Frequenzen höher als ca. 1 KHz, die Bewegung chaotisch wird und nicht mehr kontrollierbar ist. Die Gründe dafür sind in Abschnitt 6.2.1 näher beschrieben.

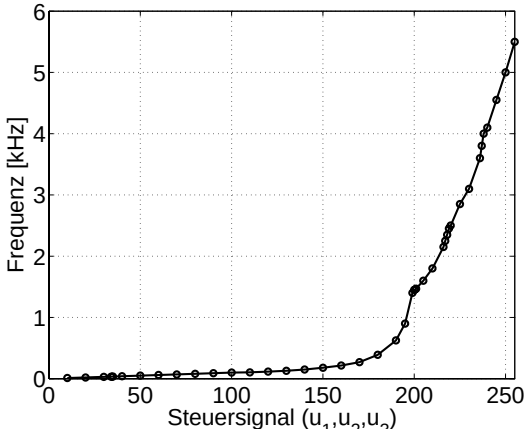
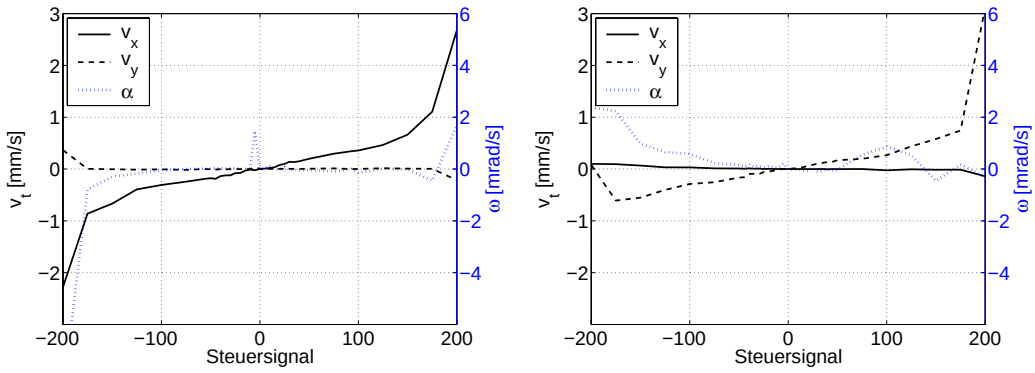


Abbildung 6.1: Zusammenhang des Steuerwerts und der resultierenden Frequenz des Miniman3-2 C

Eine Analyse der Messdaten zeigt eine über weite Strecke lineare Umsetzung des Steuersignals bis etwa $u_a = 170$. Die dazu unerwünschte senkrechte (parasitäre) Translationsbewegung hält sich in Grenzen (bei ca. 19%). Recht erheblich ist jedoch die parasitäre Rotation im Falle einer angeforderten y-Bewegung (vgl. Abbildung 6.2). Den kompletten Konfigurationsraum für Translationsbewegungen (also auch eine Kombination von x- und y) zeigt Abbildung 6.4(a).

Miniman IV

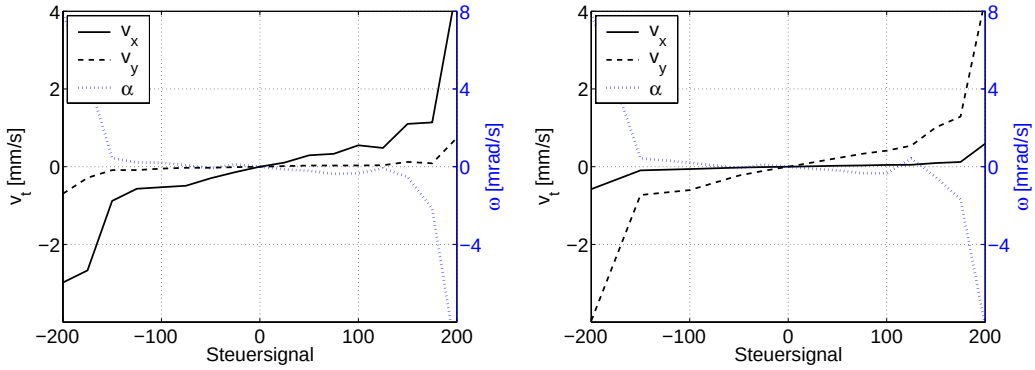
Die Signalgenerierung des Miniman-IV-Roboters erfolgt analog zum Miniman-III. Die Bewegung des Miniman IV ist deutlich stabiler und linearer als die des Miniman III. Störeinflüsse wirken sich wegen des wesentlich höheren Gewichts nur geringfügig aus. Parasitäre Störbewegungen kommen nur bei hohen Werten für den Stellbefehl (etwa ab $u_a > 150$) zustande. Es ist ferner zu beobachten,



(a) x-Translation (b) y-Translation

Abbildung 6.2: Bewegungsverhalten des Miniman III

dass der Miniman IV wegen seines höheren Gewichts eine etwas höhere Maximalgeschwindigkeit erzielt.



(a) x-Translation (b) y-Translation

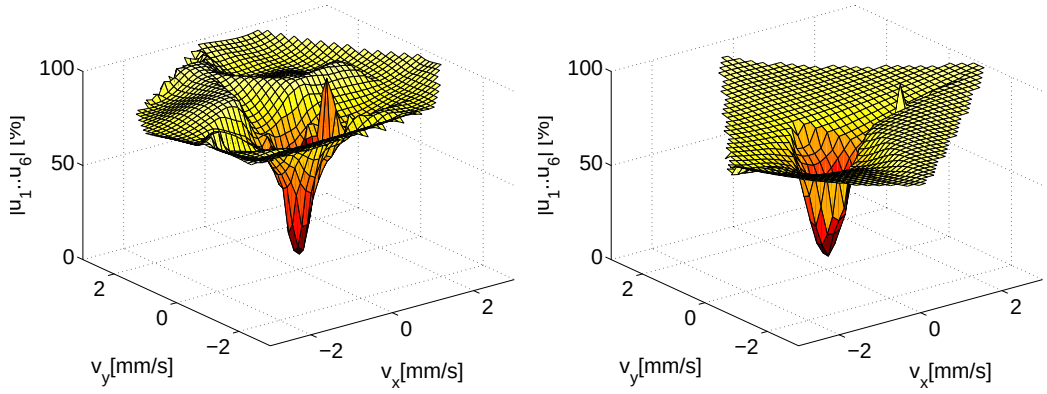
Abbildung 6.3: Bewegungsverhalten des Miniman IV

6.1.2 MiCRoN-Roboter

MiCRoN 0

Der MiCRoN 0 verfügt - im Gegensatz zu den Miniman-Robotern - über 6 Aktoren an seiner Plattform. Als Steuer- und Regelgröße wird lediglich die Frequenz (0 bis 4 kHz) verändert[1] und die Spannungamplitude konstant bei 30V gehalten.

[1]Die in den Abbildungen aufgeführten negativen Werte für die Frequenz entsprechen tatsächlich einer positiven Frequenz mit einer Phasenverschiebung von 180 °.



(a) Miniman III (b) Miniman IV

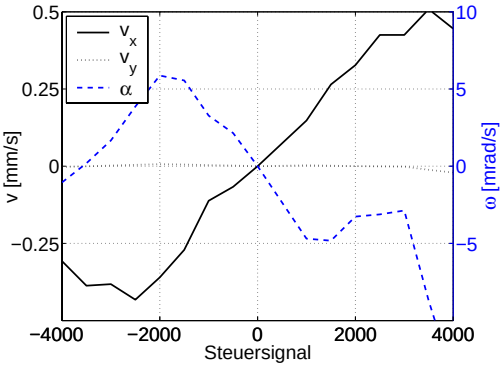
Abbildung 6.4: Bewegungsverhalten von Mikrorobotern, Translation in der X-Y-Ebene, Frequenzwert des Steuersignals in %

Beim MiCRoN 0 sieht man erhebliche parasitäre Rotationsbewegungen (vgl. Abb. 6.5(a) und 6.5(b)), die unter Umständen vom Energieversorgungskabel mitverursacht werden. Die Translationsbewegung ist über weite Bereiche nahezu linear (bis ca. $u \approx \pm 3000$), auch hält sich die parasitäre Bewegung senkrecht zur gewünschten Translationsrichtung in Grenzen. Problematisch ist vor allem das Bewegungsverhalten bei Rotation (vgl. Abb. 6.5(c)).

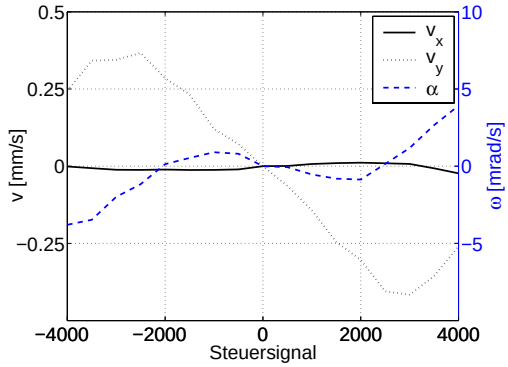
Eine interessante Frage ist die nach der Ortsabhängigkeit dieser Werte, wobei vor allem das Versorgungskabel als Einflussfaktor eine Rolle spielen dürfte. Abbildung 6.5(e) zeigt - bezogen auf die Arbeitsfläche (30x20 cm) - den Mittelwert des Betrags der Rotationsgeschwindigkeit bei angeforderter reiner x- und reiner y-Translation. Interessanterweise ist der Aufhängungspunkt des Kabels etwa bei {x=0.08; y=0.1}, also im Bereich der geringsten parasitären Rotation.

6.2 Mechanisches Modell

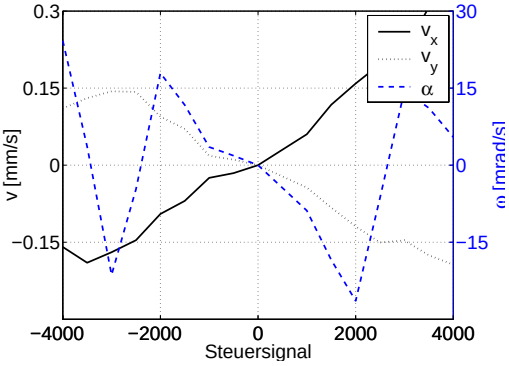
Im folgenden soll nun ein kinematisches Modell eines Mikroroboters unter Berücksichtigung aller Einflussgrößen entwickelt werden. Als geometrisches Vorbild dient dabei der Miniman-Roboter. Es ist zu überprüfen, inwieweit sich das Modell auch auf die MiCRoN-Roboter mit ihrer andersartigen Anordnung der Piezoelemente übertragen lässt.



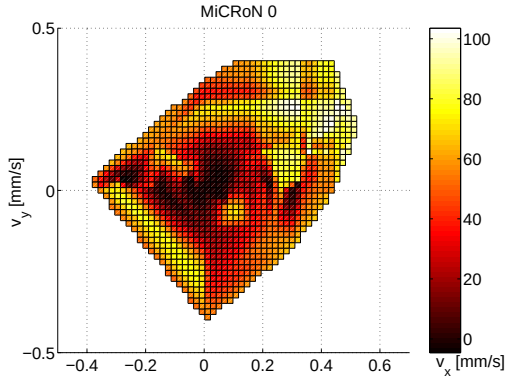
(a) Kanal 4 und 8 angesteuert



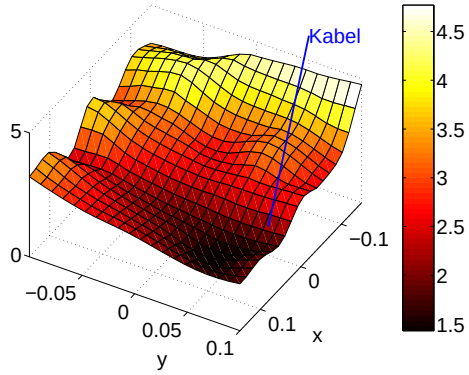
(b) Kanal 2,3,5 und 7 angesteuert (y-Translation)



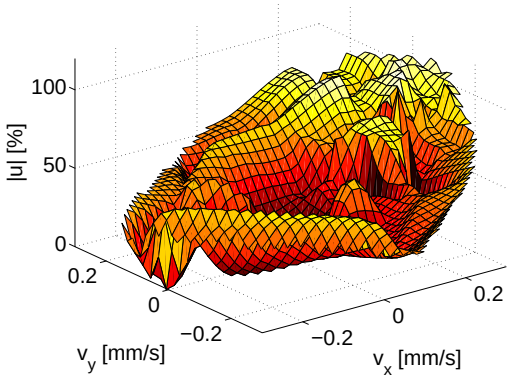
(c) Kanal 2,3,5 und 7 angesteuert (Rotation)



(d) x- und y-Translation



(e) Ortsabhängigkeit der parasitären Rotation



(f) Abhängigkeit der translatorischen Bewegung vom Stellbefehl u

Abbildung 6.5: Bewegungsverhalten des MiCRoN 0

6.2.1 Einflussgrößen

Beschleunigung

Jeder der Aktoren des Mikroroboters erzeugt beim Ausführen des inversen piezoelektrischen Effekts eine Kraft.

$$\vec{F}_i = \begin{pmatrix} F_{x,i} \\ F_{y,i} \\ F_{z,i} \end{pmatrix} \tag{6.6}$$

Diese Kraft wirkt idealerweise nur horizontal ($F_{z,i} = 0$). Durch die Kraft wird auf den starren Körper (den der Roboter darstellt) ein Drehmoment um den Massenschwerpunkt P_S erzeugt.

$$\vec{M}_i^{(P_S)} = \vec{r}_i \times \vec{F}_i \tag{6.7}$$

Der Ortsvektor $\vec{r}_i$ führt vom Fußpunkt der Kraft zum Massenschwerpunkt des Roboters. Für den Fall reiner Translationsbewegung wird die Summe der Momente aller Aktoren zu Null. Umgekehrt gilt für die Rotation, dass hier die Summe aller Kräfte (zumindest deren x- und y-Komponenten) zu Null wird. Die resultierende Kraft F_P ist mit der Geschwindigkeit gekoppelt:

$$\vec{F_P} = \sum_i \vec{F}_i = m \cdot \frac{d\vec{v}}{dt} \tag{6.8}$$

Entsprechend ist das resultierende Drehmoment

$$\vec{M_P}^{(A)} = \sum_i \vec{M}_i^{(A)} = \sum_i \vec{r}_i \times \vec{F}_i = J\frac{d\vec{\omega}}{dt} \tag{6.9}$$

Hier ist das Trägheitsmoment J des Rotationskörpers zu berücksichtigen. Der Kraftvektor $\vec{F}_P$ kann - auch um den weiteren Rechengang zu vereinfachen - in einen Betrag $F_P = |\vec{F_P}|$ und eine Richtung $\theta = \arctan(F_y, F_x)$ aufgeteilt werden. Bei unterschiedlichen Steuerwinkeln θ bestimmt sich die resultierende Kraft mit:

$$\begin{aligned} F_x &= \sum_{i=1}^{n_3} F_{x,i} = \sum_i^{n_3} F_i \cos(\theta_i) \\ F_y &= \sum_{i=1}^{n_3} F_{y,i} = \sum_i^{n_3} F_i \sin(\theta_i) \\ \Rightarrow F_P &= \sqrt{F_x^2 + F_y^2} \end{aligned} \tag{6.10}$$

Die interessierende Frage ist nun die, wie die Piezokraft F_P vom Stellbefehl $\vec{u}$ abhängt. Um den genauen Bewegungsablauf zu verstehen, ist es notwendig, vier Positionen voneinander zu unterscheiden:

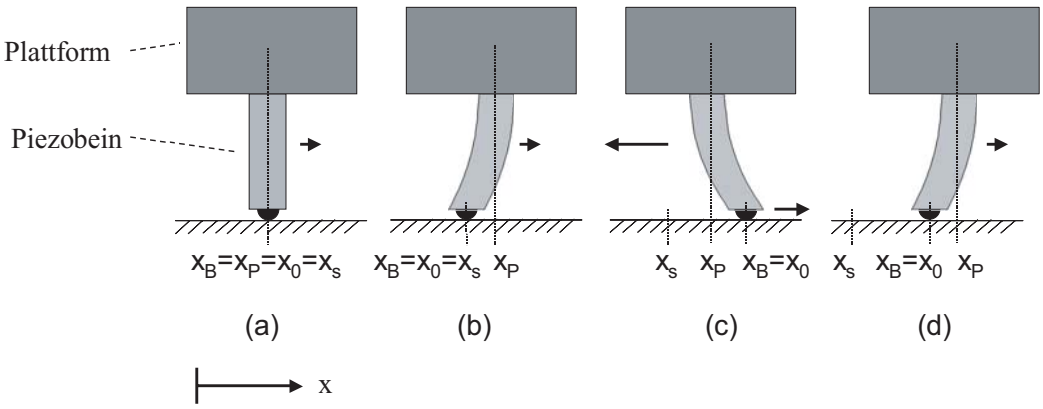


Abbildung 6.6: Idealisierter Ablauf beim Stick-Slip-Antriebsprinzip

1. x_s: Startposition des Roboters, d.h. die Position zum Zeitpunkt $t = 0$.
2. x_B: die Kontaktposition des Roboterbeins mit der Arbeitsflcähe
3. x_P: Die x-Komponente des Massenschwerpunkts der Roboterplattform
4. x_0: Grundposition des Roboters, d.h. die Position des Roboterschwerpunkts wenn keinerlei Durchbiegung der Beine vorhanden wäre. Bei einem einbeinigen Roboters wäre dies im Idealfall die Kontaktstelle des Roboterbeins mit der Arbeitsfläche ($x_0 = x_B$).

Auf Basis dieser Definition sind auch die beiden Geschwindigkeiten definiert:

- v_P: Die Relativgeschwindigkeit der Roboterplattform bezogen auf seine Grundstellung, d.h. $v_P = \frac{d(x_P - x_0)}{dt}$
- v_B: Die Absolutgeschwindigkeit der Kontaktfläche Piezobein-Arbeitsfläche: $v_B = \frac{dx_B}{dt} = \frac{d(x_B - x_s)}{dt}$.

Der Ablauf lässt sich dann in verschiedene Phasen aufteilen (vgl. Abb. 6.6 und 6.7):

(a) Startzustand: Roboterplattform befindet sich in Ruhelage, die Beine in Ausgangsstellung ($x_B = x_P = x_0 = x_s$)

(b) Stickphase: Die Spannung wird jetzt allmählich bis $+U_0$ gesteigert. Die Piezobeine werden in positive Richtung ausgelenkt, was eine Auslenkung der Plattform zur Folge hat ($x_P > x_0$). Die Beine bleiben an der Oberfläche haften ($x_B = x_0 = x_s$), bis die Haftreibungsgrenze erreicht ist. Abhängig

vom Verhältnis der Reibungskräfte zu den Antriebskräften kann es passieren, dass schon in dieser Phase eine Gleitbewegung der Piezobeine zustande kommt.

(c) Slipphase: Die Spannung wird nach Erreichen von $+U_0$ schnell auf $-U_0$ gesenkt. Dadurch kommt es zu einem sehr schnellen Zurückbiegen der Piezobeine in negativer Richtung. Wegen der im Vergleich zu den Reibungskräften großen Massenträgheitskräfte rutschen nun die Beine auf der Oberfläche durch ($x_B > x_s$,$x_P < x_B$, die Haftreibungsgrenze wird überwunden).

(d) Stickphase: Nun wird die Spannung allmählich wieder bis $+U_0$ erhöht. Die Beine werden wieder positiv ausgelenkt ($x_P > x_B$). Der gesamte Roboter wird unter Umständen noch etwas weiterrutschen, wobei diese Bewegung aber durch die Reibungskräfte und die nun wieder positive Beschleunigung der Beine abgebremst, bzw. sogar für eine kurze Zeit ins Gegenteil verkehrt wird [10]. In der Folge wechseln sich nun die Phasen (c) und (d) ab.

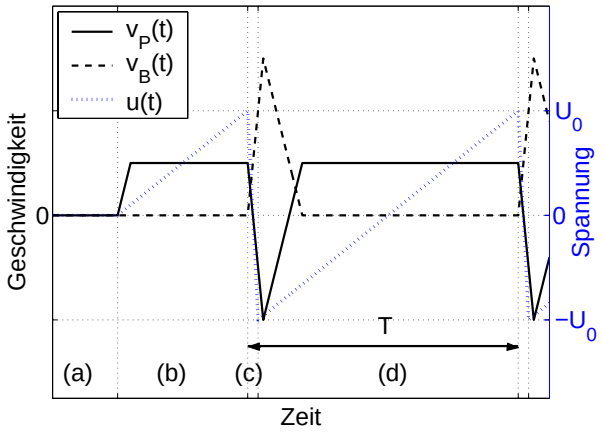


Abbildung 6.7: Verlauf von Spannung und Geschwindigkeit der Roboterplattform (v_P) und -beine (v_B)

Bei sehr hohen Frequenzen kann das Durchrutschen der Beine auch bei Beendigung der Phase (d) noch im Gange sein. Da sich nun ohnehin eine erneute Slipphase (c) anschließt, wird sich der Roboter in einem fortwährenden, geschwindigkeitsmäßig aber unregelmäßigem Durchrutschen befinden. Dies führt zu einem chaotischen und daher unerwünschten Geschwindigkeitsverhalten (siehe dazu auch [13]).

Ein Freischnitt zeigt die auftretenden Kräfte (vgl. Abbildung 6.8). An der Plattform sind neben der Kraft, die durch die angelegte Spannung entsteht (F_U),

auch der Luftwiderstand (F_W) und die durch die Durchbiegung entstehende mechanische Spannung (F_M) beteiligt.

$$F_P = F_U - F_W - F_M - F_e \tag{6.11}$$

Das mechanische System kann durch ein Feder-Masse-System mit Dämpfung modelliert werden.

$$F_M = k_1 (x_P - x_0) + k_2 v_P \tag{6.12}$$

Dabei sind k_1 und k_2 Proportionalitätskonstanten.

An den Beinen ist als Antrieb natürlich die Beschleunigung der Plattform wirksam, erneut mit einem Dämpfungsterm, allerdings in entgegengesetzter Richtung (daher das negative Vorzeichen). Die Bewegung wird durch Reibung mit der Oberfläche abgebremst (F_R), zusätzlich kann hier der Einfluss externer Kräfte - etwa hervorgerufen durch die Zugkraft eines Kabels - modelliert werden (F_e), worauf im Folgenden aber verzichtet wird. Da nun durch die Bewegung der Beine die gesamte Plattform mitbewegt wird, muss auch hier der Luftwiderstand berücksichtigt werden.

$$F_B = -(k_3 F_P + k_4 v_B) - F_R - F_W - F_e \tag{6.13}$$

Bei geringen Beschleunigungen der Plattform wird die Haftreibungsgrenze nicht überschritten und es kommt keine Beinbewegung zustande. Bevor das Differentialgleichungssystem gelöst werden kann, ist es notwendig, die auftretenden Kräfte näher zu analysieren.

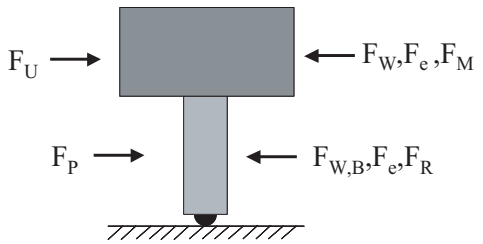


Abbildung 6.8: Auftretende Kräfte an der Plattform und an den Piezobeinen

Mechanische Verformung

Das Anlegen einer elektrischen Spannung am Piezoelement erzeugt eine Kraft, die in einer Durchbiegung der Beine resultiert. Die Durchbiegung entspricht der eines einseitig eingespannten, unter gleichmäßiger Flächenlast befindlichen Trägers [9]:

$$x_P - x_0 = -\frac{l_P^3}{8 \cdot E \cdot J_a} \cdot F_M \tag{6.14}$$

Mit der Länge der Piezoröhrchen l_P und dem (mittleren) Elastizitätsmodul E. Das axiale Flächenträgheitsmoment J_a von runden Hohlzylindern ist:

$$J_a = \frac{\pi}{4}(r_{P,a}^4 - r_{P,i}^4) \tag{6.15}$$

Dabei sind $r_{P,a}$ der Außenradius der Piezoröhrchen und $r_{P,i}$ deren Innenradius (vgl. auch Abbildung 6.10). Es ergibt sich für die elektrisch verursachte äquivalente Biegekraft:

$$F_M = \underbrace{\frac{2E\pi\left(r_{P,a}^4 - r_{P,i}^4\right)}{l_P^3}}_{k_1}(x_P - x_0) \tag{6.16}$$

Der in Gleichung 6.12 zusätzlich auftretende Term $(k_2 v_P)$ repräsentiert die durch innere Reibung hervorgerufene Dämpfung.

Reibung

Die Reibung zwischen den Roboterbeinen und der Oberfläche führt zu einem Abbremsen der Bewegung. Dabei sind verschiedene Effekte wirksam:

- Haftreibung (statische Reibung): Durch die Gewichtskraft des Roboters wird dieser auf die Oberfläche gepresst:

  $$F_H = \mu_H \cdot F_N = \mu_H \cdot m \cdot g \tag{6.17}$$

  Solange die Piezokraft der Beine unterhalb von F_H liegt, kommt es zu keiner Bewegung.

- Gleitreibung (Coulomb-Reibung): Dies stellt das klassische Modell der Reibungsphänomene dar. Die Gleitreibungskraft ist der Bewegungskraft entgegengesetzt, aber unabhängig von der Geschwindigkeit. Der Gleitreibungskoeffizient μ_G hängt vor allem von den sich an der Kontaktfläche berührenden Materialien ab.

  $$F_G = \mu_G \cdot F_N = \mu_G \cdot m \cdot g \tag{6.18}$$

- Viskose Reibung berücksichtigt die Viskosität von Schmiermitteln zwischen den Reibungspartnern. Sie hängt linear von der Geschwindigkeit ab

  $$F_V = F_N \cdot \mu_\nu \cdot \nu \tag{6.19}$$

- Als Stribeck-Effekt wird ein bestimmtes Reibverhalten bezeichnet, bei dem die Reibkörper durch einen Flüssigkeitsfilm (üblicherweise ein Schmiermittel) voneinander getrennt sind. Dieses Verhalten wird durch die sogenannte Stribeck-Kurve beschrieben.

Es gibt nun eine Reihe von Modellen, die diese Effekte in eine mathematische Formulierung bringen. Beispiele dafür sind die Modelle von Dahl [18] oder Bliman-Sorine [12] oder auch das Modell von Karnopp [43]. Das vielleicht vielversprechendste ist das Modell nach Canudas [19][2], das auch den Stribeck-Effekt berücksichtigt:

$$F_R = \sigma_0 z_k + \sigma_1 \frac{dz_k}{dt} + \sigma_2 v \tag{6.20}$$

Das Modell basiert auf der Annahme, dass sich zwei Oberflächen in Kontakt wie zwei ineinander verschobene Borsten verhalten. Die Größe z_k ist ein Wert, der die äquivalente Deformation des mechanischen Kontakts bewertet, σ_0 die Steifigkeit der Oberfläche, σ_1 ein Dämpfungsfaktor und σ_2 ein Proportionalitätsfaktor für die viskose Reibung ($\sigma_2 \sim \mu_v$). Die zeitliche Veränderung der Borstenverzahnung berechnet sich dann in diesem Modell gemäß:

$$\frac{dz_k}{dt} = v - \frac{|v| \cdot z_k \cdot \sigma_0}{F_G + (F_H - F_G) \exp\left(-v^2/v_s^2\right)} \tag{6.21}$$

Die Steifigkeit der Oberfläche σ_0 lässt sich nach [13] berechnen mit:

$$\sigma_0 = \frac{1}{\frac{1}{8a}\left(\frac{2-\nu_{q,1}}{E_1} + \frac{2-\nu_{q,2}}{E_2}\right)} \tag{6.22}$$

Mit

$$\begin{aligned} a &= \sqrt[3]{\frac{3 m_{rob} g d}{4 E'}} \\ E' &= \frac{1}{\frac{1-\nu_{q,1}^2}{E_1} + \frac{1-\nu_{q,2}^2}{E_2}} \end{aligned} \tag{6.23}$$

Dabei sind E_1 und E_2 die Elastizitätsmodule der beiden Kontaktkörper und $\nu_{q,1}$ bzw. $\nu_{q,2}$ deren Querkontraktionszahlen.

[2] Auch bekannt unter dem Name LuGre-Modell, nach der Herkunft der beteiligten Wissenschaftler: Lund und Grenoble

Luftwiderstand

Die Roboterbewegung wird durch den Widerstand der umgebenden Luft abgebremst. Die dabei entstehende Reibungskraft wird bestimmt durch:

$$F_W = c_w \cdot A_S \cdot \frac{\rho v^2}{2} \tag{6.24}$$

wobei A_S die Anströmfläche repräsentiert. Problematisch in diesem Zusammenhang ist die Bestimmung des Widerstandsbeiwerts c_w. Er hängt von der Form des Strömungskörpers, aber auch von der Reynoldszahl (Re) und der Oberflächenrauhigkeit des umströmten Körpers ab.

Es ist also zunächst zu klären, welche Strömungsform vorliegt. Die charakteristische Kenngröße, die dabei zu untersuchen ist, ist die Reynoldszahl. Sie wird beeinflusst von den Reibungs- und Trägheitskräften, die auf das Strömungsfeld wirken und charakterisiert die Strömungsform:

$$Re = \frac{v_0 L_0}{\nu_L} \tag{6.25}$$

Dabei sind v_0 die Anströmgeschwindigkeit, d.h. die Bewegungsgeschwindigkeit des Roboters, L_0 eine charakteristische Länge des Strömungsproblems und ν_L die dynamische Viskosität der Luft.

Bei sehr kleinen Reynoldszahlen (Re<1) kann von einer laminaren Strömung ausgegangen werden. Für glatte Kugeln gilt in diesem Fall der Stokes-Ansatz:

$$c_w = \frac{24}{Re} \tag{6.26}$$

Daraus folgt:

$$F_W = 12 \frac{A_S \cdot \eta_L}{L_0} v = \beta_L v \tag{6.27}$$

mit η_L = kinematischer Viskosität der Luft. Hier ist die Reibungskraft offensichtlich nur noch linear von der Geschwindigkeit abhängig. Auch für Körper, die nicht kugelförmig sind, sich aber sehr langsam durch ein Medium bewegen, kann dieser Ansatz benutzt werden.

Die charakteristische Länge des Körpers, die zur Bestimmung der Reynoldszahl erforderlich ist, kann für einen derartig komplexen Körper wie einen Mikroroboter, der ja nicht nach aerodynamischen Kriterien entwickelt wurde, nicht - wie bei Standardkörpern üblich - aus Tabellen abgelesen werden. Ein häufig benutzter Ansatz für L_0 bei komplexen Körpern ist dann die Anströmlänge, definiert als das Verhältnis der Oberfläche des Körpers zum Umfang seiner Projektionsfläche in Strömungsrichtung. Sowohl für die Miniman-Roboter ($Re \approx 0.9$) als auch für den MiCRoN 0 ($Re \approx 0.1$) ergeben sich dann bei mittlerer Geschwindigkeit kleine Reynoldszahlen, so dass die lineare Approximation nach Stokes (Gleichung 6.27) benutzt werden kann.

Spannung

Der Spannungsverlauf U_T wird sägezahnförmig aufgegeben:

$$U_T(t) = 2U_0 \cdot \epsilon\left(t - \frac{T}{2}\right) - \frac{2U_0}{T} t \cdot \epsilon(t) \tag{6.28}$$

Für die mathematische Beschreibung des Spannungsverlaufs ist eine Approximation notwendig, die das Sägezahnsignal $u(t)$ nachbildet. Notwendig ist eine periodische, stetige und differenzierbare Funktion. Eine Variante wäre die Bildung einer Fourier-Reihe:

$$u(t) = U_0 \sum_k \frac{2\pi}{k} \sin(k\pi f t) \tag{6.29}$$

Um aber eine akzeptable Approximation zu erzielen, sind zahlreiche Harmonische notwendig ($k > 40$). Ein bessere Approximation gelingt über die Ableitung von $u(t)$:

$$\frac{du}{dt} = k_5 U_0 \left(k_6 - \sin^{n_2}(\pi f t)\right) \tag{6.30}$$

Der Wert von n_2 bestimmt die Steilheit der Sägezähne. Welche Werte hier sinn-

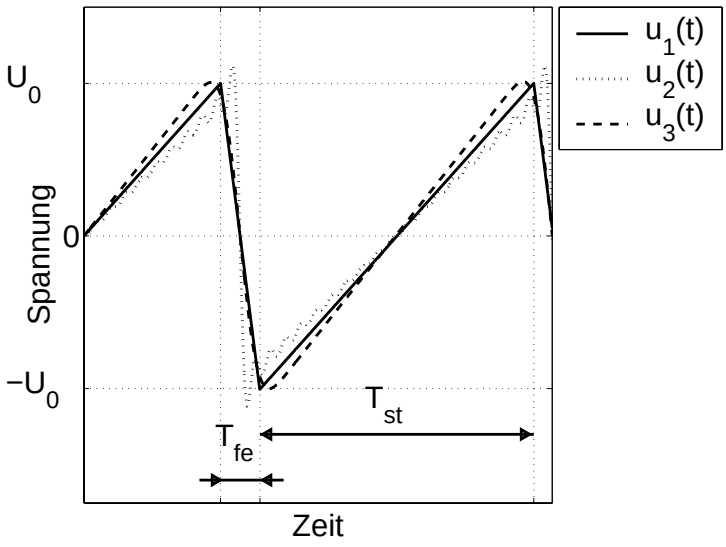


Abbildung 6.9: Vergleich verschiedener Approximationen des Spannungsverlaufs mit u_1: Ideales Signal, u_2: Fourier Reihe und u_3: Näherung gemäß Gleichung 6.30 (T_{fe}: Dauer der fallenden Flanke, T_{st}: Dauer der steigenden Flanke)

voll sind, hängt von der Signalgenerierung des jeweiligen Roboters ab. Beim MiCRoN etwa beträgt der Zeitanteil der fallenden Flanke bei etwa $1/8$ der gesamten

Periodendauer T, so dass hier etwa $n_2 \approx 20$ angesetzt werden muss. Die Parameter k_5 und k_6 sind so zu bestimmen, dass die maximale Spannung des Signals U_0 beträgt und dass für das Integral über eine Periodendauer T gilt:

$$\int_{t=0}^{T} \frac{du}{dt} = 0 \tag{6.31}$$

Der Einfluss der Spannung auf die Verformung der Piezoelemente wird durch die Verteilung des elektrischen Feldes im Piezoröhrchen beeinflusst. Die Kraftwirkung der Spannung hängt nach [27] linear von der Spannung ab und ist gegeben durch:

$$F_U(t) = k_u u(t) \tag{6.32}$$

Setzt man eine Anordnung der Elektroden gemäß Abbildung 6.10 voraus, d.h. wirkt die mechanische Kraft quer zum elektrischen Feld, ergibt sich für die Proportionalitätskonstante[3] k_u:

$$k_u = -\frac{8}{3\pi} d_{31} l^2 \frac{r_1^2 + r_1 r_2 + r_2^2}{r_2^4 - r_1^4} \sin\left(\frac{\Phi_e}{2}\right) \tag{6.33}$$

Dabei ist d_{31} die piezoelektrische Ladungskonstante, l_P die Länge des Piezoröhrchens, und Φ_e die Breite des elektrischen Feldes, gegeben durch die Ausdehnung und Anordnung der Elektroden. Für Details zur Berechnung sei auf [27] verwiesen.

6.2.2 Dynamisches Modell

Ausgehend von den erstellten Gleichgewichtsbedingungen für die angreifenden Kräfte, lassen sich nun die Differentialgleichungen aufstellen, die die angelegte Spannung und Frequenz mit der resultierenden Geschwindigkeit in Beziehung

[3] unter der Annahme, dass das elektrische Feld nur direkt zwischen der Außenelektrode, an der die Spannung angelegt wurde und der Innenelektrode auftritt, vgl. auch [28] und [27]

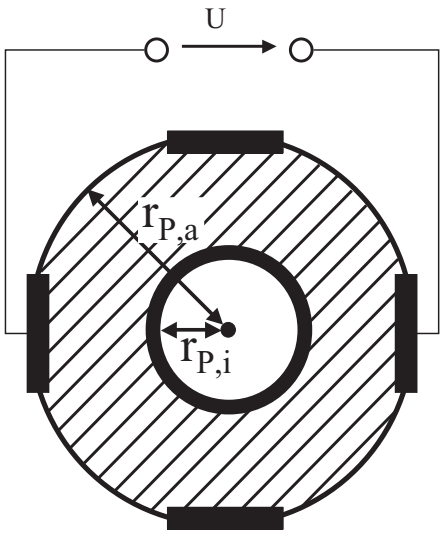


Abbildung 6.10: Anordnung der Elektroden im Querschnitt des Piezoröhrchen

setzen:

$$\frac{dv_P}{dt} = \frac{1}{m_P}(k_u u(t) - \beta_L(v_P(t) + v_B(t)) - (k_1(x_P(t) - x_0(t)) + k_2 v_P(t))) \quad (6.34)$$

$$\frac{dv_B}{dt} = \frac{1}{m_B}(-\left(k_3 m_p \frac{dv_P}{dt} + k_4 v_B(t)\right) - \left(\sigma_0 z_k(t) + \sigma_1 \frac{dz_k}{dt} + \sigma_2 v_B(t)\right) - \beta_L(v_P(t) + v_B(t))) \quad (6.35)$$

$$\frac{dz_k}{dt} = v_B(t) - \frac{|v_B(t)| \cdot z_k(t) \cdot \sigma_0}{F_G + (F_H - F_G)\exp(-v_B(t)^2/v_s^2)} \quad (6.36)$$

$$\frac{dx_P}{dt} = v_P(t) \quad (6.37)$$

$$\frac{du}{dt} = U_0(k_6 - \sin^{n_2}(\pi f t)) \quad (6.38)$$

Damit ist eine mathematische Beschreibung gelungen, die das Bewegungsverhaltens von Mikrorobotern wiedergibt, die das *Stick-Slip*-Prinzip nutzen. Eine analytische Lösung dieses nichtlinearen Differentialgleichungssystems existiert nicht. Die Plattformgeschwindigkeit $v_p(t)$ allein ist jedoch nur vom Spannungsverlauf

Symbol	Einheit	Bedeutung	Gleichung/Wert
k_1	kg/s^2	Mechanische Verformung	$\frac{2\pi E(r_{P,a}^4 - r_{P,i}^4)}{l_P^3}$
k_2	kg/s	Dämpfungsfaktor Plattformbew.	empirisch zu bestimmen
k_3	$-$	Einfluss der Plattformbewegung bei Translation	empirisch zu bestimmen
k_4	kg/s	Dämpfungsfaktor Beinbew. bei Translation	empirisch zu bestimmen
k_7	$-$	Einfluss der Platformbewegung bei Rotation	empirisch zu bestimmen
k_8	kgm^2/s	Dämpfungsfaktor Beinbewegung bei Rotation	empirisch zu bestimmen
m_B	kg	Äquiv. Masse der Piezobeine	empirisch zu bestimmen
m_P	kg	Masse der Plattform	Messung
k_u	N/V	Wirkungsfaktor der Spannung	$-\frac{8}{3\pi} d_{31} l^2 \frac{r_1^2 + r_1 r_2 + r_2^2}{r_2^4 - r_1^4} \sin\left(\frac{\Phi_e}{2}\right)$
k_5	$1/s$	Proportionalitätsfaktor Spannung	$\rightarrow U_{max} = U_0$
k_6	$-$	Verschiebung Spannungssignal	$\rightarrow \int_{t=0}^{T} \frac{du}{dt} = 0$
n_2	$-$	Steilheit des Spannungssignals	≈ 20
β_L	kg/s	Luftwiderstandsbeiwert	$12\frac{A \cdot \eta}{L_0}$
σ_0	kg/s^2	Steifigkeit der Oberfläche	$1/\left(\frac{1}{8a}\left(\frac{2-\nu_1}{E_1} + \frac{2-\nu_2}{E_2}\right)\right)$
σ_1	kg/s	Dämpfungsfaktor Reibung	empirisch zu bestimmen
σ_2	kg/s	Viskose Reibung	empirisch zu bestimmen
ν_s	m/s	Stribeck-Geschwindigkeit	empirisch zu bestimmen
F_G	N	Gleitreibungskraft	$\mu_G \cdot m_{rob} \cdot g$
F_H	N	Haftreibungskraft	$\mu_H \cdot m_{rob} \cdot g$

Tabelle 6.2: Überblick über Konstanten

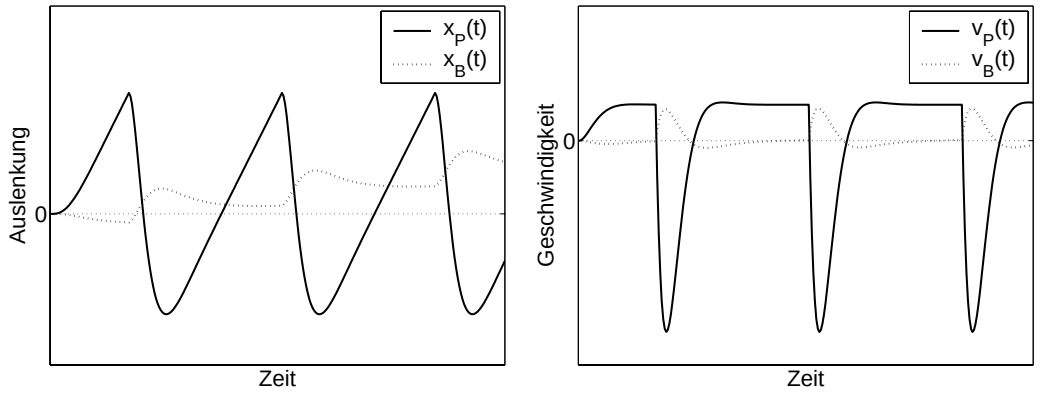
abhängig und kann daher berechnet werden:

$$\begin{aligned} \nu_p(t) = \frac{U_0 c_3}{c_{22}} \Bigg(& \exp(c_{20}t) \int [(c_7 - \sin^{n_2}(\pi f t)) \exp(-c_{20}t)\, dt] \\ & - \exp(c_{21}t) \int [(c_7 - \sin^{n_2}(\pi f t)) \exp(-c_{21}t)\, dt] \Bigg) \end{aligned} \tag{6.39}$$

Mit den Abkürzungen

$$\begin{aligned} c_1 &= \frac{k_1}{m_P}; c_2 = \frac{k_3}{m_P}; c_3 = \frac{k_2}{m_P} \\ c_{20} &= \frac{c_{22} + c_2}{2}; c_{21} = \frac{c_{22} - c_2}{2}; c_{22} = \sqrt{c_2^2 - 4c_1} \end{aligned}$$

Eine Lösung für $n_2 = 6$ ist im Anhang (Gleichung A.1) angegeben. Die Beinbewegung v_B kann mit numerischen Methoden leicht berechnet werden. Die für den *Stick-Slip*-Antrieb charakteristischen Phasen des Verharrens und Weitergleitens durch den Wechsel von Haft- und Gleitreibung sind in der numerischen Simulation (vgl. Abb. 6.11) gut zu erkennen.



(a) Positionen (b) Geschwindigkeiten

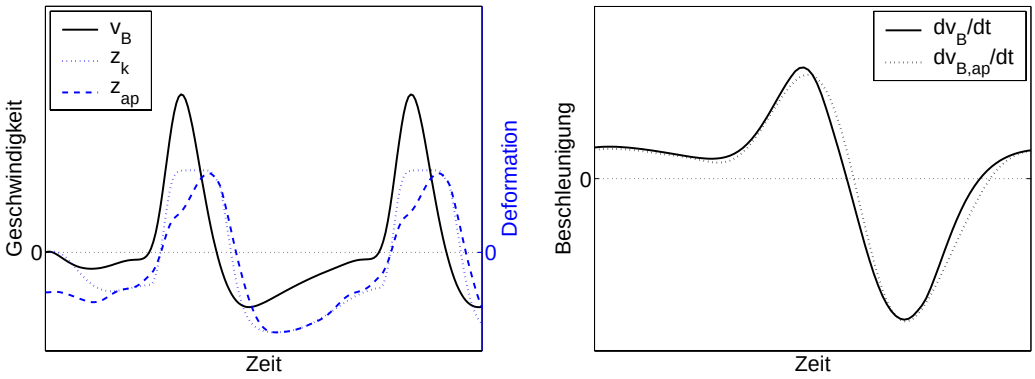
Abbildung 6.11: Verlauf der Geschwindigkeiten und Positionen anhand der numerischen Simulation des dynamischen Modells

Um die Bewegung eines realen Roboters zu simulieren, ist eine Kenntnis der in Tabelle 6.2 aufgeführten Parameter notwendig. Einige der genannten Konstanten lassen sich aus dem Roboteraufbau herleiten bzw. sind in Tabellenwerken nachzuschlagen. Eine Reihe von Parametern ist jedoch nur empirisch bestimmbar. Zu diesem Zweck müssten Messdaten der Roboter zur Verfügung stehen, aus denen detailliert das dynamische Verhalten für jeden Zyklus erkennbar wäre. Bei einer typischen Betriebsfrequenz im kHz-Bereich ist dies jedoch vom Messsystem nicht leistbar. Aus diesem Grund können die Parameter nur dahingehend bestimmt werden, dass das mittlere Geschwindigkeitsprofil der Beinbewegung nachgebildet wird (siehe Abschnitt 6.2.6).

6.2.3 Stationäres Modell

Der dynamische Verlauf ist für die Zwecke der Approximation an das inverse Bewegungsverhalten im Sinne einer Steuerung irrelevant, tatsächlich interessiert vielmehr der stationäre Zustand des Systems, nachdem für eine gewisse Zeit ein beliebiges, aber festes Stellsignal aufgegeben wurde. Dafür muss das DGL-System gelöst werden. Da vor allem die Gleichung für F_R eine analytische Lösung verhindert, muss eine Vereinfachung für $z_k(t)$ gefunden werden. Dieser Term beschreibt den Einfluss der Reibung und ist damit hauptverantwortlich dafür, dass

eine gerichtete Bewegung zustande kommt. Der Verlauf von $z_k(t)$ hängt unter anderem von den Parametern F_H,F_G und v_s ab. Sein Verhalten ist zwar periodisch mit der von der Spannung vorgegebenen Frequenz, jedoch wird sein Verhalten von der Beingeschwindigkeit maßgeblich beeinflusst, was ein asymmetrisches und von Nebenschwingungenen überlagertes Verhalten hervorruft (vgl. Abb. 6.12). Die



(a) Vergleich der Approximation $z_{ap}(t)$ und der originalen Funktion $z_k/t)$

(b) Auswirkung auf die Beingeschwindigkeit

Abbildung 6.12: Approximation von $z_k(t)$ durch eine Fourier-Reihe mit $n = 6$ gemäß Gleichung 6.40

Amplitude der Schwingung steigt exponentiell mit der Periodendauer T. Unterhalb einer kritischen Spannung U_{krit} ist die Amplitude der Schwingung zusätzlich näherungsweise linear von der Spannung U_0 abhängig. Die kritische Spannung wird wiederum von den genannten Parametern beeinflusst und lässt sich nicht pauschal angeben. Gesucht ist ein Ansatz, der die Periodizität des Verhaltens gut wiedergibt, von $v_B(t)$ entkoppelt (oder nur linear gekoppelt) und differenzierbar ist. Der klassische Ansatz zur Lösung eines solchen Problems ist eine Fourier-Reihenentwicklung. Wegen der Asymmetrie des Verhaltens ist es günstiger, die trigonometrischen Funktionen zu potenzieren. Gleichung 6.40 liefert auch bei unterschiedlichen Stribeck-Geschwindigkeiten und unterschiedlichen Haft- und Gleitreibungskräften noch akzeptable Lösungen.

$$\begin{aligned} z_k(t) &= h(U_0, f) \sum_{i=1}^{n_1} \left(a_i \cdot \sin^i(\pi f t) + b_i \cdot \cos^i(\pi f t) \right) \\ \text{mit } h(U_0, f) &= U_0 \left(c_{23} + c_{24} \exp\left(\frac{c_{25}}{f + c_{26}} \right) \right) \end{aligned} \tag{6.40}$$

Entscheidend ist bei der Approximation weniger, wie gut $z_k(t)$ selbst nachgebildet wird, sondern vielmehr wie gut das dynamische Verhalten der Beingeschwindigkeit $v_B(t)$ sich dem tatsächlichen Verlauf ähnelt (vgl. Abb. 6.12(b)).

Darüber hinaus erwies sich in der numerischen Simulation der Luftwiderstand als nahezu bedeutungslos (vgl. Abschnitt 6.2.6), so dass $F_W \approx 0$ angenommen und damit aus dem Gleichungssystem eliminiert werden kann. Mit diesen Vereinfachungen können die Gleichungen 6.34 und 6.35 ersetzt werden durch:

$$\frac{dv_P}{dt} = \frac{1}{m_P}\left[k_u u(t) - k_1(x_P(t) - x_0(t)) + k_2 v_P(t)\right] \tag{6.41}$$

$$\frac{dv_B}{dt} = \frac{1}{m_B}\left[-k_3 m_p \frac{dv_P}{dt} - (k_4 + \sigma_2)\, v_B(t) - \sigma_0 z_k(t)\right] \tag{6.42}$$

Das Gleichungssystem ist jetzt analytisch lösbar, wobei die Struktur der Lösung von den Exponenten n_1 und n_2 abhängt. Die Gleichung für die Beingeschwindigkeit nimmt folgende allgemeine Form an:

$$v_B(t) = \frac{1}{\sum_{i=0}^{n_1+n_2} \lambda_{1,i} f^{2i}} \left(\sum_{i=0}^{n_1+n_2} f^{2i}\gamma(2i)p(2i) + \sum_{i=0}^{n_1+n_2-1} f^{2i+1}\gamma(2i+1)q(2i+1) \right) \tag{6.43}$$

mit:

$$\gamma(\kappa) = \pi^\kappa \left(\prod_{\kappa+1}^{n_2}(c_{22} - c_2)(c_{22} + c_2)\right)\left(\prod_{\kappa+1}^{2n_1} c_5\right)\left(\prod_{\kappa+1}^{1} c_6\right) \tag{6.44}$$

$$\begin{aligned} p(\kappa) =& \lambda_{2,\kappa} h \exp(-c_5 t) + h \sum_{j=0}^{n_1-2} \lambda_{3,\kappa,j} \sin(\pi f t)\cos^{2j+1}(\pi f t) + \\ &+ h \sum_{j=0}^{n_1} \lambda_{4,\kappa,j} \cos^{2j}(\pi f t) + \psi_1(\kappa) + \psi_2(\kappa) \\ \psi_1(\kappa) =& \begin{cases} 0, & \text{wenn } \kappa = 0 \\ U_0 \sum_{j=0}^{n_2/2} \lambda_{5,\kappa,j} \cos^{2j}(\pi f t), & \text{wenn } \kappa > 0 \end{cases} \\ \psi_2(\kappa) =& \begin{cases} 0, & \text{wenn } \kappa < n_2 \\ \lambda_{6,\kappa} U_0 \exp(-c_5 t), & \text{wenn } \kappa \geqslant n_2 \end{cases} \end{aligned} \tag{6.45}$$

$$\begin{aligned} q(\kappa) =& \lambda_{7,\kappa} h \exp(-c_5 t) + h \sum_{j=0}^{n_1-1} \lambda_{8,\kappa,j} \sin(\pi f t)\cos^{2j+1}(\pi f t) + \\ &+ h \sum_{j=0}^{n_1-1} \lambda_{9,\kappa,j} \cos^{2j}(\pi f t) + U_0 \sum_{j=0}^{n_2/2-1} \lambda_{10,\kappa,j} \sin(\pi f t)\cos^{2j+1}(\pi f t) \end{aligned} \tag{6.46}$$

Die Koeffizientenmatrizen λ_i sind auf vielfältige Weise mit den Roboterparametern verknüpft. Um die Arbeit etwas zu erleichtern, ist zunächst zu untersuchen, ob sich die Gleichungen 6.43 bis 6.46 noch vereinfachen lassen.

Es interessiert jetzt nicht mehr der dynamische Verlauf, sondern vor allem die Relationen, die sich im stationären Zustand einstellen. Im stationären Fall ($t \to \infty$) können die Terme mit $\sin(\pi f t) \cdot \cos(\pi f t)$ vernachlässigt werden, da sie nur das periodische Verhalten - hervorgerufen durch die Plattformbewegung - wiedergeben und nichts zur mittleren Geschwindigkeit beitragen.

$$\int_{t\to\infty} \frac{1}{t} \sin(\pi f t) \cos^n(\pi f t) = 0 \tag{6.47}$$

Da c_5 immer positiv ist, laufen alle Terme, die $\exp(-c_5 t)$ enthalten gegen Null. Die Ausdrücke $p(\kappa)$ und $q(\kappa)$ vereinfachen sich dann zu:

$$\begin{aligned}
\overline{p}(\kappa) =& h \sum_{j=0}^{n_1} \lambda_{4,\kappa,j} \overline{\cos^{2j}}(\pi f t) + U_0 \sum_{j=0,\kappa>0}^{n_2/2} \lambda_{5,\kappa,j} \overline{\cos^{2j}}(\pi f t) \\
\overline{q}(\kappa) =& h \sum_{j=0}^{n_1-1} \lambda_{9,\kappa,j} \overline{\cos^{2j}}(\pi f t)
\end{aligned} \tag{6.48}$$

Der Mittelwert (Effektivwert) der Kosinusse kann berechnet werden durch:

$$\begin{aligned}
\overline{\cos^n}(\pi f t) &= \frac{\overline{(\cos^{n-2}}(\pi f t))}{2^{n-1}} \left(2 + \frac{(\frac{1}{2}n - 1)^2}{\frac{1}{8}n^2 - \frac{1}{4}n} \right) \\
&= \frac{1}{2^{n-1}} rd \left(\frac{2 \cdot 4^{\left(\frac{n}{2}-1\right)} \Gamma\left(\frac{n+1}{2}\right)}{\sqrt{\pi}\Gamma\left(\frac{n}{2}+1\right)} \right)
\end{aligned} \tag{6.49}$$

Die Gamma-Funktion ist bekanntermaßen definiert mit:

$$\Gamma(\kappa) = \int_{t=0}^{\infty} \exp(-t) t^{\kappa-1} \tag{6.50}$$

Damit ist nun eine geschlossene, nicht rekursive Lösung des Modells gewonnen, die jedoch immer noch recht unübersichtlich ist:

$$\overline{v_B} = \frac{1}{\sum_{i=0}^{n_1+n_2} \lambda_{1,i} f^{2i}} \left(\sum_{i=0}^{n_1+n_2} f^{2i} \gamma(2i) \overline{p}(2i) + \sum_{i=0}^{n_1+n_2-1} f^{2i+1} \gamma(2i+1) \overline{q}(2i+1) \right) \tag{6.51}$$

Auch hängt die Lösung immer noch von n_1 und n_2 ab. Bei der numerischen Simulation anhand des dynamischen Modells haben sich hier Werte von $n_1 \geqslant 6$ und $n_2 \geqslant 16$ als sinnvoll herausgestellt. Für eine computergestützte Berechnung ist der dabei entstehende Ausdruck kompakt genug, um eine Echtzeitfähigkeit zu gewährleisten. Eine Darstellung dieser Lösung würde jedoch den Rahmen dieser Arbeit sprengen, so dass aus Platzgründen im Anhang exemplarisch die Lösung für $n_1 = 2$ und $n_2 = 2$ aufgeführt wurde (Gleichung A.2).

6.2.4 Inverses Modell

Das Modell gemäß Gleichungen 6.43 und 6.48 stellt den Zusammenhang zwischen dem Einfluss der Stellsignale (U_0, f), dem Systemverhalten (k_i, m_B, m_P) und der Reibung ($\sigma_0, \sigma_1, \sigma_2$) einerseits und der resultierenden mittleren Endgeschwindigkeit des Roboters andererseits her. Damit ist ein stationäres kinematisches Modell eines mobilen Roboters gefunden, der das *Stick-Slip*-Prinzip als Antrieb nutzt. Dieses Modell kann nun für Simulation und Reglerentwurf genutzt werden. Gesucht ist aber für die Steuerungsebene ein inverses kinematisches Modell, um direkt eine Geschwindigkeit vorgeben zu können und die entsprechende Frequenz und Spannungsamplitude zu berechnen. Eine direkte Invertierung des Vorwärts-Modells ist jedoch nicht möglich. Es bieten sich nun zwei Möglichkeiten an:

(a) Es wird ein numerisches Verfahren eingesetzt, um die Nullstellen der Gleichung

$$\overline{v_B}(f, U_0) - v_{B,soll} = 0 \tag{6.52}$$

zu bestimmen. Dabei ist $v_{B,soll}$ die gewünschte mittlere Geschwindigkeit des Roboters und $\overline{v_B}(f, U_0)$ die berechnete, sich ergebende Geschwindigkeit gemäß 6.51 unter Vorgabe von f und U_0. Die Optimieraufgabe besteht dann darin, ein f^* bzw. U_0^* so zu finden, dass diese Gleichung erfüllt ist. Diese Optimierung muss während des Roboterbetriebs bei jedem neuen Stellsignal erneut durchgeführt werden.

(b) Es wird anhand der Struktur des stationären Vorwärtsmodells approximativ ein Ansatz gewählt, der invertierbar ist und gut genug ist - bei Wahl geeigneter Werte für die Parameter - das Bewegungsverhalten der Roboter noch wiederzugeben.

Variante (b) ist wegen der besseren Echtzeitfähigkeit vorzuziehen. Reduziert man die Gleichungen des stationären Modells auf das Wesentliche, so erhält man folgende Struktur:

$$\overline{v_B}(f, U_0) = h(f, U_0) g_1(f) + U_0 g_2(f) \tag{6.53}$$

Eine Analyse der Ergebnisse der Simulationen anhand des stationären Modells ergibt, dass der Ausdruck $U_0 g_2(f)$ nur geringen Einfluss auf das Gesamtergebnis hat. Ferner kann beobachtet werden, dass die Funktion $g_1(f)$ für alle untersuchten Roboter nur geringe Abhängigkeit von der Stellgröße f aufweist. Damit kann in einer groben Annäherung an das Systemverhalten der Ansatz

$$\overline{\nu_B}(f, U_0) \approx c_{27} \cdot h(f, U_0) = c_{27} \cdot U_0 \left(c_{23} + c_{24} \exp\left(\frac{c_{25}}{f + c_{26}} \right) \right) \tag{6.54}$$

aufgestellt werden. Dieser Ansatz lässt sich nun leicht invertieren und man erhält bei Vorgabe der Spannungsamplitude:

$$f = \frac{c_{25}}{\ln\left(\frac{\nu_{B,soll} - c_{23} c_{27} U_0}{c_{24} c_{27} U_0} \right)} - c_{26} \tag{6.55}$$

bzw. bei Vorgabe der Frequenz:

$$U_0 = \frac{\nu_{B,soll}}{c_{27} \left(c_{23} + c_{24} \exp\left(\frac{c_{25}}{f + c_{26}} \right) \right)} \tag{6.56}$$

Bei einer reinen Translation ist die Spannungsamplitude bei beiden untersuchten Robotermodellen konstant ($U_0 = U_{0,max}$) und nur die Frequenz wird geregelt, so dass nur Gleichung 6.55 ausgewertet werden muss. Die Parameter c_{23} bis c_{27} aus Gleichung 6.55 sind in einem Optimiervorgang zu bestimmen, bleiben aber während des Roboterbetriebs konstant, so dass im Rahmen der Steuerung diese Gleichung für jedes neue Stellsignal nur einmal ausgewertet werden muss. Es bleibt jedoch festzuhalten, dass dieses Modell aufgrund der zahlreichen Vereinfachungen nur eine grobe Approximation darstellen kann.

6.2.5 Rotation

Die bisherigen Ausführungen bezogen sich auf rein translatorische Bewegung. Verantwortlich für die Rotation um eine Drehachse ist das Drehmoment gemäß Gleichung 6.9. Die Winkelbeschleunigung ergibt sich mit:

$$\frac{d\omega}{dt} = \frac{1}{J} \sum_{i}^{n_3} \vec{r}_i x \vec{F}_i \tag{6.57}$$

mit n_3 = Anzahl der Aktoren des Roboters. Unter der Annahme, dass weder Hebel r noch die Kraft eine vertikale Komponente aufweisen, vereinfacht sich der Ausdruck zu:

$$\frac{d\omega}{dt} = \frac{1}{J} \sum_{i}^{n_3} |\vec{r}_i||\vec{F}_i| sin(\alpha_i) \tag{6.58}$$

mit α_i = Winkel zwischen Hebelarm und Kraftwirkrichtung θ des Aktors i (vgl. Abb. 6.13). Da sowohl die Hebel $r_i = |\vec{r}_i|$ als auch die Winkel α_i der Aktoren im Allgemeinen unterschiedlich sind, unterscheidet sich der Berechnungsablauf von dem einer reinen Translation. Es wird daher für jeden Aktor zunächst einzeln eine resultierende Kraft $F_{P,i}$ analog Gleichung 6.11 gebildet. Unter Ausnutzung von Gleichung 6.57 berechnet sich dann die Winkelbeschleunigung der Plattform:

$$\frac{d\omega_P}{dt} = \frac{1}{J_P} \sum_i^{n_3} r_i \left(F_{U,i} - F_{W,i} - F_{M,i}\right) \sin(\alpha_{P,i}) \tag{6.59}$$

Obwohl die Berechnung der Kräfte $F_{U,i}$,$F_{W,i}$,$F_{M,i}$ formell genauso vonstatten

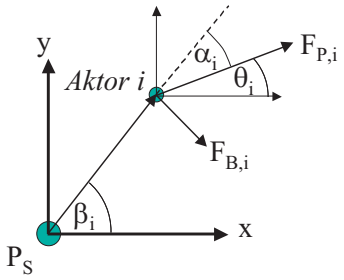


Abbildung 6.13: Kräfte und Winkel bei der Rotation ($\alpha = \beta - \theta$)

geht wie im Falle der Translation, ist jedoch zu beachten, dass die entsprechenden Parameter hier im Prinzip für jeden Aktor unterschiedliche Werte annehmen können. Werden die Aktoren mit gleichen Frequenzen und Spannungsamplituden, aber unterschiedlichen Winkeln angesteuert, lässt sich die Annahme treffen, dass $F_{P,i} = F_P/n_3$. Damit folgt:

$$\frac{d\omega_P}{dt} = \frac{F_P}{n_3 J_P} \sum_i^{n_3} r_i \sin(\alpha_{P,i}) \qquad \text{wenn } \forall f_i : f_i = f \tag{6.60}$$

Bei Kenntnis des Geschwindigkeitsverlaufs der Plattform kann die Beinbeschleunigung analog der Translation berechnet werden. Die Beine werden auch bei rotatorischen Bewegungen von der Beschleunigung der Plattform insgesamt angetrieben:

$$\frac{d\omega_B}{dt} = \frac{1}{J_B} \left(- \left(k_7 J_P \frac{d\omega_P}{dt} + k_8 \omega_B \right) - \sum_i^{n_3} r_i \left(F_{R,i} + F_{W,i}\right) \right) \tag{6.61}$$

Die Geschwindigkeiten, die zur Berechnung der Reibungskräfte $F_{R,i}$ herangezogen werden, sind bei der Rotation für jeden Aktor unterschiedlich[4]. An dieser Stelle muss daher Gleichung 6.36 für jeden Aktor mit der Beingeschwindigkeit

$$v_{B,i}(t) = r_i \omega_B(t) \tag{6.62}$$

ausgewertet werden.

Das Differentialgleichungssystem für rotatorische Bewegungen hat also ein sehr ähnliches Aussehen wie das für Translation, so dass es nicht überraschen kann, dass auch die Lösung eine ähnliche Struktur und ähnliche Probleme aufweist. Das stationäre Modell basiert dann auf folgender Gleichung

$$\frac{d\omega_B}{dt} = \frac{1}{J_B}\left(-k_7 J_P \frac{d\omega_P}{dt} + \left(k_8 + \sigma_2 \sum_i^{n_3} r_i\right)\omega_B(t) + \sigma_0 \sum_i^{n_3} r_i z_{k,i}(t)\right) \tag{6.63}$$

Die Abhängigkeit vom Steuerwinkel θ ist über $\dot{\omega}_P$ gewährleistet. Für $z_{k,i}(t)$ kann dann wieder eine Approximation wie in Gleichung 6.40 gewählt werden, die von $U_{0,i}$ und f_i abhängt. Das sich ergebende Gleichungssystem kann erneut gelöst werden, wobei hier aber aus Platzgründen auf seine Darstellung verzichtet wird. Man kann sich aber leicht klar machen, dass die Lösung sich nur in wenigen Details von der Translation unterscheidet.

Inverses Modell

Die stationäre Lösung hat im Falle der Rotation die Struktur

$$\omega_B(f_i, U_{0,i}, \theta_i) = \sum_i^{n_3} h_i(f_i, U_{0,i}) g_1(f_i) + \sum_i^{n_3} U_{0,i} g_2(f_i, \theta_i) \tag{6.64}$$

Da nun nicht mehr nur eine Zielgröße (im Falle der Translation die Frequenz f), sondern $2n_3$ Zielgrößen (Frequenzen bzw. Spannungsamplituden und Winkel) gesucht sind, ist eine Invertierung nicht mehr eindeutig möglich.

Rotation lässt sich grundsätzlich auf zwei Wegen erzielen: Es können entweder die Aktoren mit unterschiedlichen Frequenzen oder Spannungsamplituden beaufschlagt werden oder auch die Ansteuerwinkel θ unterschiedlich gewählt werden. Eine reine Rotation (ohne überlagerte translatorische Bewegungen) wird er-

[4]Da der Richtungsvektor bei rein rotatorischer Beinbewegung immer senkrecht auf dem Hebelarm steht, entfällt der Term $sin(\alpha_{B,i})$ in Gleichung 6.61

zielt, wenn die Gleichgewichtsbedingungen

$$\begin{aligned} F_{B,x} &= \sum_{i}^{n_3} |F_i| \cos(\theta_i) = 0 \\ F_{B,y} &= \sum_{i}^{n_3} |F_i| \sin(\theta_i) = 0 \\ M_t &= \sum_{i}^{n_3} r_i |F_i| \sin(\alpha_i) = J \frac{d\omega_B}{dt} \sin(\alpha_i) \end{aligned} \tag{6.65}$$

erfüllt sind. Bei einem Roboter mit drei Aktoren sind hierdurch die erforderlichen Kräfte und damit die Frequenzen bzw. Spannungsamplituden in Abhängigkeit der Winkel festgelegt. Bei Robotern mit mehr als drei Aktoren können Kräfte bzw. Winkel teilweise frei gewählt werden. Als Beispiel sei hier nur die Lösung für drei Aktoren angegeben ($n_3 = 3$):

$$\begin{aligned} F_{B,1} &= J \frac{d\omega_B}{dt} \left(\frac{\cos(\theta_3)\sin(\theta_2) - \sin(\theta_3)\cos(\theta_2)}{\varphi} \right) = J \frac{d\omega_B}{dt} \frac{\phi_1}{\varphi} \\ F_{B,2} &= J \frac{d\omega_B}{dt} \left(\frac{\sin(\theta_3)\cos(\theta_1) - \sin(\theta_1)\cos(\theta_3)}{\varphi} \right) = J \frac{d\omega_B}{dt} \frac{\phi_2}{\varphi} \\ F_{B,3} &= J \frac{d\omega_B}{dt} \left(\frac{\cos(\theta_2)\sin(\theta_1) - \cos(\theta_1)\sin(\theta_2)}{\varphi} \right) = J \frac{d\omega_B}{dt} \frac{\phi_3}{\varphi} \end{aligned} \tag{6.66}$$

mit

$$\begin{aligned} \varphi = {} & r_1 \sin(\alpha_1) \left(\cos(\theta_3)\sin(\theta_2) - \cos(\theta_2)\sin(\theta_3)\right) \\ & + r_2 \sin(\alpha_2) \left(\cos(\theta_1)\sin(\theta_3) - \cos(\theta_3)\sin(\theta_1)\right) \\ & + r_3 \sin(\alpha_3) \left(\cos(\theta_2)\sin(\theta_1) - \cos(\theta_1)\sin(\theta_2)\right) \end{aligned}$$

$$\text{wenn } \theta_1 \neq \theta_2 \vee \theta_1 \neq \theta_3$$

Die Kräfte sind also durch die Winkelbeschleunigung, die Steuerwinkel und die geometrische Anordnung der Aktoren vorgegeben. Im einfachsten Fall wird der Ansteuerwinkel so gewählt, dass er senkrecht auf dem Hebelarm steht:

$$\alpha_{P,i} = \pm\pi/2 \Rightarrow \theta_i = \beta_i \mp \pi/2 \tag{6.67}$$

Ein Sonderfall ist dann gegeben, wenn die Aktoren auch auf einem (gedachten) Kreis um den Roboterschwerpunkt gleichmäßig angeordnet sind ($\Delta\beta = \frac{2\pi}{n_3}$). Gleichung 6.66 vereinfacht sich dann zu:

$$F_{B,i} = \frac{J}{3} \frac{d\omega_B}{dt} \tag{6.68}$$

In diesem Fall müssten alle Aktoren mit der gleichen Frequenz und Spannungsamplitude angesteuert werden. Allgemein folgt aus Gleichung 6.66 für die tangentiale Geschwindigkeit des Aktors i:

$$v_{B,i}(t) = \frac{J}{m_i}\phi_i\omega_B(t) \tag{6.69}$$

Mit der gleichen Argumentation wie bei der Invertierung des translatorischen Modells in Abschnitt 6.2.4 lässt sich analog zu Gleichung 6.55 und 6.71 herleiten:

$$f_i = \frac{c_{25}}{\ln\left(\frac{\frac{J}{m_i}\phi_i\,\omega_{B,soll} - c_{23}c_{27}U_0}{c_{24}c_{27}U_0}\right)} - c_{26} \tag{6.70}$$

bzw.

$$U_{0,i} = \frac{\frac{J}{m_i}\phi_i\omega_{B,soll}}{c_{27}\left(c_{23} + c_{24}\exp\left(\frac{c_{25}}{f+c_{26}}\right)\right)} \tag{6.71}$$

Damit ist eine Berechnung der Frequenz oder Spannungsamplitude jedes Aktors als Funktion der gewünschten Winkelgeschwindigkeit und unter Berücksichtigung der Geometrie des Roboters möglich. Die für die Berechnung erforderliche äquivalente Masse m_i jedes Aktors erschliesst sich nicht unmittelbar und ist natürlich von der Anordnung der Aktoren abhängig. Mit guter Näherung kann aber für alle untersuchten Roboter davon ausgegangen werden, dass $m_i = m_{rob}/n_3$ gilt.

Überlagerte Translation und Rotation

Um eine vollständige Holonomie des Roboters zu erzielen, muss zugelassen werden, dass Translation und Rotation gleichzeitig auftreten. Das dynamische wie auch das stationäre Modell für Translation und für Rotation gilt genauso für diesen Fall. Bei der Invertierung ist jedoch zu beachten, dass die Bedingungen nach Gleichung 6.65 nicht mehr gelten. Stattdessen gilt:

$$\begin{aligned} F_{B,x} &= \sum_i^{n_3} |F_i|\cos(\theta_i) = F_x \\ F_{B,y} &= \sum_i^{n_3} |F_i|\sin(\theta_i) = F_y \\ M_t &= \sum_i^{n_3} r_i|F_i|\sin(\alpha_i) = J\frac{d\omega_B}{dt} \end{aligned} \tag{6.72}$$

Dies führt bei drei Aktoren zu:

$$
\begin{aligned}
F_1 &= J\frac{d\omega_B}{dt}\frac{\phi_1}{\varphi} + m_1\frac{dv_{B,x}}{dt}\frac{\phi_{x,1}}{\varphi} + m\frac{dv_{B,y}}{dt}\frac{\phi_{y,1}}{\varphi} \\
F_2 &= J\frac{d\omega_B}{dt}\frac{\phi_2}{\varphi} + m_2\frac{dv_{B,x}}{dt}\frac{\phi_{x,2}}{\varphi} + m\frac{dv_{B,y}}{dt}\frac{\phi_{y,2}}{\varphi} \\
F_3 &= J\frac{d\omega_B}{dt}\frac{\phi_3}{\varphi} + m_3\frac{dv_{B,x}}{dt}\frac{\phi_{x,3}}{\varphi} + m\frac{dv_{B,y}}{dt}\frac{\phi_{y,3}}{\varphi} \\
&\text{mit} \\
\phi_{x,1} &= r_2\sin(\alpha_2)\sin(\theta_3) - r_3\sin(\alpha_3)\sin(\theta_2) \\
\phi_{y,1} &= r_3\sin(\alpha_3)\cos(\theta_2) - r_2\sin(\alpha_2)\cos(\theta_3) \\
\phi_{x,2} &= r_3\sin(\alpha_3)\sin(\theta_1) - r_1\sin(\alpha_1)\sin(\theta_3) \\
\phi_{y,2} &= r_1\sin(\alpha_1)\cos(\theta_3) - r_3\sin(\alpha_3)\cos(\theta_1) \\
\phi_{x,3} &= r_1\sin(\alpha_1)\sin(\theta_2) - r_2\sin(\alpha_2)\sin(\theta_1) \\
\phi_{y,3} &= r_2\sin(\alpha_2)\cos(\theta_1) - r_1\sin(\alpha_1)\cos(\theta_2)
\end{aligned}
\tag{6.73}
$$

Sobald die Winkel α_i alle gleich sind (wie dies typischerweise bei Rotation der Fall ist) und die Entfernungen vom Drehpunkt auch gleich sind, wird $\sum \phi_{x,i}$ und $\sum \phi_{y,i}$ zu Null, es kommt dann also zu keiner Translation (wie zu erwarten war). Sind hingegen die Winkel θ_i alle gleich (wie bei reiner Translation) sind die Ausdrücke ϕ_i nicht mehr definiert. Es wird damit klar, dass diese Gleichung nur dann anzuwenden ist, wenn eine rotatorische Komponente vorhanden ist.

Damit lässt sich nun bei Kenntnis bzw. Vorgabe der translatorischen Geschwindigkeitskomponenten $v_{B,x}(t)$ und $v_{B,y}(t)$, der Winkelgeschwindigkeit $\omega_B(t)$ und der Steuerwinkel $\theta(i,t)$ unter Berücksichtigung der Anordnung der Aktoren auf dem Roboter die benötigten Kräfte für jeden Aktor und damit - unter Zuhilfenahme der Invertierung - die Frequenz bzw. die Spannungsamplitude bestimmen. Die notwendigen Steuerwinkel sind hier jedoch unbekannt. Bei der reinen Translation ergaben sie sich aus $\theta = \arctan(F_y, F_x)$, bei der reinen Rotation aus $\theta = \beta \pm \pi/2$. Bei drei unabhängigen Gleichungen und sechs Unbekannten sind drei Größen im Prinzip frei wählbar. Wenn die drei Winkel gewählt werden, können die dazugehörigen Frequenzen bzw. Spannungsamplituden berechnet werden. Es ist dabei jedoch zu bedenken, dass es für die Frequenz wie für die Amplitude Ober- und Untergrenzen gibt, die bei ungeeigneter Wahl der Winkel evtl. über- oder unterschritten werden. Aus folgender einfacher geometrischer Beziehung können ge-

eignete Winkel jedoch leicht berechnet werden:

$$\begin{aligned} m v_B \cos(\theta_i) &= \frac{J}{r_i}\omega \cos(\theta_r) + m v_t \cos(\theta_t) \\ m v_B \sin(\theta_i) &= \frac{J}{r_i}\omega \sin(\theta_r) + m v_t \sin(\theta_t) \\ &\text{mit} \\ v_B &= \sqrt{v_{B,x}^2 + v_{B,y}^2} \\ \theta_r &= \beta_i \pm \pi/2 \end{aligned} \tag{6.74}$$

Dabei ist θ_t der Winkel, der zu wählen wäre, wenn nur Translation erforderlich ist, v_t die gewünschte Translationsgeschwindigkeit und ω die gewünschte Winkelgeschwindigkeit

Aus Gleichung 6.74 folgt:

$$\theta_i = \arctan(\theta_{x,i}, \theta_{y,i})$$

mit

$$\theta_{x,i} = \frac{J\omega \sin(\beta_i)\cos(\frac{\pi}{2}) - J\omega\cos(\beta_i)\sin(\frac{\pi}{2}) + m v_t \sin(\theta_t) r_i}{\sqrt{2 r_i J \omega m v_t (\cos(-\theta_r)\cos(\theta_t) - \sin(-\theta_r)\sin(\theta_t)) + (m v_t)^2 r_i^2 + (J\omega)^2}}$$

$$\theta_{y,i} = \frac{J\omega \cos(\beta_i)\cos(\frac{\pi}{2}) + J\omega\sin(\beta_i)\sin(\theta_r) + m v_t \cos(\theta_t) r_i}{\sqrt{2 r_i J \omega m v_t (\cos(-\theta_r)\cos(\theta_t) - \sin(-\theta_r)\sin(\theta_t)) + (m v_t)^2 r_i^2 + (J\omega)^2}} \tag{6.75}$$

Damit sind die Steuerwinkel als Funktion des gewünschten Drehmoments und Translationskraft bestimmt und können in Gleichung 6.73 eingesetzt werden. Eine Invertierung erfolgt dann analog zu reiner Translation und reiner Rotation.

6.2.6 Ergebnisse

In die zu untersuchenden Modelle fließt eine Vielzahl von Parametern ein, die zum Teil eine physikalische Bedeutung haben (vgl. Tabelle 6.2). Für diese Parameter kann bezogen auf den jeweiligen Roboter eine Abschätzung gegeben werden. Ein anderer Teil der Parameter hat jedoch ausschließlich mathematische Bedeutung und muss in einem Optimierungsprozess gefunden werden.

Eine dynamische Simulation ist auf der bestehenden Datenbasis nur sehr eingeschränkt möglich. Zum einen lässt das Messsystem Messzyklen von weniger als ~200 ms nicht zu. Zum anderen entziehen sich bestimmte Größen der Messbarkeit: Es ist im Prinzip nur eine Messung von $v_B + v_P$ möglich, da die Messmarken

	Dynamisches Modell		Stationäres Modell		Inverses Modell [b]	
	Transl.	Rotation	Transl.	Rotation	Transl.	Rotation
Miniman3	3.4	4.0	2.2	2.0	4.8	5.0
Miniman4	3.9	5.7	1.3	1.0	4.6	6.7
MiCRoN0	11.3	3.5	10.4	3.5	13.6	6.4

Tabelle 6.3: Minimaler Fehler (MAE) in % der mechanischen Modelle

auf der Roboteroberseite angebracht sind, was eine getrennte Messung unmöglich macht. Die Abgleichbarkeit des dynamischen Modells mit den Messdaten ist somit nur bedingt möglich.

Es wurden die Simulationen und Parameteranpassungen für drei Modelle durchgeführt, nämlich für

- das dynamische Modell gemäß Abschnitt 6.2.2
- das stationäre Modell[5] gemäß Abschnitt 6.2.3
- und das inverse Modell Typ b gemäß Abschnitt 6.2.4

Die Ergebnisse (vgl. Tabelle 6.3) zeigen eine recht gute Approximation aller Modelle an das Bewegungsverhalten der Miniman-Roboter. Es wird jedoch deutlich, dass das kinematische Modell des MiCRoN 0 nur bedingt brauchbar ist, die Modellfehler liegen hier durchweg über 10%. Einer der Gründe dürfte in der prozentual stärkeren Streuung der Messdaten zu suchen sein. Natürlich kann das entwickelte Modell aber auch nur eingeschränkt auf den MiCRoN-Roboter übertragen werden, da die Anordnung der Piezoelemente hier etwas anders ist.

Ein Vergleich der Modelle untereinander zeigt, dass das stationäre Modell dem dynamischen Modell bei allen Robotern überlegen ist. Das dürfte zum einen auf das erwähnte Problem der Abgleichbarkeit mit dem dynamischen Verhalten liegen, zum anderen aber auch daran, dass beim stationären Modell mehr Parameter zur Anpassung vorhanden sind (wegen der Fourier-Reihen) und dadurch eine bessere Approximation erzielt werden kann.

Sowohl beim Miniman III als auch beim Miniman IV können einzelne, begrenzte Bereiche identifiziert werden, in denen der Modellfehler stark ansteigt. Der Modellfehler bei der Approximation an rotatorisches Verhalten beim MiCRoN ist vergleichsweise gering, allerdings nur im Bereich niedriger Frequenzen.

Auffällig ist die Diskrepanz bei den Modellfehlern beim MiCRoN 0 zwischen Translation und Rotation. Durch die parasitäre Rotation während der Translation

[5] $n_1 = 6$

kommt es zu starken Beeinflussungen und einem sehr unruhigen Kurvenverlauf. Umgekehrt beeinflusst eine parasitäre Translation die Rotation kaum.

Parasitäre Rotation, d.h. unerwünschte Rotation, die bei rein translatorischer Ansteuerung auftritt, hat folgende mögliche Ursachen:

1. Ungenauigkeiten bei der Verarbeitung der Piezoelemente resultierend in unterschiedlichen Spannungswerten und/oder Auslenkungen der Piezoelemente
2. Ungenaue Anordnung der Piezobeine
3. Abnutzung/Verschleiß der Piezoelemente

Diese drei Ursachen sind analytisch nur schwer zu modellieren. Dies stellt ein Argument für den Einsatz von *Blackbox*-Modellen dar, bei denen die Modellfindung und -optimierung mit numerischen Methoden erfolgt (vgl. Abschnitt 6.3 und 6.4).

Der Einfluss des Luftwiderstands auf das dynamische Verhalten ist als gering einzuschätzen. Unabhängig davon, welche charakteristische Länge und Anströmfläche gewählt wird, liegt β_L in der Größenordnung von maximal 10^{-5} kg/s (wegen der kinematischen Viskosität von Luft $\eta_{Luft} = 1.5 \cdot 10^{-5} m^2/s$). Wird dieser Wert in einer Simulationsrechnung benutzt, so stellt sich heraus, dass der Einfluss von F_W auf das Gesamtsystem bei weniger als 1% liegt. Somit hat sich die Annahme bestätigt, dass diese Größe vernachlässigt werden kann.

6.3 Neuronales Modell

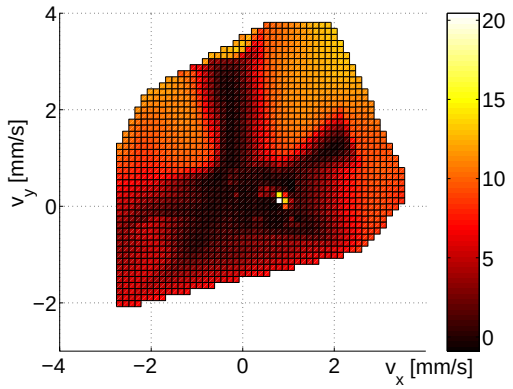
Der Einsatz von künstlichen neuronalen Netzen stellt eine Methode dar, die sich an der Nervenstruktur des Gehirns orientiert und versucht, diese auf mathematischen Wege nachzubilden. Sie wurde in den 1940er Jahren vom Neurophysiologen Warren McCulloch und vom Mathematiker Walter Pitts entworfen. Seitdem wurden - insbesondere seit den 1980er Jahren - zahlreiche Modelle entwickelt, die versuchen, diese Technik zu verbessern. Eine Vielzahl von unterschiedlichen Netztypen stehen zur Verfügung, von denen im Rahmen dieser Arbeit jedoch nur zwei näher untersucht werden.

6.3.1 Netzarten

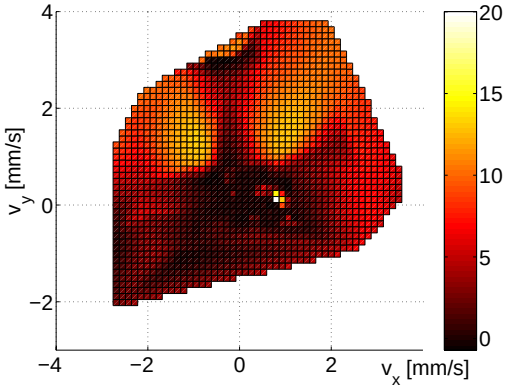
MLP-Netze

MLP-Netze[6] bilden die klassische Form der Feedforward-Netze, bei denen die Neuronen in Schichten angeordnet sind, wobei jeder Ausgang eines Neurons in

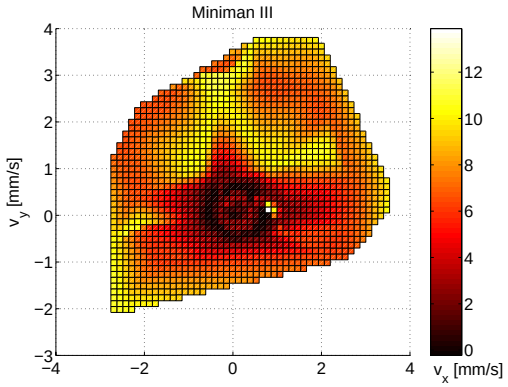
[6] MLP: ***M**ulti-**L**ayer-**P**erceptron*



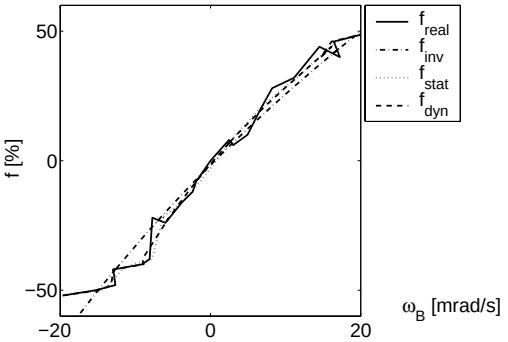
(a) Dynamisches Modell, Translation, Modellfehler in %



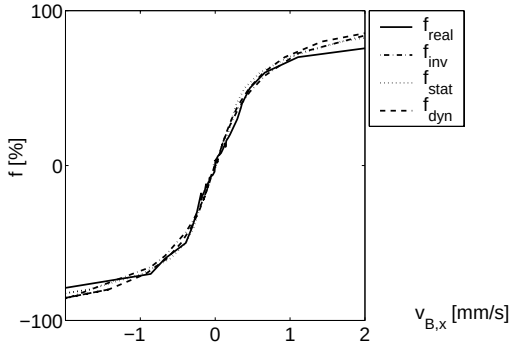
(b) Inverses Modell, Translation, Modellfehler in %



(c) Stationäres Modell, Translation, Modellfehler in %

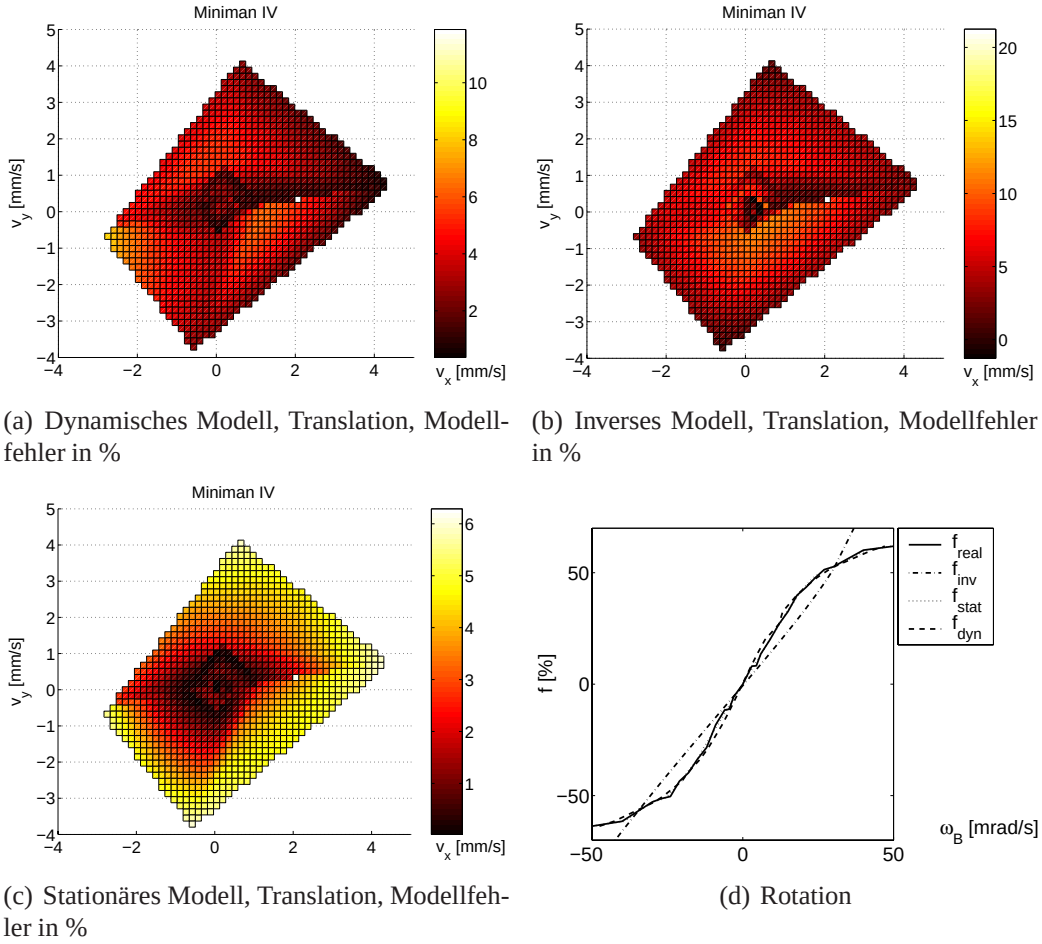


(d) Rotation



(e) x-Translation($v_y = 0$)

Abbildung 6.14: Ergebnisse für Miniman III



(a) Dynamisches Modell, Translation, Modellfehler in %

(b) Inverses Modell, Translation, Modellfehler in %

(c) Stationäres Modell, Translation, Modellfehler in %

(d) Rotation

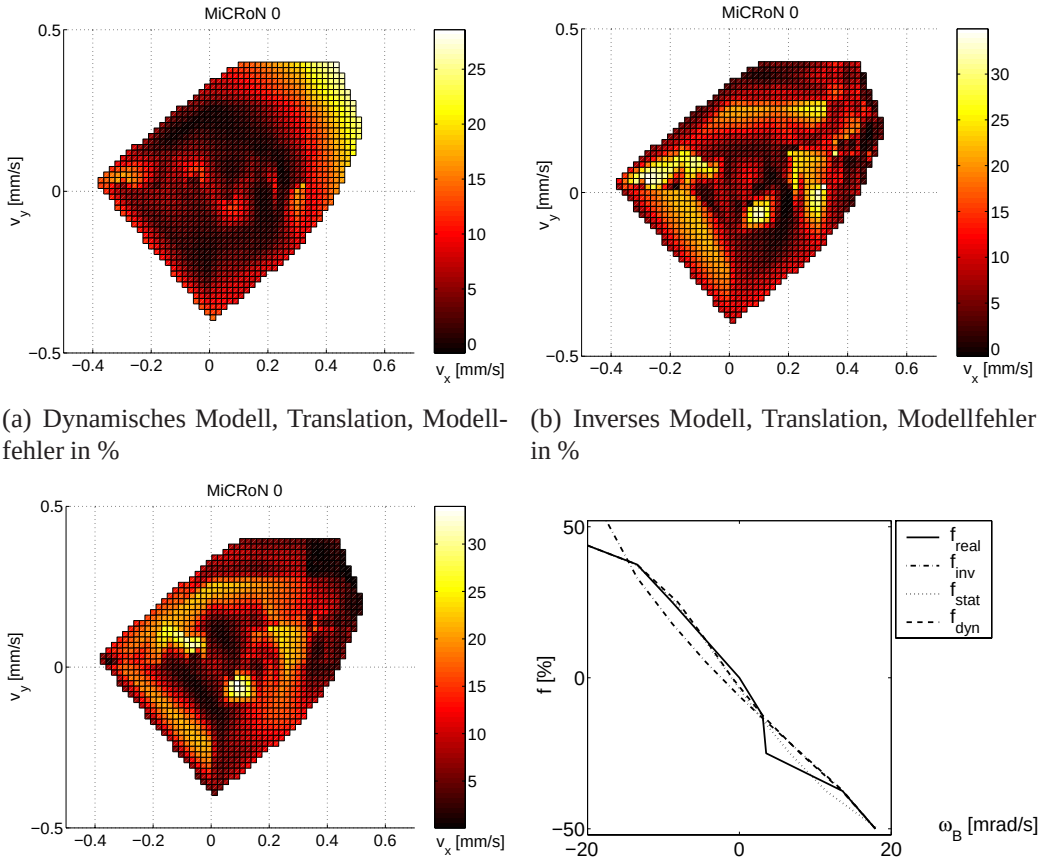
Abbildung 6.15: Ergebnisse für Miniman IV

einer Schicht mit allen Eingänge der Neuronen in der folgenden Schicht verbunden ist. Die Berechnung des Ausgang erfolgt über die Aktivierung a:

$$a = f(\sum_{i} (w_{N,i} p_i) + b) \tag{6.76}$$

Das Ausgangssignal a des Neurons stellt dann das Eingangssignal p von Neuronen der folgenden Schicht dar.

Darauf aufbauend kann ein Netz mit einer beliebigen Anzahl von Schichten gebildet werden, die ihrerseits eine beliebige Anzahl von Neuronen besitzen. Dies gilt uneingeschränkt für sogenannte verdeckte Schichten, die also nicht mit der äußeren Welt verbunden sind. Die erste und die letzte Schicht eines neuronalen Netzes haben jedoch die Funktion, Modelleingangsgrößen aufzunehmen bzw. die vom Netz berechneten Modellausgänge abzugeben. Deshalb ist die Anzahl der



(a) Dynamisches Modell, Translation, Modellfehler in %

(b) Inverses Modell, Translation, Modellfehler in %

(c) Stationäres Modell, Translation, Modellfehler in %

(d) Rotation

Abbildung 6.16: Ergebnisse für MiCRoN 0

Neuronen in diesen Schichten durch die Anzahl der Ein- und Ausgänge vorgegeben.

RBF-Netze

Eine besondere Form der *Feedforward*-Netze sind die Radialen Basisfunktionsnetze (RBF). Dies sind Netze, die aus drei Schichten bestehen: Zwischen der Ein- und Ausgabeschicht existiert eine versteckte Schicht mit einer variablen Anzahl an Neuronen. Die Ausgabe eines Neurons in der versteckten Schicht wird durch den Abstand seines Zentrums $\vec{c_k}$, seiner Ausbreitung σ_N und dem Eingabewert beeinflusst. Typischerweise ist die Aktivierungsfunktion die Gauss-Kennlinie

$$a = \exp(-\frac{1}{2\sigma_N^2}\|c_k - x_m\|^2) \tag{6.77}$$

Die Gaussglocke führt dazu, dass Neuronen, deren Zentren nahe der Eingangsgröße liegen, einen sehr viel stärkeren Einfluss haben als Neuronen mit entfernten Zentren. Da die Anzahl der Neuronen im Prinzip beliebig ist, wird bei der Optimierung die Anzahl der Neuronen solange erhöht bis sie die maximale Grenze erreicht hat. Ist diese Grenze gleich oder größer der Anzahl der Eingangsdatensätze ist, kommt es zu einer perfekten Übereinstimmung. Das ist natürlich nicht sinnvoll, da dadurch keine gute Generalisierung erzielt wird. Es ist daher erforderlich, die Anzahl der Neuronen der verborgenen Schicht deutlich kleiner als die Anzahl der Datensätze zu halten.

6.3.2 Ergebnisse

Die eingesetzten Netze bestehen aus jeweils 4 Schichten mit insgesamt 18 Neuronen. Nach Versuchen mit verschiedenen Varianten hat sich der Tangens hyperbolicus (tanhyp) als geeignete Aktivierungsfunktion für die versteckten Schichten herausgestellt. Gegenstand der Optimierung sind sämtliche Gewichte und Bias-Werte.

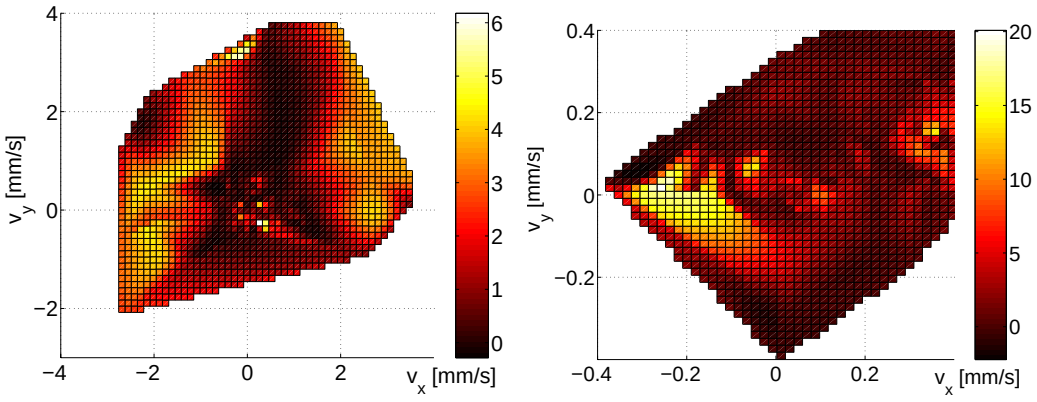
RBF-Netze benötigen mehr Neuronen als klassische Netze, da die Gaussglocke je nach eingestellter Breite zwischen den Zentren stark abfällt. Darüber hinaus ist die Fähigkeit zur Extrapolation bei RBF-Netzen aus dem gleichen Grund nicht sehr ausgeprägt. Die dargestellten Ergebnisse beziehen sich daher auf Netze mit 40 Neuronen.

Die Modellfehler der besten Netze zeigen einen über den Konfigurationsraum sehr uneinheitlich verteilten Fehler. Die Streuung der Fehler ist beim Miniman III und IV größer als beim MiCRoN. Am Beispiel einer reinen Translation wie in Abbildung 6.17 lassen sich Bereiche sehr guter Approximation aber auch Geschwindigkeitskombinationen mit einem großen Fehler erkennen.

Die Ortsabhängigkeit des Fehlers - vor allem durch Kräfte, die durch die Versorgungsleitung auf den Roboter ausgeübt werden - spielt einen nicht zu unterschätzenden Einfluss. Wird auch die Position des Roboters den Traingsdaten hinzugefügt und dem Netz zum Lernen gegegeben, kann der Fehler um einiges reduziert werden (vgl. Abb 6.19).

Beim Miniman IV fällt die Fehlerreduktion deutlich geringer aus als bei den anderen Robotern. Dies hat seinen Grund in der größeren Masse und Kraft des Miniman IV Roboters und damit einem geringeren Einfluss des Kabels. Bei der Berücksichtigung der Ortsabhängigkeit ist aber zu beachten, dass dies mit zwei entscheidenden Nachteilen erkauft wird:

- Ein erheblich höherer Messaufwand ist erforderlich, da ja nun an jedem Ort (bzw. an eng beeinander liegenden Stützpunkten) der gesamte Konfigurationsraum durchgemessen werden muss.



(a) Miniman III (b) MiCRoN 0

Abbildung 6.17: Modellfehler (MAE, in %) inverses Modell bei reiner Translation der neuronalen Netze

- Bei Änderungen des Arbeitsraumes - etwa durch Veränderung der Kabelaufhängungen - sind erneut Messung und Training durchzuführen.

Es ist ferner zu beachten, dass durch das Lernen der Ortsabhängigkeit die durch das Kabel eingebrachten parasitären Bewegungen nie vollständig kompensiert werden können. Die Ergbnisse für das inverse Modell wie auch für das Vorwärtsmodell sind in Tabelle 6.4 zusammengefasst.

Roboter	Invers	Vorwärts
Miniman III	1.4	1.8
Miniman IV	1.0	0.7
MiCRoN 0	4.2	8.9

Tabelle 6.4: Minimaler Fehler (MAE) in % der besten neuronalen Netze

6.4 Evolutionäres Modell

Genetische Programme sind sehr gut in der Lage, Approximationen an unbekannte Funktionen - speziell bei stark nichtlinearen Verläufen - herzustellen. In Kapitel 5 wurde ein Verfahren vorgestellt, das über die herkömmliche Methode hinausgeht. Dieses Verfahren wurde an Testfunktionen gemessen und hat sich dort auch bewährt. Interessant ist es jetzt, zu untersuchen, inwieweit sich dieser Trend auch bei der Modellierung vom Bewegungsverhalten von Mikrorobotern bestätigt. Dafür wurden diese Verfahren mit den Messdaten der Mikroroboter getestet und die

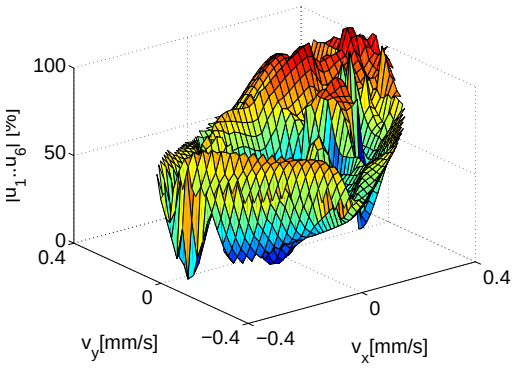


Abbildung 6.18: Ausgabe eines neuronalen Netzes beim MiCRoN 0 für reine Translation

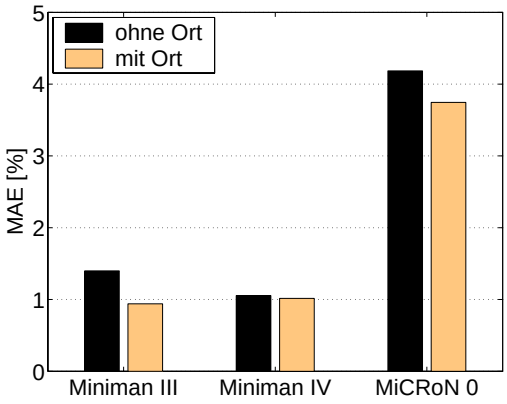


Abbildung 6.19: Reduzierung des Modellfehlers bei neuronalen Netzen durch Einführung der Ortsabhängigkeit

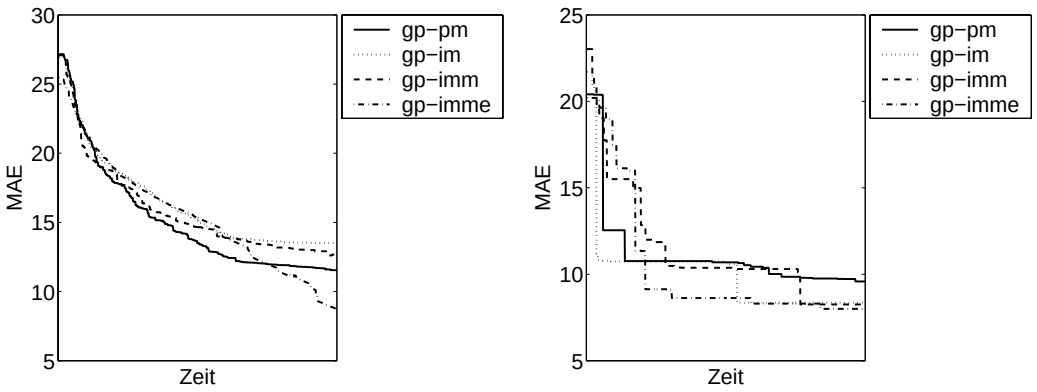
Ergebnisse näher analysiert. In Tabelle 6.5 ist im Vergleich der Modellfehler der vier Varianten dargestellt.

Typ	Roboter	PM	IM	IMM	IMME
GP	Miniman III	9.6	8.4	8.3	8.0
	Miniman IV	8.5	10.3	7.8	8.3
	MiCRoN 0	13.1	15.7	16.0	8.2
GE	Miniman III	10.2	9.8	9.9	9.1
	Miniman IV	9.5	10.5	7.2	9.2
	MiCRoN 0	10.1	12.9	17.2	10.1

Tabelle 6.5: Minimaler Fehler (MAE) in % der genetischen inversen Modelle

Man erkennt, dass sich zwischen den drei klassischen Varianten *PM*, *IM* und *IMM* nur geringe Unterschiede zeigen, der Fehler jedoch beim Insel-Modell mit Meta-Evolution (*IMME*) abnimmt, beim MiCRoN 0 sogar deutlich. Bei den drei erstgenannten hängen die Ergebnisse in beträchtlichem Maße von der Wahl der Steuerparameter ab. Diese Schwierigkeit entfällt bei der eingeführten Methode und mag die besseren Ergebnisse erklären.

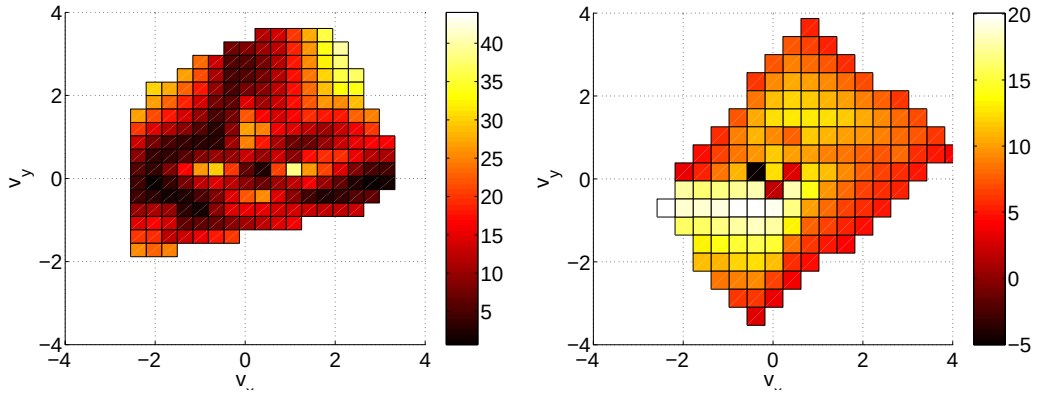
Eine weitere Frage, die sich in diesem Zusammenhang stellt, ist, ob nun die baumbasierte Struktur (GP) oder die grammatikorientierte Variante (GE) besser geeignet ist, Mikroroboter zu modellieren. Die Ergebnisse (vgl. Tabelle 6.5) geben hier jedoch keine klare Aussage. Während bei den Miniman-Roboter GP überlegen ist, ergibt sich für GE ein Vorteil beim MiCRoN. Ob dies nun systematische Gründe hat oder zufällig durch den Verlauf der Evolution zustande gekommen ist, kann nur spekuliert werden. Es fällt jedoch auf, dass schon im Kapitel 5 die



(a) Mittlerer Fehler (b) Minimaler Fehler

Abbildung 6.20: Verlauf des Modellfehlers des Miniman III (MAE, in %) bei genetisch erzeugten Programmen

einfachere Funktion (Binomial3-Problem) besser von GP und die komplexere besser von GE approximiert wurde. Eine genauere Untersuchung dieses Phänomens würde jedoch den Rahmen dieser Arbeit sprengen.



(a) Miniman III (b) Miniman IV

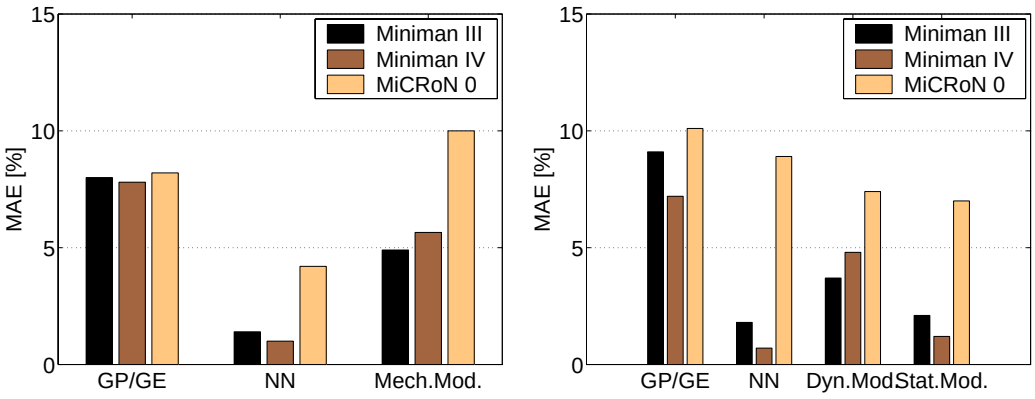
Abbildung 6.21: Modellfehler (MAE, in %) bei reiner Translation der genetisch erzeugten Modelle

Bei einer Analyse des Fehlerverlaufs (vgl. Abb. 6.20(a)) zeigt sich die Überlegenheit von *IMME*: Während bei anderen Verfahren der evolutionäre Verlauf etwa bei einem mittleren Fehler von 10-12% stoppt, läuft bei *IMME* die Verbesserung weiter. Grund dafür ist die höhere Diversität der Population und die fortwährende Anpassung der von Evolutionsparametern. Bei Betrachtung des minimalen Feh-

lers (Abb. 6.20(b)) - d.h. die beste aller gefundenen Lösungen - wird deutlich, dass u.U. auch sehr schnell überlegene Lösungen gefunden werden können.

6.5 Bewertung

Es wurden in den vorangegangen Abschnitten eine Reihe unterschiedlicher Verfahren vorgestellt, mit denen ein Steuerungsalgorithmus für Mikroroboter realisiert werden kann. Die Ergebnisse für jedes einzelne Modell sind in den entsprechenden Abschnitten dargestellt und erläutert worden. Was im Zusammenhang mit den Einsatzmöglichkeiten von evolutionären Strategien interessiert ist nun der Vergleich der Modelle untereinander. Ein Blick auf Abbildung 6.22(a) verrät, das die evolutionären Modelle - was den Modellfehler betrifft - hinter den besten neuronalen Netzen liegen, sich jedoch durchaus mit den kinematischen Modellen messen können. Insbesondere bei stark nichtlinearen Verläufen wie beim MiCRoN 0 ist das evolutionäre Verfahren den anderen Methoden teilweise sogar überlegen.



(a) Inverses Modell (b) Vorwärtsmodell

Abbildung 6.22: Modellfehler der untersuchten Modelle

Die ortsabhängigen Ergebnisse liegen im Allgemeinen deutlich besser als die ohne Berücksichtigung des Ortes. Dies wird jedoch mit zwei großen Nachteilen erkauft, nämlich höherem Messaufwand und der Notwendigkeit, die Arbeitsumgebung (insbesondere die Kabelaufhängung) konstant zu halten.

Kapitel 7

Regelung

Ziel der Regelung ist es, aufbauend auf den Information der Positionsmessung (vgl. Abschnitt Abschnitt 3.2) die Geschwindigkeiten der Roboter so zu berechnen, dass eine präzise Navigation möglich wird. Zu beachten ist hierbei, dass ein Regler auf einem spezifischen Steuerungsmodell (vgl. Kapitel 6) basiert, d.h. die Ergebnisse einer Topologie- und Parameteroptimierung immer in Bezug zu dieser Steuerung zu sehen sind. Aus Sicht des Reglers besteht die Regelstrecke (G_S) aus Steuerungsmodell (G_T), Hardware-Controller, dem Roboter selbst (G_{Rob}) sowie dem Messsystem (G_M, vgl. Abb 7.1).

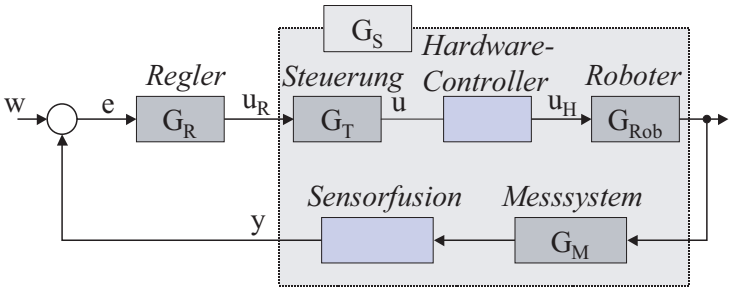


Abbildung 7.1: Aufbau der Regelstrecke

Regelgröße ist die Geschwindigkeit. Dabei sollte die Roboterbewegung so koordiniert werden, dass sowohl im Makro- als auch im Mikrobereich Bahnabweichungen im tolerablen Bereich liegen. Die genaue Toleranzgrenze hängt natürlich von der speziellen Aufgabe ab, man kann aber davon ausgehen, dass Abweichungen von der Sollposition im Bereich bis zu 30 μm akzeptabel sind.

7.1 Ausgangspunkt

7.1.1 Streckenanalyse

Die Basis für die klassische Reglerauslegung ist eine Analyse der Regelstrecke. Allgemein besteht die Aufgabe darin, Gleichungen zu finden, die die Eingänge des Systems (hier: die Reglerausgänge $u_R(t)$) auf die Ausgänge (hier: die Geschwindigkeiten des Roboters $y(t)$) abbilden:

$$\begin{aligned} x_z(t+T) &= f\left[x_z(t), x_z(t-T), .., x_z(t-n_1T), u_R(t), u_R(t-T), .., u_R(t-n_2T)\right] \\ y(t) &= g\left[x_z(t), x_z(t-T), .., x_z(t-n_3T)\right] \end{aligned} \tag{7.1}$$

Die diskrete Zeitfolge $x_z(t)$ wird auch als (innerer) Zustand des Systems bezeichnet. Es sind nun drei Szenarien denkbar:

1. f und g sind bekannt und der Zustand $x_z(t)$ ist mess- oder beobachtbar. Dies entspricht dem Einsatz von analytisch hergeleiteten Modellen.

2. f und g sind unbekannt aber der Zustand $x_z(t)$ ist mess- oder beobachtbar. In diesem Fall lassen sich Zustandsmodelle anwenden.

3. f und g sind ebenso unbekannt wie der Zustand $x_z(t)$. Man spricht in diesem Zusammenhang von *Blackbox*-Modellen. Hierbei ist noch noch zu unterscheiden, ob es sich um lineare oder nichtlineare Funktionen handelt.

Aufbauend auf dieser sehr allgemeinen Systembeschreibung können eine Reihe von Modellen abgeleitet werden. Ziel ist es, einen Vergleich der Modelle untereinander durchzuführen und geeignete wie auch weniger geeignete Ansätze zu identifizieren. Darauf soll im Folgenden näher eingegangen werden.

Analytisches Modell

In Kapitel 6 wurden zahlreiche (Vorwärts-)Modelle zur Beschreibung des Roboterverhaltens hergeleitet und deren Parameter anhand von Messdaten angepasst. Ferner wurden Steueralgorithmen gebildet, die auf dieses Bewegungsverhalten einwirken. Damit ist im Grunde das komplette Verhalten der Regelstrecke bekannt.

Um die Komplexität des Vorgehens nicht zu groß werden zu lassen, wird hier ausschließlich das stationäre Modell der Strecke (Abschnitt 6.2.3) zusammen mit dem erfolgreichsten Steuermodell benutzt. Als am günstigsten hat sich für die Steuerung ein neuronales Netz herausgestellt (Abschnitt 6.3). Die Kombination aus diesen beiden Teilmodellen wird im Folgenden als Analytisches Modell (*AM*) bezeichnet und im weiteren Verlauf dieses Kapitels näher untersucht.

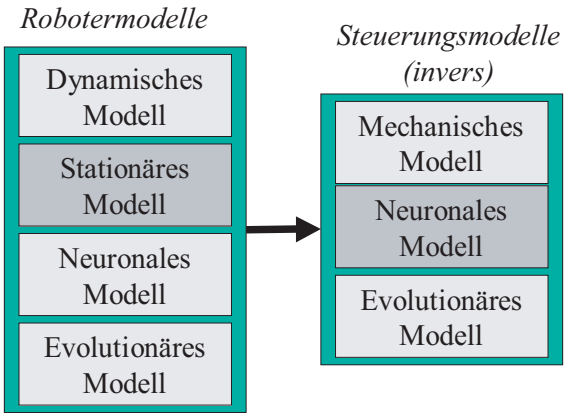


Abbildung 7.2: Kombination aus Robotermodell und Steuerungsmodell zum Regelstreckenmodell (Analytisches Modell)

Zustandsmodell

Zustands(größen)modelle (*ZM*) sind verbreitete Formulierungen zur Beschreibung von dynamischen Systemen. Sie werden durch Zustands- und Ausgangsgleichung beschrieben:

$$\begin{aligned} \vec{x}_z(t+T) &= \mathbf{A}\vec{x}_z(t) + \mathbf{B}\vec{u}_R(t) \\ \vec{y}(t) &= \mathbf{C}\vec{x}_z(t) + \mathbf{D}\vec{u}_R(t) \end{aligned} \tag{7.2}$$

Dabei heißen **A** Systemmatrix ($\mathbf{A} \in \mathbb{R}^{n_z \times n_z}$), **B** Eingangsmatrix ($\mathbf{B} \in \mathbb{R}^{n_z \times n_r}$), **C** Ausgangsmatrix ($\mathbf{C} \in \mathbb{R}^{n_f \times n_z}$), **D** Durchgangsmatrix ($\mathbf{D} \in \mathbb{R}^{n_f \times n_r}$), n_r Anzahl der Reglerausgangsgrößen, n_z Anzahl der Zustandsgrößen und n_f Anzahl der Freiheitsgrade des Roboters.

Bei Zustandsmodellen werden also zusätzlich zu den Ein- und Ausgangsgrößen des System noch zusätzliche Größen eingeführt. Diese Zustandsgrößen berücksichtigen die Dynamik des Systems und können physikalischen Größen entsprechen.

Das Modell kann sehr bequem mit MATLAB identifiziert werden, so dass hier nicht näher auf die Bestimmung der Parametermatrizen eingegangen werden muss. Zusätzlich zur Bestimmung der Matrizen ist die Anzahl der Zustandsgrößen, d.h. die Dimension des Vektors x_z zu ermitteln. Sinnvolle Ergebnisse lieferte ein Modell mit $n_z \geqslant 4$ Zustandsgrößen.

Lineare *Blackbox*-Modelle

Für die Analyse einer unbekannten Regelstrecke werden häufig sogenannte *Blackbox*-Modelle eingesetzt. Voraussetzung dafür ist, dass Messdaten in ausreichender

Modell	Gleichung
ARX	$A_R(q)y(t) = B_R(q)u(t-nk) + e(t)$
ARMAX	$A_R(q)y(t) = B_R(q)u(t-nk) + C_R(q)e(t)$
Output-Error (OE)	$y(t) = \frac{B_R(q)}{F_R(q)}u(t-nk) + e(t)$
Box-Jenkins (BJ)	$y(t) = \frac{B_R(q)}{F_R(q)}u(t-nk) + \frac{C_R(q)}{D_R(q)}e(t)$

Tabelle 7.1: Blackbox-Modelle zur Streckenanalyse

Menge zur Verfügung stehen. Ein sehr häufig eingesetztes Modell ist das ARX-Modell[1]:

$$y(t) + a_1y(t-1) + \ldots + a_{na}y(t-n_a) = b_1u(t-n_k) + b_2u(t-n_k-1)+ \ldots + b_{nb}u(t-n_k-n_b+1) + e(t) \tag{7.3}$$

Oder in symbolischer Schreibweise mit dem Verschiebungsoperator q:

$$A_R(q)y(t) = B_R(q)u(t-n_k) + e(t) \tag{7.4}$$

Dabei charakterisiert $e(t)$ das Rauschen, im mehrdimensionalen Fall die Kovarianzmatrix. Dies ist jedoch nur eine spezielle Ausprägung der allgemeinen Formulierung:

$$A_R(q)y(t) = \frac{B_R(q)}{F_R(q)}u(t-n_k) + \frac{C_R(q)}{D_R(q)}e(t) \tag{7.5}$$

Dabei sind A_R, B_R, C_R, D_R und F_R Koeffizientenmatrizen, die mit Hilfe von Optimierverfahren dann an die Messdaten angepasst werden können. Dadurch ist eine rekursive Berechnung des dynamischen Streckenverhaltens möglich. Es gibt nun eine Reihe von Modellen (vgl. Tabelle 7.1), die auf dieser Struktur basieren. Neben den eigentlichen Modellparametern $A_R..F_R$ ist auch der Zeithorizont, also n_a, n_b und n_k, festzulegen. Es gibt dafür kein etabliertes Verfahren, so dass diese Größen in der Regel empirisch bestimmt werden.

Alternativ zur polynomialen Darstellung gibt es auch die in der klassischen Regelungstechnik gebräuchliche Formulierung der Übertragungsfunktionen, die auch zur Klasse der *Blackbox*-Modelle gehört:

$$y(t) = Gu(t) + He(t) \tag{7.6}$$

Durch Umformen von Gleichung 7.5 kann leicht eine Übertragungsfunktion gemäß 7.6 gewonnen werden.

[1] ARX:**A**uto **r**egressive with e**x**ogenous input

Nichtlineare *Blackbox*-Modelle

Für die Nachbildung des invertierten Roboterverhaltens allein wurden in Abschnitt 6 schon erfolgreich neuronale Netze und genetisch erzeugte Modelle benutzt. Auf dieser Basis wurde eine Steuerung entworfen. Die gleichen Verfahren können auch zur Streckenanalyse benutzt werden. Dabei kann es sich auch um ein rekursives Modell handeln, wobei neben den Momentanwerten auch die Streckendynamik berücksichtigt wird. Es hat sich in diesem Zusammenhang das sogenannte *NARMA*-Modell[2] etabliert:

$$y(t+T) = f\left(y(t), y(t-T), ..., y(t-n_y T), u(t), u(t-T), ..., u(t-n_u T)\right) \tag{7.7}$$

Die Funktion f wird üblicherweise über ein neuronales Netz approximiert. Dabei wird die Dynamik der Regelgröße y über n_y Zeitschritte berücksichtigt, die des Stellbefehls u über n_u Zeitschritte.

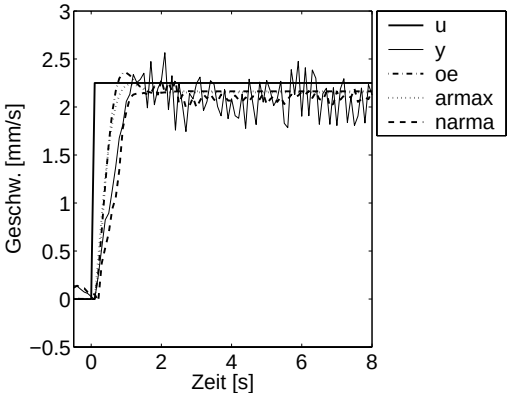
Für die genetische Programmierung stellt diese Optimieraufgabe ebenfalls kein Problem dar, analog zum *NARMA*-Modell werden - nach Festlegung des Zeithorizonts - Stellbefehle und Geschwindigkeiten als Eingänge betrachtet und ansonsten die bereits in Kapitel 5 festgelegten mathematischen Operatoren benutzt.

Ergebnisse

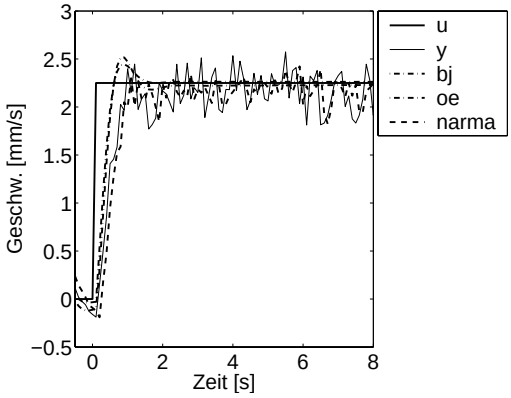
Um die Parameter der Modelle anzupassen und die Qualität der Modellansätze zu testen, wurden Messungen durchgeführt. Diesmal wurde bei der Ansteuerung der Roboter der erfolgreichste Steueralgorithmus aus Kapitel 6 miteinbezogen. Um eine aussagekräftige Datenmenge zusammenzustellen, wurde der gesamte Konfigurationsraum diskretisiert. Konkret bedeutet das, dass über den gesamten Konfigurationsraum verteilte Stützpunkte einzeln gemessen wurden. Um den Umfang der Datenerhebung zu begrenzen, wurde eine Beschränkung auf nur 5 gleichmäßig verteilte Stützpunkte je Freiheitsgrad, also für jede Dimension aus dem Bereich $\{-1; -0.5; 0; 0.5; 1\} \cdot v_{max}$ vorgenommen. Um statistische Probleme zu vermeiden, wurde erneut eine Vergleichsprüfung (*Cross-validation*) mit einer separaten Testmenge durchgeführt. Die Anzahl der Stützpunkte hierbei war jedoch geringer und beschränkte sich auf $\{-0.75; 0; 0.75\} \cdot v_{max}$ für jede Dimension. Die folgenden Fehlerangaben beziehen sich stets auf die Testmenge.

Die Vorgaben wurden dem offenen Regelkreis als Sprünge von einer Dauer von 10 s aufgegeben und die Sprungantwort aufgezeichnet. Die Messung der Sprungantworten wurde wegen der Limitierung des Messsystems in einem Takt von ca. 150-250 ms durchgeführt.

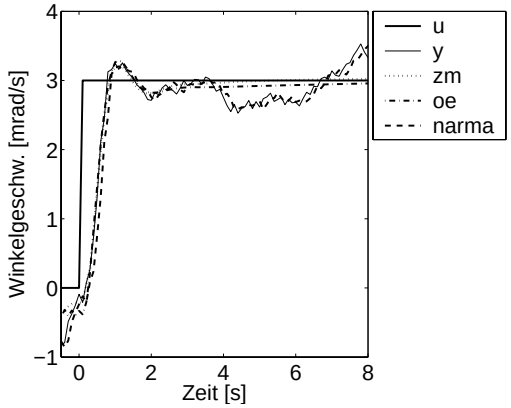
[2] *NARMA*:**N**onlinear **A**uto**r**egressive **M**oving **A**verage



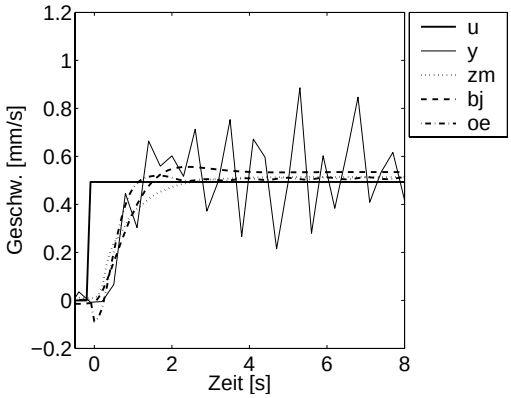
(a) Miniman III, x-Translation



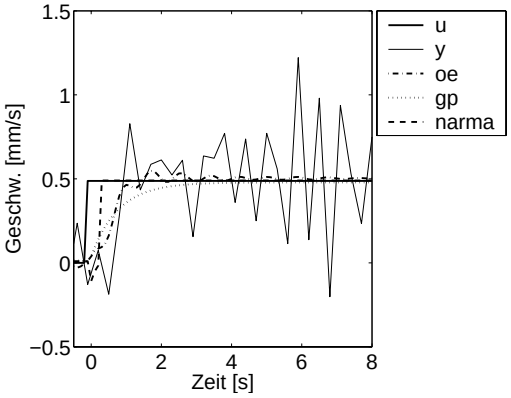
(b) Miniman III, y-Translation



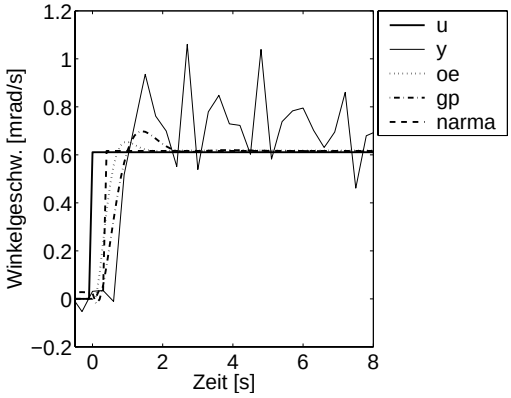
(c) Miniman IV, Rotation



(d) MiCRoN 0, x-Translation



(e) MiCRoN 0, y-Translation



(f) MiCRoN 0, Rotation

Abbildung 7.3: Approximation der Regelstrecke durch Blackbox-Modelle: Beispiele für Sprungantworten

Roboter	AM	ZM	ARX	BJ	OE	ARMAX	NARMA	GP
Miniman III	4.0	0.9	1.0	1.2	0.9	1.0	0.6	1.4
Miniman IV	4.1	1.0	0.9	1.3	0.6	0.8	0.5	1.3
MiCRoN 0	23.8	7.0	7.0	6.9	6.7	7.0	6.9	7.7

Tabelle 7.2: Mittlerer absoluter Fehler der Streckenmodelle (in %)

Eine Analyse der Ergebnisse offenbart, dass es sich bei dem System um ein im Prinzip schwingfähiges Übertragungsverhalten handelt, dass jedoch von starkem Rauschen so überlagert wird, dass eine eindeutige Auswertung schwer fällt (vgl. Abb 7.3). Aufgrund der Steigung der Sprungantwort kann davon ausgegangen werden, dass die Strecke eine Verzögerung erster Ordnung besitzt und durch ein PT_1-Glied angenähert werden kann. Lediglich bei der Rotation zeigt sich eine stärker verzögerte Anfangssteigung (PT_2- Verhalten).

Beim MiCRoN zeigt sich ein wesentlich stärkeres Rauschen was zum großen Teil - aber nicht ausschliesslich - den größeren Modellfehler erklärt. Natürlich schlagen hier auch Messfehler prozentual stärker durch, weil die Geschwindigkeiten insgesamt langsamer sind. Ein Problem beim MiCRoN ist insbesondere der hohe Fehler bei der Rotation. Die Verzögerung bei der Sprungantwort ist ebenfalls beim MiCRoN am grössten, vermutlich durch eine Totzeit, die durch die IR-Kommunikation verursacht wird.

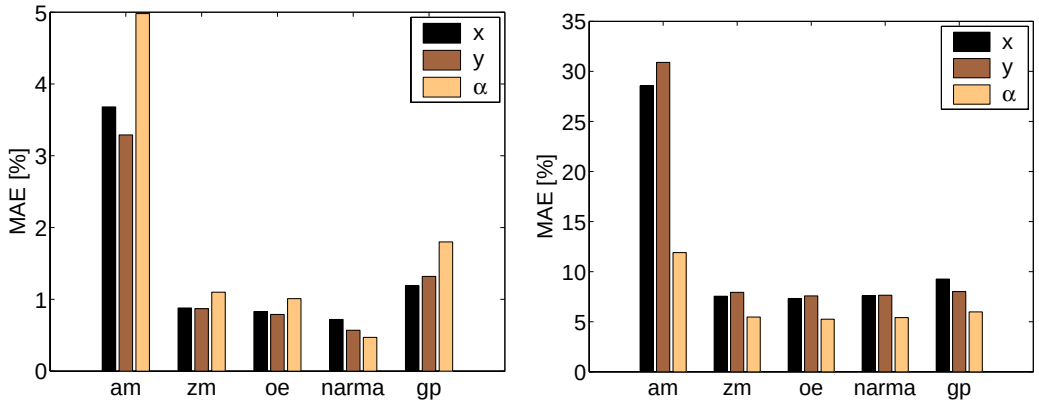
Das analytische Modell (*AM*) schneidet bei allen Robotern am schlechtesten ab, insbesondere die Nachbildung des MiCRoN 0 Verhaltens ist praktisch unbrauchbar. Dies mag seinen Grund darin haben, dass das mechanische Modell aus Kapitel 6.2 eher auf den Miniman zugeschnitten ist. Am besten gelingt die Appoximation mit dem *NARMA*-Modell, hier ist der Modellfehler insgesamt am geringsten. Der andere nichtlineare Ansatz - die genetische Programmierung - hat sich durchaus bewährt und ist nur etwas schlechter als die linearen *Blackbox*-Modelle und die Zustandsmodelle.

Da das auf neuronalen Netzen basierende Modell (*NARMA*) sich bei allen Robotern als das beste gezeigt hat, soll im weiteren Verlauf der Untersuchungen dieses Modell als Basis für den Reglerentwurf herangezogen werden.

7.1.2 Reglertest

Das Bilden eines Streckenmodells erlaubt es nun, vielfältige Simulationen mit beliebigen Reglern durchzuführen. Natürlich müssen aber die entwickelten Regler anschließend auch mit der tatsächlichen Roboter-Hardware getestet werden. Insbesondere beim MiCRoN 0 musste festgestellt werden, dass die Streckenmodellbildung nur eine ungefähre Wiedergabe der Realität darstellt.

Für die Tests mit Robotern wurden zwei Arten von Abläufen definiert:



(a) Miniman III (b) MiCRoN 0

Abbildung 7.4: Mittlerer absoluter Fehler (MAE) des Streckenmodells bezogen auf Freiheitsgrad und Modell

1. Der gesamte Konfigurationsraum des Roboters wird systematisch durchfahren. Dabei entspricht die Aufteilung der der Validierung des Streckenmodells.

2. Der besseren Anschaulichkeit wegen wird darüber hinaus eine Testtrajektorie durchfahren. Die Auswahl der Bahn erfolgt so, dass Kombinationen aus Bewegungen in allen Freiheitsgraden enthalten sind.

Diese beiden Varianten werden für jeden Reglertyp mindestens einmal durchgeführt und zwar mit dem Miniman III mit Pisa-Greifer, Miniman IV und MiCRoN 0[3]. Damit das Einregelverhalten beobachtet werden kann, ist nach jedem Sollwertsprung der Sollwert für 10 s konstant zu halten. Danach wird, um zu einem definierten Anfangszustand zurück zu kommen, der Sollwert auf Null gesetzt, bevor der nächste Sprung erfolgt.

Für die Bewertung der Reglergüte kommen zwei Arten von Fehlern in Betracht:

- Der Geschwindigkeitsfehler $e_{m,v}$ gibt die mittlere absolute Abweichung der Sollgeschwindigkeit von der Istgeschwindigkeit wieder:

$$e_{m,v}(t) = |w(t) - y(t)| \tag{7.8}$$

[3]Bei der Darstellung der Ergebnisse wurde aus Platzgründen häufig nur der Miniman III und der MiCRoN 0 aufgeführt, da sich die Dynamik und Regelgüte des Miniman IV und III meist ähnelten

- Der Positionsfehler $e_{m,p}$ gibt die Ortsabweichung vom Zielpunkt an, bezogen auf die Zeit, die vorgesehen war, dieses Ziel zu erreichen:

$$e_{m,p} = w \sum_{i=1}^{n_y} (t_i - t_{i-1}) - \sum_{i=1}^{n_y} y_i \cdot (t_i - t_{i-1}) \tag{7.9}$$

  y_i sind die gemessenen Geschwindigkeiten zum Zeitpunkt t_i. Dieser Fehler stellt einen kumulierten Geschwindigkeitsfehler, also eine Bahnabweichung, dar.

Es ist zu erwarten, dass der Geschwindigkeitsfehler höher liegt als der Positionsfehler, da sich beim letzteren positive und negative Abweichungen kompensieren können. Da sich die Schrittweite bzw. maximale Geschwindigkeit der Roboter doch erheblich unterscheidet, hat es sich als sinnvoll herausgestellt, vor allem den relativen Fehler, d.h. bezogen auf die jeweilige maximale Geschwindigkeit, zu betrachten. Dies erleichtert auch den Vergleich von translatorischen und rotatorischen Regelfehlern.

7.2 Linearer Reglerentwurf

Voraussetzung für den Entwurf eines Reglers mit Methoden der klassischen Regelungstechnik ist ein näherungsweise lineares bzw. linearisiertes Verhalten der Strecke. Bei nichtlinearen Strecken muss die Berechnung des Führungs- und Störverhaltens getrennt erfolgen, da das Superpositionsprinzip theoretisch nicht gilt. In der Praxis werden dennoch häufig lineare Auslegungsprinzipien für nichtlineare Regelstrecken benutzt.

Regelziele sind neben der Stabilität vor allem das Vermeiden von bleibenden Regelabweichungen und das Verhindern von parasitären Bewegungen. Es handelt sich bei der Regelstrecke um ein MIMO-System[4], da mehrere Eingangs- und mehrere Ausgangsgrößen das System beeinflussen. Bei einem Mehrgrößensystem kann der Regler nicht in einem Entwurfsschritt bestimmt werden. Das Übertragungsverhalten des Gesamtsystems berechnet sich bei einem Dreigrößensystem (drei Eingänge, drei Ausgänge) folgendermaßen:

$$Y(s) = \begin{pmatrix} G_{s,x,x}G_{R,x} & G_{s,y,x}G_{R,y} & G_{s,\alpha,x}G_{R,\alpha} \\ G_{s,x,y}G_{R,x} & G_{s,y,y}G_{R,y} & G_{s,\alpha,y}G_{R,\alpha} \\ G_{s,x,\alpha}G_{R,x} & G_{s,y,\alpha}G_{R,y} & G_{s,\alpha,\alpha}G_{R,\alpha} \end{pmatrix} U(s) \tag{7.10}$$

Es wird klar, dass jede der drei Reglerübertragungsfunktion $\mathbf{G_R}$ auf jeden der Ausgänge gleichzeitig wirkt, wobei allerdings die Einflüsse durch die Stärke der Streckenübertragungsfunktion $\mathbf{G_S}$ gemindert oder verstärkt werden kann. Diese Wirkung ist aber ungewollt und stellt für den jeweiligen Ausgang eher eine Störgröße

[4] MIMO: **M**ultiple **I**nput, **M**ultiple **O**utput

als einen Stellbefehl dar. Deswegen soll nun im Folgenden untersucht werden, ob sich das Systemverhalten mit mehreren unabhängigen Reglern gut beeinflussen lässt bzw. ob noch weitergehende Maßnahme (etwa eine Störgrößenkompensation) ergriffen werden müssen.

7.2.1 PID-Regler

Ein einfacher, in der technischen Praxis aber sehr häufig eingesetzter Regler ist der PID-Regler. In seiner rekursiven Form lautet der Algorithmus für digitale Regelungen:

$$\begin{aligned} y_k =& y_{k-1} + b_0 e_k + b_1 e_{k-1} + b_2 e_{k-2} \\ & \text{mit} \\ & b_0 = K_R \left(1 + \frac{T_V}{T} + \frac{T}{2T_N}\right) \\ & b_1 = -K_R \left(1 - \frac{T}{2T_N} + 2\frac{T_V}{T}\right) \\ & b_2 = K_R \frac{T_V}{T} \end{aligned} \tag{7.11}$$

wobei T die Taktrate (Abtastzeit), K_R die Reglerverstärkung, T_N die Nachstellzeit und T_V die Vorhaltzeit bezeichnet. Der PID-Regelalgorithmus gilt eigentlich nur für lineare Regelstrecken, wird aber wegen seiner Robustheit und Vielseitigkeit auch für nichtlineare bzw. fastlineare Regelstrecken erfolgreich eingesetzt.

Zu beachten ist - da es sich um eine Geschwindigkeitsregelung handelt - dass der Arbeitspunkt u_0 zum Reglerausgang addiert wird. Da die unterlagerte Steuerung bereits Geschwindigkeiten als Eingangsgröße erwartet und in Frequenzen und Spannungswerte umsetzt, kann als Arbeitspunkt der Geschwindigkeitssollwert w angenommen werden (vgl. Abb. 7.5).

Für die Bestimmung der unbekannten Reglerparameter werden üblicherweise Einstellregeln eingesetzt, die von experimentell ermittelten Kenngrößen des offenen Regelkreises ausgehen. Die meisten dieser Regeln gelten für analoge Regelkreise, z.B. die Ziegler-Nichols Einstellregeln oder die Regeln nach Chien, Hrones und Reswick. Diese Regeln können bei digitalen Regelkreisen eingesetzt werden, wenn die Abtastzeit so gering ist, dass der Regelkreis quasikontinuierlich arbeitet. Es muss deshalb zunächst die notwendige Abtastzeit berechnet werden und überprüft werden, ob diese vom System überhaupt geleistet werden kann. Die Berechnung geschieht mit Hilfe von Kenngrößen der Sprungantwort (für das genaue Vorgehen vgl. z.B. [52]).

Die gemessenen Kenngrößen sind in Tabelle 7.3 aufgelistet. Unter Anwendung der Regeln zur Bestimmung einer geeigneten Abtastzeit ergibt sich eine

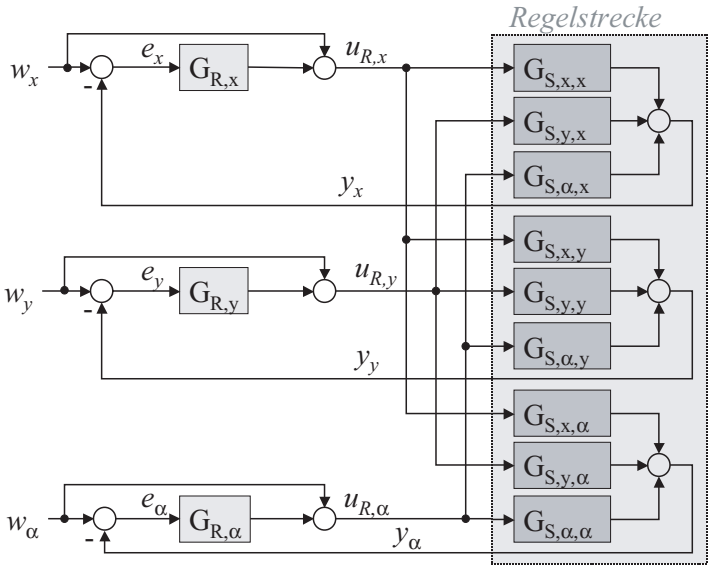
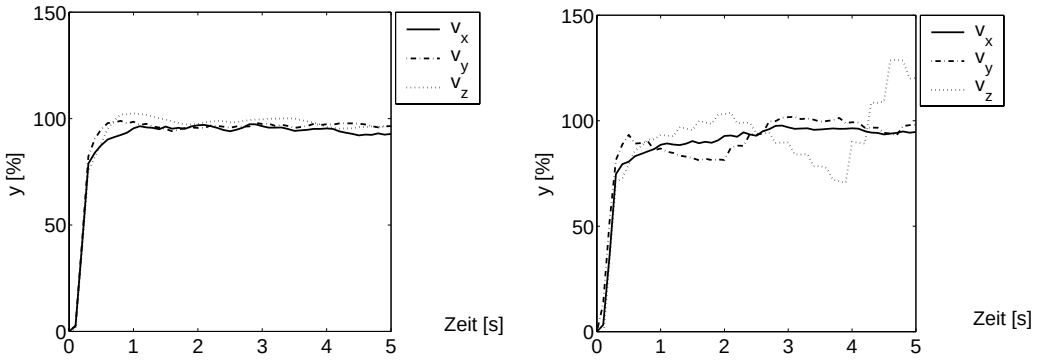


Abbildung 7.5: Drei SISO-Regler für ein System mit drei Ein- und drei Ausgangsgrößen

maximale Taktrate von ca. 50 ms für Miniman III und IV, sowie von 30 ms für MiCRoN 0. Diese Zeitraten sind aus technischen Gründen nicht zu realisieren, einerseits begründet im Messsystem, zum anderen aber auch in hardwarespezifischen Beschränkungen, insbesondere im Zusammenhang mit der Signalgenerierung und -übertragung. Die unter Berücksichtigung aller Beschränkung günstigstenfalls zu realisierende Zeitrate liegt bei den Miniman-Robotern bei ca. 250 ms, beim MiCRoN 0 etwa bei 150 ms. Klassische Einstellregeln für analoge Regelkreise können also nicht angewendet werden. Es gibt jedoch speziell für digitale Regelungen entwickelte Einstellregeln, die auch noch bei höheren Taktraten gültig sind (für Details zur Vorgehensweise sei auf [52] verwiesen). Geeignet ist die Regel nach Takahashi, die für Abtastraten von bis zu $T \leqslant 2T_u$ gültig ist (siehe Tabelle 7.4). Dies ist zumindest näherungsweise von der Regelstrecke zu leisten. Die Regel bezieht sich auf einen PID-Regler der additiven Form.

Der berechnete PID-Regler ist jedoch für einige Freiheitsgrade theoretisch nicht stabil, wie ein Blick auf Abbildung 7.7(a) verrät. Es gibt einen Pol (Nullstelle der charakteristischen Gleichung) mit einem positiven Realteil. Es gelingt - unter Beibehaltung der Reglerstruktur - nur mit einer deutlichen Erhöhung des Differentialanteils, den Pol in die linke Hälfte zu verschieben. Wünschenswert ist ein deutlicher Abstand zur imaginären Achse, da damit gewährleistet ist, dass auch bei leichten Modellabweichungen (etwa durch Veränderungen von Materialeigenschaften) noch eine stabile Regelung möglich ist. Leider ist der so berechnete



(a) Miniman III (b) MiCRoN 0

Abbildung 7.6: (Gemittelte und normierte) Sprungantworten

Roboter	Stellgröße	Verzugszeit T_u [s]	Ausgleichszeit T_g [s]	Einstellzeit T_{95} [s]
Miniman III	u_x	0.095	0.262	0.982
	u_y	0.096	0.251	0.505
	u_θ	0.099	0.270	0.601
Miniman IV	u_x	0.094	0.269	0.921
	u_y	0.101	0.267	0.764
	u_θ	0.092	0.256	0.617
MiCRoN 0	u_x	0.099	0.280	2.603
	u_y	0.057	0.294	2.621
	u_θ	0.111	0.270	1.257

Tabelle 7.3: Kenngrößen zur Bestimmung der Abtastzeit

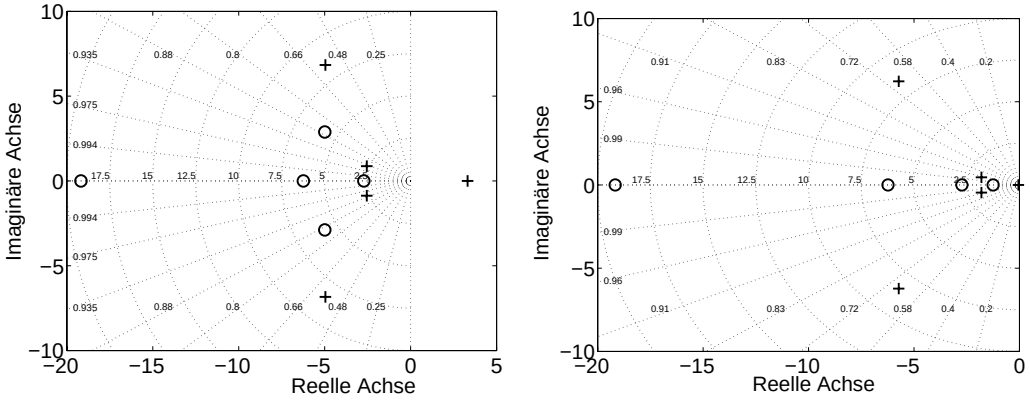
Reglergröße	Bestimmungsgleichung nach Takahashi
K_R	$\frac{1.2T_g}{K_s(T_u+T)}$
T_N	$\frac{2\left(T_u+\frac{T}{2}\right)^2}{T_u+T}$
T_V	$0.5\,(T_u+T)$

Tabelle 7.4: Einstellregeln für PID-Regler nach Takahashi

Regler wegen des höheren T_V sehr empfindlich für Rauschen, so dass er für den praktischen Betrieb ungeeignet war. Als Alternative ist ein Regler vom PPT_1-Typ denkbar. Seine Übertragungsfunktion lautet im Bildbereich:

$$G_R(s) = K_p \left(\frac{1 + T_V s}{1 + T_1 s} \right) \tag{7.12}$$

Abbildung 7.7(b) zeigt exemplarisch Pol- und Nulstellen eines stabilen PPT_1-Reglers für die x-Translation des MiCRoN 0.

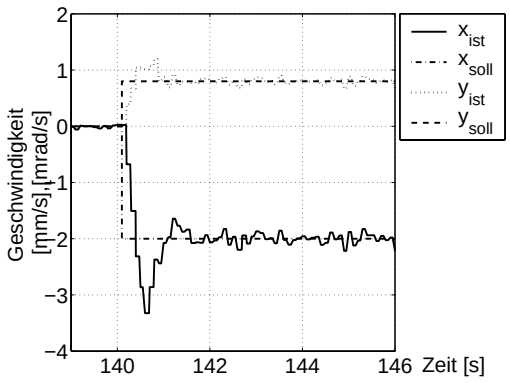


(a) PID-Regler mit Parametern nach Takahashi (b) PPT1-Regler mit K_p=1.0, T_1=1.0 und T_V=0.4

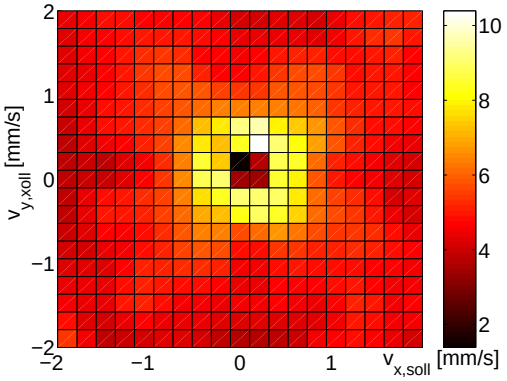
Abbildung 7.7: Pol-Nulstellen-Plan zweier Regelkreise des MiCRoN 0 für x-Translation

Die auf Basis der Regeln aus Tabelle 7.4 erzielten Regelparameter wurden mit den Robotern über den gesamten Konfigurationsraum eingesetzt. Als Vergleich wurde ein ein PPT1-Regler, dessen Parameter unter Berücksichtigung von Stabilitätsbetrachtungen empirisch angepasst wurden, dem gleichen Test unterzogen.

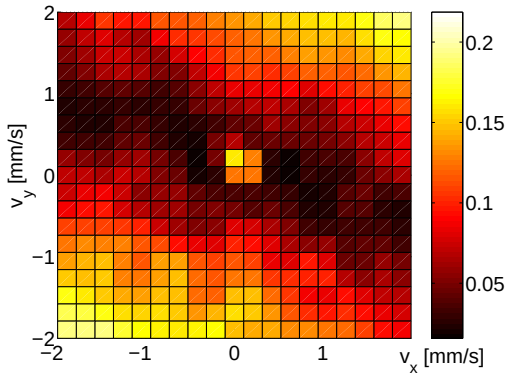
Die Ergebnisse zeigen ein gutes Regelverhalten sowohl in Bezug auf die Führungsgröße als auch das Ausregeln von Störungen. Der PID-Regler erwies sich - trotz der ungewissen Stabilität - als geeignet, das Regelziel zu erreichen (vgl. Abb. 7.8(b) und 7.9(b)). Lediglich bei einem Sollwertsprung sind naturgemäß anfangs größere Abweichungen zu verzeichnen (vgl. Abb. 7.8(a) und 7.9(a)), die jedoch schnell ausgeregelt werden. Auffällig ist hingegen die Abweichung bei der Orientierung: Durch parasitäre Rotation ergibt sich schnell ein Orientierungsfehler, der auch nicht ausgeregelt wird. Auch bei einer erwünschten Rotation ergibt sich eine bleibende Regelabweichung. Dies betrifft insbesondere den MiCRoN 0, bei dem auch starke Schwingungen ein Ausregeln der Orientierung fast unmöglich machen.



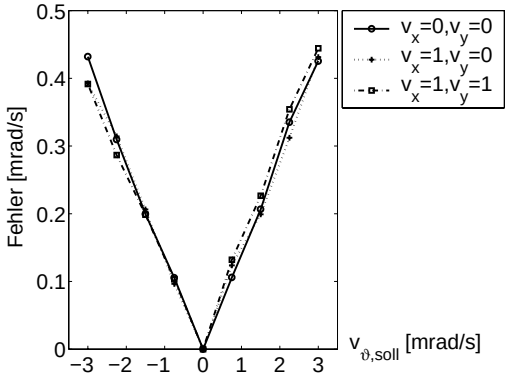
(a) Typisches Sprungverhalten



(b) Relativer Geschwindigkeitsfehler (bezogen auf v_{soll}) der Translation in %



(c) Parasitäre Rotation in mrad/s



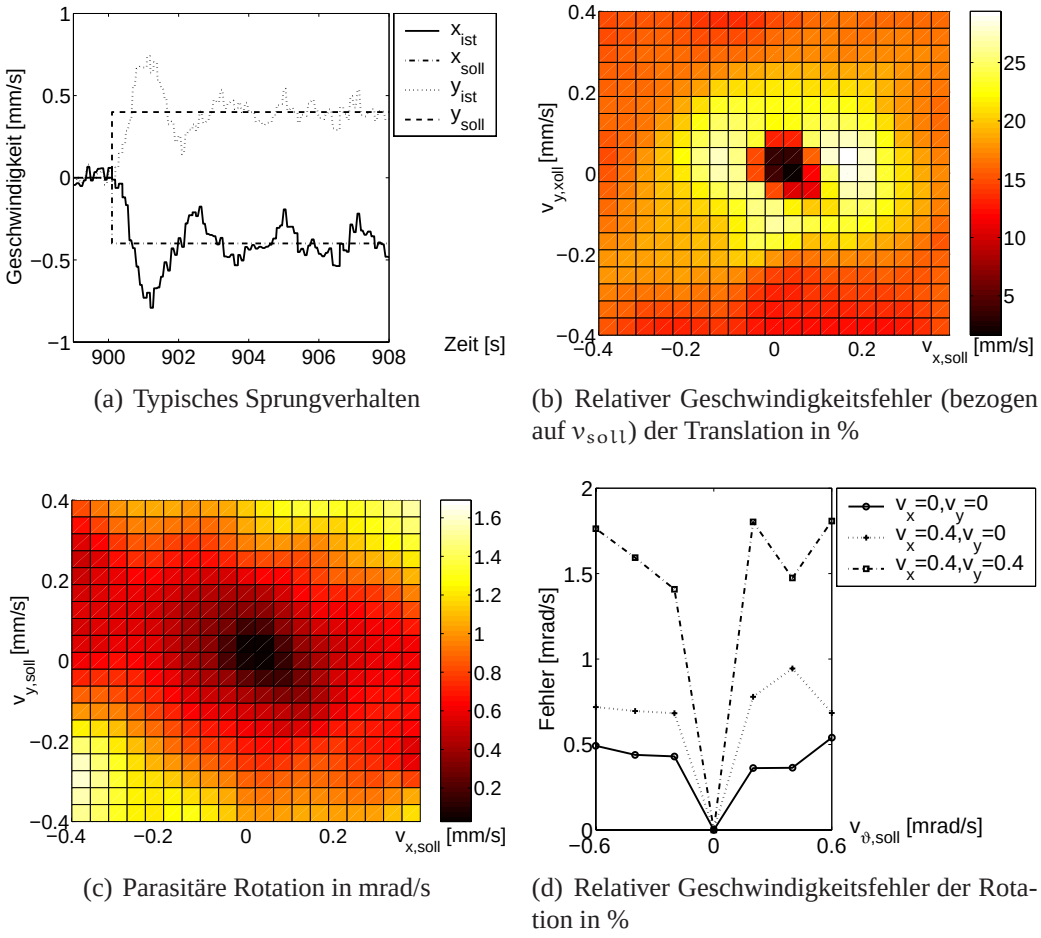
(d) Relativer Geschwindigkeitsfehler der Rotation in %

Abbildung 7.8: PID-Geregelte Bewegung des Miniman III

Generell ist der relative Fehler (dieser ist hier - abweichend von der sonstigen Definition - der besseren Darstellbarkeit wegen bezogen auf die Sollgeschwindigkeit) bei sehr niedrigen Geschwindigkeiten gering, bei mittleren Geschwindigkeiten am größten und bei hohen Geschwindigkeiten auf einem mittleren Niveau. Der absolute Fehler steigt jedoch fast monoton mit der Geschwindigkeit an.

Der Differentialanteil erwies sich als nachteilhaft, da er zu starken Schwingungen des Systems führte. Grund dafür dürfte das ausgeprägte Rauschen sein. Die Ergebnisse können wahrscheinlich durch manuelle Feinabstimmung der Parameter noch etwas verbessert werden.

Die Positionsabweichung am Ende einer Bahn ist tatsächlich niedriger als die Aufsummierung der einzelnen Geschwindigkeitsfehler (vgl. Tabelle 7.6), denn negative und positive Bahnabweichung heben sich größtenteils auf.



(a) Typisches Sprungverhalten

(b) Relativer Geschwindigkeitsfehler (bezogen auf v_{soll}) der Translation in %

(c) Parasitäre Rotation in mrad/s

(d) Relativer Geschwindigkeitsfehler der Rotation in %

Abbildung 7.9: PID-Geregelte Bewegung des MiCRoN 0

Der PID-Regler erwies sich im Vergleich zum PPT1-Regler als leicht vorteilhaft in der Regelgüte, was zu einem Teil auf das Überschwingen durch den hohen D-Anteil zurückzuführen ist. Auffällig ist ferner, dass beim PID-Regler insbesondere die Regelgüte in Bezug auf den Positionsfehler deutlich besser ist.

7.2.2 Störgrößenaufschaltung

Ein großes Problem, das bei der Approximation des System durch mehrere unabhängige Regler (SISO) auftrat, waren parasitäre Bewegungen, insbesondere Rotationen. Obwohl das System über mehrere Eingangs- und mehrere Ausgangsgrößen verfügt, handelt es sich hier nicht um ein MIMO-System eigentlicher Prägung. Vielmehr ist die Wirkung eines Stellbefehls auf andere Dimensionen (para-

sitäre Schleichbewegungen) als Störgröße aufzufassen. Da ihr Übertragungsverhalten aus dem Streckenmodell bekannt ist, kann eine Adaption im Sinne einer Störgrößenaufschaltung zum Zwecke ihrer Eliminierung durchgeführt werden.

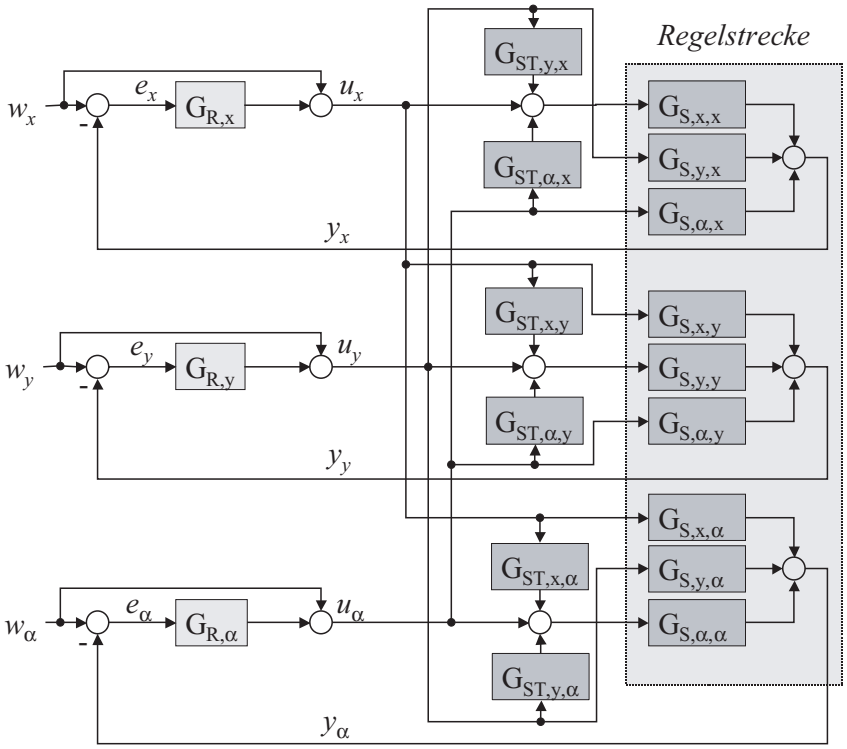


Abbildung 7.10: Signalflussbild eines Reglers mit Störgrößenaufschaltung

In Abbildung 7.10 ist eine mögliche Realisierung einer solchen Störgrößenkompensation aufgeführt. Es gilt:

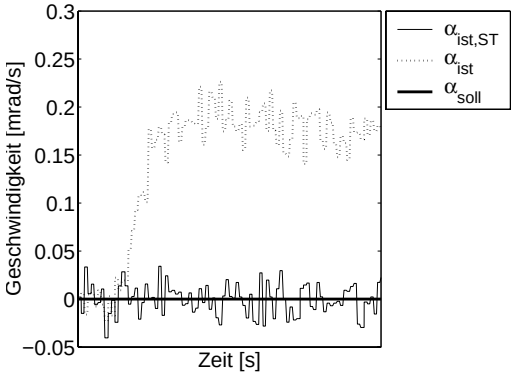
$$\vec{y} = A\vec{u}'_R$$

mit

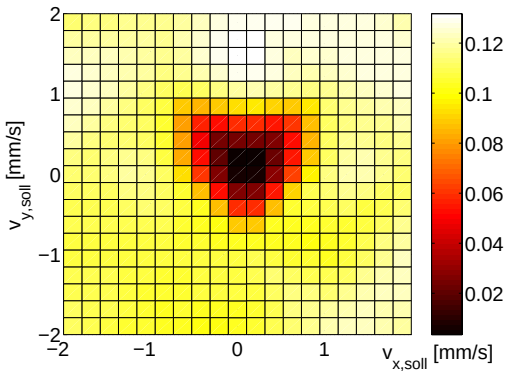
$$A = \begin{bmatrix} G_{S,x,x} & G_{S,y,x} + G_{S,x,x}G_{ST,y,x} & G_{S,\alpha,x} + G_{S,x,x}G_{ST,\alpha,x} \\ G_{S,x,y} + G_{S,y,y}G_{ST,x,y} & G_{S,y,y} & G_{S,\alpha,y} + G_{S,y,y}G_{ST,\alpha,y} \\ G_{S,x,\alpha} + G_{S,\alpha,\alpha}G_{ST,x,\alpha} & G_{S,y,\alpha} + G_{S,\alpha,\alpha}G_{ST,y,\alpha} & G_{S,\alpha,\alpha} \end{bmatrix} \tag{7.13}$$

Die Streckenübertragungsfunktionen $\mathbf{G_S}$ sind aus der Streckenanalyse bekannt. Auslegungsziel muss es sein, möglichst alle Element außerhalb der Hauptdiagonalen zu minimieren. Dies gelingt über die Anpassung der Störgrößenfilter G_{ST}. Für einen technisch realisierbaren Regler darf allerdings in der Übertragungsfunktion der Zählergrad nicht größer als der Nennergrad sein.

Bei der Realisierung wurden für den Regler G_R die selben Parameter benutzt wie im vorangegangenen Abschnitt (PID-Regler). Die Ergebnisse zeigen eine starke Reduktion der parasitären Bewegungen (siehe Abb. 7.11), was der wesentliche Grund für die Verminderung des Gesamtfehlers darstellt (vgl. Tabelle 7.6 auf Seite 132).



(a) Bei einem Sollwertsprung von v_y (α_{ist}: Rotation ohne Störgrößenkompensation, $\alpha_{ist,ST}$: Rotation mit Störgrößenkompensation)



(b) Bei reiner Translation in mrad/s

Abbildung 7.11: Parasitäre Rotation nach Störgrößenkompensation beim Miniman III

Es darf nicht unerwähnt bleiben, dass das vorgestellte Störgrößenschema nur ungefähr gilt. Idealerweise müsste eine weitere Rückkopplung auf die Ausgangsgröße nach der Störgrößenaufschaltung berücksichtigt werden.

7.2.3 Zustandsregelung

Bei einem Zustandsregler werden zusätzlich zu den Ein- und Ausgangsgrößen des Systems weitere, innere Größen des Systems betrachtet. Diese fließen ebenfalls in die Berechnung der Stellbefehle ein. Diese inneren Größen können mit tatsächlichen physikalischen Größen korrespondieren, können aber auch frei gewählt sein. Ein Zustandsregler basiert auf einem Zustandsmodell der Strecke, so wie es in Abschnitt 7.1.1 bestimmt wurde. Im Allgemeinen (aber nicht zwingend) ist die Durchgangsmatrix **D** dabei Null.

$$\begin{aligned} \vec{x}_z(t+T) &= \mathbf{A}\vec{x}_z(t) + \mathbf{B}\vec{u}(t) \\ \vec{y}(t) &= \mathbf{C}\vec{x}_z(t) \end{aligned} \tag{7.14}$$

Voraussetzungen für die Realisierung eines Zustandsregler sind:

- Ein mathematisches Modell der Strecke muss vorhanden sein. Dieses wurde bereits in Abschnitt 7.1.1 aufgestellt.
- Die Zustandsgrößen müssen messbar oder zumindest beobachtbar sein. Auch dann, wenn die Zustandsgrößen tatsächlichen physikalischen Größen entsprechen, ist eine Messung der Zustandsvariablen selten mit einem vertretbaren Aufwand realisierbar, so dass die Beobachtung als Möglichkeit bleibt. Dies ist auch in unserem Fall zwingend notwendig. Dazu muss die Beobachtbarkeit des Systems untersucht werden.
- Das System muss steuerbar sein.

Die Beobachtbarkeit eines Systems kann über die Determinante der Beobachtbarkeitsmatrix ermittelt werden. Ein System ist vollständig beobachtbar wenn gilt:

$$\det \mathbf{Q}_B \neq 0 \qquad \text{mit } \mathbf{Q}_B = \begin{bmatrix} \mathbf{C} \\ \mathbf{C} \cdot \mathbf{A} \\ \mathbf{C} \cdot \mathbf{A}^2 \\ \vdots \\ \mathbf{C} \cdot \mathbf{A}^{n-1} \end{bmatrix} \tag{7.15}$$

Ein System ist vollständig steuerbar, wenn für die Determinante der Steuerbarkeitsmatrix gilt:

$$\det \mathbf{Q}_S \neq 0 \qquad \text{mit } \mathbf{Q}_S^T = \begin{bmatrix} \mathbf{B} \\ \mathbf{A} \cdot \mathbf{B} \\ \mathbf{A}^2 \cdot \mathbf{B} \\ \vdots \\ \mathbf{A}^{n-1} \cdot \mathbf{B} \end{bmatrix} \tag{7.16}$$

Ist das System steuer- und beobachtbar können bei einer Zustandsregelung die Eigenwerte bzw. Pole der Regelung direkt vorgegeben werden.

Die Pole der Zustandsübertragungsfunktion $x_z(t)/u_w(t)$ können durch den proportional wirkenden Zustandsregler (G_{zr}) vorgegeben werden. G_{zr} wird mit Hilfe der homogenen Zustandsdifferenzengleichung der Regelstrecke bestimmt:

$$\vec{x}_z(t+T) = \mathbf{A}\vec{x}_z(t) + \mathbf{B}\left(\mathbf{G}_{zr}\vec{x}_z(t) + \mathbf{G}_v\vec{w}(t)\right) \tag{7.17}$$

Für Details zu Bestimmungsmethoden siehe [52] bzw. für MIMO-Systeme [42]. Die Pole des geschlossenen Regelkreises sollten große negative Realteile haben, um eine schnelle Regelung zu ermöglichen. Dabei muss natürlich berücksichtigt werden, dass die Begrenzung der Stellbefehle nicht über- bzw. unterschritten wird.

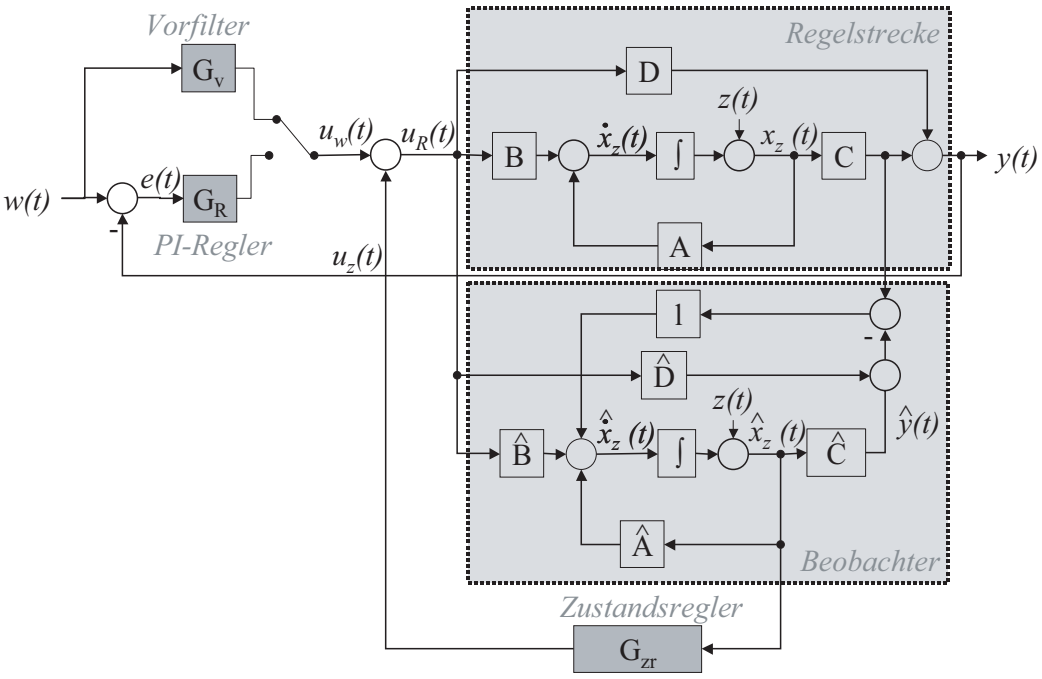


Abbildung 7.12: Prinzip eines Zustandsreglers mit Beobachter und Vorfilter bzw. PI-Regler

Häufig dient ein Vorfilter G_v der Kompensation von bleibenden Regelabweichungen. Er wirkt als rein proportionales Glied: $u_w(t) = G_v \cdot w(t)$. Normiert man seine Ausgabe über die Einheitssprungantwort, so ergibt sich als Berechnungsgleichung:

$$\mathbf{G}_v = -\left[\mathbf{C} \cdot (\mathbf{A} + \mathbf{B} \cdot \mathbf{G}_{zr})^{-1} \cdot \mathbf{B}\right]^{-1} \tag{7.18}$$

Wegen der Robustheit der Regelung gegenüber Parameteränderungen[5] oder auch Abweichungen des Modells gegenüber der Realität wird meist ein PI-Regler anstelle eines nur verstärkend wirkenden Vorfilters einsetzen. Damit sind jedoch die Regelabweichungen $\vec{e}(t) = \{e_x(t), e_y(t), e_\theta(t)\}$ als zusätzliche Zustandsgrößen beim Zustandsregler zu berücksichtigen. Für Details zur Realisierung siehe z.B. [52].

Idealerweise ist das Übertragungsverhalten der Strecke gleich des Verhaltens des Beobachters: $\mathbf{A} = \hat{\mathbf{A}}, \mathbf{B} = \hat{\mathbf{B}}, \mathbf{C} = \hat{\mathbf{C}}, \mathbf{D} = \hat{\mathbf{D}} \Rightarrow \vec{y}(t) = \hat{\vec{y}}(t)$. Tatsächlich ergibt sich jedoch eine Differenz, die über die Beobachtungsmatrix $\mathbf{L}$ in das Modell einfließt:

$$\hat{\vec{x}}_z(t+T) = \hat{\mathbf{A}}\hat{\vec{x}}_z(t) + \mathbf{L}(\vec{y}(t) - \hat{\vec{y}}(t)) + \hat{\mathbf{B}}\vec{u}(t) \tag{7.19}$$

[5] Etwa durch Verschleiß von Bauteilen

Die Realisierung des Zustandsreglers für die untersuchten Mikroroboter ergab, dass alle Roboter mit ihren Zustandsmodellen steuer- und beobachtbar sind. Eine Durchführung zeigt ein im Vergleich zum PID-Regler mit Störgrößenkompensation sogar schlechteres Verhalten (siehe Tabelle 7.6). Der Grund dafür dürfte in den stärkeren Schwingungen des Reglers zu suchen sein. Dies kann möglicherweise durch eine geschicktere Wahl der Pole noch verbessert werden.

Ein Vergleich der beiden Zustandsregler-Varianten (mit Vorfilter und mit PI-Regler) zeigt eine grundsätzliche Überlegenheit der PI-Variante, die jedoch nicht sehr ausgeprägt ist. Es konnten damit vor allem bleibende Regelabweichungen verringert werden. Im Vergleich zu anderen Reglern schneidet die Zustandsregelung schlechter ab, weil häufig Stellbefehle angefordert werden, die oberhalb der zulässigen Grenze liegen. Ferner hat sich als negativ herausgestellt, dass eine starke Belastung der Piezoelemente durch häufiges und ausgeprägtes Regeln bis an die Stellbefehlsgrenzen hervorgerufen wurde. Diese Tendenz wurde durch den PI-Anteil nicht ausreichend gemindert, so dass ein praktischer Einsatz des Zustandsreglers kaum gegeben ist.

Besonders problematisch war die Rotationsregelung des MiCRoN 0. Die Schwingungen ebenso wie die parasitäre Rotation waren hier besonders deutlich und störend. Bei der Variante mit dem Vorfilter war dieses Problem überhaupt nicht in den Griff zu bekommen.

7.2.4 Regelung auf Basis eines inversen Modells

Der *Linear Inverse Model Controller* (LIMC) ist ein sogenannter *deadbeat*-Regler, der ein sehr schnelles und robustes Verhalten zeigt [76]. Ausgangspunkt des Designs ist ein linearisiertes Robotermodell:

$$\vec{y}(t+T) = \mathbf{A}(t)\vec{y}(t) + \mathbf{B}(t)T\dot{\vec{y}}(t) + \mathbf{E}(t) \tag{7.20}$$

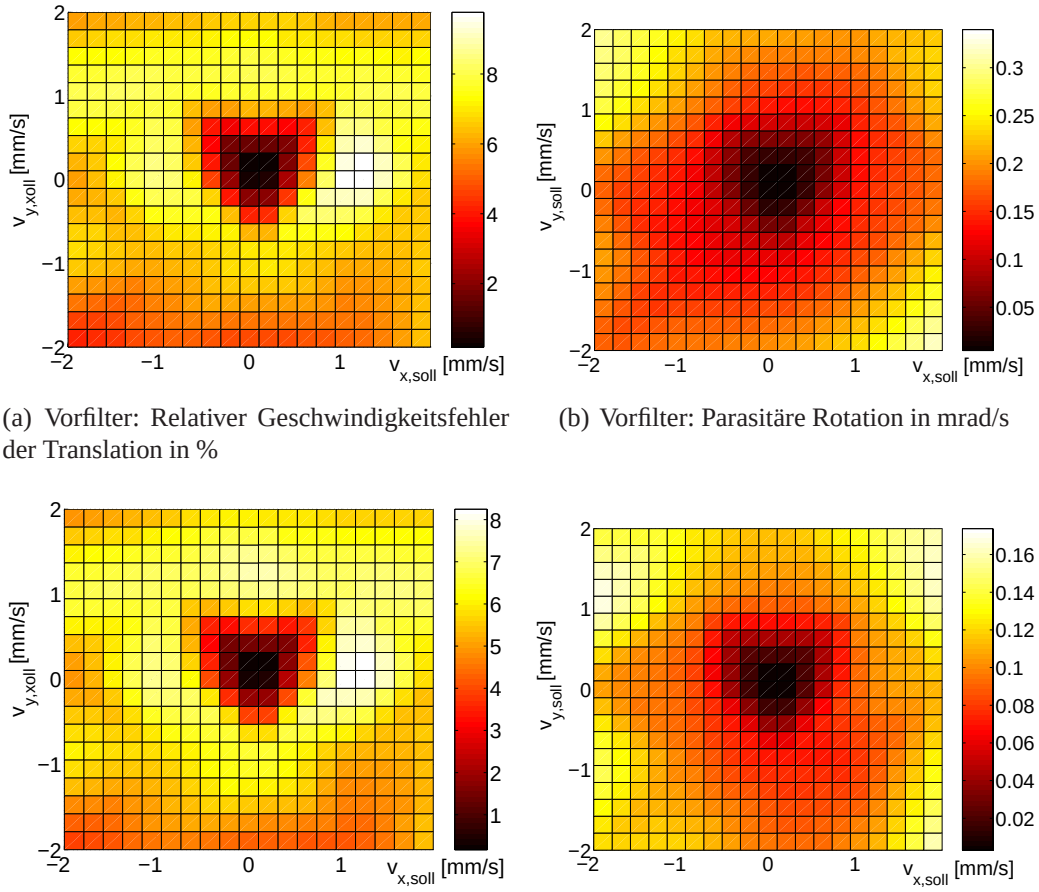
Anders als bei einem Zustandsregler werden hier keine inneren Zustände des Systems berücksichtigt und der Ausgang kann direkt berechnet werden.

Die Werte für die Parameter-Matrizen **A**, **B** und **E** hängen vom Roboter ab und müssen mit Hilfe eines geeigneten Schätzers approximiert werden. Die Invertierung des Modells liefert eine Gleichung für den Reglerausgang:

$$\vec{u}_R(t) = \dot{\vec{y}}(t) = \frac{1}{T}\mathbf{B}^{-1}(t)[\vec{y}_w(t+T) - \mathbf{A}(t)\vec{y}(t) - \mathbf{E}(t)] \tag{7.21}$$

Der Term $\vec{y}_w(t)$ ist die gewünschte Position im nächsten Zeitschritt $t+\Delta t$. Da die Taktrate in der Praxis nicht immer konstant und im voraus bekannt ist[6], ist diese

[6]Die tatsächliche Zeit zwischen zwei Berechnungsschritten hängt u.a. von der Belastung des Prozessors ab



(a) Vorfilter: Relativer Geschwindigkeitsfehler der Translation in %

(b) Vorfilter: Parasitäre Rotation in mrad/s

(c) PI: Relativer Geschwindigkeitsfehler der Translation in %

(d) PI: Parasitäre Rotation in mrad/s

Abbildung 7.13: Zustandsgeregelte Bewegung des Miniman III

Form nicht praktisch. Eine Umwandlung zu

$$\vec{u}_R(t) = \mathbf{B}^{-1}(t)\vec{w}(t) - \frac{1}{T}\mathbf{B}^{-1}(t)(\mathbf{A}(t)\vec{y}(t) + \mathbf{E}(t) - \vec{y}(t)) \tag{7.22}$$

erlaubt die direkte Vorgabe der Sollgeschwindigkeit $\vec{w}(t)$. Anstelle der Bias-Matrix $\mathbf{E}(t)$ kann auch die Dynamik des Prädiktions-Fehlers berücksichtigt werden.

Die Parameter-Matrizen $\mathbf{A}, \mathbf{B}$ und die Bias-Matrix $\mathbf{E}$ werden im laufenden Betrieb von einem Optimieralgorithmus adaptiert. Die Methode der rekursiven kleinsten Fehlerquadrate (RLS) minimiert den Fehler und ist schnell genug um in Echtzeitanwendungen eingesetzt zu werden [45]. Da die Parameter des Roboters und damit die Parameter-Matrizen mit der Zeit variieren, wurde eine Lernrate eingeführt. Damit werden neuere Daten stärker berücksichtigt. Um einen *Windup*

des Schätzers zu vermeiden, wurde ein Algorithmus eingebaut, der als *Regularized Constant Trace* Algorithmus [5] bekannt ist. Wegen Instabilitäten bei hohen Geschwindigkeiten und bei plötzlichen Richtungsänderungen werden hohe Schätzfehler diskontiert (*Robust Estimation*).

Der vorgestellte Regler hat sich als stabil und präzise herausgestellt (vgl. Tabelle 7.6 auf Seite 132). Für die meisten sinnvollen Trajektorien liegt der mittlere Regelfehler bei weniger als 30 µm, bei niedrigen Geschwindigkeiten noch erheblich darunter. Etwas negativ hat sich die starke Belastung der Aktoren durch ausgeprägte Stellbefehlssprünge herausgestellt. Dieses führte in einigen Fällen zu Schwingungen und Instabilitäten.

7.3 Nichtlinearer Reglerentwurf

Die bisher aufgeführten Regler gelten streng genommen nur für lineare Regelstrecken. Wie schon gezeigt, ist die betrachtete Regelstrecke jedoch nicht linear, so dass es sich lohnen sollte, auch nichtlineare Regelansätze zu untersuchen.

7.3.1 Modellprädiktiver Regler

Die modellprädiktive Regelung (*Model Predictive Control*, MPC) ist ein in der Industrie (insbesondere der Petrochemie) weit verbreiteter Ansatz für Mehrgrößen-Systeme mit Stellgrößenbeschränkung [83]. Es wird dabei in jedem Zeitschritt mit Hilfe eines Modells das zukünftige Prozessverhalten über eine bestimmte Anzahl von Zeitschritten - dem sogenannten Prädiktionshorizont - vorausgesagt. In seiner ursprünglichen Form wurde ein lineares Modell benutzt, so daß eine direkte Berechnungsgleichung für den Reglerausgang hergeleitet werden kann (analog Abschnitt 7.2.4). Wenn nichtlineare Modelle benutzt werden, muss eine geeignete Optimierroutine benutzt werden, um anhand eines Gütekriteriums den günstigsten Reglerausgang zu finden. Das Prädiktionsmodell kann bei Kenntnis der Antwort der Regelstrecke auch fortlaufend adaptiert werden.

In die Gütefunktion fließen nicht nur der Regelfehler des betrachteten Zeithorizontes ein, sondern es können auch die Stellinkremente berücksichtigt werden. Ziel dabei ist es, eine möglichst gleichmäßige Bewegung des Roboters ohne ruckartige Bewegungen zu erreichen.

$$\vec{J}_G(\vec{y},\vec{u}) = \sum_{i=1}^{T_p} (\vec{w}_i - \vec{y_{m,i}})^T \mathbf{Q_y} (\vec{w}_i - \vec{y_{m,i}}) + \sum_{i=0}^{T_c} (\vec{u}'_{R,i} - \vec{u}_{R,i})^T \mathbf{Q_u} (\vec{u}'_{R,i} - \vec{u}_{R,i}) \tag{7.23}$$

Hierbei ist J_G die Gütefunktion, T_p der Prädiktionshorizont der Regelgröße, T_c der

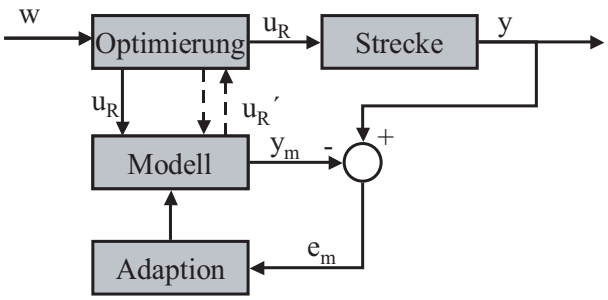


Abbildung 7.14: Schema eines modellprädiktiven Reglers

Prädiktionshorizont der Stellgröße[7], w der Sollwert, y_m die Modellausgabe, u_R' der vom Optimierer angefragte Stellbefehl und u_R der dann letztendlich ausgegebene Stellbefehl. Statt des quadratischen Fehlers wie in Gleichung 7.23 können natürlich auch andere Fehlerkriterien benutzt werden. Über die Matrizen $\mathbf{Q_u}$ und $\mathbf{Q_y}$ kann der Einfluss der Stellgröße bzw. der Regelgröße auf die Güte der Modellausgabe gewichtet werden.

Es ist offensichtlich, daß der Rechenaufwand und damit die Berechnungsdauer mit steigendem Zeithorizont ansteigt und damit die Echtzeitfähigkeit beeinträchtigt. Andererseits sinkt dabei die Wahrscheinlichkeit, ein instabiles oder zumindest unerwünschtes Streckenverhalten zu bekommen [29]. Da die Sensitivität und das Antwortverhalten bei den zu regelnden Robotern nicht bei allen Aktoren identisch sind, müssen unterschiedliche Prädiktionshorizonte für die einzelnen Freiheitsgrade berücksichtigt werden.
Als Modell kommen hier sowohl lineare als auch nichtlineare Ansätze in Frage. Im nichtlinearen Fall werden typischerweise neuronale Netze benutzt [73], aber einen Einsatz anderer Modelle ist genauso möglich [83]. Für den Einsatz in der Mikrorobotik können im Prinzip alle in Abschnitt 7.1.1 erstellten Modelle benutzt werden. Die Adaption des Modells während des laufenden Betriebs ist nicht zwingend notwendig, erhöht aber auf längere Sicht die Regelgüte.

Bei einem Einsatz eines genetisch erzeugten Modells ist jedoch die Größe der Gleichung so zu beschränken, dass eine Auswertung die Taktrate nicht übermäßig in die Höhe treibt. Dies betrifft sowohl die Optimierung - also das Finden eines geeigneten Stellbefehls - wie auch die Adaption des Modells. Bei den evolutionär erzeugten Modellen ist zu beachten, daß auch diskontinuierliche Zustände erzeugt werden können. Eine Optimierroutine sollte mit diesem Sachverhalt umgehen können.

Dieser Regler hat sich in der Praxis wenig bewährt. Die sonst übliche Taktzeit

[7] im Allgemeinen ist $T_c \leqslant T_p$

von 150-250 ms konnte wegen des Optimierbedarfs nicht eingehalten werden, insbesondere dann, wenn auch noch eine Adaption des Modells durchgeführt wurde. Die evolutionär erzeugten Modelle waren ebenso wie die analytischen Modelle wegen der ausgeprägten Nichtlinearitäten besonders ungeeignet, hier lagen realisierbare Taktzeiten durchweg im Bereich > 1 Sekunde. Etwas schnellere Ergebnisse konnten mit Zustandsmodellen und NARMA-Modellen erzielt werden. Damit war dann eine stabile Regelung möglich, jedoch mit einem recht beträchtlichen Regelfehler (vgl. Tabelle 7.6).

7.3.2 Feedback Linearisierung

Feedback Linearisierung (auch *NARMA*-L2 genannt) ist ein Verfahren, das auf zwei vorab trainierten neuronalen Netzen basiert [58]. Diese stellen eine Approximation an das *NARMA*-Modell dar (vgl. Abschnitt 7.1.1).

$$\begin{aligned} w(t+hT) =& N_1 + N_2 \cdot u(t+T) \\ & \text{mit} \\ N_1 =& f\left(y(t), y(t-T), .., y(t-n_y T), u(t), u(t-T), ..., u(t-n_u T)\right) \\ N_2 =& f\left(y(t), y(t-T), .., y(t-n_y T), u(t), u(t-T), ..., u(t-n_u T)\right) \end{aligned} \tag{7.24}$$

wobei $h \geqslant 2$ sein muss und N_1 und N_2 zwei (in der Regel unterschiedliche) neuronale Netze sind. Der Vorteil gegenüber einem *NARMA*-Modell ist darin zu sehen, dass die gesuchte Stellgröße $u(t+T)$ linear in diesem Modell auftaucht und daher leicht berechnet werden kann:

$$u(t+T) = \frac{w(t+hT) - N_1}{N_2} \tag{7.25}$$

Die Netze N_1 und N_2 können durch (statische) Backpropagation oder ein anderes Optimierverfahren identifiziert werden. Im Rahmen dieser Arbeit wurde die Backpropagation benutzt.

Die Ergebnisse zeigen einen sehr stabilen und schnellen Regler, der für alle untersuchten Mikroroboter nur geringe Bahnabweichungen zulässt (vgl. Tabelle 7.6).

7.3.3 Evolutionär optimierter Regler

Die bisher diskutierten und eingesetzten Regler basieren auf bewährten Auslegungsprinzipien, die jedoch einige Nachteile aufweisen:

- Sie basieren teilweise auf einer linearen Regelstrecke.

- Die Auslegung erfordert theoretische Kenntnisse bzw. eine Analyse der Regelstrecke und ist oft zeitaufwändig.
- Die Reglertopologie ist meist vorab schon festgelegt
- Die Anzahl der arithmetischen Operationen ist begrenzt, komplexe Programmstrukturen können nicht einbezogen werden.

Dies alles motiviert die Suche nach einem flexibleren Werkzeug. Die in Kapitel 5 vorgestellte Methode basiert auf genetischer Programmierung und ist flexibel genug, nicht nur Parameter, sondern auch die Topologie eines Reglers festzulegen. Es sind erst wenige Versuche unternommen worden, mit evolutionären Mitteln Regler zu identifizieren, meist für relativ einfache Regelstrecken. In [47] beispielsweise wird eine Strecke mit der Übertragungsfunktion

$$G(s) = \frac{K}{(1 + T_1 s)^2} \tag{7.26}$$

mit einem genetisch erzeugten Regler geregelt (T_1 ist eine beliebige Zeitkonstante).

Da bei der genetischen Programmierung immer mehrere Lösungen in Konkurrenz stehen, vom realen Roboter aber immer nur eine Reglerausgabe umgesetzt werden kann, folgt die Notwendigkeit, ein Modell zu benutzen. Dieses Modell schätzt die vermutliche Reaktion des Roboters auf einen bestimmten Stellbefehl. Mit diesem Ansatz folgt jedoch unmittelbar, dass der Regler nie besser als dieses Modell sein wird (vgl. Abschnitt 7.1.1).

Gegenüber einer Modellbestimmung in einem offenen Regelkreis - wie für die Steuerung in Kapitel 6 geschehen - ist es möglich und sinnvoll, bei der Untersuchung eines geschlossenen Regelkreises zusätzlich zur Verfügung stehende Informationen zu nutzen. Neben der Sollgeschwindigkeit ($w(t)$) als eigentlicher Regelgröße ist dabei auch die Regelabweichung des letzten Zeitschritts ($e(t-T)$) zu nennen, die als zusätzliche Eingangsgröße in das System betrachtet werden kann. Darüber hinaus können aber auch die Istgeschwindigkeit ($\dot{y}(t-T)$) und die aktuelle Position ($y(t-T)$) einbezogen werden. Zu bedenken ist aber, dass ein Regler, der die aktuelle Position in die Berechnung des Stellbefehls einbezieht, seine Allgemeingültigkeit verliert. Er kann dann vermutlich nicht auf anderen Arbeitsflächen eingesetzt werden. Auch der Stellbefehl des letzten Zeitschritts ($u(t-T)$) kann einbezogen werden, wodurch eine direkte Rückkopplung des eigenen Regelerausgangs möglich wird. Zu beachten ist, dass in der benutzten Version von der Methode immer nur ein Reglerausgang berechnet werden kann, der Vorgang also für jeden Freiheitsgrad wiederholt werden muss. Da nun aber für die Strecke immer alle Eingangsgrößen bekannt sein müssen, stellt sich die Frage, welche Werte für die jeweils anderen Streckeneingänge anzunehmen sind, d.h. für

diejenigen Stellbefehle die nicht vom betrachteten Programm selbst abhängen. Es gibt nun mehrere Möglichkeiten diesem Problem zu begegnen.

- Ein GP, das mehrere Ausgänge produziert, wird benutzt. Dies würde jedoch der bisher entwickelten Struktur widersprechen.
- Mehrere GP laufen parallel und werden auch parallel entwickelt und gemeinsam bewertet. Dies würde jedoch eine bewertungsmäßige Trennung von guten und schlechten Programmen unmöglich machen.
- Es wird für die jeweils anderen Aus-/Eingänge ein Referenzwert angenommen und der Strecke aufgegeben. Dieser Referenzwert könnte im einfachsten Fall der Sollwert sein, da ja die Steuerung - als Teil des Streckenmodells - schon ein weitestgehende Umsetzung der Sollgeschwindigkeit vorsieht.

Eine interessanter Aspekt ist auch der Fitnesswert. In die Fitness sollte nicht nur die momentane Regelabweichung einfließen, sondern vielmehr der Reglerfehler über einen gewissen Zeithorizont hinweg. Es bietet sich hier das Integral der Regelabweichung an. Als zusätzliche Eingabe kann hier auch eine Bewertung der Stellinkremente einbezogen werden, wodurch eine zu starke Beanspruchung der Aktoren vermieden wird:

$$\begin{aligned} \Phi =& \frac{1}{1+\lambda_y e_{m,y}+\lambda_u e_{m,u}} \\ \text{mit} & \\ e_{m,y} =& \frac{\sum_{i=0}^{n_y} |e(t-iT)|}{(n_y+1)} \\ e_{m,u} =& \frac{\sum_{i=0}^{n_u} |u(t-iT)-u(t-(i+1)T)|}{(n_u+1)} \end{aligned} \tag{7.27}$$

Über die Faktoren λ_y und λ_u kann eine Gewichtung bzw. Normierung vorgenommen werden, die den Einfluss des Stellbefehls bzw. der Regeldifferenz auf die Fitness-Funktion reguliert. Um eine vollständige Bewertung eines Reglers zu erzielen, muss die Anzahl der betrachteten Zeitschritte maximal gewählt werden ($n_y = n_u =$ Anzahl der Modellausgaben).

In Tabelle A.1 im Anhang wird die Menge der mathematischen Funktionen vorgestellt, die implementiert sind. Nicht alle sind jedoch für Regelungszwecke geeignet oder sinnvoll. Um den Suchraum einzuschränken, findet deshalb eine Beschränkung auf die in Tabelle 7.5 aufgeführten Operatoren statt. Für den Einsatz einiger dieser Operatoren ist eine Kenntnis der Taktrate erforderlich. Diese

Taktrate wurde im Rahmen der Auslegung als konstant 250 ms angenommen, ein Wert, der dann auch im realen Betrieb erreicht wurde.

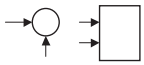
Name	Blockschaltbild	Name	Blockschaltbild
Summe		Differenz	
Integrierer		Bedingung Typ 1	
Differenzierer		Bedingung Typ 2	
Verstärkung		PT1	
Produkt		PT2	
Verzögerung			

Tabelle 7.5: Mathematische Operatoren für den genetischen Regler

Um der Tendenz des Algorithmus entgegenzuwirken, triviale Lösungen (wie etwa $u = w$) zu finden, wurde der Anfang des Programmbaums mit Additionen vorbelegt, d.h. nicht nur der Wurzelknoten sondern auch ein Kind- und Enkelknoten sind zu Beginn der Evolution Summen. Andernfalls führen einfache Lösungen wie die erwähnte häufig in die Sackgasse, da das genetische Material so klein ist, dass auch Kreuzungen kaum noch zu Variationen führen.

Im ersten Auslegungsschritt wurde nun für jeden Freiheitsgrad getrennt ein Regler anhand der Streckenmodellausgaben gemäß Abschnitt 7.1.2 berechnet. Nach dem auf diese Weise die besten Regler identifiziert wurden, konnte in einem Praxistest mit realen Robotern die gleichen Sollwertsprünge erneut aufgegeben werden. Die im Modell erzielten Ergebnisse haben sich dabei im Wesentlichen bestätigt.

Die Ergebnisse (vgl. Tabelle 7.6) zeigen für den Miniman III (und gleiches gilt auch für den Miniman IV) ein recht gutes Bewegungsverhalten, mit dem sich auch praktische Aufgaben zuverlässig lösen lassen. Relativ ernüchternd war das Ergebnis in Bezug auf den MiCRoN 0: Ein Bewegungsfehler von ca. 200 µm/s bzw. 5 mrad/s sind deutlich zu hoch, um hier noch eine Praxistauglichkeit zu bescheinigen.

Die Topologie des genetisch erzeugten Reglers zeigt Ähnlichkeiten sowohl mit PI-Reglern als auch mit Zustandsreglern. Es sind jedoch auch diskrete Schaltelemente enthalten und einige recht unkonventionelle Kombinationen. Die PT_1-

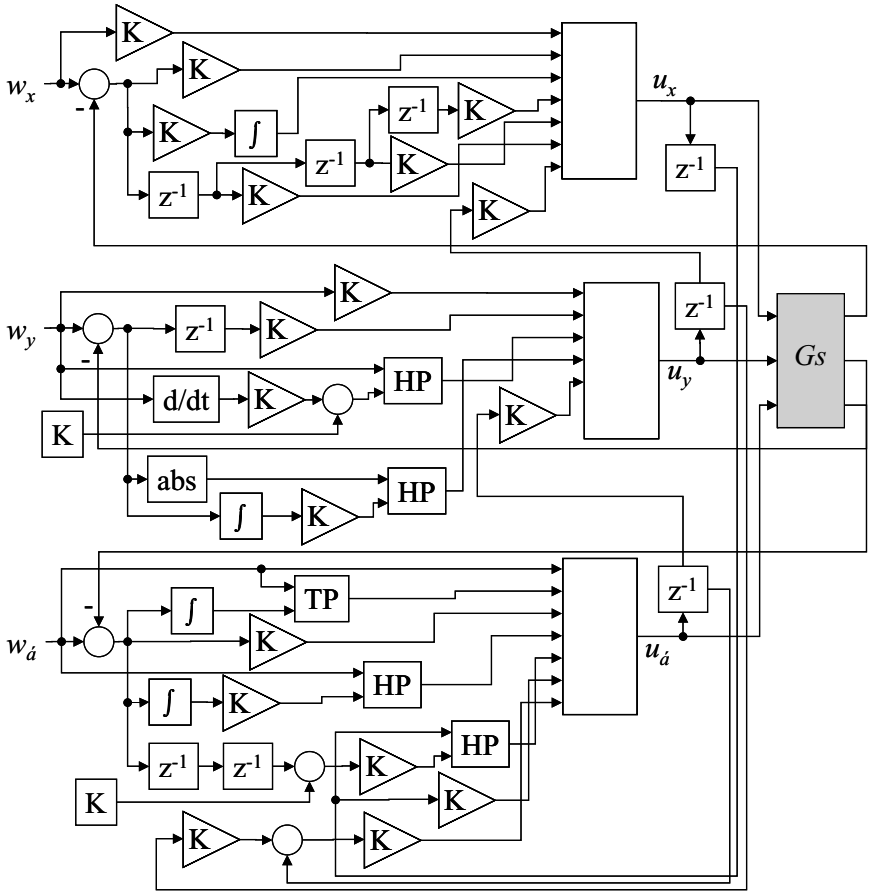
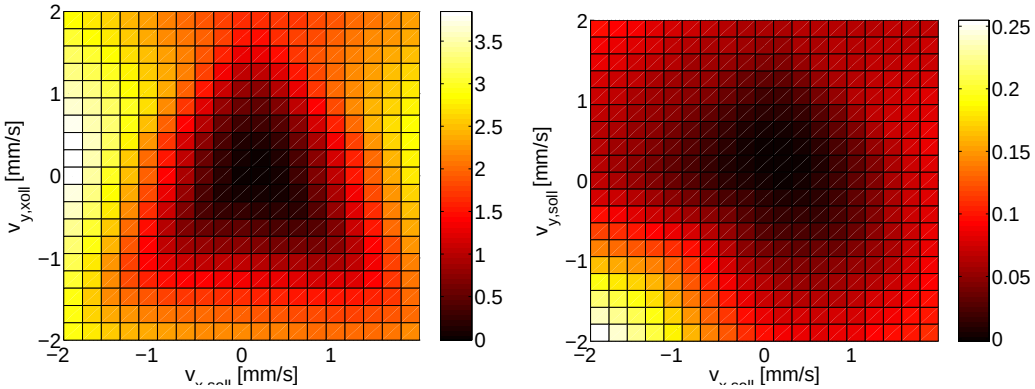


Abbildung 7.15: Schaltbild eines genetisch erzeugten Reglers für den Miniman III (vereinfacht)

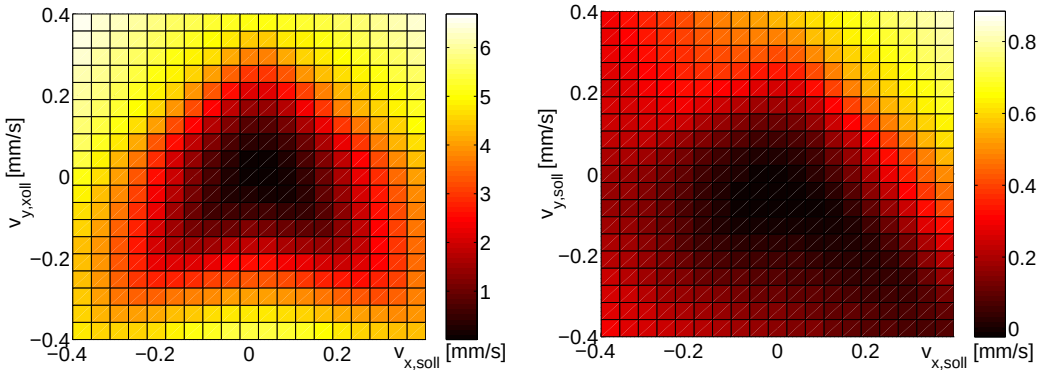


(a) Relativer Geschwindigkeitsfehler der Translation in %

(b) Parasitäre Rotation in mrad/s

Abbildung 7.16: Bewegung des Miniman III mit dem genetisch erzeugten Regler

und PT_2-Funktionen haben sich nicht bewährt, vermutlich weil der Einfluss der Konstanten (d.h. Speicherwerte, vgl. Gleichung 5.9 und 5.10) sehr groß ist und diese nicht Gegenstand eines Optimierungsprozesses sind. Auffällig häufig treten Fallunterscheidungen auf (Bedingung Typ 1 und 2), die offensichtlich gut geeignet sind, den teilweise unstetigen Verlauf der Strecke zu regeln.



(a) Relativer Geschwindigkeitsfehler der Translation in %

(b) Parasitäre Rotation in mrad/s

Abbildung 7.17: Bewegung des MiCRoN 0 mit dem genetisch erzeugten Regler

Bewährt hat sich die Einbeziehung des Stellinkrements in die Fitness-Funktion. Ein Anteil von ca. 30% lieferte hier die besten Ergebnisse (gegenüber 70 % Einfluss der Regeldifferenz) und führte dazu, dass der Roboter wesentlich gleichmäßiger lief.

Regler	Fehler	Miniman III			MiCRoN 0		
		x	y	θ	x	y	θ
PID	Geschw	0.0493	0.0463	0.114	0.0339	0.0362	2.18
	Pos	0.0121	0.0143	0.0332	0.00302	0.00206	1.78
PPT1	Geschw	0.0503	0.0458	0.267	0.0182	0.0199	1.53
	Pos	0.039	0.0201	0.238	0.00547	0.00827	1.43
Störgrößenk.	Geschw	0.049	0.0483	0.108	0.0177	0.0172	2.01
	Pos	0.00834	0.0156	0.0195	0.00321	0.00205	1.77
Zustandsr. VF	Geschw	0.0664	0.211	0.444	0.0859	0.0936	1.17
	Pos	0.0436	0.17	0.406	0.0763	0.0864	1.08
Zustandsr. PI	Geschw	0.0968	0.142	0.363	0.0259	0.0246	0.266
	Pos	0.0581	0.0502	0.313	0.0056	0.00785	0.114
LIMC	Geschw	0.0828	0.0599	0.4001	0.0412	0.0425	1.9254
	Pos	0.032	0.054	0.142	0.005	0.010	1.626
GenRegl	Geschw	0.0798	0.0947	0.118	0.214	0.22	4.91
	Pos	0.0772	0.0793	0.103	0.0154	0.0528	1.41
MPC	Geschw	0.131	0.131	0.312	0.0615	0.0981	1.28
	Pos	0.118	0.115	0.267	0.0456	0.0453	0.352
NARMA-L2	Geschw	0.0462	0.0454	0.194	0.0245	0.0246	0.732
	Pos	0.0223	0.0247	0.158	0.0102	0.0105	0.438

Tabelle 7.6: Absoluter mittlerer Fehler (MAE) [mm/s] bzw. [mrad/s] für verschiedene Reglertypen

7.4 Bewertung

Es wurden in diesem Kapitel insgesamt neun unterschiedliche Regler vorgestellt und an Mikrorobotern getestet. Tabelle 7.6 liefert einen Überblick über die erzielten Ergebnisse. Als besonders erfolgreich hat sich der *NARMA-L2*-Regler herausgestellt, der bei allen Robotern den geringsten Geschwindigkeitsfehler aufweist und nur beim Positionsfehler der Miniman-Roboter geringfügig über den Abweichungen des PID- und des störgrößenkompensierten Reglers liegt. Die Zustandsregler schneiden vergleichsweise schlecht ab, hier sind auch die stärksten Schwingungen der Stellgröße und - als Konsequenz - auch der Regelgröße zu beobachten.

Der *MPC*-Regler ist wegen des hohen Zeitbedarfs der Optimierung weniger für den Echtzeitbetrieb geeignet, mit ihm konnte eine Taktzeit von 150-250 ms nicht eingehalten werden, stattdessen musste T auf > 500 ms erhöht werden.

Ein weiterer Aspekt ist die Belastung des Roboters. Durch eine Begrenzung der Stellinkremente ist beim MPC- und - trotz einiger Unstetigkeiten - auch beim genetischen Regler ein deutlich glatterer Verlauf der Bewegung zu beobachten.

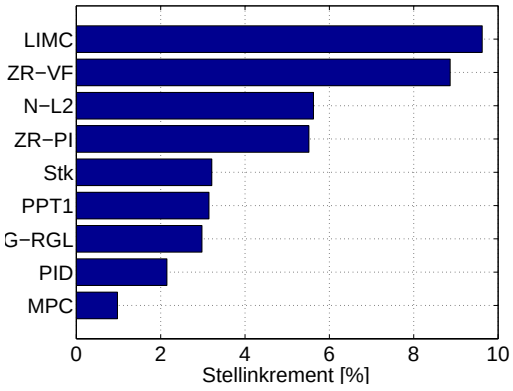


Abbildung 7.18: Mittleres Stellinkrement Δu_R [%] der Regler (gemittelt über alle untersuchten Mikroroboter)

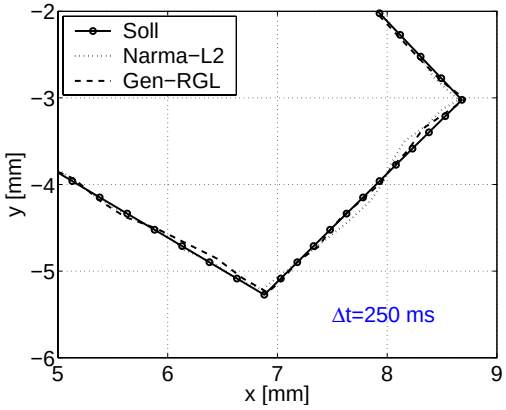
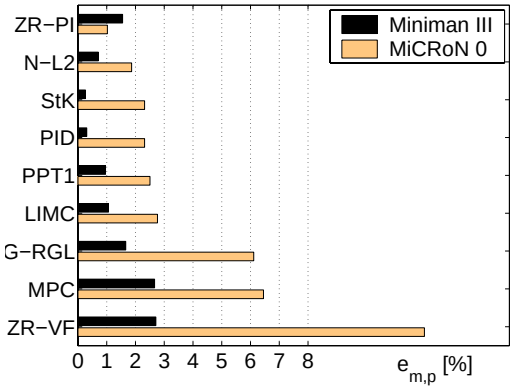
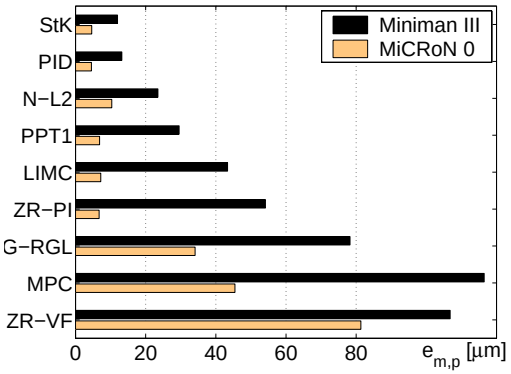


Abbildung 7.19: Beispieltrajektorie des Miniman IV

Starke Schwingungen - wie sie insbesondere bei den Zustandsreglern und dem LIMC-Regler auftreten - können vermieden werden (vgl. Abb. 7.18). Es muss natürlich festgehalten werden, dass dieses Verhalten in starkem Maße von der Parametrierung der einzelnen Regler abhängt.



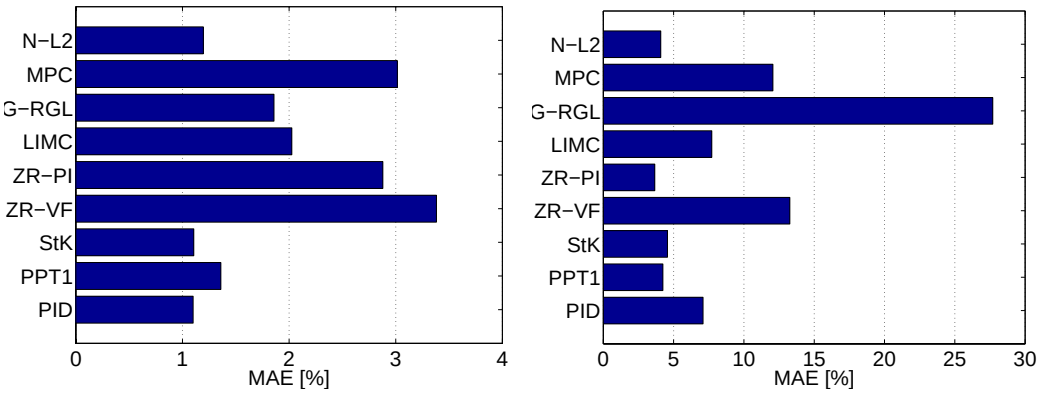
(a) Relative Bahnabweichung (Translation und Rotation)



(b) Mittlere Positionsabweichung der Translation

Abbildung 7.20: Gemittelter Positionsfehler

Eine Adaption des Reglers im Betrieb hat sich als nicht sehr erfolgreich herausgestellt. Beide eingesetzten adaptiven Regler - MPC und LIMC - leiden etwas unter dem ausgeprägten Rauschen des Systems und der Ortsabhängigkeit des Bewegungsverhaltens, was sich in beständigen (signifikanten) Parameteränderungen des Modells bemerkbar macht. Auch das Variieren der Lernrate hat hier nur bedingt Abhilfe geschaffen, vermutlich könnte nur eine grundsätzliche Modeller-



(a) Miniman III (b) MiCRoN 0

Abbildung 7.21: Relativer gemittelter Geschwindigkeitsfehler verschiedener Reglertypen

weiterung eine deutliche Verbesserung herbeiführen.

Der PI-Zustandsregler war der einzige, mit dem sich der MiCRoN 0 gut regeln ließ, insbesondere bei der Rotation war hier ein deutlicher Vorteil zu sehen. Die angestrebte Bahnabweichung von maximal 30 µm konnte von einer Reihe von Reglern teilweise deutlich unterboten werden (vgl. Abb. 7.20).

Der evolutionär erzeugte Regler hat sich für die Miniman-Roboter im Vergleich mit anderen Reglertypen bewährt und in der Praxis zu einem stabilen Regelbetrieb geführt, wenngleich festgehalten werden muss, dass andere Regler teilweise eine bessere Regelgüte aufweisen. Ein Blick auf Abbildung 7.19 zeigt, dass nur eine geringe Bahnabweichung auftritt, die sich im Mittel nicht wesentlich von der anderer Regler unterscheidet. Eine weitere Verbesserung der Regelgüte kann aber zweifellos durch Verfeinerungen der Lernmethode und auch durch weitere Trainingsläufe erzielt werden.

Kapitel 8

Zusammenfassung und Ausblick

Die Mikrorobotik ist ein Forschungsgebiet, das ein stets wachsendes Interesse der Öffentlichkeit hervorruft. Grund dafür sind die Anwendungsmöglichkeiten der Mikroroboter, beispielsweise in der Mikromontage und der Medizintechnik.

Evolutionäre Methoden bilden Prozesse nach, nach denen die Natur strukturierte und hochangepasste Lebewesen herausgebildet hat. Mit Hilfe der Evolution konnte Intelligenz entstehen.

In dieser Arbeit wurde der Versuch unternommen, diese beiden Forschungsgebiete miteinander zu kombinieren. Dabei war nicht die Nachbildung von komplexen Verhaltensmustern, sondern die Geschwindigkeitsregelung von Mikrorobotern mit evolutionären Methoden Gegenstand der Untersuchung.

Der in dieser Arbeit verwendete evolutionäre Ansatz basiert auf der genetischen Programmierung bzw. grammatischen Evolution. Beide Methoden bieten größtmögliche Flexibilität und sind in der Lage, auch stark nichtlineares Verhalten zu approximieren. Nachteil beider Verfahren sind jedoch der hohe Rechenzeitbedarf und die schwierige Parametrierung, die oft ein hohes Maß an Erfahrung des Benutzers erfordert. Diese beiden Nachteile werdem im vorgestellten Konzept der Meta-Evolution vermindert: Die Gesamtpopulation wird auf mehrere Gruppen aufgeteilt, die separat voneinander den evolutionären Prozess ablaufen lassen. Die Steuerungsparameter dieser Prozesse unterscheiden sich jedoch von Gruppe zu Gruppe und werden von einer übergeordneten Instanz ihrerseits einem evolutionären Optimierungsprozess unterworfen. Dadurch entledigt sich der Benutzer der Aufgabe, die Parametrierung selbst vorzunehmen. Durch die Aufteilung in Gruppen kann der Ablauf sehr leicht auf mehreren Prozessoren parallel durchgeführt werden, was in einer Beschleunigung des gesamten Prozesses resultiert.

Diese Methode stellt die Basis des Einsatzes für die Steuerung und Regelung von Mikrorobotern dar. Um ein flexibles, auf jeden Typ von Mikroroboter anwendbares Konzept zu haben, wurde ein zweistufiger Aufbau eingeführt: Die Steuerungsebene sorgt dafür, dass Geschwindigkeitsvorgaben in hardwarenahe

Signale übersetzt werden und dabei gleichzeitig Nichtlinearitäten in der Roboterbewegung weitgehend kompensiert werden. Die Steuerung wurde mit Hilfe von vorab gelernten Modellen realisiert. Die Regelungsebene bildet mit Hilfe einer Positionssensorik einen geschlossenen Regelkreis und berechnet das Eingangssignal für die Steuerung.

Sowohl für Steuerung als auch für die Regelung wurden neben dem evolutionären auch verschiedene andere Ansätze eingesetzt und miteinander verglichen. Als Robotermodell wurde ausgehend vom geometrischen Aufbau der Mikroroboter ein kinematisches Modell abgeleitet, das in seiner invertierten Form auch zur Steuerung eingesetzt werden kann. Verschiedene lineare und auch nichtlineare Regler wurden entwickelt und mit einem genetisch erzeugten Regler verglichen.

8.1 Ergebnisse

Genetische Verfahren sind keine deterministischen Verfahren und können somit keine Konvergenz garantieren. Häufig werden demzufolge völlig ungeeignete Lösungen produziert, die aber durch den Selektionsprozess schnell wieder von der Evolution selbst verworfen werden. Am Ende stehen nicht selten brauchbare bis gute Lösungen, die teilweise so unkonventionell sind, dass ein von Menschen erfolgter Entwurfsprozess niemals zu solch einem Ergebnis führen würde.

Die Ergebnisse zeigen, dass evolutionäre Methoden durchaus mit anderen Verfahren erfolgreich konkurrieren können. Als bestes Steuerungsmodell hat sich jedoch ein neuronales Netz herausgestellt. Dieses war sowohl dem evolutionären als auch dem analytisch hergeleiteten Modell überlegen. Speziell bei Robotern mit stark nichtlinearem Bewegungsverhalten fällt der Vorsprung der neuronalen Netze gegenüber evolutionär erzeugten Modellen geringer aus.

Auch bei den Reglern ist die insgesamt erfolgreichste Variante von neuronaler Struktur (*NARMA-L2*). Die evolutionär erzeugten Regler sind aber zumindest bei einigen der untersuchten Roboter ebenfalls geeignet, die Vorgaben in Bezug auf Genauigkeit zu erreichen. Als Vorteil eines evolutionären Reglerentwurfs hat sich hier auch die Möglichkeit herausgestellt, Nebenbedingungen in den Entwurfsprozess einzubeziehen. Auf diese Weise konnte das mittlere Stellinkrement und damit die Belastung der Aktoren deutlich reduziert werden.

8.2 Ausblick

Der Einsatz von evolutionären Methoden ist nicht auf die Bewegungsregelung von Mikrorobotern beschränkt. In künftigen Anwendungen können diese Methoden auf alle Bereiche der Verhaltenssteuerung ausgeweitet werden und damit

auch komplexe Verhaltensmuster gelernt werden. Ein Szenario ist in diesem Zusammenhang die Schwarmintelligenz, die auf einem dezentralen Multi-Agenten-Prinzip beruht. Natürlich kann die Evolution auch gewinnbringend bei eher hierarchischen Planungsansätzen sowie in der Muster- bzw. Objekterkennung genutzt werden.

Anhang A

Anhang

Lösung für $v_p(t)$ des dynamischen Modells mit $n_2 = 6$ (mit MAPLE berechnet):

$$
\begin{aligned}
v_P(t) =& (2(c_2\pi f\cos(2\pi ft)c_1^2 - 4c_2^3\pi^3f^3 + 8\pi^3f^3c_7c_2^3 + 32\pi^5f^5c_7c_2 - 16\sin(\pi ft)\cos(\pi ft)\pi^4f^4c_1 \\
&+ 8\pi^4f^4\sin(2\pi ft)c_1 + 16\pi^3f^3c_7c_2c_1 - 2c_7t\pi fc_1^3 + 2\pi^2f^2\sin(2\pi ft)c_1^2 + \pi ftc_1^3 \\
&- 8c_7t\pi^3f^3c_2^2c_1 - 16c_7t\pi^3f^3c_1^2 + 4\pi^3f^3tc_2^2c_1 + 8\pi^3f^3tc_1^2 + 16\pi^5f^5tc_1 - c_2\pi fc_1^2 - \\
&32c_7t\pi^5f^5c_1 + 2c_2\pi fc_7c_1^2 - 16\pi^5f^5c_2 - \sin(\pi ft)\cos(\pi ft)c_1^3 - 8c_2\pi^3f^3c_1 - \\
&4\sin(\pi ft)\cos(\pi ft)\pi^2f^2c_2^2c_1 - 8\sin(\pi ft)\cos(\pi ft)\pi^2f^2c_1^2 + 2c_2^2\pi^2f^2\sin(2\pi ft)c_1)c_3U_0)/ \\
&((c_2^2 + c_2c_{22} + 2c_1 + 8\pi^2f^2)(c_2^2 - c_2c_{22} + 2c_1 + 8\pi^2f^2)\pi fc_1^2)
\end{aligned}
\tag{A.1}
$$

Lösung für $\overline{v_B}$ des stationären Modells mit $n_1 = 2$ und $n_2 = 2$ (mit MAPLE berechnet):

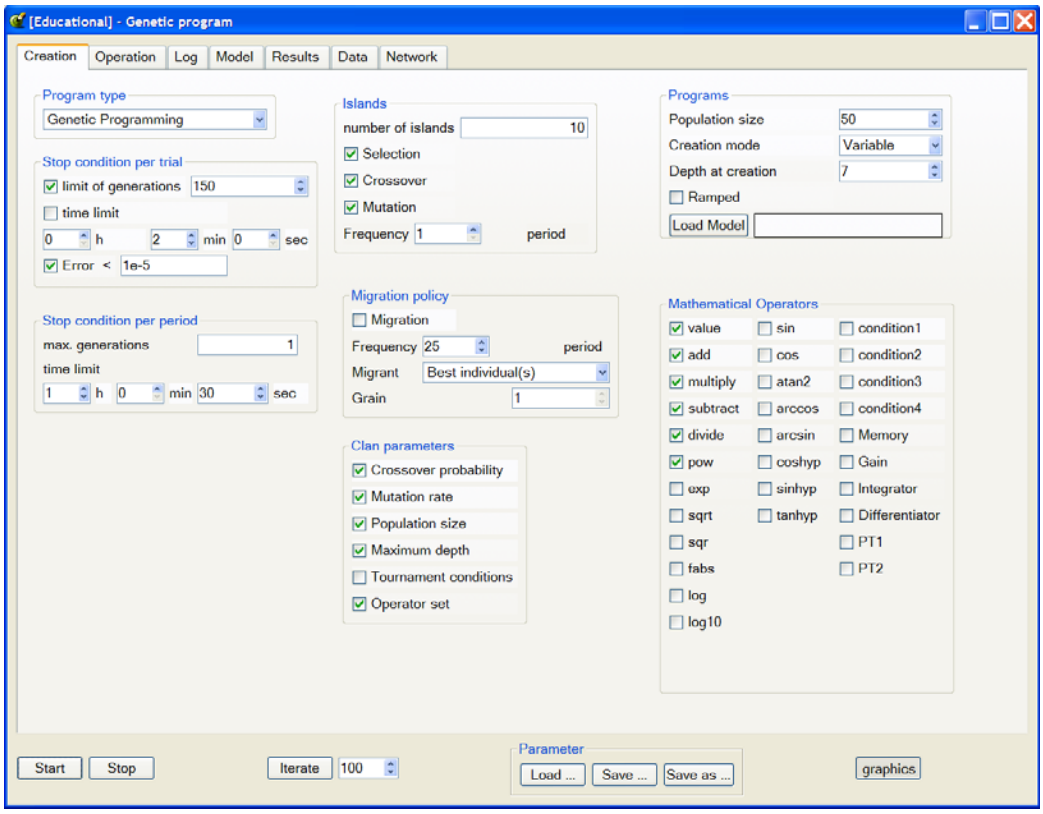
$$
\begin{aligned}
\overline{v_B} = {} & ((-512c_6h\pi^8b_2 - 512c_6h\pi^8a_2)f^8 + (-1024c_6h\pi^7a_1c_5\cos(\pi tf)^2 + 512c_6h\pi^7a_1c_5)f^7 + \\
& ((-256c_6h\pi^6b_2c_5^2 + 256c_6h\pi^6a_2c_5^2)\cos(\pi tf)^4 - 256c_6h\pi^6c_1b_2 + (-512c_6h\pi^6b_1c_5^2 \\
& - 256c_6h\pi^6a_2c_5^2 - 256c_4c_3U_0\pi^6c_5 + 256c_6h\pi^6b_2c_5^2)\cos(\pi tf)^2 - 256c_6h\pi^6c_1a_2 \\
& - 128c_6hc_2^2\pi^6a_2 + 256c_6h\pi^6b_1c_5^2 - 128c_6h\pi^6a_2c_5^2 - 128c_6hc_2^2\pi^6b_2 - 192c_6h\pi^6b_2c_5^2 \\
& + 128c_4c_3U_0\pi^6c_5)f^6 + (32c_6h\pi^5a_1c_5^3 + 256c_6h\pi^5c_1a_1c_5 + 128c_6hc_2^2\pi^5a_1c_5 \\
& + (-256c_6hc_2^2\pi^5a_1c_5 - 64c_6h\pi^5a_1c_5^3 - 512c_6h\pi^5c_1a_1c_5)\cos(\pi tf)^2)f^5 \\
& + (-32c_4c_3U_0\pi^4c_2c_5^2 + (-64c_6h\pi^4b_2c_5^4 + 128c_6h\pi^4c_1a_2c_5^2 + 64c_5^4c_6ha_2\pi^4 \\
& - 64c_6hc_2^2\pi^4b_2c_5^2 - 128c_6h\pi^4c_1b_2c_5^2 + 64c_6hc_2^2\pi^4a_2c_5^2)\cos(\pi tf)^4 + 16c_6h\pi^4b_1c_5^4 \\
& + 128c_6h\pi^4c_1b_1c_5^2 + 64c_6hc_2^2\pi^4b_1c_5^2 + 8c_4c_3U_0\pi^4c_5^3 - 16c_6h\pi^4b_2c_5^4 - 64c_6h\pi^4c_1a_2c_5^2 \\
& - 96c_6h\pi^4c_1b_2c_5^2 + (-128c_6h\pi^4c_1a_2c_5^2 - 64c_4c_3U_0\pi^4c_1c_5 + 128c_6h\pi^4c_1b_2c_5^2 \\
& - 64c_5^4c_6ha_2\pi^4 + 64c_4c_3U_0\pi^4c_2c_5^2 - 16c_4c_3U_0\pi^4c_5^3 + 64c_6h\pi^4b_2c_5^4 + 64c_6hc_2^2\pi^4b_2c_5^2 \\
& - 256c_6h\pi^4c_1b_1c_5^2 - 64c_6hc_2^2\pi^4a_2c_5^2 - 128c_6hc_2^2\pi^4b_1c_5^2 - 32c_6h\pi^4b_1c_5^4)\cos(\pi tf)^2 \\
& - 32c_6hc_1^2a_2\pi^4 - 32c_6hc_1^2b_2\pi^4 - 48c_6hc_2^2\pi^4b_2c_5^2 - 32c_6hc_2^2\pi^4a_2c_5^2 + 32c_4c_3U_0\pi^4c_1c_5)f^4 \\
& + (8c_6hc_2^2\pi^3a_1c_5^3 + 16c_6h\pi^3c_1a_1c_5^3 + 32c_6hc_1^2a_1c_5\pi^3 + (-16c_6hc_2^2\pi^3a_1c_5^3 \\
& - 32c_6h\pi^3c_1a_1c_5^3 - 64c_6hc_1^2a_1c_5\pi^3)\cos(\pi tf)^2)f^3 + (-12c_6hc_1^2b_2c_5^2\pi^2 - 2c_4c_3U_0\pi^2c_2c_5^4 \\
& + 4c_6hc_2^2\pi^2b_1c_5^4 - 4c_6hc_2^2\pi^2b_2c_5^4 + (-32c_6hc_1^2b_1c_5^2\pi^2 - 8c_6hc_2^2\pi^2b_1c_5^4 - 4c_4c_3U_0\pi^2c_1c_5^3 \\
& + 32c_6h\pi^2c_1b_2c_5^4 + 4c_4c_3U_0\pi^2c_2c_5^4 - 16c_6h\pi^2c_1b_1c_5^4 - 32c_5^4c_1c_6ha_2\pi^2 + 16c_6hc_1^2b_2c_5^2\pi^2 \\
& - 16c_6hc_1^2a_2c_5^2\pi^2 + 16c_6hc_2^2\pi^2b_2c_5^4 - 16c_5^4c_6ha_2c_2^2\pi^2)\cos(\pi tf)^2 + (-32c_6h\pi^2c_1b_2c_5^4 \\
& + 16c_6hc_1^2a_2c_5^2\pi^2 - 16c_6hc_1^2b_2c_5^2\pi^2 + 32c_5^4c_1c_6ha_2\pi^2 - 16c_6hc_2^2\pi^2b_2c_5^4 \\
& + 16c_5^4c_6ha_2c_2^2\pi^2)\cos(\pi tf)^4 + 8c_6h\pi^2c_1b_1c_5^4 + 16c_6hc_1^2b_1c_5^2\pi^2 + 2c_4c_3U_0\pi^2c_1c_5^3 \\
& - 8c_6hc_1^2a_2c_5^2\pi^2 - 8c_6h\pi^2c_1b_2c_5^4)f^2 + (-4c_6hc_1^2a_1c_5^3\pi\cos(\pi tf)^2 + 2c_6hc_1^2a_1c_5^3\pi)f \\
& + c_6hc_1^2b_1c_5^4 - c_6hc_1^2b_2c_5^4 + (-4c_6hc_1^2b_2c_5^4 + 4c_6hc_1^2a_2c_5^4)\cos(\pi tf)^4 + (4c_6hc_1^2b_2c_5^4 \\
& - 2c_6hc_1^2b_1c_5^4 - 4c_6hc_1^2a_2c_5^4)\cos(\pi tf)^2) \\
& /((16\pi^4f^4 + 4c_2^2\pi^2f^2 + 8\pi^2c_1f^2 + c_1^2)(16\pi^2f^2 + c_5^2)(4\pi^2f^2 + c_5^2)c_5)
\end{aligned}
\tag{A.2}
$$

mit

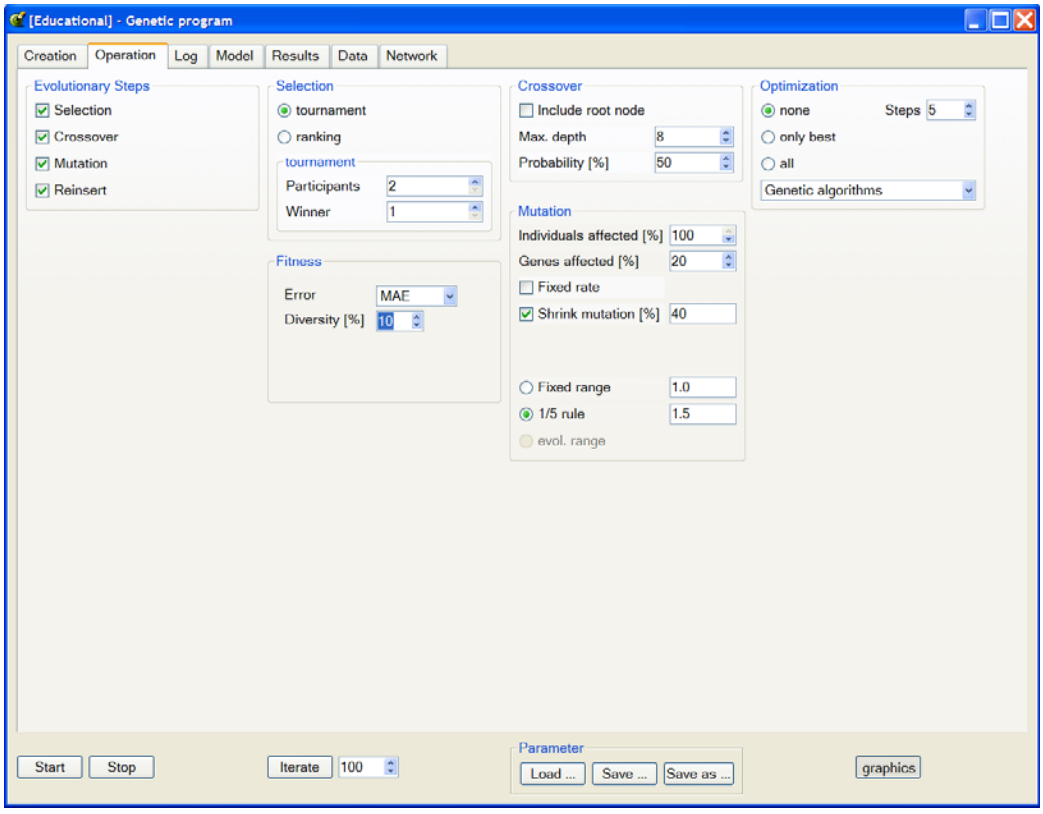
$$
\begin{aligned}
c_1 &= \frac{k_1}{m_P} & c_2 &= \frac{k_3}{m_P} \\
c_3 &= \frac{k_2}{m_P} & c_4 &= \frac{k_6}{m_B} \\
c_5 &= \frac{k_4 + \sigma_2}{m_B} & c_6 &= \frac{\sigma_0}{m_B}
\end{aligned}
$$

```
– <GeneticParameter
DataFile="E:/Daten/robots/GP–Data/Binomial3.txt"
   evolutionPeriod="3"
ClanOperator="1" migration="1" optMode="0" ClanPopSize="1"
ClanMutRate="1" mutationOn="1" clanSelection="1" ramped="0"
mutateGene="2" SelectionType="tournament" migrateGrain="1"
reinsertOn="1" period="1" periodTimeLimit="3630"
   whichMigrant="Best
individual(s)" migratePeriod="10" optAlgo="0"
   mutateInd="100"
mutPara="1.5" ClanTourn="1" tournamentParticipants="2"
clanCrossover="1" n_pop="10" allowedDepth="4"
   clanMutation="1"
maxDepth="3" crossoverOn="1" type="0" mutMode="1"
programType="Genetic␣Programming" Generations="150"
   selectionOn="1"
ClanMaxDepth="1" ClanCrossProb="1" tournamentWinners="1"
crossProb="50" errorType="MAE" diversPerc="0"
   mutateFixed="1"
mutShrink="0"
   modelName="E:/Daten/robots/GP–Result/GPResC_13.xml"
n_clans="10">
  <Operators tanhyp="0" sqrt="1" condition1="1"
     condition2="1" atan2="0" cos="0" sin="0" pow="0"
     arcsin="0" arccos="0" multiply="1" Differentiator="0"
     Integrator="0" exp="0" sinhyp="0" coshyp="0"
     subtract="1" divide="1" value="1" fabs="0" add="1"
     sqr="1" log="0" Memory="0" />
  </GeneticParameter>
```

Listing A.1: Parameterblock der Ergebnisdatei



(a) Voreinstellungen



(b) Ablauf

Abbildung A.1: Benutzeroberfläche

Operator	Arg. (x_m)	Werte (k_s)	Berechnung
Konstante	0	1	k_s
Addition	2	0	$x_{m,1} + x_{m,2}$
Multiplikation	2	0	$x_{m,1} \cdot x_{m,2}$
Subtraktion	2	0	$x_{m,1} - x_{m,2}$
geschützte Division	2	0	$\begin{cases} x_{m,1}/x_{m,2}, & \text{wenn } x_{m,2} > 1e-15 \\ 1, & \text{wenn } x_{m,2} \leqslant 1e-15 \end{cases}$
Potenzfunktion	2	0	$x_{m,1}^{\text{int}(x_{m,2})}$
Exponentialfunktion	1	0	$\exp(x_m)$
Wurzel	1	0	$\sqrt{x_m}$
Quadrat	1	0	x_m^2
Absolutwert	1	0	$\lvert x_m \rvert$
Natürlicher Log.	1	0	$\ln(x_m)$
Logarithmus	1	0	$\log(x_m)$
Sinus	1	0	$\sin(x_m)$
Cosinus	1	0	$\cos(x_m)$
Arccos	1	0	$\arccos(x_m)$
Arcsin	1	0	$\arcsin(x_m)$
Coshyp	1	0	$\cosh(x_m)$
Sinhyp	1	0	$\sinh(x_m)$
Tanhyp	1	0	$\tanh(x_m)$
Integrator	1	1	$k_s += x_m \Delta t$
Differentiator	1	1	$(x_m - k_s)/\Delta t; k_s = x_m$
Verstärkung	1	1	$x_m \cdot k_s$
Verzögerung	1	1	$k_s; k_s = x_m$
PT_1	1	3	f_{PT1}
PT_2	1	4	f_{PT2}
Bedingung Typ 1 (HP)	2	0	$\begin{cases} x_{m,2}, & \text{wenn } x_{m,1} > 0 \\ 0, & \text{wenn } x_{m,1} \leqslant 0 \end{cases}$
Bedingung Typ 2	4	0	$\begin{cases} \arg(3), & \text{wenn } x_{m,1} > x_{m,2} \\ \arg(4), & \text{wenn } x_{m,1} \leqslant x_{m,2} \end{cases}$
Bedingung Typ 3 (TP)	2	0	$\begin{cases} x_{m,2}, & \text{wenn } x_{m,1} < 0 \\ 0, & \text{wenn } x_{m,1} \geqslant 0 \end{cases}$
Bedingung Typ 4	1	2	$\begin{cases} k_{s,1}, & \text{wenn } x_{m,1} > 0 \\ k_{s,2}, & \text{wenn } x_{m,1} \leqslant 0 \end{cases}$

Tabelle A.1: Operatorenliste bei GP

Symbolverzeichnis

Griechische Symbole

α	Winkel zwischen Ortsvektor des Aktors und dem Ansteuerwinkel [mrad]
β	Winkel des Ortsvektors zum Aktor [mrad]
β_L	Luftwiderstandskoeffizient [kg/s]
ϵ	Einheitssprungfunktion
η_L	Kinematische Viskosität der Luft [kg/(m·s)]
ι	Abbruchbedingung
λ_u	Gewichtungsfaktor Stellgröße
λ_y	Gewichtungsfaktor Regelgröße
μ_G	Gleitreibungskoeffizient
μ_H	Haftreibungskoeffizient
μ_ν	Koeffizient der viskosen Reibung
ν_L	Dynamische Viskosität der Luft [m^2/s]
ν_q	Querkontraktionszahl, Poissonzahl
ω	Winkelgeschwindigkeit [mrad/s]
ω_0	Eigenkreisfrequenz des ungedämpften Systems [1/s]
ω_B	Winkelgeschwindigkeit der Piezobeine [mrad/s]
ω_P	Winkelgeschwindigkeit der Roboterplatform [mrad/s]
Φ	Fitness-Funktion
Φ_e	Die Breite des elektrischen Feldes
ρ	Dichte des Mediums [kg/m^3]
σ	Standardabweichung
σ_0	Steifigkeit der Oberfläche [kg/s^2]

σ_1	Dämpfungsfaktor Reibung [kg/s]
σ_2	Korrekturfaktor viskose Reibung [kg/s]
σ_N	Ausbreitung eines RBF-Neurons
τ_1, τ_2	Korrekturfaktoren für die Mutation reeller Zahlen
θ	Ansteuerwinkel (=Kraftwirkrichtung) im lokalen (Roboter-)Koordinatensystem [mrad]
θ_r	Ansteuerwinkel Rotation [mrad]
θ_t	Ansteuerwinkel Translation [mrad]
E_m	Mittlerer absoluter Modellfehler

Lateinische Symbole

$\vec{r}$	Ortsvektor einer Kraft [mm,mm,mm]
A	Systemmatrix
a	Aktivierung eines Neurons
A_S	Anströmfläche [mm^2]
B	Eingangsmatrix
b	Bias eines Neurons
C	Ausgangsmatrix
c_k	Zentrum eines RBF-Neurons
c_w	Luftwiderstandsbeiwert
D	Durchgangsmatrix
D_R	Dämpfungsfaktor
d_{31}	Transversale piezoelektrische Ladungskonstante [C/N;m/V]
E	Elastizitätsmodul [N/mm^2]
e	Regeldifferenz, Regelabweichung
E_m	Absoluter Modellfehler
e_m	Relativer Modellfehler
E_t	Rechenaufwand einer Population
F	Kraft [N]
f	Frequenz [Hz]
F_B	Auf das Piezobein ausgeübte Kraft [N]

F_e	Durch äußere Einflüsse entstehende Kraft, v.a. Zugkraft des Versorgungskabels [N]
F_H	Haftreibungskraft [N]
F_M	Durch Durchbiegung der Piezoaktoren erzeugte Kraft [N]
F_N	Normalkraft [N]
F_P	Resultierende Kraft [N]
F_R	Reibungskraft der Kontaktfläche [N]
F_U	Durch Piezospannung erzeugte Kraft [N]
F_W	Kraft durch Luftwiderstand [N]
G	Übertragungsfunktion
g	Erdbeschleunigung [m/s^2]
g_f	Funktionaler Zusammenhang zwischen Ansteuerbetrag und resultierender Frequenz beim Miniman
G_R	Reglerübertragungsfunktion
G_S	Streckenübertragungsfunktion
G_v	Vorfilter
G_{ST}	Übertragungsfunktion der Störgrößenkompensation
G_{zr}	Übertragungsfunktion des Zustandsregler
J	Trägheitsmoment [$kg\ m^2$]
J_a	Axiales Flächenträgheitsmoment [m^4]
J_B	Trägheitsmoment der Piezobeine [kgm^2]
J_G	Gütefunktion
J_P	Trägheitsmoment der Roboterplatform [kgm^2]
k_1	Koeffizient der mechanischen Verformung [kg/s^2]
k_2	Dämpfungsfaktor Plattformbewegung [kg/s]
k_3	Einfluss der Plattformbewegung
k_4	Dämpfungsfaktor Beinbewegung [kg/s]
k_5	Proportionalitätsfaktor Spannung [1/s]
k_6	Verschiebung Spannungssignal
k_7	Einfluss der Platformbewegung bei Rotation
k_8	Dämpfungsfaktor Beinbewegung bei Rotation [kgm^2/s]

K_R	Reglerverstärkung
k_s	Gespeicherter Zwischenwert (Evolution)
k_u	Wirkungsfaktor der Spannung [N/V]
L	Beobachtungsmatrix
L_0	Charakteristische Länge [mm]
l_P	Länge der Piezoröhrchen [mm]
M	Drehmoment [Nm]
m	Masse [Nm]
m_B	Äquivalente Masse der Piezobeine [kg]
m_p	Masse der Plattform [kg]
N	Normalverteilte Zufallszahl
n_1	Anzahl der Fourierreihenglieder
N_1, N_2	Neuronale Netze
n_2	Steilheit des Spannungssignals
n_3	Anzahl der Aktoren des Roboters
n_b	Anzahl der Bits in einem String
n_d	Anzahl der Datensätze
n_f	Anzahl der Freiheitsgrade pro Roboter
n_m	Anzahl der Messwerte
n_p	Anzahl der Teilnehmer an einem Turnier (Evolution)
n_q	Anzahl der Messbereiche auf der Arbeitsfläche
n_r	Anzahl der Reglerausgangsgrößen pro Roboter
n_s	Anzahl der Stellsignale pro Aktor
n_u	Anzahl Zeitschritte Stellbefehl
n_w	Anzahl der Gewinner eines Turniers (Evolution)
n_y	Anzahl Zeitschritte Regelgröße
n_z	Anzahl der Zustandsgrößen pro Roboter
n_{gene}	Anzahl der Gene einer Lösung
n_{ins}	Anzahl der Inseln (Evolution)
n_{lev}	Anzahl an Ebenen (Baumstruktur)
N_{me}	Anzahl der Perioden, nach denen Meta-Evolution stattfindet

N_{mig}	Anzahl der Perioden, nach denen Migration stattfindet
n_{oper}	Anzahl an mathematischen Operatoren
n_{op}	Häufigkeit eines mathematischen Operatoren in einer Lösung
n_{pop}	Anzahl der Lösungen in einer Population
p	Eingang eines Neurons
p_c	Wahrscheinlichkeit mit der eine Kreuzung (Rekombination) durchgeführt wird
p_m	Wahrscheinlichkeit mit der Mutation durchgeführt wird
p_N	Quantil der Normalverteilung
P_S	Massenschwerpunkt des Roboters [mm,mm,mm]
q	Verschiebungsoperator
Q_B	Beobachtbarkeitsmatrix
Q_S	Steuerbarkeitsmatrix
Q_u	Gewichtungsmatrix der Stellgröße
Q_y	Gewichtungsmatrix der Regelgröße
r_m	Mutationsbereich
r_{div}	Anteil der Diversität
$r_{P,a}$	Außenradius der Piezoröhrchen [mm]
$r_{P,i}$	Innenradius der Piezoröhrchen [mm]
Re	Reynoldszahl
Rg	Rang eines Elements in einer Liste
Rg_{div}	Diversitätsbasierter Rang in einer Liste
Rg_{err}	Fehlerbasierter Rang in einer Liste
T	Abtastzeit,Taktzeit,Periodendauer [s]
t	Zeit [s]
T_1, T_2	Zeitkonstanten [s]
T_c	Prädiktionshorizont der Stellgröße
T_g	Ausgleichszeit [s]
T_N	Nachstellzeit [s]
T_p	Prädiktionshorizont der Regelgröße
T_u	Verzugszeit [s]

T_V	Vorhaltezeit [s]
T_{95}	Einstellzeit [s]
T_{si}	Zeitabstand zwischen zwei Spannungsstufe [s]
T_{st}	Periodendauer der steigenden Flanke [s]
U	Elektrische Spannung [V]
u	Stellbefehl [mm/s]
U_0	Versorgungsspannung [V]
u_a	Ansteuerbetrag
u_H	Steuersignal, Hardwaresignal
u_r	Reglerausgang [mm/s]
U_T	Spannungsverlauf einer Periode [V]
u_w	Sollwert nach Vorfilter
v	Geschwindigkeit [mm/s]
v_0	Anströmgeschwindigkeit [mm/s]
v_B	Geschwindigkeit des Piezobeins [mm/s]
v_P	Geschwindigkeit der Roboterplatform [mm/s]
v_s	Stribeck-Geschwindigkeit [mm/s]
w	Sollwert
w_N	Gewicht einer neuronalen Verbindung
x	Position des Roboters [mm,mm,mm]
x_0	Grundstellung [mm]
x_B	Position des Piezobeins [mm]
x_i	Gemessener Ausgangswert des i-ten Datensatzes
x_m	Modelleingangsgröße
x_P	Momentaner Schwerpunkt (Position) der Roboterplatform [mm]
x_s	Startposition [mm]
x_z	Zustandsgröße
y	Regelgröße, Istzustand des Systems, Istgeschwindigkeit [mm/s]
y_m	Modellausgangsgröße
y_s	Systemausgangsgröße, gemessener Ausgang
y_w	Sollposition [mm]

z	Störgröße	
z_k	Äquivalente Deformation durch mechanischen Kontakt	[mm]
GE	Genetisches Programm mit grammatischer Struktur	
GP	Genetisches Programm mit Baumstruktur	
IM	Insel-Modell ohne Migration	
IMM	Insel-Modell mit Migration	
IMME	Insel-Modell mit Migration und Meta-Evolution	
PM	Panmiktische Population	

Literaturverzeichnis

[1] R. Abiyev and V. Abiyev. Genetic Algorithm Learning Parameters of Neural Systems. In 2nd FAE International Symposium 'Creating the Future', Lefke, TRNC, Türkei, 2002.

[2] E. Alba and J. M. Troya. Genetic Algorithms for Protocol Validation. In PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, pages 870–879, London, UK, 1996. Springer-Verlag.

[3] E. Alba and J. M. Troya. A Survey of Parallel Distributed Genetic Algorithms. Complex., 4(4):31–52, 1999.

[4] D. Andre and J. R. Koza. A parallel implementation of genetic programming that achieves super-linear performance. Inf. Sci., 106(3-4):201–218, 1998.

[5] K. Astrom and B. Wittenmark. Adaptive Control. Addison-Wesley Series in Electrical Engineering: Control Engineering. Addison-Wesley, 1995.

[6] J. Baker. Adaptive selection methods for genetic algorithms. In J. Grefenstette, editor, Proceedings of the 1st Int. Conf. on Genetic Algorithms and their Applications, pages 101–111, Pittsburgh, 1985. Lawrence Erlbaum Associates.

[7] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. Genetic Programming - An Introduction: On the Automatic Evolution of Computer Programs and its Applications. dpunkt - Morgan Kaufmann, Heidelberg - San Francisco, 1997.

[8] T. Bäck. Evolutionary Algorithms in Theory and Practice. Oxford Univ. Press, 1996.

[9] W. Beitz and K. Küttner. Taschenbuch für den Maschinenbau. Springer-Verlag, 18 edition, 1995.

[10] A. Bergander. Control, wear testing and integration of stick-slip micropositioning. PhD thesis, EPFL, Lausanne, Switzerland, 2003.

[11] A. Bethke. Comparison of genetic algorithms and gradient-based optimizers on parallel processors: Efficiency of use of processing capacity. Technical Report 197, University of Michigan, Logic of Computers Group, 1976.

[12] P.-A. Bliman and M. Sorine. Friction modelling by hysterisis operators. application to dahl, stickton, and stribeck effects. In Int. Conference on Models of Hysterisis, Trento, Italien, 1991.

[13] J. Breguet. Actionneurs Stick and Slip Pour Micro-Manipulateurs. PhD thesis, EPFL, Lausanne, Schweiz, 1998.

[14] K. Bryant. Genetic algorithms and the traveling salesman problem. Master's thesis, Harvey Mudd College, 2000.

[15] G. Caprari, P. Balmer, R. Piguet, and R. Siegwart. The Autonomous Micro Robot Alice: a Platform for Scientific and Commercial Applications. 1998.

[16] R. Caruana, L. Eshelman, and J. Schaffer. Representation and Hidden Bias ii: Eliminating Defining Length Bias in Genetic Search via Shuffle Crossover. 11th Int. Conf. on Artificial Intelligence, 1:750–755, 1989.

[17] M. Chhabra, A. Nahar, N. Agrawal, T. Jain, A. Mukerjee, A. Mathad, and S. Chaudhuri. Novel approaches to vision and motion control for robot soccer. 2004.

[18] P. Dahl. A Solid Friction Model. TOR 158(3107-18), The Aerospace Corporation, El Segundo, California, USA, 1968.

[19] C. C. de Wit, H. Olssson, K. Astrom, and P.Lischinsky. A New Model for Control of Systems with Friction. IEEE Transactions on Automatic Control, 40(3), 1995.

[20] K. Deb and R. Agrawal. Simulated binary crossover for continuous search space. Complex Systems, 9:115–148, 1995.

[21] B. Donald, C. Levey, C. McGray, I.Paportny, and D.Rus. An untethered, electrostatic, globally controllable mems micro-robot. Technical report, 2006.

[22] B. Edmonds. Meta-genetic programming: Co-evolving the operators of variation. Technical Report 98-32, Centre for Policy Modelling, Manchester Metropolitan University, Aytoun St., Manchester, M1 3GH. UK, 1998.

[23] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. IEEE Trans. on Evolutionary Computation, 3(2):124–141, 1999.

[24] S. E. Eklund. A diffusion architecture for genetic programming. IT-lyftet, 2002.

[25] R. Estana. Entwicklung und Test eines moirébasierten hochgenauen Positionserfassungssystems. PhD thesis, Universität Karlsruhe, IPR, 2006.

[26] R. Estana. Micron public report. Technical report, UNIKARL, DMS, EPFL, FhG, NTUA, SHU, SSSA, UB, 2006.

[27] G. Felsö. Theory of the motion principle of a piezoelectric driven microrobot. Periodica Polytechnica Ser. El., 44(3):pp. 227–239, 2000.

[28] I. T. U. Ferroelectr. ANSI/IEEE Standard on Piezoelectricity. In ANSI/IEEE Standard 176-1987, 1987.

[29] R. Findeisen and F. Allgöwer. An introduction to nonlinear model predictive control. In Proceedings of the 21st Benelux Meeting on Systems and Control, Veldhoven, 2002.

[30] D. Fogel. Evolving artificial intelligence. PhD thesis, University of California, San Diego, 1992.

[31] L. Fogel, A. Owens, and M. Walsh. Artificial Intelligence through Simulated Evolution. Wiley, New York, 1966.

[32] R. Genov, S. Madhavapeddi, and G. Cauwenberghs. Learning to navigate from limited sensory input: Experiments with the khepera microrobot.

[33] D. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodalfunction optimization. In J. Grefenstette, editor, Proceedings of the 2nd Int. Conf. on Genetic Algorithms, Cambridge, MA, 1987. Lawrence Erlbaum Associates.

[34] M. Golub and D. Jakobovic. A new model of global parallel genetic algorithm. In Proc. 22th Int. Conference ITI'00, Pula, 2000.

[35] J. Gomez. Self adaptation of operator rates in evolutionary algorithms. In GECCO (1), pages 1162–1173, 2004.

[36] J. Grefenstette. Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Vanderbilt University, Computer Science Department, Nashville, 1981.

[37] R. Hinterding, Z. Michalewicz, and A. Eiben. Adaptation in evolutionary computation: A survey. In IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, 1997.

[38] H. Hülsen, S. Garnica, and S. Fatikow. Extended kohonen networks for the pose control of microrobots in an nanohandling station. 2003.

[39] J. Holland. Adaptation in natural and artifiifcial systems. Technical report, University of Michigan Press, Ann Arbor, MI, 1975.

[40] J. Holland. Adaptation in natural and artificial systems. MIT Press, Cambridge, MA, USA, 1992.

[41] M. Hutter. Fitness uniform selection to preserve genetic diversity. Technical report, IDSIA, Manno-Lugano, Schweiz, 2001.

[42] J.Kautsky, N.K.Nichols, and P. Doren. Robust pole assignment in linear state feedback. International Journal of Control, (5):1129–1155, 1985.

[43] D. Karnopp. Computer simulation of stick-slip friction in mechanical dynamic systems. ASME J. of Dynamic Systems and Machine Theory, 1985.

[44] C. Karr. Design of an adaptive fuzzy logic controller using genetic algorithm. In Proc. 4th Int. Conf. on Genetic Algorithms, pages 450–456, 1991.

[45] S. Kay. Fundamentals of Statistical Signal Processing - Estimation Theory. Prentice Hall, Englewood Cliffs, NJ, USA, 1993.

[46] Z. Konfrst. Parallel genetic algorithms: Advances, computing trends, applications and perspectives. In 18th Int. Parallel and Distributed Processing Symposium, 2004.

[47] J. Koza, M. Keane, F. Bennett, J.Yu, W.Mydlowec, and O. Stiffelman. Automatic creation of both the topology and parameters for a robust controller by means of genetic programming. In Proceedings of the 1999 IEEE, Cambridge, MA, 1999.

[48] J. R. Koza. Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA, USA, 1992.

[49] J. R. Koza. Human-competitive applications of genetic programming. Springer-Verlag New York, Inc., pages 663–682, 2003.

[50] S. Loizou and K. Kyriakopoulos. Motion planning of piezoelectrically driven micro-robots via navigation functions. In 13th IEEE Mediterranean Conference on Control and Automation, 2005.

[51] S. Luke and L. Spector. A revised comparison of crossover and mutation in genetic programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, Genetic Programming 1998: Proceedings of the Third Annual Conference, pages 208–213, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Morgan Kaufmann.

[52] H. Lutz and W. Wendt. Taschenbuch der Regelungstechnik. Verlag Harri Deutsch, Frankfurt am Main, 4 edition, 2002.

[53] S. Martel, M. Sherwood, C. Helm, W. G. de Quevedo, T. Fofonoff, R. Dyer, and J. Bevilacqua. Three-legged wireless miniature robots for mass-scale operations at the sub-atomic scale. In ICRA, pages 3423–3428, 2001.

[54] J. Mathieu, S. Martel, L. Yahia, G. Soulez, and G. Beaudoin. Mri systems as a mean of propulsion for a microdevice in blood vessels. 2003.

[55] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin, 1992.

[56] M. Miki, T. Hiroyasu, and M. Kaneka. A parallel genetic algorithm with distributed environment scheme. In H. Arabnia, editor, Proceedings of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications. CSREA Press, 2000.

[57] F. Mondada, E. Franzi, and P. Ienne. Mobile robot miniaturisation: A tool for investigation in control algorithms. In The 3rd International Symposium on Experimental Robotics III, pages 501–513, London, UK, 1994. Springer-Verlag.

[58] K. Narendra and S. Mukhopadhyay. Adaptive control using neural networks and approximate models. In IEEE Transactions on Neural Networks, volume 8 of 3, 1997.

[59] B. Nelson and R. Ramajani. Biomedical micro-robotic system. In 8th Intl. Conf. on Medical Image Computing and Computer Assisted Intervention (MICCAI 2005), Palm Springs, California, USA, 2005.

[60] A. Neubauer. A theoretical analysis of the non-uniform mutation operator for the modified genetic algorithm. In Evolutionary Computation, 1997., IEEE International Conference on, 1997.

[61] 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, 1994.

[62] P. Nordin, W. Banzhaf, and M. Brameier. Evolution of a world model for a miniature robot using genetic programming. Robotics and Autonomous Systems, 25(1-2):105–116, 1998.

[63] J. Page, R. Poli, and W. B. Langdon. Smooth uniform crossover with smooth point mutation in genetic programming: A preliminary study. In R. Poli, P. Nordin, W. B. Langdon, and T. C. Fogarty, editors, Genetic Programming, Proceedings of EuroGP'99, volume 1598, pages 39–49, Goteborg, Sweden, 26-27 1999. Springer-Verlag.

[64] R. Poli and W. B. Langdon. Genetic programming with one-point crossover and point mutation. Technical Report CSRP-97-13, Birmingham, B15 2TT, UK, 15 1997.

[65] J. B. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, and R. A. Watson. Evolutionary techniques in physical robotics, pages 511–523. Morgan Kaufman Publishers Inc., San Francisco, CA, USA, 2002.

[66] I. Rechenberg. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart, 1973.

[67] C. Ryan and M. O'Neill. Grammatical evolution: A steady state approach. In J. R. Koza, editor, Late Breaking Papers at the Genetic Programming 1998 Conference, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Stanford University Bookstore.

[68] K. Santa. Intelligente Regelung von Mikrorobotern in einer automatisierten Mikromontagestation. PhD thesis, Universität Karlsruhe (TH), 1998.

[69] J. Schaffer, R. Caruana, L. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In Proceedings of the 3rd Int. Conf. on Genetic Algorithms, 1989.

[70] F. Schmoeckel. Mobile Mikroroboter im Rasterelektronenmikroskop. PhD thesis, Universität Karlsruhe, IPR, 2004.

[71] H. Schwefel. Numerical Optimization for Computer Models. John Wiley & Sons, Inc., Chichester, UK, 1981.

[72] H. Schwefel. Evolution and Optimum Seeking: The Sixth Generation. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[73] D. Soloway and P. J. Haley. Neural generalized predictive control: A newton-raphson implementation. TM 110244, NASA, February 1997.

[74] L. Sun, M. Li, and Y. Chu. Control system of mobile micro-robot based on dsp. In Proceedings of the IEEE Int. Conf. on Robotics, Intelligent Systems and Signal Processing, 2003.

[75] G. Syswerda. Uniform crossover in genetic algorithms. In Proceedings of the 3rd International Conference on Genetic Algorithms, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[76] M. Szymanski. Entwurf und implementierung einer flexiblen positionsregelung für mikroroboter. Master's thesis, Universität Karlsruhe, IPR, 2003.

[77] R. Tanese. Distributed genetic algorithms. In Proc. of 3rd Int. Conf. On Genetic Algorithms, pages 434–439, 1989.

[78] Z. Tao, T. Dalong, and Z. Jiangbo. Development of a high resolution wireless mobile microrobot.

[79] A. Teller. Evolving programmers: the co-evolution of intelligent recombination operators, volume 2 of Advances in Genetic Programming, pages 45–68. MIT Press, Cambridge, MA, USA, 1996.

[80] M. Tomassini, L. Vanneschi, F. V. Fernandez, and G. G. Gil. Experimental investigation of three distributed genetic programming models. In PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, pages 641–650, London, UK, 2002. Springer-Verlag.

[81] M. Tomassini, L. Vanneschi, F. Fernández, and G. Galeano. A study of diversity in multipopulation genetic programming. In P. Liardet, P. Collet, C. Fonlupt, E. Lutton, and M. Schoenauer, editors, Evolution Artificielle, 6th Int. Conf., Lecture Notes in Computer Science, pages 243–255, Marseilles, France, 2003. Springer. Revised Selected Papers.

[82] A. Tuson and P. Ross. Self-adaption by co-evolution. Technical Report 788, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, UK, 1996.

[83] J. B. Waller. Concepts and methodologies in non-linear model predictive control. Technical report, Process Control Laboratory, Abo Akademi, Abo, Finnland, 2000.

[84] P. Whigham. Grammatically-based genetic programming. In J. P. Rosca, editor, Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, pages 33–41, Tahoe City, California, USA, 1995.

[85] D. Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. Schaffer, editor, Proceedings of the 3rd Int. Conf. on Genetic Algorithms, pages 116–123, San Mateo, California, 1989. Morgan Kaufmann Publishers, Inc.

[86] M. Willis, H. Hiden, P. Marenbach, B. McKay, and G. Montague. Genetic programming: An introduction and survey of applications. In 2nd Int. Conf. Genetics Algorithms in Engineering Systems: Innovations and Applications, number 446 in IEE Conference Publications, pages 314–319, Glasgow, UK, 1997.

[87] H. Woern, F. Schmoeckel, A. Buerkle, J. Samitier, M. PuigVidal, S. J. U. Simu, J. Meyer, and M. Biehl. From decimeter- to centimeter sized mobile microrobots: the development of the miniman system. In Proceedings of the SPIE. Microrobotics and Microassembly III, volume 4568, pages 175–186, 2001.

[88] J. Ziegler. Kalman-filterung für sensorsysteme mobiler mikroroboter. Master's thesis, Universität Karlsruhe, IPR, 2003.