Evolutionäre Algorithmen zur Optimierung von Modellen für laufende Roboter

Doktorarbeit

Matthias Hebbel

Genehmigte Dissertation zur
Erlangung des akademischen Grades eines Doktors an der
Fakultät für Elektrotechnik und Informationstechnik
Technische Universität Dortmund

Abteilung Informationstechnik
Institut für Roboterforschung

2009

Danksagung

Mein Dank gilt allen, die es mir ermöglicht haben, diese Doktorarbeit zu verfassen. Besonderer Dank gilt meinem Doktorvater Prof. Dr.-Ing. Uwe Schwiegelshohn für die Unterstützung meiner Forschungsarbeiten und das große Vertrauen, das er mir jederzeit entgegengebracht hat. Außerdem danke ich Prof. Dr. Hans-Dieter Burkhard, der als Vizepräsident der RoboCup Federation maßgeblich an der Popularität von Roboterfußball als Forschungsdisziplin an deutschen Universitäten beteiligt ist.

Weiterhin möchte ich mich bei den ehemaligen Arbeitskollegen des Instituts für Roboterforschung der Technischen Universität Dortmund bedanken. Allen voran Walter Nisticó, mit dem es sehr viel Spaß gemacht hat, unsere Roboterfußball-Teams zu betreuen, aber auch den weiteren ehemaligen Kollegen Dr.-Ing. Ingo Dahm, Lars Schley, Stefan Czarnetzki, Joachim Lepping und Christian Grimme sowie zahlreichen Studenten, die als studentische Hilfskräfte oder als Teilnehmer von Projektgruppen weit über das übliche Maß hinaus für das Roboterfußball-Team gearbeitet haben. Weiterer Dank gilt außerdem Dr.-Ing. Thomas Röfer, Tim Laue, Matthias Jüngel, Dr. Jan Hoffmann und Daniel Göhring, die als Mitstreiter im GermanTeam oder den BreDoBrothers immer wieder für fruchtreiche Diskussionen gesorgt haben.

Abschließend möchte ich mich auch bei meinen Eltern Ursula und Prof. Dr. Hartmut Hebbel bedanken, die es mir ermöglicht haben, diese Laufbahn einzuschlagen. Meinem Vater gilt darüber hinaus noch besonderer Dank für das geduldige Diskutieren von mathematischen Problemstellungen.

Erstgutachter:	Prof. Dr.-Ing. Uwe Schwiegelshohn
Zweitgutachter:	Prof. Dr. Hans-Dieter Burkhard
Tag der Prüfung:	30. Oktober 2009

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Arbeit befasst sich primär mit der Modellierung und Optimierung von Laufbewegungen für zwei- und vierbeinige Roboter. Laufende Roboter können sich sowohl in der künstlich geschaffenen Umgebung des Menschen als auch in naturbelassener Umgebung besser und flexibler bewegen als rollende Roboter. Von Menschen strukturierte Umgebungen wie Wohnungen oder Häuser, sind auf den Körper des Menschen angepasst. Treppen oder auch kleinere Stufen sind für Menschen leicht zu überwindende Hindernisse, für Roboter hingegen sind sie sehr schwierig zu erklimmen. Radgetriebene Roboter können zwar äußerst effektiv auf glatten Flächen, wie z. B. in Lagerhallen, agieren, Stufen können sie jedoch nur mit weiteren Hilfsmitteln überwinden. Ebenso können bereits kleine Steigungen zu unüberwindbaren Hindernissen werden, laufende Roboter hingegen können diese Hindernisse wie ein Mensch überwinden. Eine geeignete Perzeption vorausgesetzt, können laufende Roboter sogar Treppen wie ein Mensch steigen [11].

Noch deutlicher wird die Überlegenheit der Fortbewegung auf Beinen gegenüber Rollen auf weichem oder zerklüftetem Untergrund. Auf weichem Untergrund, wie z. B. Sand, können Räder schnell durchdrehen oder einsinken, die Bewegung auf Beinen ist hier viel effektiver und unter Umständen auch energetisch effizienter [57]. Auf unebenem oder zerklüftetem Boden ist eine Fortbewegung auf Rädern gegebenenfalls gar nicht möglich, wenn Hindernisse wie z. B. Löcher oder Felsen den Weg blockieren. Das Laufen hingegen erfordert keinen kontinuierlichen Bodenkontakt, so dass Löcher im Boden oder kleinere Felsen einfach überstiegen werden können.

Der Fokus dieser Arbeit liegt neben der Modellierung von Laufbewegungen auf dem selbständigen Laufenlernen der Roboter, so dass der eigentliche Lernprozess vollständig autonom von dem Roboter durchgeführt werden kann. Ein weiterer wichtiger Aspekt, der in dieser Arbeit untersucht wird, ist die Reduktion der Lerndauer und der dazu nötigen Trainingsläufe.

Zur Förderung der Forschung auf dem Gebiet (laufender) autonomer Roboter wurde das Fußballspielen von Robotern als Benchmark definiert. In dem so genannten *RoboCup* [33]

sind mittlerweile über 300 Teams aktiv, die Fußball spielende Roboter entwickeln und programmieren.

1.1 Roboterfußball

Nachdem es 1996 dem Supercomputer *DeepBlue* von IBM gelang, gegen den amtierenden Schachweltmeister Garri Kasparow zu gewinnen, wurde klar, dass Computerschach nicht mehr als Benchmark für die Künstliche Intelligenz dienen konnte. Durch die gestiegene Rechen- und Speicherleistung von Computern wurden Verfahren entwickelt, die zahlreiche zukünftige Spielzüge simulieren können und schließlich den vielversprechendsten Zug auswählen. Dieses Verfahren hat jedoch mit Künstlicher Intelligenz, für die Computerschach lange als Messlatte galt, nicht viel gemein. Daher wurde eine neue Messlatte für die Künstliche Intelligenz gesucht und schließlich das Fußballspiel von autonomen Robotern als neuer Benchmark auserkoren.

Zum Vergleich der erreichten Leistungen der zahlreichen Forschergruppen, die sich mit Roboterfußball beschäftigen, finden seit 1997 jährlich Roboterfußballturniere statt. In dem Verband *RoboCup* wird in zahlreichen verschiedenen Ligen Roboterfußball gespielt. Jede Liga hat dabei einen eigenen Forschungsschwerpunkt. So gibt es z. B. Simulationsligen, in denen das gesamte Roboterteam simuliert wird, Ligen mit fahrenden Robotern und Ligen mit laufenden Robotern. Die beiden in dieser Arbeit genutzten laufenden Roboter nahmen an der so genannten *Four-Legged League* und der *Humanoid League* teil.

Prinzipiell sind die größten zu lösenden Probleme im Roboterfußball, die aber nicht zwangsläufig in allen Ligen anzutreffen sind:

- Erkennung von Objekten auf dem Spielfeld (Sensorik)
- Lokalisierung der eigenen Position (Selbstlokalisierung)
- Modellierung von spielrelevanten Objekten (z. B. des Balles)
- Fortbewegung des Roboters (Aktorik)
- Fertigkeiten auf unterer Verhaltensebene (z. B. Ball schießen)
- Planung von Spielzügen
- Koordination der Handlungen und Spielzüge im Team

Klassischerweise wurden die meisten der obigen Punkte durch reine Programmierung von entsprechenden Algorithmen gelöst. Im Laufe der Zeit wurde erkannt, dass häufig Lösungen nicht mehr nur von Hand entwickelt werden können, so dass immer mehr maschinelle Lernverfahren zur Unterstützung in der Entwicklung eingesetzt wurden. Mittlerweile werden in allen oben genannten Problemen von einigen Forschergruppen Lernverfahren eingesetzt [6].

1.2 Lernende Roboter

In den letzten Jahren wurde sehr viel an selbständig lernenden autonomen Robotern geforscht, so dass diesem Forschungsbereich ein eigener Name *Evolutionary Robotics* zuteil wurde. Das Ziel von Evolutionary Robotics ist die Entwicklung von Methoden zum automatischen Generieren von intelligenten Robotersteuerungen, ohne dass eine direkte Programmierung von Menschen nötig ist [43].

Im Folgenden wird ein kleiner Überblick über lernende Roboter in der Domäne RoboCup gegeben. Das Erlernen von Fertigkeiten erstreckt sich dabei über viele Bereiche. Peter Stone hat den Begriff *Layered Learning* geprägt [64], der aussagt, dass Lernverfahren auf verschiedenen Ebenen sinnvoll eingesetzt werden können und das Lernen auf oberen Ebenen, wie z. B. der Verhaltenssteuerung von den unteren, zuvor erlernten Ebenen, wie z. B. dem Laufen profitieren. Das Erlernen von Laufbewegungen wird von zahlreichen Teams im RoboCup, insbesondere in der Four-Legged League, genutzt und wird später in Kapitel 2 näher betrachtet.

Beispielsweise wurde auf unterer Verhaltensebene für Aibo Roboter gelernt, den Ball effektiv zu greifen. Dabei klemmt der Roboter den Ball zwischen Kopf und Boden ein und kann den Ball so sicher führen und präzise einen Schuss vorbereiten. Dieses einfache Verhalten kann beim Roboterfußball spielentscheidend sein. Das Team, dessen Roboter den Ball zuerst unter Kontrolle bringt, hat einen großen Vorteil. Insbesondere ab dem Jahr 2005 wurde dieser Spielzug dominant, da mit dem „gegriffenen“ Ball dem erkannten Gegner ausgewichen werden konnte. Das reine Programmieren dieser Bewegung von Hand ist kaum durchführbar, da zahlreiche Parameter, wie z. B. die Distanz zwischen Roboter und Ball, ab der der Kopf gesenkt wird, einzustellen sind. Außerdem sind Anpassungen an den verwendeten Teppich oder geänderte Laufarten des Roboters durchzuführen. Neben dem möglichst zuverlässigen Greifen ist auch die Zeitdauer dessen sehr wichtig, da auch hier das schnellere Team klar im Vorteil ist. Daher wurde von dem RoboCup Team *UT Austin Villa* eine durch fünf Parameter beschriebene Greifbewegung aufgestellt, die von Robotern autonom erlernt wurde [10].

Die RoboCup *Simulation League* bietet ein großes Spektrum zum Lernen von Verhaltensmustern, da hier keine Roboter durch die zahlreichen Trainingsläufe verschleißen und Situationen einfach mehrfach nachgespielt werden können. Ein Team simulierter Roboter kann z. B. selbständig die Strategie erlernen, den Ball möglichst lange unter Kontrolle zu halten und dabei ein vorgegebenes Areal nicht zu verlassen. Gleichzeitig versuchen gegnerische Spieler, den Ball unter ihre Kontrolle zu bringen. Das Verhalten wird dabei aus mehreren einfachen vorgegebenen Aktionen, wie z. B. den Ball zu einem Mitspieler passen oder sich mit dem Ball möglichst weit von den Angreifern entfernen, zusammengesetzt. Die erlernte Strategie konnte sich dabei gegen alle handgeschriebenen Verhaltensmuster durchsetzen [63].

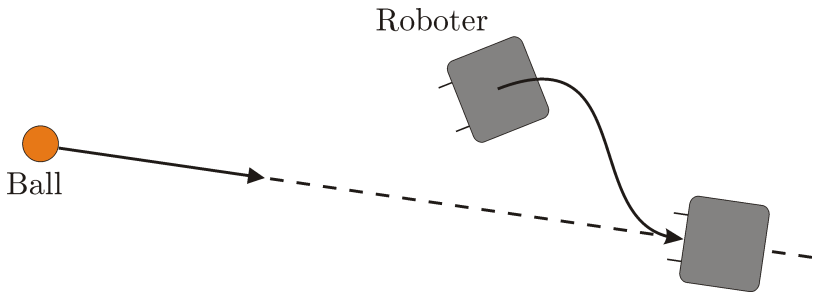


Abbildung 1.1: Der Roboter fängt einen rollenden Ball ab.

Ebenfalls werden in der *Midsize League*, einer Liga mit realen fahrenden autonomen Robotern, häufig Lernverfahren eingesetzt, um eine Spielstrategie zu optimieren. Beispielsweise kann ein Roboter einen Spielzug erlernen, indem er einen rollenden Ball möglichst effizient abfängt. Diese Situation ist in Abbildung 1.1 dargestellt. Um den Ball unter Kontrolle zu bringen, muss der Roboter auf eine Linie mit dem Ball fahren, muss ihn mit seiner Frontseite berühren und gleichzeitig eine annähernd gleiche Geschwindigkeit wie der Ball einnehmen. Zur Reduktion der nötigen realen Trainingsläufe wurde die Strategie zunächst in einem Robotersimulator erlernt und anschließend erfolgreich auf den realen Roboter angepasst [39].

Das Lernen kann auch auf kognitiver Ebene eingesetzt werden, indem ein Roboter die für die Bildverarbeitung nötige Farbzuordnung lernt. In der Four-Legged League werden die spielrelevanten Objekte, wie z. B. der Ball oder die Tore durch eindeutige Farben gekennzeichnet. Die Bildverarbeitung nutzt zur Erkennung dieser Objekte Farbklassen, denen alle Farbtöne der entsprechenden Objekte zugeordnet sind. Ein Ball ist z. B. orange. Je nach Lichteinfall erscheint seine Farbe mal heller und mal dunkler. Diese möglichen Farbtöne müssen nun alle der Farbklasse Orange zugeordnet werden, damit der Roboter in möglichst jeder Position den Ball erkennen kann. Dabei muss die Farbklasse möglichst alle auftretenden Farbtöne enthalten, darf aber gleichzeitig nicht Farbtöne anderer Objekte, wie z. B. das Rot anderer Roboter enthalten, da diese ansonsten möglicherweise als Ball erkannt werden könnten. Diese Farbzuordnung händisch durchzuführen ist sehr zeitaufwändig und muss bei jeder Änderung des Lichteinfalls angepasst werden. Daher wurde ein Verfahren entwickelt, mit dem der Roboter selbständig die Farbklassen erlernt und während eines Roboterfußballspiels mögliche Änderungen des Lichteinfalls erkennt und autonom die Farbklassen anpassen kann [60].

Maschinelle Lernverfahren werden aber nicht nur zur Optimierung der Steuerung der Roboter eingesetzt, sondern können ebenfalls zur Designphase des Roboters unterstützen. Es wurde gezeigt, dass durch geeignete Lernverfahren sogar von Grund auf ein Roboter aus zur Verfügung stehenden Bauteilen, wie z. B. Verbindungselementen oder Antrieben erzeugt werden kann [37].

1.3 Struktur der Arbeit

In dieser Arbeit werden zunächst in Kapitel 2 parametrisierte Laufmodelle für zwei verschiedene Robotertypen hergeleitet. Diese Modelle können Laufbewegungen erzeugen, die jedoch in ihrer Effektivität stark von den Parametern des Modells abhängen. Ziel der Arbeit ist, Verfahren zu entwickeln und zu untersuchen, mit denen die Parameter des Modells, die zu einem möglichst schnellen Laufen führen, selbständig von den Robotern erlernt werden können. Evolutionäre Algorithmen werden als Grundlage für das Erlernen der Parameter verwendet und werden in Kapitel 3 zusammen mit modellgestützten Ansätzen näher erläutert.

Für den Vergleich und die Analyse verschiedener Lernverfahren sind sehr viele Testläufe der Roboter nötig. Da dies zu einem sehr hohen Verschleiß des Roboters führt und auch sehr zeitaufwändig ist, wird in dieser Arbeit teilweise ein physikalischer Robotersimulator genutzt, der in Kapitel 4 so optimiert wird, dass er sich möglichst ähnlich zu einem realen Roboter verhält.

In Kapitel 5 werden Verfahren vorgestellt, mit denen vier- und zweibeinige Roboter selbständig Laufen lernen können. Es werden die Ergebnisse der Lernverfahren für die benutzten Robotermodelle präsentiert und aufgezeigt, welche Lernverfahren sich am besten für die Optimierung des Laufens eignen. Abschließend werden in Kapitel 6 die Ergebnisse dieser Arbeit zusammengefasst.

Kapitel 2

Modellierung von Laufbewegungen

In diesem Kapitel werden parametrisierte Modelle hergeleitet, die Laufbewegungen für zwei Arten von Robotern generieren. Die Modelle beruhen dabei auf einem Abfahren von vorgegebenen Trajektorien, die mittels inverser Kinematik in Bewegungen der Gelenke umgerechnet werden. Bei den Modellen liegt die Einfachheit der Berechnung im Vordergrund. Es wird nicht auf Modelle eingegangen, die dynamisch, abhängig von Stabilitätskriterien wie z. B. dem weit verbreiteten Zero-Moment-Point [67] oder Foot-Rotation-Indicator [14], die Platzierung der Füße berechnen.

Das erste Modell wird für den vierbeinigen Roboter Sony Aibo vorgestellt.

2.1 Sony Aibo

Der Sony Aibo ist ein programmierbarer vierbeiniger Roboter, der erstmals 1999 öffentlich zum Verkauf angeboten wurde. Im Laufe der Zeit wurden verschiedene Modelle herausgebracht. In dieser Arbeit wurde ausschließlich das letzte Modell, der so genannte Aibo ERS 7 benutzt.

Der ERS 7 (dargestellt in Abbildung 2.1) ist ein autonomer Roboter, der durch die Herausgabe eines SDK's von Sony in der Programmiersprache C++ frei programmierbar ist. Der wichtigste Sensor, mit dem der Roboter seine Umwelt wahrnehmen kann, ist eine CMOS Kamera mit einer Auflösung von 208×160 Pixeln, daneben stehen noch Berührungs-, Beschleunigungs- und Abstandssensoren zur Verfügung. Jedes Bein besitzt drei Gelenke, die über Servomotoren mit Positionsmessung angetrieben werden. Dank seines 64-Bit MIPS Mikroprozessors mit einer Taktrate von 576 MHz kann der Roboter alle kognitiven und reaktiven Berechnungen in Echtzeit ausführen. Zur Kommunikation mit anderen Robotern oder PCs verfügt der ERS 7 über eine WLAN 802.11 Schnittstelle. Das auszuführende Programm wird auf einen Memorystick geschrieben und schließlich vom Roboter ausgeführt.

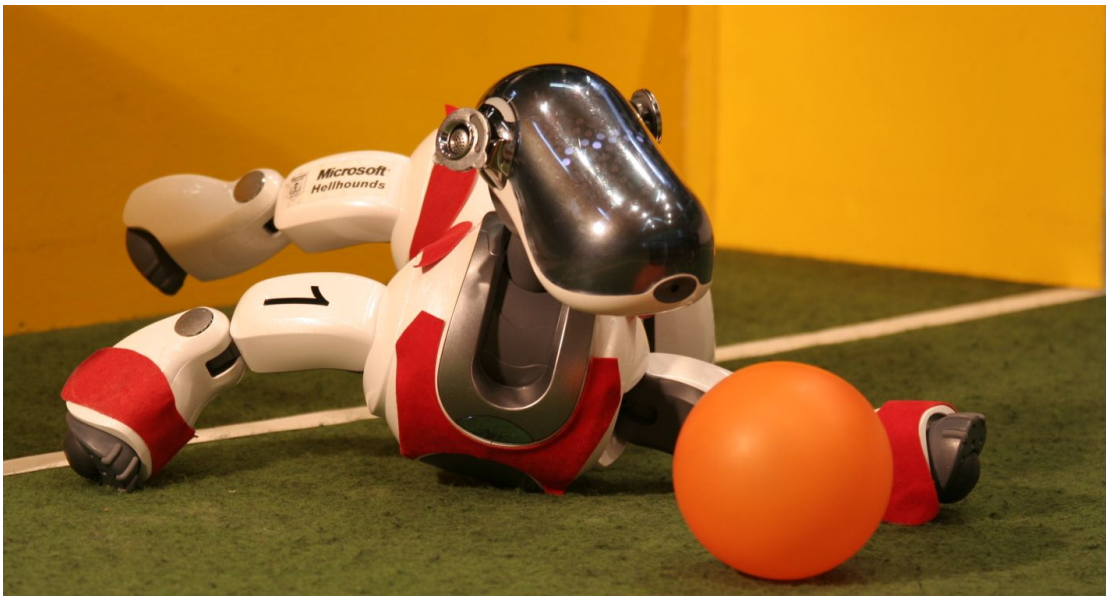


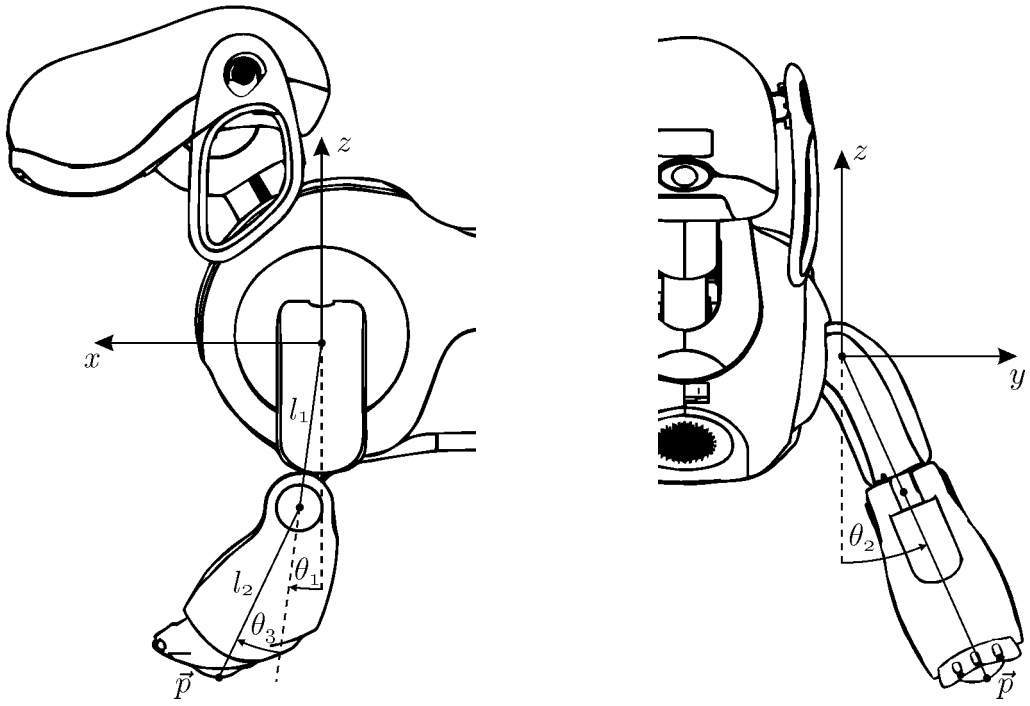
Abbildung 2.1: Ein Sony Aibo ERS 7 als Torwart beim Roboterfußball

2.1.1 Kinematik

Wie schon erwähnt, wird die Bewegung der Beine durch Trajektorien beschrieben. Der Fußpunkt (der unterste Punkt des Beines) wird immer auf der Trajektorie gehalten, die Bewegung der Gelenke des Beines ergeben sich dann automatisch. Bei dem Aibo ERS 7 wird alle 8 ms eine neue Sollposition des Fußes berechnet und die entsprechenden einzunehmenden Gelenkwinkel an die Motoren gesendet. Integrierte PID Regler überwachen das Anfahren der Position. Die Gelenkpositionen müssen also mit einer Frequenz von 125 Hz berechnet werden. Aus dieser hohen Frequenz ergibt sich die Forderung nach einer möglichst effizienten Berechnung.

Zum Abfahren von Trajektorien ist es notwendig, eine Zuordnungsfunktion zu kennen, die für einen Punkt $\vec{p} = (x\ y\ z)^T$ im Raum die entsprechenden Gelenkwinkel $\vec{\theta} = (\theta_1\ \theta_2\ \theta_3)^T$ berechnet. Diese Transformation wird als *inverse Kinematik* bezeichnet. Zur Herleitung der inversen Kinematik wird zunächst die *Vorwärtskinematik* beschrieben, die aus gegebenen Gelenkwinkeln den resultierenden Fußpunkt im Raum ausrechnet.

Die Lage des Koordinatensystems, der Gelenkwinkel und die entsprechenden Längen der Beine sind in der Abbildung 2.2 gezeigt und an die Herleitung in dem Teamreports des GermanTeams von 2005 angelehnt [51].



(a) Ansicht in der x-z–Ebene (b) Ansicht in der y-z–Ebene

Abbildung 2.2: Bemaßung und Lage des Koordinatensystems der Beine

Vorwärtskinematik

Die Lage des Punktes $\vec{p}$ wird mit Hilfe von Koordinatensystemtransformationen bestimmt. Zunächst liegt das Ursprungskoordinatensystem in der Schulter des Roboters. Die x-Achse zeigt nach vorne, die y-Achse zu der linken Seite und die z-Achse nach oben. Die Lagen und Richtungen des Ursprungskoordinatensystems und der nachfolgend verwendeten Rotationswinkel sind in Abbildung 2.2 dargestellt. Eine Rotation mit Drehachse n um dem Winkel θ entgegen dem Uhrzeigersinn wird mit $\mathrm{Rot}(n, \theta)$ bezeichnet, eine Translation entlang der Achse n mit der Länge d mit $\mathrm{Trans}(n, d)$.

Die Transformation setzt sich aus fünf Einzeltransformationen zusammen, die in ihrer Reihenfolge durch die kinematische Kette der einzelnen Gelenke definiert sind:

1. Rotation um die y-Achse um den Winkel $-\theta_1$, $\mathrm{Rot}(y, -\theta_1)$
2. Rotation um die x-Achse um den Winkel θ_2, $\mathrm{Rot}(x, \theta_2)$
3. Translation entlang der negativen z-Achse mit der Länge l_1, $\mathrm{Trans}(z, -l_1)$
4. Rotation um die y-Achse um den Winkel $-\theta_3$, $\mathrm{Rot}(y, -\theta_3)$
5. Translation entlang der z-Achse mit der Länge l_2, $\mathrm{Trans}(z, -l_2)$

Die Gesamttransformation ist daher definiert durch

$$\vec{p} = (x\ y\ z)^T = \mathrm{Rot}(y, -\theta_1) \cdot \mathrm{Rot}(x, \theta_2) \cdot \mathrm{Trans}(z, -l_1) \cdot \mathrm{Rot}(y, -\theta_3) \cdot \mathrm{Trans}(z, -l_2). \quad (2.1)$$

Durch die Einführung von homogenen Koordinaten lässt sich die Gleichung multiplikativ aufstellen und der Fußpunkt $\vec{p}$ relativ zur Schulter ist gegeben durch

$$\begin{aligned}
\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} &= \begin{pmatrix} \cos\theta_1 & 0 & -\sin\theta_1 & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta_1 & 0 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_2 & -\sin\theta_2 & 0 \\ 0 & \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -l_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \\
&\quad \begin{pmatrix} \cos\theta_3 & 0 & -\sin\theta_3 & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta_3 & 0 & \cos\theta_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -l_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} l_1 \sin\theta_1 \cos\theta_2 + l_2 \cos\theta_1 \sin\theta_3 + l_2 \sin\theta_1 \cos\theta_2 \cos\theta_3 \\ l_1 \sin\theta_2 + l_2 \sin\theta_2 \cos\theta_3 \\ -l_1 \cos\theta_1 \cos\theta_2 + l_2 \sin\theta_1 \sin\theta_3 - l_2 \cos\theta_1 \cos\theta_2 \cos\theta_3 \\ 1 \end{pmatrix}. \quad (2.2)
\end{aligned}$$

Hierbei sei beachtet, dass es sich bei dem Koordinatensystem um ein Rechtssystem handelt und die Matrizen zur Drehung des Koordinatensystems durch die Transponierte der Rotationsmatrix für Punkte gegeben ist.

Inverse Kinematik

Die inverse Kinematik ist gegeben durch die Umkehrfunktion der Vorwärtskinematik und berechnet damit für einen gegebenen Punkt $\vec{p}$ im Raum die zugehörigen Gelenkwinkel $\vec{\theta}$. Nur für bestimmte Fälle kann die inverse Kinematik durch geometrische Berechnungen geschlossen gelöst werden. Dies ist nur bei Robotern mit wenigen Gelenken und nur in bestimmten Anordnungen der Fall. Ansonsten kann die inverse Kinematik nur über Approximationsverfahren genähert werden, wie z. B. über die Invertierung der Jacobi-Matrix [38]. Die Inverse der Jacobi-Matrix kann, soweit überhaupt existent, nur näherungsweise gelöst werden und ist zudem sehr rechenaufwändig.

Für dem Aibo lässt sich die inverse Kinematik geometrisch lösen. In der Herleitung wird davon ausgegangen, dass der Punkt $\vec{p}$ innerhalb des erreichbaren Arbeitsraumes liegt. Die Implementierung der inversen Kinematik hingegen fängt diese Fälle ab und berechnet gegebenenfalls den Punkt $\vec{p'}$ mit dem kleinsten Abstand zu $\vec{p}$.

Zunächst wird der Kniegelenkswinkel θ_3 betrachtet. Dieser Winkel definiert eindeutig die Länge $d = \sqrt{x^2 + y^2 + z^2}$ der Strecke von der Schulter zu der Fußposition. Mit Hilfe des

Kosinussatzes gilt dann die Beziehung

$$\begin{aligned}\cos(\pi - \theta_3) &= \frac{l_1^2 + l_2^2 - d^2}{2l_1 l_2} \\ \cos\theta_3 &= \frac{d^2 - l_1^2 - l_2^2}{2l_1 l_2} \\ \theta_3 &= \pm \arccos\left(\frac{d^2 - l_1^2 - l_2^2}{2l_1 l_2}\right). \end{aligned} \tag{2.3}$$

Diese Gleichung liefert zwei mögliche Lösungen, anschaulich bedeutet dies, dass das Kniegelenk in beide Richtungen eingeknickt werden kann. Da bei dem Aibo die Bewegung in die gemäß Abbildung 2.2 positive Richtung (also mit dem Uhrzeigersinn) einen wesentlich größeren Winkel abdecken kann, wird immer nur mit dem positiven Ergebnis gerechnet, die endgültige Gleichung lautet damit

$$\theta_3 = \arccos\left(\frac{x^2 + y^2 + z^2 - l_1^2 - l_2^2}{2l_1 l_2}\right).$$

Durch Kenntnis von θ_3 lässt sich der abspreizende Schultergelenkswinkel θ_2 berechnen, da laut Gleichung 2.2

$$y = l_1 \sin\theta_2 + l_2 \sin\theta_2 \cos\theta_3$$

gilt und nur θ_2 unbekannt ist. Da durch die Bauform des Roboters $|\theta_2| \leq \frac{\pi}{2}$ garantiert ist, ist der Winkel θ_2 damit eindeutig definiert durch

$$\theta_2 = \arcsin\left(\frac{y}{l_1 + l_2 \cos\theta_3}\right).$$

Die Berechnung von θ_1 kann nun mittels der ersten Zeile aus Gleichung 2.2 erfolgen, es gilt

$$\begin{aligned} x &= l_1 \sin\theta_1 \cos\theta_2 + l_2 \cos\theta_1 \sin\theta_3 + l_2 \sin\theta_1 \cos\theta_2 \cos\theta_3 \\ &= \cos\theta_1 \left(l_2 \sin\theta_3\right) + \sin\theta_1 \left(l_1 \cos\theta_2 + l_2 \cos\theta_2 \cos\theta_3\right) \\ &= a\cos\theta_1 + b\sin\theta_1 \end{aligned} \tag{2.4}$$

mit den Hilfsvariablen

$$\begin{aligned} a &:= l_2 \sin\theta_3 \\ b &:= l_1 \cos\theta_2 + l_2 \cos\theta_2 \cos\theta_3. \end{aligned}$$

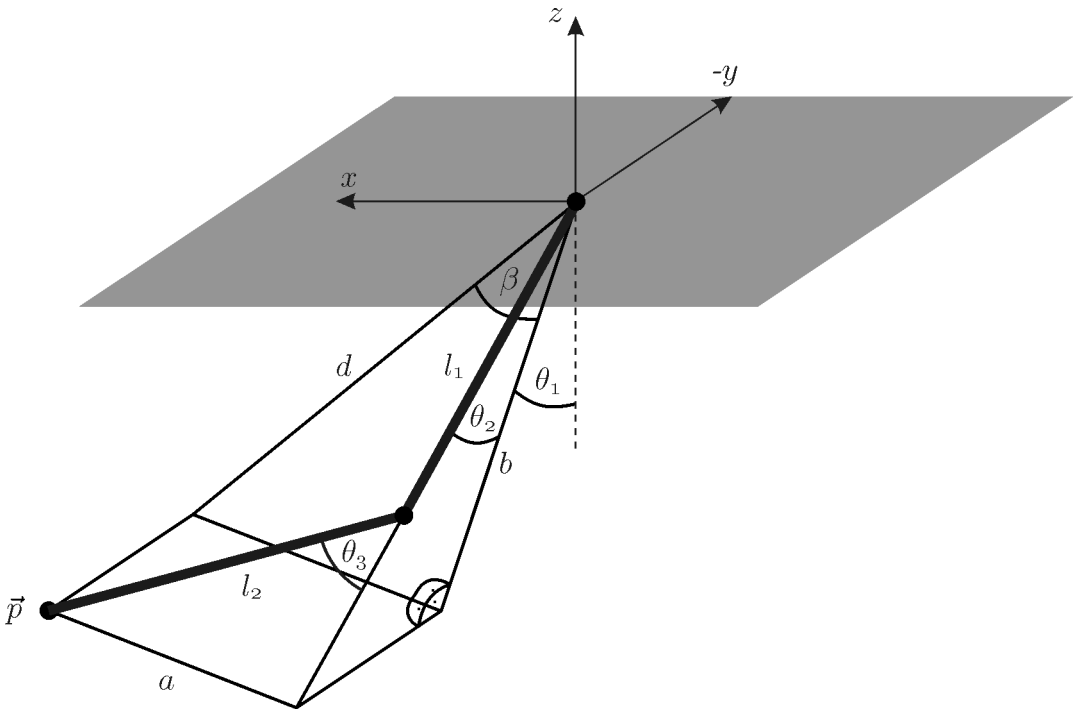


Abbildung 2.3: Schematische Darstellung des Roboterbeines im Raum

Die geometrische Interpretation von a und b ist in Abbildung 2.3 veranschaulicht. Basierend auf dieser Abbildung können die folgenden weiteren Beziehungen aufgestellt werden[1]:

$$\begin{aligned}\beta &= \arctan\left(\frac{a}{b}\right)\\ d &= \sqrt{a^2+b^2}\\ a &= d\sin\beta\\ b &= d\cos\beta.\end{aligned}$$

Gleichung 2.4 lässt sich damit umschreiben zu

$$\begin{aligned}x &= d\sin\beta\cos\theta_1 + d\cos\beta\sin\theta_1\\ &= d(\sin\beta\cos\theta_1 + \cos\beta\sin\theta_1)\\ &= d\sin(\theta_1+\beta).\end{aligned} \tag{2.5}$$

Die Berechnung für z aus Gleichung 2.2 wird nun herangezogen und analog zu den obigen

[1] In dieser Arbeit wird für den Arcustangens keine Unterscheidung bezüglich seines Vorzeichens gemacht, da sich dieses durch eine Fallunterscheidung zurückrechnen lässt

Umformungen für x vereinfacht:

$$\begin{aligned} z &= -l_1 \cos\theta_1 \cos\theta_2 + l_2 \sin\theta_1 \sin\theta_3 - l_2 \cos\theta_1 \cos\theta_2 \cos\theta_3 \\ &= \sin\theta_1 (l_2 \sin\theta_3) - \cos\theta_1 (l_1 \cos\theta_2 + l_2 \cos\theta_2 \cos\theta_3) \\ &= a \sin\theta_1 - b \cos\theta_1 \\ &= d(\sin\beta \sin\theta_1 - \cos\beta \cos\theta_1) \\ &= -d \cos(\theta_1 + \beta). \end{aligned} \tag{2.6}$$

Aus Gleichung 2.5 und Gleichung 2.6 ergibt sich damit

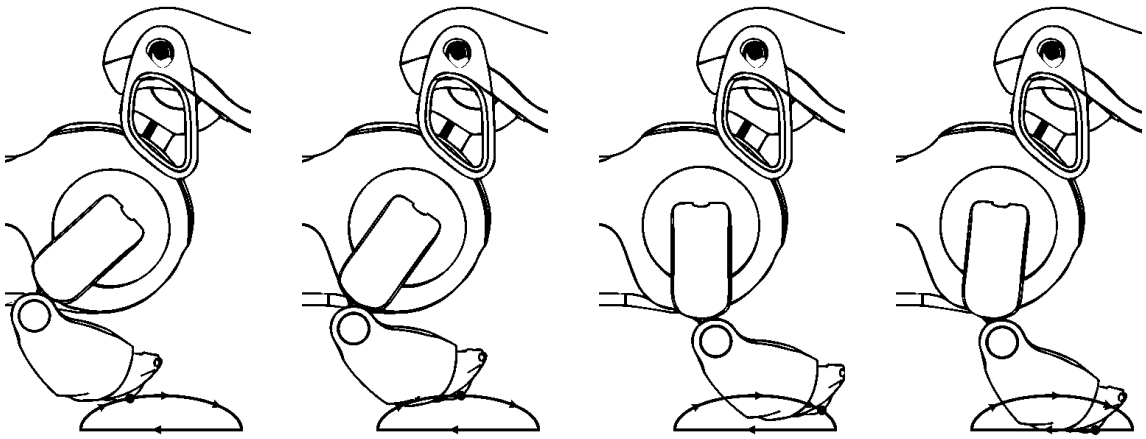
$$\begin{aligned} \theta_1 + \beta &= -\arctan\left(\frac{x}{z}\right) \\ \theta_1 &= -\arctan\left(\frac{x}{z}\right) - \beta \\ \theta_1 &= -\arctan\left(\frac{x}{z}\right) - \arctan\left(\frac{l_2 \sin\theta_3}{l_1 \cos\theta_2 + l_2 \cos\theta_2 \cos\theta_3}\right). \end{aligned} \tag{2.7}$$

Damit ist die inverse Kinematik für den Aibo ERS 7 bekannt, so dass mit den hergeleiteten Gleichungen die Gelenkwinkel in Abhängigkeit eines anzufahrenden Fußpunktes berechnet werden können.

2.1.2 Bahnkurven für die Beinbewegung

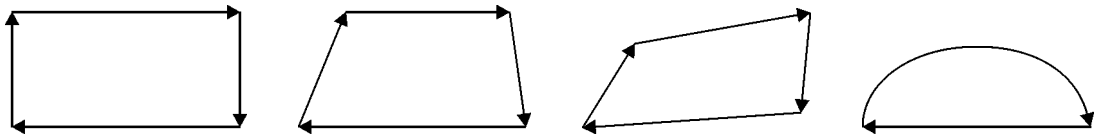
Die Füße des Roboters werden entlang von vordefinierten Bahnkurven bewegt. Abbildung 2.4 zeigt exemplarisch, wie die Füße des Roboters entlang der Trajektorien geführt werden. Zum einen soll der Roboter so schnell wie möglich laufen, zum anderen soll der resultierende Lauf aber auch vibrationsarm sein, damit die Kamera des Roboters ein möglichst gutes Bild liefert. Als Gangart wird hier immer der „Trab“ gewählt, das heißt, dass die diagonal gegenüberliegenden Füße immer in die gleiche Richtung bewegt werden. Während der Fuß vorne links und hinten rechts in der Luft sind, halten die anderen beiden Füße Kontakt mit dem Boden – und umgekehrt.

Zahlreiche Anstrengungen wurden von RoboCup Teams unternommen, um die „optimale“ Bahnkurve zu finden. Die Bahnkurven sind in der Regel über Parameter beschrieben, die die Form (Ausmaße) der Trajektorien und die Bahngeschwindigkeit für verschiedene Abschnitte der Trajektorie beeinflussen. Das Team Austin Villa nutzte eine halb-elliptische Bahnkurve, die über 17 Parameter beschrieben wurde [62], das GermanTeam bis 2004 eine rechteckige Bahnkurve [50], definiert über 14 Parameter, und das Team UPenn der University of Pennsylvania Vierecke, die mit 19 Parametern beschrieben wurden [8]. Alle Bahnkurven haben gemein, dass sie auf einer Ebene im Raum liegen. Abbildung 2.5 zeigt verschiedene gängige Formen der Bahnkurven.



(a) (b) (c) (d)

Abbildung 2.4: Bewegung eines Roboterfußes entlang einer halb-elliptischen Trajektorie

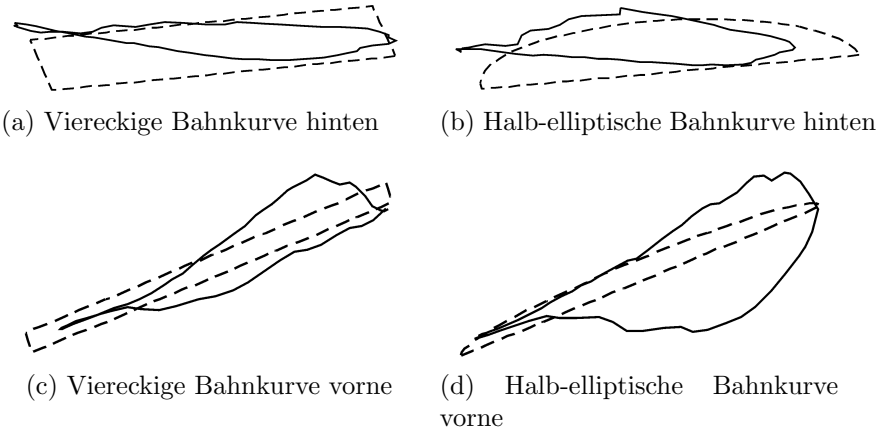


(a) Rechteckig (b) Trapezförmig (c) Viereckig (d) Halb-elliptisch

Abbildung 2.5: Verschiedene Trajektorien für die Fußbewegung

Eine Untersuchung des resultierenden Laufes anhängig von der Form der Trajektorie wurde durchgeführt, um den Effekt der Form der Fußtrajektorie zu verstehen. Dazu lief ein Roboter mit verschiedenen Bahnkurven und aus den tatsächlichen Gelenkwinkeln der Beingelenke wurde über die Vorwärtskinematik die tatsächliche Fußposition ermittelt [18]. Abbildung 2.6 zeigt die Soll-Trajektorien (gestrichelt) und die tatsächliche Trajektorien (durchgezogene Linie) für eine viereckige und eine halb-elliptische Ansteuerung jeweils eines Vorder- und Hinterbeins beim geradeaus Laufen. Die dargestellten Ansteuerungs-Trajektorien unterscheiden sich nur in der Phase, in der das Bein in der Luft ist, die Länge und Höhe der viereckigen und halb-elliptischen Kurven sind identisch. Ebenso die Phase, in der der Fuß den Boden berühren sollte. Die tatsächliche Bewegung weicht relativ stark von der gewünschten Bahnkurve ab. Besonders auffällig ist der Unterschied in der Bahnkurve der Vorderbeine zwischen der viereckigen Ansteuerung (Abbildung 2.6c) und der halb-elliptischen Ansteuerung (Abbildung 2.6d). Obwohl die unterschiedliche Ansteuerung nur die Beinbewegung in der Luft beeinflussen sollte, unterscheiden sich die Kurven insbesondere in dem Bereich, in dem die Füße den Boden berühren, der von der Ansteuerung her identisch ist.

Die Gründe für diese starke Abweichung wurden nicht eindeutig identifiziert. Ein Grund könnte sein, dass für die Kinematik eine Vereinfachung getroffen wurde, so dass der „Unterschenkel“ des Roboters als Linie betrachtet wird. Unter gewissen Gegebenheiten führt



(a) Viereckige Bahnkurve hinten (b) Halb-elliptische Bahnkurve hinten

(c) Viereckige Bahnkurve vorne (d) Halb-elliptische Bahnkurve vorne

Abbildung 2.6: Vergleich der angesteuerten und tatsächlichen Fußbewegung

dies zu einem Fehler, wie z. B. in Abbildung 2.4c ersichtlich: Der berechnete Fußpunkt, der den Boden berühren soll, stimmt nicht mit dem tatsächlich tiefsten Punkt des Fußes überein. Dieser Effekt wird umso mehr verstärkt, je mehr der Roboter auf den „Ellenbogen“ läuft. Als Ausweg wurde von dem Team NuBots die Form des Beines approximiert, eine eindeutige Verbesserung der Situation konnte aber nicht festgestellt werden [44]. Eine weitere Erklärung für die Abweichung liegt in der fehlenden Regelung zur Einhaltung der Ansteuerungskurve. Während in der industriellen Robotik großer Aufwand zur Einhaltung der Ansteuerung betrieben wird, existiert bei dem hier betrachteten Robotermodell lediglich der PID-Controller in den Gelenken. Er sorgt jedoch nicht explizit für eine Einhaltung der Ansteuerungskurve, sondern vielmehr zum Anfahren einer vorgegebenen Position. Bei einer Ansteuerung, die an den physikalischen Grenzen der Motoren – oder sogar darüber – liegt, versagt die Regelung. Um dieses Problem zu umgehen, wurde z. B. für den Sony Aibo eine Regelung, basierend auf einem neuronalen Netz, entwickelt, die eine Annäherung der tatsächlichen und der angesteuerten Bahnkurve ermöglicht [15].

In dieser Arbeit wird jedoch kein weiterer Aufwand betrieben, damit die Einhaltung der Soll-Trajektorie gewährleistet wird, sondern es werden mehr Freiheitsgrade für die Trajektorien zugelassen und das Problem als „Black-Box“ Problem betrachtet und später die Parameter der Trajektorien mit Hilfe von Lernverfahren optimiert. Schließlich ist es nicht das Ziel, dass die Trajektorien so abgefahren werden, wie sie angesteuert werden, sondern vielmehr, dass der Roboter schnell damit laufen kann. Die eigentliche resultierende Form der Trajektorien spielt keine Rolle.

Die Trajektorien werden nun durch 3-dimensionale Polygone beschrieben um mehr Freiheitsgrade zu gewährleisten und es der späteren Optimierung zu überlassen, wie die Parameter gewählt werden. Die Parameter, die die Polygone definieren, werden später in Kapitel 5.2.1 genauer beschrieben.

2.1.3 Omnidirektionale Steuerung

Mit der erläuterten Ansteuerung ist es nun möglich, den Roboter mit einer gewissen Geschwindigkeit geradeaus Laufen zu lassen. Gewünscht ist natürlich eine Steuerung der Geschwindigkeit und auch der Bewegungsrichtung. Daher wird im Folgenden die omnidirektionale[2] Steuerung des Roboters erläutert. Diese Arbeit baut auf dem so genannten Rädermodell auf, das von dem RoboCup Team UNSW entwickelt wurde [24] und sich quasi als Standard in der Four-Legged-League und somit zur Steuerung der Sony Aibos etabliert hat. Die Realisierung der Steuerung nimmt näherungsweise an, dass die tatsächliche Trajektorie mit der Ansteuerung übereinstimmt.

Steuerung der Geschwindigkeit

Die Laufgeschwindigkeit des Roboters ließe sich zum einen über die Bewegungsgeschwindigkeit des Fußes auf dem Boden steuern, d. h., dass die gesamte Trajektorie für einen langsam laufenden Roboter langsamer abgefahren wird. In dem hier vorgestellten Modell bleibt hingegen die Zeit, die der Fuß zum Abfahren der gesamten Trajektorie benötigt, konstant. Die Regelung der Geschwindigkeit wird über eine räumliche Skalierung der Trajektorien realisiert [9]. Soll der Roboter z. B. nur mit halber Maximalgeschwindigkeit (in x-Richtung) laufen, wird die Trajektorie auf die halbe Länge in x-Richtung skaliert, soll der Roboter auf der Stelle treten, ohne sich zu bewegen, wird ihre Länge auf 0 in x-Richtung skaliert.

Steuerung der Bewegungsrichtung

Die Steuerung der Bewegungsrichtung basiert auf der Annahme, dass sich die Beine des Roboters auf dem Boden wie Räder verhalten [24]. Abbildung 2.7 zeigt schematisch die Bewegung der Beine, damit sich der Roboter um den Punkt P_{ICR} dreht. Für Räder, die nur einen Kontaktpunkt auf dem Boden haben, stimmt diese Annahme. Da sich die Beine aber nicht exakt auf einer Kreisbahn bewegen, sondern nur tangential dazu auf einer Geraden, stimmt dieses Modell nur näherungsweise. Es hat sich in der Praxis aber als hinreichend genau erwiesen.

Die gewünschte Bewegung der Beine wird erreicht, indem die bisher entlang der x-Achse ausgerichteten Trajektorien zuerst auf die gewünschte Länge (Geschwindigkeit) skaliert werden und anschließend um die z-Achse gedreht werden. Die entsprechende Skalierung und Rotation lässt sich einfach durch Vektoraddition berechnen und ist in Abbildung 2.8 veranschaulicht. Die Länge der eingezeichneten Vektoren entspricht der Länge der Lauftrajektorien, die Richtung gibt an, wohin die Trajektorien um die z-Achse gedreht werden

[2] Omnidirektional bedeutet in diesem Zusammenhang, dass sich der Roboter unabhängig von seiner Position in jede gewünschte Richtung in der Ebene bewegen kann.

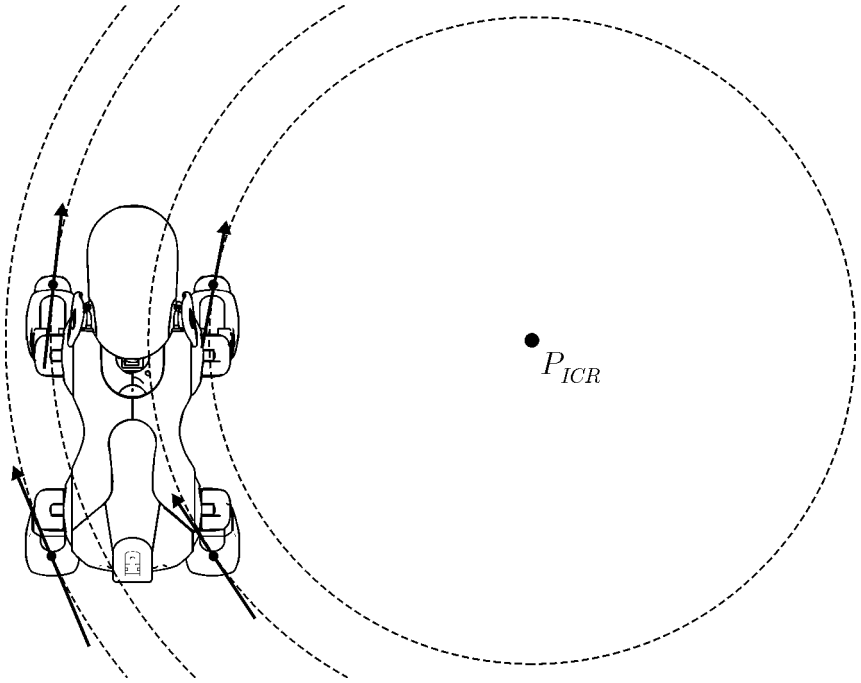


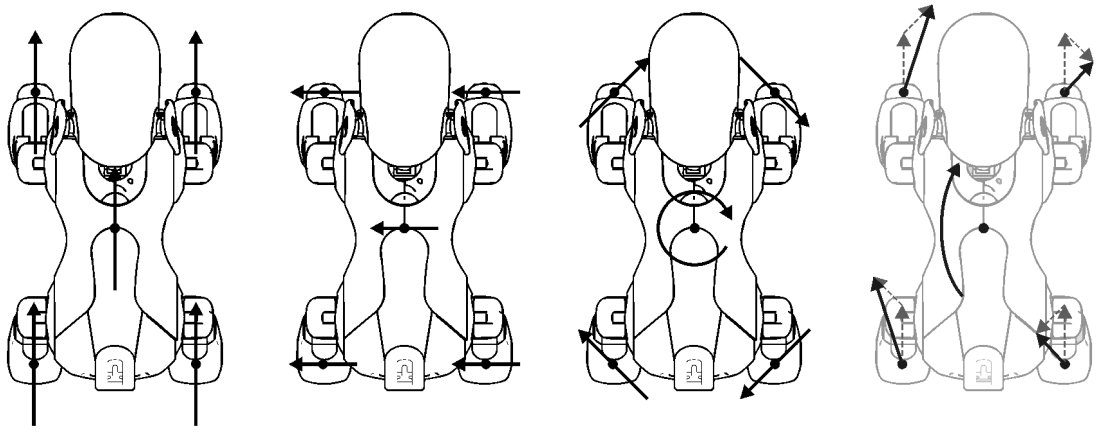
Abbildung 2.7: Rotation des Roboters um den Punkt P_{ICR}

müssen. Abbildungen 2.8a - 2.8c zeigen Steuerungen, bei denen jeweils nur eine Komponente der gewünschten Geschwindigkeit $\vec{v} = (v_x\ v_y\ v_r)^T$ ungleich Null ist. In Abbildung 2.8d ist die Vektoraddition für ein Vorwärtslaufen in Kombination mit einer Rotation dargestellt - in diesem Fall würde der Roboter in einem Kreis laufen. Es sei beachtet, dass zur Veranschaulichung der Vektorrechnung in diesem Beispiel nur die Draufsicht der halben Längen der Lauftrajektorien abgebildet sind.

Interpolation von Parametersätzen

Da Trajektorien, die einen schnellen Geradeauslauf gewährleisten, nicht zwangsläufig (in rotierter Form) auch einen schnellen Seitwärtslauf oder eine schnelle Rotation bewirken, werden verschiedene Trajektorien (beschrieben durch Parameter) für verschiedene Bewegungsrichtungen genutzt [18, 22]. Um einen flüssigen Lauf auch bei Änderung der Bewegungsrichtung zu gewährleisten, wird nicht „hart“ zwischen den Trajektorien umgeschaltet, sondern abhängig von der Bewegungsrichtung zwischen verschiedenen Trajektorien mehrstufig linear interpoliert. Abbildung 2.9 zeigt die Interpolation der zur Verfügung stehenden Parametersätze.

Zuerst werden abhängig von der x-Geschwindigkeit die entsprechenden Parameter aus Gruppe X interpoliert. Damit der Roboter bei langsameren Bewegungen möglichst vibrationsarm läuft, stehen speziell für diesen Fall weitere Parametersätze zur Verfügung. Soll der Roboter z. B. mit 75% seiner Maximalgeschwindigkeit vorwärts laufen, wird zwischen den



(a) Vorwärts (b) Seitwärts (c) Rotation (d) Vorwärts kombiniert mit Rotation

Abbildung 2.8: Bewegung der Beine für verschiedene Laufrichtungen

Parametersätzen für langsames Vorwärtslaufen und schnelles Vorwärtslaufen linear interpoliert – in diesem Fall also gemittelt. Abhängig von der y-Geschwindigkeit wird zwischen den entsprechenden Parametersätzen aus Gruppe Y^+, bzw. Y^- interpoliert. Diese beiden resultierenden Parametersätze wiederum werden abhängig von y miteinander kombiniert. Schlussendlich wird noch abhängig von der Rotationsgeschwindigkeit der Parametersatz für die Rotation proportional eingemischt.

2.2 Zweibeiniger Roboter Kondo KHR-1

Ein weiterer in dieser Arbeit genutzter Roboter ist der in Abbildung 2.10 dargestellte humanoide Roboter KHR-1 von der japanischen Firma Kondo. Dieser Roboter ist als Bausatz erhältlich mit einer Software, mit der vorgegebene Bewegungen abgespielt werden können.

Damit der Roboter autonom agieren kann, wurde im Rahmen einer Diplomarbeit eine Steuerungselektronik entwickelt, die Servopositionen über eine serielle Schnittstelle entgegennimmt und in die servospezifischen Pulslängen umwandelt, weiterhin sind noch ein Beschleunigungssensor, ein Gyroskop sowie ein Spannungsregler für die Servomotoren integriert. Als Recheneinheit wurde ein PDA benutzt, an den eine Kamera angeschlossen wurde und der wiederum über die serielle Verbindung mit der Steuerungselektronik kommuniziert.

Der Kondo KHR-1 besitzt 17 rotatorische Freiheitsgrade, die durch die Verwendung von folgenden Servomotoren (vgl. Abbildung 2.10c) realisiert sind: Einen Servo, um den Kopf zu drehen, je drei Servos in den Armen und je fünf Servos in den Beinen.

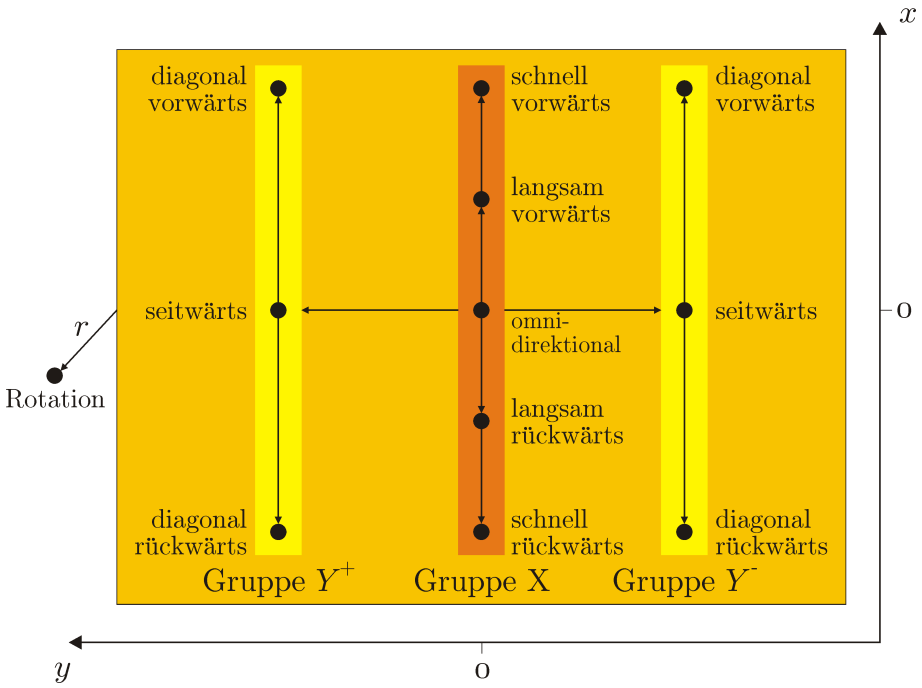


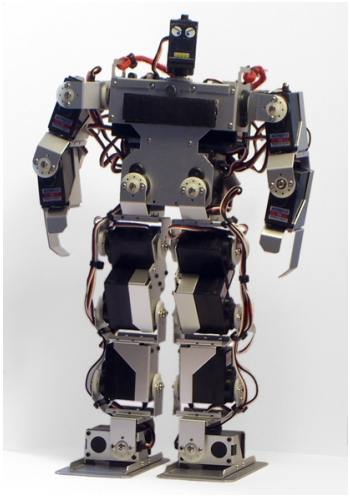
Abbildung 2.9: Interpolation von verschiedenen Parametersätzen

Eine Einschränkung stellt die Anordnung der Servos in den Beinen dar, da es ist dem Roboter nicht möglich ist, die Beine um die z-Achse zu drehen. Die Beinservos können die Füße des Roboters zwar entlang aller drei Raumachsen unterhalb des Oberkörpers bewegen, die Ausrichtung der Füße bleibt, bezogen auf den Oberkörper, jedoch stets gleich.

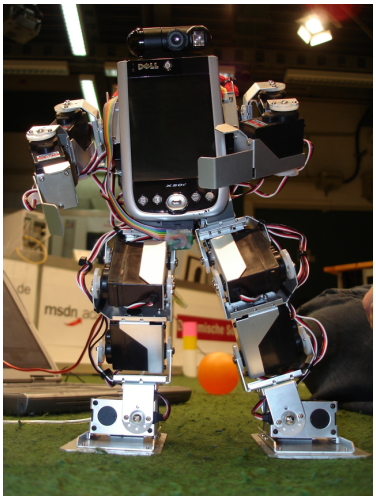
2.2.1 Inverse Kinematik

Wie zuvor schon für den Aibo, wird auch für diesen Roboter ein Modell, basierend auf dem Abfahren von Trajektorien, hergeleitet. Daher muss auch für den KHR-1 eine Rechenvorschrift gefunden werden, die aus einem zu erreichenden Punkt im Raum die zugehörigen Gelenkwinkel für die Servomotoren berechnet.

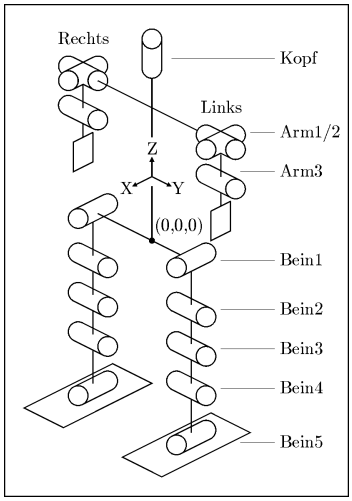
Die Berechnung der inversen Kinematik nutzt die Bezeichnungen aus Abbildung 2.11. Die Mittelpunkte der Gelenke sind mit $P_1, \ldots, P_5$ bezeichnet, der jeweilige einzunehmende Winkel des Gelenks mit $\theta_1, \ldots, \theta_5$ und die Längen der Verbindungen zwischen den Gelenkmittelpunkten P_i und P_j mit l_{ij}. Der Winkel θ_2 ist der Übersicht halber in Abbildung 2.11b nicht eingezeichnet und ist gegeben durch die Hilfswinkel $\alpha_1 + \alpha_2$. Das Koordinatensystem des Beines liegt in dem obersten Gelenk P_1. Berechnet werden nun die einzunehmenden Gelenkwinkel $\theta_1, \ldots, \theta_5$, damit sich der Referenzpunkt P_5 des Fußes mit dem anzufahrenden Punkt $\vec{p} = (p_x\ p_y\ p_z)^T$ deckt. Es wird bei der Herleitung davon ausgegangen, dass der Punkt $\vec{p}$ im Arbeitsraum des Beines liegt und somit erreichbar ist. Weiterhin sei gefordert,



(a) Der Kondo KHR-1 im Auslieferungszustand



(b) Der Kondo KHR-1 mit Steuerungselektronik



(c) Die kinematische Struktur des Kondo KHR-1

Abbildung 2.10: Der Kondo KHR-1

dass sich der Fuß des Roboters immer parallel zum Boden bewegt.

Zunächst lässt sich der Winkel θ_1 berechnen, da er allein bestimmt, wie weit das Bein seitlich in y Richtung gestreckt wird.

$$\theta_1 = \arctan\left(\frac{p_y}{p_z}\right) \tag{2.8}$$

Aus der Forderung, dass sich der Fuß parallel zum Boden befinden soll, ergibt sich der Winkel θ_5 zu

$$\theta_5 = -\theta_1 = -\arctan\left(\frac{p_y}{p_z}\right). \tag{2.9}$$

Durch die Kenntnis von θ_1 und θ_5 lassen sich die Koordinaten der Punkte P_2 und P_4 bestimmen zu

$$P_2 = \begin{pmatrix} 0 \\ l_{12}\sin\theta_1 \\ -l_{12}\cos\theta_1 \end{pmatrix}$$

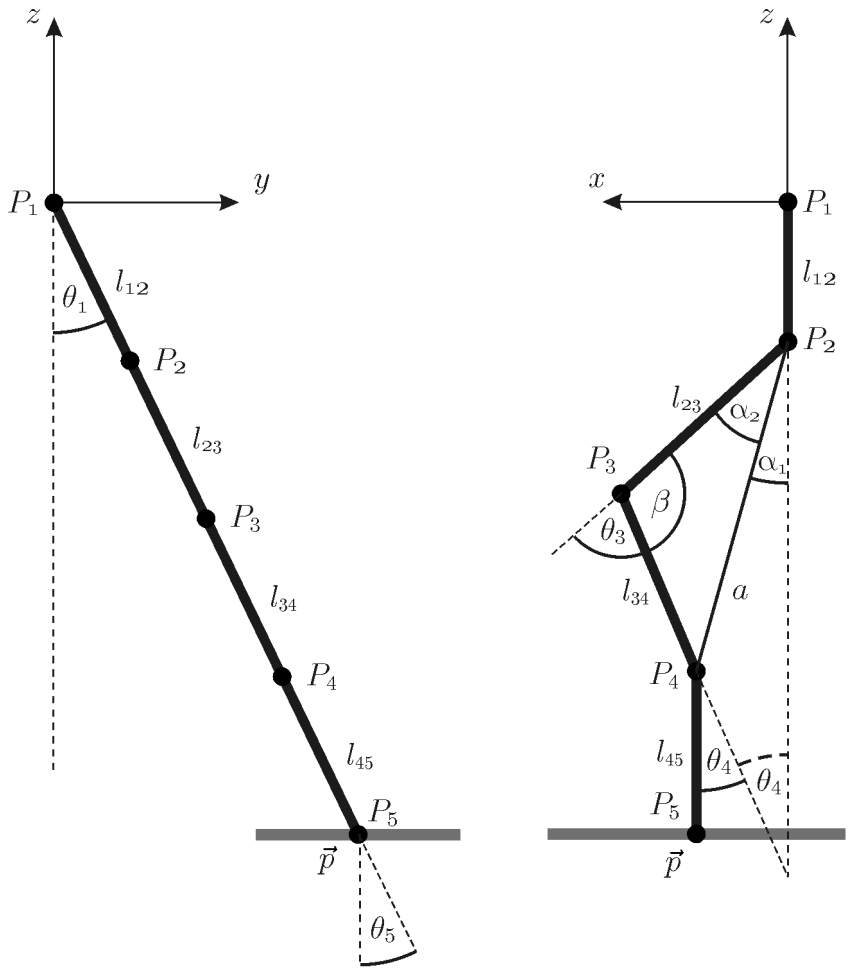
und

$$P_4 = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + \begin{pmatrix} 0 \\ l_{45}\sin\theta_5 \\ l_{45}\cos\theta_5 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y + l_{45}\sin\theta_5 \\ p_z + l_{45}\cos\theta_5 \end{pmatrix}.$$

Zur weiteren Berechnung wird das Dreieck, welches durch die Punkte P_2, P_3, P_4 aufgespannt wird, betrachtet. Die Länge der Seite a des Dreiecks ist gegeben durch

$$a = \sqrt{(P_{4x} - P_{2x})^2 + (P_{4z} - P_{2z})^2} \tag{2.10}$$

$$= \sqrt{p_x^2 + (p_z + l_{45}\cos\theta_5 + l_{12}\cos\theta_1)^2} \tag{2.11}$$



(a) Ansicht in der y-z Ebene (b) Ansicht in der x-z Ebene

Abbildung 2.11: Die Bezeichnungen der Winkel und Abschnittslängen eines Beines des Kondo KHR-1

und der Hilfswinkel α_1 dementsprechend durch

$$\alpha_1 = \arctan\left(\frac{P_{4x} - P_{2x}}{P_{4z} - P_{2z}}\right) = \arctan\left(\frac{p_x}{p_z + l_{45}\cos\theta_5 + l_{12}\cos\theta_1}\right).$$

Mit Hilfe des Kosinussatzes lässt sich der Winkel α_2 bestimmen zu

$$\alpha_2 = \pm\arccos\left(\frac{l_{23}^2 - l_{34}^2 + a^2}{2l_{23}a}\right).$$

Die zwei Lösungsmöglichkeiten für α_2 bedeuten anschaulich, dass das Knie des Roboters (Punkt P_3) entweder nach vorne oder nach hinten eingeknickt werden kann. Für den KHR-1 wird festgesetzt, dass das Kniegelenk wie auch beim Menschen nach hinten, also entgegen

des Uhrzeigersinns eingeknickt wird. Bei Betrachtung von Abbildung 2.11b wird klar, dass dann für den Winkel α_2 nur die negative Lösung (Bewegung im Uhrzeigersinn) gültig ist. In diesem Fall verbleibt also nur die Lösung

$$\alpha_2 = -\arccos\left(\frac{l_{23}^2 - l_{34}^2 + a^2}{2l_{23}a}\right).$$

Der Winkel θ_2 kann nun bestimmt werden durch

$$\begin{aligned}\theta_2 =& \alpha_1 + \alpha_2 \\ =& \arctan\left(\frac{p_x}{p_z + l_{45}\cos\theta_5 + l_{12}\cos\theta_1}\right) - \arccos\left(\frac{l_{23}^2 - l_{34}^2 + a^2}{2l_{23}a}\right).\end{aligned} \tag{2.12}$$

Ebenfalls mittels des Kosinussatzes lässt sich der Winkel β ausdrücken durch

$$\beta = \pm\arccos\left(\frac{l_{23}^2 + l_{34}^2 - a^2}{2l_{23}l_{34}}\right)$$

und da $\theta_3 = \pi - \beta$ gilt, folgt

$$\begin{aligned}\theta_3 =& \pi \mp \arccos\left(\frac{l_{23}^2 + l_{34}^2 - a^2}{2l_{23}l_{34}}\right) \\ =& \mp\arccos\left(\frac{a^2 - l_{23}^2 - l_{34}^2}{2l_{23}l_{34}}\right).\end{aligned}$$

Wie schon erwähnt, wird vorausgesetzt, dass das Knie nur entgegen des Uhrzeigersinns (also im mathematisch positiven Sinn) eingeknickt wird, daher bleibt die Lösung

$$\theta_3 = \arccos\left(\frac{a^2 - l_{23}^2 - l_{34}^2}{2l_{23}l_{34}}\right). \tag{2.13}$$

Der noch verbleibende Winkel θ_4 stellt sicher, dass der Fuß bezüglich der x-Achse parallel zum Boden steht. Da die Winkelsumme im Dreieck stets π beträgt und in diesem Fall, wie in Abbildung 2.11b dargestellt, die Winkel θ_2 und θ_4 mit negativen Werten belegt sind (sie sind im Uhrzeigersinn gedreht), wird θ_4 schließlich bestimmt durch

$$\theta_4 = -\theta_2 - \theta_3. \tag{2.14}$$

2.2.2 Trajektorien der Roboter-Gliedmaßen

Die Laufbewegung des Kondo KHR-1 wird ebenfalls durch Trajektorien für die Robotergliedmaßen beschrieben. Hier werden neben den Beinen auch die Arme und der Oberkörper entlang von definierten Bahnen bewegt. Die Füße werden entlang zweidimensionaler Trajektorien bewegt, die Arme bewegen sich über jeweils ein eigenes Gelenk entlang eindimensionaler Bahnen. Separat wird noch der Oberkörper eindimensional bewegt. Durch diese Bewegungsmöglichkeiten lässt sich im Groben die Laufbewegung des Menschen imitieren.

Es wird nur die Bewegung für ein Bein, bzw. einen Arm beschrieben. Die Bewegung des anderen Beines und Armes ergibt sich durch eine Phasenverschiebung analog.

Alle zur Laufsteuerung benutzten Trajektorien werden in der Zeit, die die Länge eines Schrittes definiert, einmal komplett abgefahren. Dabei gibt die Variable t mit $0 \leq t < 1$ an, an welcher Position im Schritt (und somit der Trajektorie) sich die Laufsteuerung gerade befindet. Der Wert $t = 0$ beschreibt den der Anfangspunkt der Trajektorie und stimmt mit dem Endpunkt für $t = 1$ wieder überein. Der Parameter t_{Step} definiert die Zeit für einen vollen Schritt und stellt somit quasi den Skalierungsfaktor zwischen realer Zeit und dem Parameter t dar. Bei dem KHR-1 erfolgt die Ansteuerung der Beine mit 50 Hz.

Anders als bei den AIBO Robotern werden die Servos dieses humanoiden Roboters weit weniger am Limit der Motoren bewegt, so dass die real abgefahrenen Trajektorien den angesteuerten Trajektorien sehr nahe kommen. Daher werden im Folgenden vier mögliche Formen für die Trajektorien der Fußbewegung in der Luft vorgestellt und diskutiert. Zunächst werden die Trajektorien für die einzelnen Gliedmaßen für geradeaus Laufen mit konstanter Geschwindigkeit erläutert und später zur Richtungs-, bzw. Geschwindigkeitssteuerung erweitert [19].

Trajektorien für die Fußbewegung

Prinzipiell kann für die Bewegung der Beine zwischen einer Phase der Bodenberührung und einer Phase, während der das Bein durch ein Anheben in Laufrichtung vorgesetzt wird, unterschieden werden. In der Literatur wird der Zeitpunkt, in dem beide Beine den Boden berühren mit *double support phase* und die Zeit, in der der Roboter nur auf einem Bein steht mit *single support phase* bezeichnet. Durch die Phase des Voransetzens wird das Bein, in Bezug auf den Körper, in die Laufrichtung bewegt und wieder abgesenkt. Während des Bodenkontaktes bewegt sich das Bein dann entgegen der Laufrichtung, um den Roboterkörper in die Laufrichtung zu bewegen. Für das hier genutzte Laufmodell wird festgelegt, dass die Zeit, in der ein Fuß in der Luft ist, gleich der Zeit der Bodenberührung ist. Die *double support phase* ist dementsprechend unendlich kurz und der Roboter befindet sich (fast) immer in der *single support phase*. Während die Bewegung des Fußes auf dem Boden als Gerade im Raum festgelegt wird, ist die Bewegung des Fußes in der Luft beliebig wählbar.

Die Bewegung eines einzelnen Fußes findet im dreidimensionalen Raum statt, zwei Dimensionen reichen hier aber aus, um die Bewegung zu beschreiben. Die Trajektorie wird durch zwei Vektoren aufgespannt. Ein Vektor zeigt in die Laufrichtung, der andere Vektor entgegen der wirkenden Gravitationskraft. Eine dritte Dimension in der Trajektorienbeschreibung würde weder zu einer Fortbewegung in die geplante Laufrichtung, noch zu einer Änderung der Schritthöhe führen. Weiterhin wird für die Trajektorie gefordert, dass sie einen geschlossenen Pfad beschreibt, so dass der Fuß nach Vollendung des Schrittes zum Zeitpunkt $t = 1$ exakt an der Stelle steht, wo der nächste Schritt für $t = 0$ beginnt. Ab-

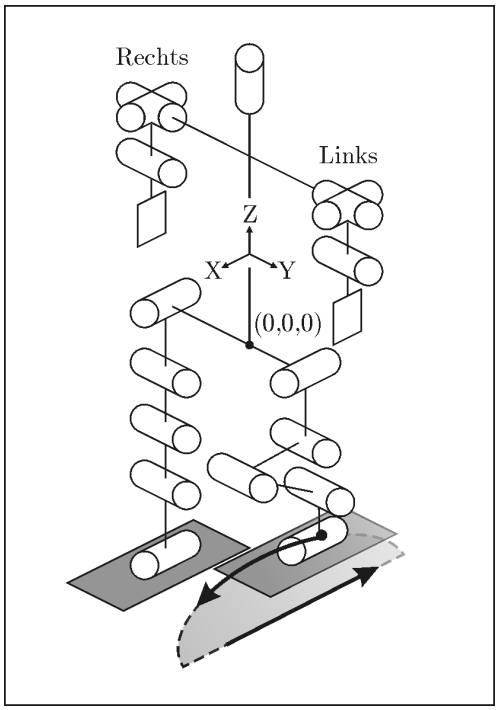


Abbildung 2.12: Bewegung des Fußes entlang einer Trajektorie

bildung 2.12 zeigt schematisch, wie der Roboterfuß entlang einer Trajektorie bewegt wird. Die einzunehmenden Gelenkwinkel werden über die inverse Kinematik berechnet.

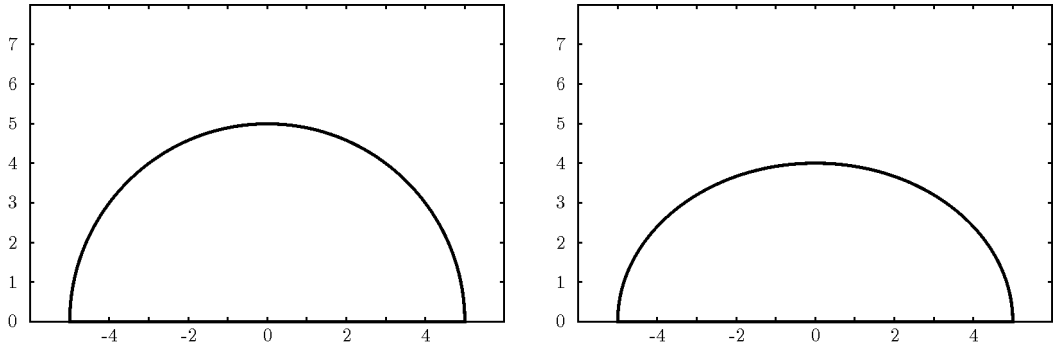
Die Bewegung beider Füße wird durch dieselben Trajektorien definiert, sie ist jedoch zeitlich um den Offset $t_o = 0,5$ verschoben, so dass der eine Fuß gerade in dem Moment angehoben wird, in dem andere Fuß den Boden berührt.

Die Unterseite der Trajektorie soll dabei wegen des hier betrachteten ebenen Untergrundes eine Gerade sein und wird beschrieben durch die Gleichung

$$\vec{p}(t) = \begin{pmatrix} x(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} \frac{s_l}{2} - 2s_l(t - 0.5) \\ 0 \end{pmatrix}. \tag{2.15}$$

Die Fußbewegung auf dem Boden findet in dem Zeitraum $0,5 \leq t < 1$ statt. Wie schon zuvor erläutert, werden bei diesem Laufmodell die Füße stets parallel zum Boden ausgerichtet, so dass der Fuß beim Aufsetzen den Boden komplett berührt und ausgeschlossen wird, dass der Fuß in der Luft durch Abwinkeln den Boden berühren kann.

Bei der Entwicklung des Laufmodells für den KHR-1 wurden verschiedene Arten von Trajektorien für die Fußbewegung in der Luft für den Zeitraum $0 \leq t < 0,5$ untersucht und werden im Folgenden kurz erläutert.



(a) Halbkreis (b) Halb-Ellipse

(c) Rechteck (d) Sinusform

Abbildung 2.13: Verschiedene Trajektorien für die Fußbewegung

Halb-Kreis Exemplarisch ist die Halb-Kreis Trajektorie in Abbildung 2.13a dargestellt. Als Konsequenz der kreisförmigen Bewegung ergibt sich, dass die Schritthöhe die Hälfte der Schrittlänge entspricht. Weiterhin ist sichergestellt, dass die Füße senkrecht auf den Boden aufsetzen. Diese Trajektorie wird mit nur einem Parameter, der Schrittlänge s_l, eindeutig beschrieben durch die Gleichung

$$\vec{p}(t) = \begin{pmatrix} x(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} -(\cos 2\pi t) \cdot \frac{s_l}{2} \\ (\sin 2\pi t) \cdot \frac{s_l}{2} \end{pmatrix}. \tag{2.16}$$

Rechteck Für eine rechteckförmige Bewegung der Füße (Abbildung 2.13c) lässt sich die Form der Trajektorie über die zwei Parameter Schrittlänge s_l und die Schritthöhe s_h beschreiben. Für eine Bewegung, die unterschiedliche Geschwindigkeiten für das Anheben und Absenken, sowie der Vorwärtsbewegung des Fußes zulässt, müssen zusätzlich noch die Zeiten für diese Abschnitte angegeben werden. Während der Zeit t_1 wird der Fuß angehoben, in der Zeitspanne $t_2 - t_1$ wird der Fuß in der Luft in die Laufrichtung bewegt. Die Zeitdauer zum Absetzen des Fußes ergibt sich dann automatisch durch $0.5 - t_2$. Es muss also die Bedingung $0 \leq t_1 \leq t_2 < 0.5$ erfüllt sein. Diese Trajektorie ist über die zuvor

erläuterten Parameter schließlich definiert durch

$$\vec{p}(t) = \begin{pmatrix} x(t) \\ z(t) \end{pmatrix} = \begin{cases} \begin{pmatrix} -\frac{s_l}{2} \\ \frac{s_h}{t_1} \cdot t \end{pmatrix} & \text{für } 0 \leq t < t_1 \\ \begin{pmatrix} -\frac{s_l}{2} + \frac{s_l}{t_2 - t_1} \cdot (t - t_1) \\ s_h \end{pmatrix} & \text{für } t_1 \leq t < t_2 \\ \begin{pmatrix} \frac{s_l}{2} \\ s_h - \frac{s_h}{0.5 - t_2} \cdot (t - t_2) \end{pmatrix} & \text{für } t_2 \leq t < 0.5 \end{cases} . \tag{2.17}$$

Halb-Ellipse Die Halb-Ellipse (2.13b) ist quasi eine Erweiterung der halbkreisförmigen Bewegung. Sie entkoppelt die Schrittlänge s_l von der Schritthöhe s_h. Es wird hier wieder eine konstante Bewegungsgeschwindigkeit festgesetzt, so dass diese Trajektorie durch

$$\vec{p}(t) = \begin{pmatrix} x(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} -(\cos 2\pi t) \cdot \frac{s_l}{2} \\ (\sin 2\pi t) \cdot s_h \end{pmatrix} \tag{2.18}$$

beschrieben wird.

Sinus Die sinusförmige Trajektorie nach [1] ist dargestellt in Abbildung 2.13d und wird definiert durch

$$\vec{p}(t) = \begin{pmatrix} x(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} \left(4t - 1 - \frac{\sin(4\pi t)}{\pi}\right) \cdot \frac{s_l}{2} \\ (\sin 2\pi t) \cdot s_h \end{pmatrix} . \tag{2.19}$$

Die Form der Trajektorie ähnelt auf den ersten Blick stark der halbelliptischen Trajektorie. Bei dieser Trajektorie sind wiederum Schritthöhe s_h und Schrittlänge s_l voneinander entkoppelt.

Auswahl einer Fuß-Trajektorie

In Experimenten mit den verschiedenen Trajektorien wurden deren Vor- und Nachteile in Hinsicht auf die resultierende Laufbewegung untersucht. Die Ergebnisse werden hier kurz erläutert.

Die halbkreisförmige Ansteuerung führt insbesondere bei langen Schritten durch die Abhängigkeit von Schrittlänge und Schritthöhe zu einem unvorteilhaft hohen Anheben des Fußes. Dies führt bei großen Schritten zu einer Instabilität des Roboters, da die angehobenen Beine den Schwerpunkt weit nach oben verlagern und so ein instabiles Laufen erzeugen. Eine geringe Schritthöhe hingegen führt automatisch auch zu einer geringen Schrittweite, was dann bei gleicher Schrittdauer in einer langsameren Laufgeschwindigkeit resultiert.

Vorteilhaft hingegen ist zu erwähnen, dass durch die ruckfreien Bewegungen der Fußbewegung in der Luft und das senkrechte Aufsetzen der Füße auf den Boden die Laufbewegung (für nicht zu große Schritte) prinzipiell einen stabilen Eindruck macht.

Die rechteckförmige Ansteuerung bietet zwar eine Entkopplung von Schritthöhe und Schrittlänge, sowie auch ein senkrechtes Absetzen der Füße, hat jedoch den gravierenden Nachteil, dass sich die Füße in der Luft sehr ruckartig bewegen, was an den rechtwinkligen Ecken der Trajektorie liegt. In diesen Ecken werden die Füße in ihrer Richtung stark umgelenkt, was sich in Erschütterungen des Roboterkörpers niederschlägt.

Bei der halbelliptischen Ansteuerung werden die Vorteile der halbkreisförmigen Ansteuerung mit den Vorteilen der rechteckigen Bahnkurve kombiniert, so dass die Bewegung ruckfrei ist, die Füße senkrecht auf den Boden aufgesetzt werden und die Schrittlänge von der Schritthöhe unabhängig ist. Nachteile sind bei dieser Ansteuerung nicht zu erkennen.

Wie die halbelliptische Bahnkurve, erzeugt auch die sinusförmige Trajektorie eine sehr glatte und natürlich aussehende Beinbewegung, allerdings verursacht sie ein Stolpern des Roboters, weil die Füße kurz vor dem Aufsetzen auf den Boden noch leicht in Laufrichtung bewegt werden. Bei Lauftests auf Teppichboden „stolperte“ der Roboter mehr als bei der elliptischen Bewegung und fiel häufiger um.

Für die spätere Optimierung wird aufgrund der genannten Vorteile nur die halbelliptische Trajektorie verwendet.

Trajektorien der Arme

Die menschliche Laufbewegung zeigt, dass eine mit den Beinen synchronisierte Armbewegung die Laufbewegung unterstützen kann. Die Arme können zum einen beim Balancieren helfen und zum anderen auch zur Fortbewegung beitragen, indem sie Momente in die Laufrichtung erzeugen. Jogger und Läufer machen daher auch ausgiebig Gebrauch der Arme. Hier wird auf die Definition von Armbewegungen entlang zweidimensionaler Trajektorien mittels inverser Kinematik verzichtet. Stattdessen werden die Schultergelenke direkt eindimensional im Gelenkraum bewegt. Es werden nur die oberen Schultergelenke (Arm 1/2 aus Abbildung 2.10c) für die Bewegung genutzt, das Ellbogengelenk (Arm 3) wird nicht bewegt. Abbildung 2.14 veranschaulicht die mögliche Armbewegung. Die Gelenke ermöglichen eine Bewegung in der x-z und der y-z Ebene.

Die Geschwindigkeit und die Phasenlage der Armbewegung sind über den selben Parameter t fest an die Fußbewegung gekoppelt. Die Zeit für ein Vor- und Zurückschwingen der Arme entspricht dementsprechend der Zeit für einen Schritt. Als Bewegungsform der Arme hat sich eine Sinusschwingung als deutlich effizienter als eine lineare Ansteuerung proportional zu t erwiesen, da, wie schon bei der rechteckigen Fußtrajektorie, die abrupten Richtungsänderungen zu Instabilität führen. Die sinusförmige Ansteuerung bremst die Bewegung vor dem Wendepunkt sanft ab, was als Nebeneffekt zur Folge hat, dass die Arme länger die

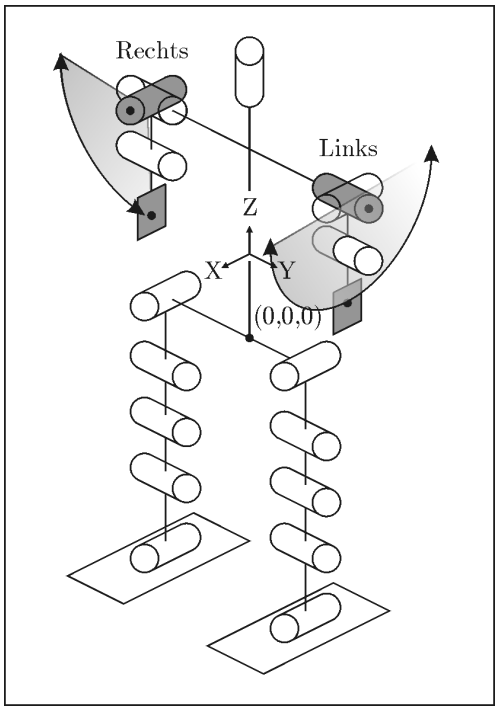


Abbildung 2.14: Bewegungsmöglichkeiten der Arme in der x-z und y-z Ebene

Funktion von Ausgleichsgewichten für die Beine erfüllen können.

Armbewegung in der x-z-Ebene Der Winkel $\omega_{arm_{xz}}$ der Armbewegung in der x-z-Ebene ist über die Zeit definiert durch

$$\omega_{arm_{xz}}(t) = \frac{\pi}{2} x_{arm_{max}} \cdot \sin 2\pi \left(t + \varphi_{arm_{xz}}\right) \tag{2.20}$$

mit einem Skalierungsfaktor $x_{arm_{max}}$ und einem Phasenversatz relativ zu der Beinbewegung von $\varphi_{arm_{xz}}$, der auf $\varphi_{arm_{xz}} = 0,25$ festgelegt wird. Das bedeutet, dass (wie beim menschlichen Lauf) der linke Arm nach vorne schwingt, wenn sich das rechte Bein nach vorne bewegt.

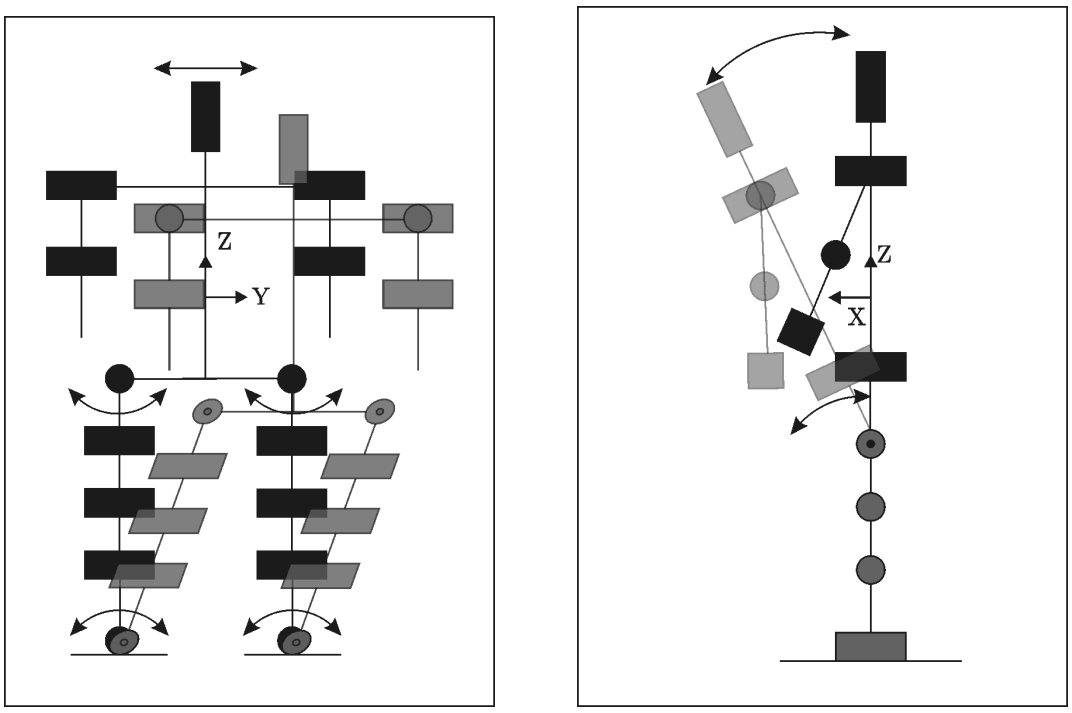
Armbewegung in der y-z-Ebene Da die Bewegung der Arme in der y-z-Ebene durch den Körper des Roboters nach innen beschränkt ist, wird die Armbewegung definiert durch

$$\omega_{arm_{yz}}(t) = \frac{\pi}{4} y_{arm_{max}} \cdot (\sin 2\pi(t + \varphi_{arm_{yz}}) + 1) \tag{2.21}$$

mit einer festgelegten Phase relativ zu der Beinbewegung von $\varphi_{arm_{yz}} = 0,5$. Der Arm wird also nur bis zur Waagerechten und damit nicht nicht über die Schulterhöhe hinaus angehoben.

Bewegung des Oberkörpers

Um die Schwerpunktverlagerung des Roboters durch das Anheben eines Fußes noch weiter zu kompensieren als dies alleine durch die Armbewegungen möglich ist, wird zudem eine Bewegung des Oberkörpers seitlich (in y Richtung) und vorwärts (in x Richtung) modelliert. Abbildung 2.15 zeigt schematisch diese Bewegungen. Wie schon bei den Armbewegungen werden die Bewegungen des Oberkörpers direkt im Gelenkraum ausgeführt.



(a) Die Seitwärtsbewegung (b) Die Vorwärtsbewegung

Abbildung 2.15: Schematische Darstellung der Oberkörperbewegung

Seitwärtsschwingung Die Seitwärtsschwingung ist gerechtfertigt, um den Schwerpunkt des Roboters über den Fuß, der auf dem Boden steht, zu bewegen. Das seitliche Ausstrecken der Arme beim Laufen hat denselben Effekt, jedoch deutlich geringer, da die Masse der Arme relativ gering ist. Wenn der Roboter entsprechend große Füße hat, ist durch diese Verlagerung des Schwerpunktes sogar ein statisch stabiles Laufen möglich. Realisiert wird diese Bewegung nur durch die Gelenke Bein 1 (oberstes Beingelenk) und Bein 5 (unteres Beingelenk) gemäß Abbildung 2.10c. Um eine ruhige Bewegung zu gewährleisten, wird abermals eine sinusförmige Ansteuerung genutzt, die über den selben Parameter t mit der Beinbewegung gekoppelt ist.

$$\omega_{body_y}(t) = \frac{\pi}{2} y_{body_{max}} \cdot \sin 2\pi(t + \varphi_{body_y}) \tag{2.22}$$

Für eine Phase von $\varphi_{body_y} = 0$ bewegt sich der Oberkörper derart, dass er seine maximale seitliche Auslenkung erreicht hat, wenn der gegenüberliegende Fuß in der Luft ist und sich gerade in der Mitte der Luftbewegung befindet. In dem Moment, in dem beide Füße den Boden berühren, befindet sich der Oberkörper in Mittelstellung. Der resultierende Winkel $\omega_{body_y}(t)$ wird nun einfach zu dem durch die Fußtrajektorie vorgegebenen Winkel des Gelenkes Bein 1 hinzuaddiert und entsprechend von dem unteren Gelenk Bein 5 subtrahiert.

Vorwärtsschwingung Der Vollständigkeit halber wird noch eine Schwingung in x-Richtung realisiert. Vögel beispielsweise schwingen beim Laufen den Körper für jeden Schritt leicht nach vorne. Da diese Bewegung für jeden Einzelschritt durchzuführen ist, wird sie im Vergleich zu den anderen Trajektorien mit der doppelten Frequenz betrieben und ist definiert durch

$$\omega_{body_x}(t) = \frac{\pi}{4} x_{body_{max}} \cdot (1 + \sin 4\pi(t + \varphi_{body_x})) \tag{2.23}$$

Der Winkel $\omega_{body_x}(t)$ wird auf den Winkel der Gelenke Bein 2 (bezogen auf Abbildung 2.10c) addiert, da durch die inverse Kinematik ja sichergestellt ist, dass der Oberkörper stets aufrecht ausgerichtet ist. So kann, unabhängig von der Position der Füße, mit einer Addition dieses Winkels die gewünschte Auslenkung des Oberkörpers erreicht werden. Die Phase wird mit $\varphi_{body_x} = 0$ festgelegt.

2.2.3 Steuerung der Bewegungsrichtung

Die bisher erläuterte Bewegung der Beine resultiert in einem Laufen in die x-Richtung, also geradeaus. Gewünscht ist aber, wie für den Aibo Roboter auch, eine möglichst omnidirektionale Steuerung.

Eine translatorische Bewegung in eine Richtung α in der x-y Ebene kann, wie zuvor bei dem Aibo erläutert, einfach durch eine Rotation der Trajektorie mit dem Winkel α um die z Achse erfolgen. Da die Füße normalerweise relativ dicht (in y Richtung) beieinander stehen, entsteht hier allerdings für zu große Seitwärtsschritte ein Problem. Die Füße würden in der Mitte kontinuierlich kollidieren. Daher werden die Beine proportional zu der Seitwärtsgeschwindigkeit in y Richtung gespreizt. In Abhängigkeit von der Bewegungsrichtung α ergibt sich eine Spreizung von

$$y_o = \frac{s_l}{2} |\sin \alpha| \tag{2.24}$$

mit der Schrittlänge s_l. Dieser Offset y_o wird zu der gewünschten anzusteuernden y Koordinate vor der Berechnung der inversen Kinematik addiert. Der rechte Fuß wird um den Wert y_o verschoben und der linke Fuß entsprechend um den Wert $-y_o$.

Wie zuvor erwähnt, ist der Kondo KHR-1 aufgrund seiner Bauart nicht in der Lage, die Beine um die z Achse zu drehen. Als Ausweg kann sich der Roboter auf der Stelle drehen, indem die Beine auseinandergezogen werden (durch einen Halbschritt nach vorne oder nach hinten), und der Roboter dann mit beiden Füßen auf dem Boden die Beine wieder

zusammenzieht. Dadurch dreht sich der Roboter auf einem Boden, der den Füßen ein Rutschen gestattet, in die Richtung des Fußes, der zuvor hinten stand. Dies funktioniert jedoch nur auf rutschigem Boden. Ferner kann nicht genau errechnet werden, wie weit sich der Roboter drehen wird, da es dabei einen Versatz durch die Bodenreibung gibt.

Die Geschwindigkeit wird, wie für den Aibo Roboter, über die (räumliche) Schrittlänge gesteuert. Wenn der Roboter beispielsweise mit halber Maximalgeschwindigkeit laufen soll, wird die Trajektorie in x Richtung auf die Länge $s_l/2$ skaliert. Für ein Treten auf der Stelle beträgt die Schrittlänge entsprechend 0.

Kapitel 3

Optimierung mit Evolutionären Algorithmen

Evolutionäre Algorithmen gehören zu der Klasse der randomisierten Suchverfahren. Zur Optimierung von Problemen orientieren sie sich in ihrer Vorgehensweise an der biologischen Evolution und dem grundlegenden Paradigma *survival of the fittest*. Durch Nachahmung grundlegender Evolutionsprinzipien entsteht so ein leistungsfähiges Optimierverfahren.

Klassische Optimierverfahren, wie z. B. einfache Gradientenverfahren, scheitern häufig an einer oder mehreren der folgenden Eigenschaften der zu optimierenden Zielfunktion [2]:

- Nicht differenzierbare Zielfunktion
- Multimodale oder hochdimensionale Zielfunktion
- Restriktionen in dem Definitionsbereich der Zielfunktion
- Zielfunktion unterliegt stochastischem Rauschen

Schwefel hat gezeigt, dass sich Evolutionäre Algorithmen für die Optimierung von Zielfunktionen mit den genannten Eigenschaften eignen [54]. Die Oberklasse der Evolutionären Algorithmen gliedert sich auf in *Evolutionsstrategien*, *Genetische Algorithmen* [13], *Evolutionäre Programmierung* [12] und *Genetische Programmierung* [35].

Abbildung 3.1 zeigt das allen evolutionären Algorithmen zugrunde liegende Prinzip. Eine Elterngeneration pflanzt sich fort durch *Reproduktion*. Auf die Gene der Eltern können dabei folgende Operationen wirken:

- *Replikation*,
- *Mutation* und
- *Rekombination*.

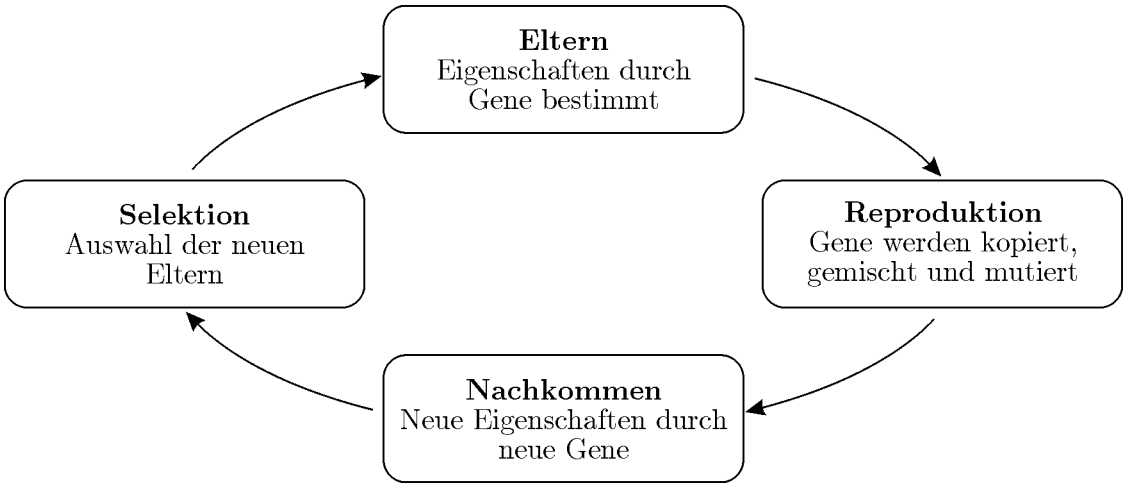


Abbildung 3.1: Evolutionskreislauf

Auf diese Weise entsteht eine Anzahl von *Nachkommen*, die sich in der Regel von ihren Eltern in ihren Genen unterscheiden. Andere Gene bedeuten allgemein auch andere Eigenschaften. Durch die *Selektion* wird bestimmt, welche Nachkommen in der nächsten Generation zu Eltern werden dürfen und schließlich wieder neue Nachkommen erzeugen, die restlichen Nachkommen sterben aus. Wenn die Selektion nach dem oben genannten Paradigma *survival of the fittest* vorgeht und nur die Nachkommen mit den besten Eigenschaften zu Eltern werden dürfen, findet im Laufe der Generationen eine Optimierung der Eigenschaften statt. Die Qualität der Eigenschaften eines Elters oder Nachkommens wird bei evolutionären Algorithmen als *Fitness* bezeichnet.

Damit dieses Evolutionsprinzip auf Optimierungsprobleme anwendbar ist, muss die Problemrepräsentation in geeigneter Form vorliegen. Die Eltern– und Nachkommenpopulation bestehen aus so genannten Individuen. Ein Individuum repräsentiert einen Punkt im Suchraum. Zur Bewertung von Individuen muss eine Fitnessfunktion existieren, die aus den Eigenschaften eines Individuums die jeweilige Fitness bestimmt. Zur Erzeugung der anfänglichen Population der Eltern muss eine Initialisierungsmethode bestehen.

Durch die Reproduktion der Eltern werden die Gene verändert, was auf das Optimierungsproblem bezogen, einer „Bewegung“ im Suchraum entspricht. Daher müssen für einen evolutionären Algorithmus (genetische) Operatoren existieren, die auf die Individuen anwendbar sind.

Durch den Selektionsprozess der Nachkommen wird dem Optimierungsprozess eine Richtung verliehen. Die ausgewählten Nachkommen, die die neue Elterngeneration bilden, werden abhängig von ihrer Fitness ausgewählt. Zur Durchführung dieser Selektion wird für den evolutionären Algorithmus ein entsprechender Selektionsoperator benötigt.

Generell lässt sich ein evolutionärer Algorithmus durch verschiedene Parameter charakterisieren. *Exogene Parameter* bestimmen insbesondere die Populationsgrößen des Algo-

rithmus, also die Anzahl der Eltern und Nachkommen, und bleiben in der Regel konstant. *Endogene Parameter* wirken sich auf die genetischen Operationen aus und können während eines Optimierungsprozesses variiert werden.

In dieser Arbeit werden zur Optimierung von Laufmodellen für Roboter nur Evolutionsstrategien verwendet. Daher wird im Folgenden das grundlegende Prinzip der Evolutionsstrategie genauer erläutert. Auf die übrigen Evolutionären Algorithmen wird hier nicht näher eingegangen. Ausführliche theoretische Abhandlungen der Evolutionsstrategie sind in den Standardwerken [48, 55] gegeben.

3.1 Die Standard Evolutionsstrategie

Evolutionsstrategien wurden in den sechziger Jahren von Rechenberg und Schwefel entwickelt [47, 52]. Ursprünglich wurden die Evolutionsstrategien als Hilfsmittel zur experimentellen Optimierung eingesetzt, um praktische technische Probleme zu optimieren. Dabei wurden eine variable Tragfläche in einem Windkanal, sowie die Form eines Winkelstücks eines Rohres für den Flüssigkeitstransport optimiert.

Evolutionsstrategien eignen sich zum Auffinden von Optima von Funktionen $F(\mathbf{x})$. Für den d-dimensionalen Parametervektor $\mathbf{x}$ gilt in der Regel $\mathbf{x} = (x^{(1)} \cdots x^{(d)})^T \in \mathbb{R}^d$, aber auch andere Suchräume sind durch Anpassung der genetischen Operatoren denkbar. Die einzelnen Komponenten $x^{(l)}$ mit $l = 1, \ldots, d$ des Parametervektors $\mathbf{x}$ werden als *Entscheidungsvariable* oder als *Objektparameter* bezeichnet.

Jedes Individuum $\mathbf{a}_k$ einer Population $\mathcal{P}$ beinhaltet neben einer Belegung der Entscheidungsvariablen $\mathbf{x}_k$ auch eine Menge endogener Strategieparameter $\mathbf{s}_k$. Die endogenen Strategieparameter beeinflussen die Stärke des genetischen Mutationsoperators, der auf das Individuum einwirkt. Auf diese Weise kann die Suche zwischen eher lokaler oder globaler Charakteristik gesteuert werden. Weiterhin enthält jedes Individuum die zugehörige Fitness $F_k := F(\mathbf{x}_k)$. Für jedes Individuum $\mathbf{a}_k$ gilt also

$$\mathbf{a}_k := (\mathbf{x}_k, \mathbf{s}_k, F_k). \tag{3.1}$$

Die Elternpopulation $\mathcal{P}_e^{(t)}$ der Generation t besteht aus μ Individuen. Aus der Elternpopulation geht die Nachkommenpopulation $\mathcal{P}_n^{(t)}$ mit λ Individuen hervor. Die Parameter μ und λ, die die Populationsgrößen bestimmen, gehören zu den *exogenen Strategieparametern*. Die exogenen Strategieparameter werden initial in Abhängigkeit der erwarteten Charakteristik und Dimensionalität des Suchraumes gewählt und bleiben bei der hier erläuterten Standard Evolutionsstrategie konstant. Eine generelle optimale Einstellung für μ und λ kann nicht gegeben werden. Prinzipiell lässt sich aber sagen, dass je größer die Dimensionalität der Zielfunktion ist, umso größer sollte auch μ gewählt werden. Der Verhältnis λ/μ wird in der Literatur als *Selektionsdruck* bezeichnet. Ein zu geringer Selektionsdruck kann zu einer sehr langsamen Konvergenz oder sogar zur Divergenz der Evolutionsstrategie führen.

Ein zu hoher Selektionsdruck hingegen birgt die Gefahr der frühzeitigen Konvergenz zu einem lokalen Optimum in sich, da die momentan besten Individuen die Elternpopulation dominieren können und so die Vielfalt in der Population verloren gehen kann. Schwefel empfiehlt einen Selektionsdruck im Verhältnis $\lambda/\mu = 5, \ldots, 7$, dessen Verhältnis im Übrigen keinesfalls ganzzahlig sein muss [55].

```
1   Begin
2     t:=0;
3     initialisiere P_e^(0) := {a_1^(0), ..., a_μ^(0)};
4     Repeat
5       For k:=1 To λ Do
6       Begin
7         E_k^(t):=verheirate(P_e^(t), ρ);
8         s̃_k:=Strategieparameter_Rekombination(E_k);
9         x̃_k:=Objektvariablen_Rekombination(E_k);
10        s_k:=Strategieparameter_Mutation(s̃_k);
11        x_k:=Objektvariablen_Mutation(x̃_k, s_k);
12        F_k:=F(x_k);
13      End;
14      P_n^(t):={(x_1, s_1, F_1), ..., (x_λ, s_λ, F_λ)};
15      if (μ , λ)–ES then P_e^(t+1) :=Selektion(P_n^(t), μ);
16      if (μ + λ)–ES then P_e^(t+1) :=Selektion(P_n^(t) ∪ P_e^(t), μ);
17      t:=t+1;
18    Until Abbruchbedingung;
19  End
```

Abbildung 3.2: Pseudo–Code einer $(\mu/\rho \dot{+} \lambda)$ Evolutionsstrategie

Bevor auf die Operatoren von Evolutionsstrategien näher eingegangen wird, soll die Funktionsweise anhand des Pseudo–Codes aus Abbildung 3.2 verdeutlicht werden. Zunächst werden die Individuen $a_1^{(0)}, \ldots, a_\mu^{(0)}$ der ersten Elterngeneration $\mathcal{P}_e^{(0)}$ initialisiert (Zeile 3). Diese Initialisierung kann zufällig oder, falls ein Vorwissen über das Problem vorliegt, mit aussichtsreichen Belegungen der Objektvariablen geschehen. In der „Generationsschleife“ (Zeilen 4–13) werden zunächst λ Individuen der Nachkommenpopulation $\mathcal{P}_n^{(t)}$ aus jeweils ρ Eltern erzeugt. Für jedes Individuum werden sowohl die Strategieparameter als auch die Objektparameter durch Rekombination der Parameter der Eltern und anschließende Mutation erzeugt. Für jedes Nachkommen wird in Zeile 12 die Fitness bestimmt und zugewiesen. Aus der erzeugten Nachkommengeneration $\mathcal{P}_n^{(t)}$ werden abhängig von dem Selektionstyp die Individuen der nächsten Elternpopulation $\mathcal{P}_e^{(t+1)}$ bestimmt (Zeile 15 und 16). Ist ein Abbruchkriterium noch nicht erreicht, wird die nächste Generation erzeugt.

Folgende Kriterien lassen sich als Abbruchbedingung nutzen:

- Erreichen eines vorgegebenen Fitnesswertes,

- Konvergenz von Fitnesswerten,
- Konvergenz von Objektparametern,
- Konvergenz von Strategieparametern,
- Erreichen einer maximal vorgegebenen Generationenanzahl oder
- Überschreiten einer vorgegebenen Rechenzeit.

Im Folgenden werden die weiteren Einstellungen der hier genutzten $(\mu/\rho \dot{+} \lambda)$ Evolutionsstrategie eingegangen.

3.1.1 Selektion

Der Selektionsoperator bestimmt die Elternpopulation der nächsten Generation $\mathcal{P}_e^{(t+1)}$ aus den Individuen der Nachkommenpopulation $\mathcal{P}_n^{(t)}$. Durch die Auswahl der Individuen aus $\mathcal{P}_n^{(t)}$ mit der besten Fitness wird dem Optimierungsprozess durch die Selektion eine Richtung im Suchraum verliehen. Bei Evolutionstrategien werden häufig zwei Selektionsoperatoren eingesetzt: die *Komma–Selektion* oder die *Plus–Selektion*.

Bei einer Komma–Strategie $(\mu/\rho, \lambda)$ werden aus den λ Nachkommen aus $\mathcal{P}_n^{(t)}$ die μ Individuen mit der besten Fitness bestimmt. Diese Auswahl ergibt die Elternpopulation der nächsten Generation $\mathcal{P}_e^{(t+1)}$. Für die Komma–Strategie gilt sinnvollerweise $\lambda > \mu$.

Die Plus–Selektion $(\mu/\rho + \lambda)$ wählt die μ Individuen mit der besten Fitness für die nachfolgende Elternpopulation $\mathcal{P}_e^{(t+1)}$ aus der Nachkommenpopulation $\mathcal{P}_n^{(t)}$ **und** der Elternpopulation $\mathcal{P}_e^{(t)}$ aus. Einschränkungen für die Größe von λ und μ existieren bei der Plus–Strategie wegen $\mu < \mu + \lambda$ nicht.

Da Plus–Strategien das Überleben des bislang besten Individuums garantieren, ist die Gefahr gegeben, dass sie in lokalen Optima „steckenbleiben“. Außerdem kann ein fälschlicherweise (z. B. durch Messfehler) zu gut bewertetes Individuum sehr lange (oder für immer) überleben und so die Strategie in eine falsche, nicht optimale Richtung lenken. Komma–Strategien lassen im Gegensatz zu einer Plus–Strategie auch Verschlechterungen von Generation zu Generation zu. Durch diese Eigenschaft des Vergessens der letzten Elterngeneration können Komma–Strategien auch Berge einer Fitnesslandschaft auf dem Weg zum (globalen) Optimum überwinden. Zu gute falsch bewertete Individuum stellen für die Komma–Selektion auch nur ein kleines Risiko dar, da sie nicht in die nächste Elterngeneration übernommen werden. Eine Kombination aus Plus– und Komma–Strategie wird geschaffen, indem den Eltern eine maximale Lebensdauer von κ Generationen erlaubt wird.

3.1.2 Mutation und Anpassung

Die Hauptrolle bei der genetischen Veränderung von Individuen spielt der Mutationsoperator. Er sollte die Eigenschaften der *Erreichbarkeit*, *Driftlosigkeit* und *Skalierbarkeit* aufweisen [5]. Erreichbarkeit bedeutet, dass jeder Punkt $(\widetilde{\mathbf{x}}, \widetilde{\mathbf{s}})$ nach einer endlichen Anzahl von Mutationen von $(\mathbf{x}_e, \mathbf{s}_e)$ aus erreicht werden kann. Die Driftlosigkeit wird empfohlen, da durch die Mutation keine Richtung in dem Suchraum vorgegeben sein soll. Das Einschlagen einer vielversprechenden Suchrichtung sollte ausschließlich durch den Selektionsoperator geschehen. Die Forderung nach Skalierbarkeit ist nötig, um die bevorzugte Schrittweite der Mutation anpassen zu können.

Für reellwertige Suchräume erfüllt eine normalverteile Zufallsvariable die geforderten Eigenschaften. Daher bildet für diese Suchräume in der Regel eine Normalverteilung

$$\mathcal{N}(\nu, \sigma^2) \quad \text{mit Dichte} \quad \phi(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\frac{(x-\nu)^2}{\sigma^2}\right) \tag{3.2}$$

die Basis für den Mutationsoperator. ν gibt den Erwartungswert und σ die Standardabweichung der Normalverteilung an.

Eine Mutation eines Wertes $x^{(i)} \in \mathbb{R}$ erfolgt durch Addition eines $\sigma \cdot \mathcal{N}(0,1)$–verteilten Zufallswertes $z^{(i)}$, die Standardabweichung σ entspricht der Mutationsstärke, d. h.

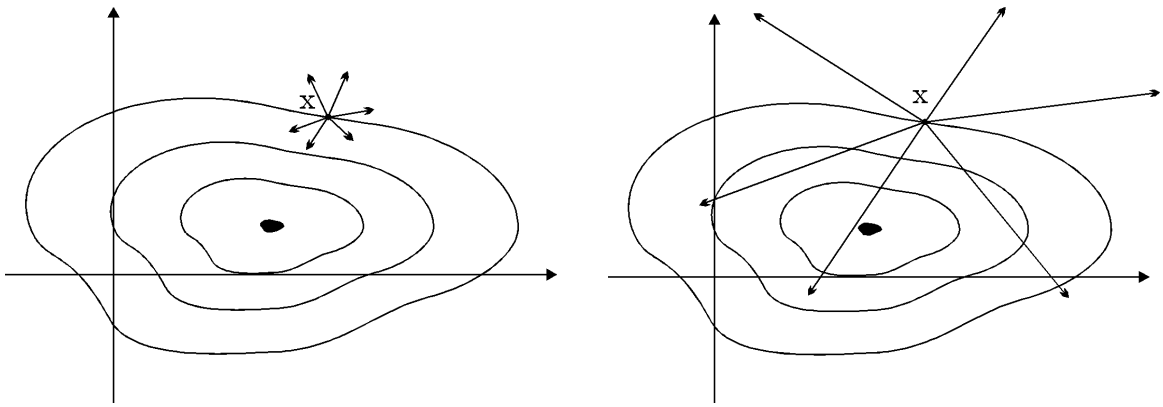
$$\widetilde{x}^{(i)} := x^{(i)} + z^{(i)}. \tag{3.3}$$

Die Normalverteilungsdichte des zufälligen Mutanten $\widetilde{x}^{(i)}$ lautet dann

$$\phi(\widetilde{x}^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\frac{(\widetilde{x}^{(i)} - x^{(i)})^2}{\sigma^2}\right). \tag{3.4}$$

Offensichtlich werden bei der Mutation mittels Normalverteilung kleine Veränderungen in Abhängigkeit von σ gegenüber großen Veränderungen bevorzugt ausgeführt. Auf die biologische Mutation bezogen, entspräche dies in etwa dem Sprichwort „Der Apfel fällt nicht weit vom Stamm“.

Die benutzte Mutationsstärke spielt eine wichtige Rolle bei Evolutionsstrategien im Bezug auf die Konvergenzgeschwindigkeit. Ist die Mutationsstärke zu groß, besteht die Gefahr, dass die erzeugten Individuen über das gesuchte Optimum „hinausschießen“. Bei zu kleiner Mutationsstärke hingegen ist die Annäherung an das Optimum nur sehr langsam möglich. Außerdem besteht die Gefahr, dass durch zu kleine Veränderungen lokale Optima nicht mehr verlassen werden können. Abbildung 3.3 veranschaulicht dies. Die geschlossenen Linien deuten einen Bereich gleicher Fitness an. Das Optimium ist durch die mittlere schwarze Fläche gekennzeichnet. Der Punkt x wird in Abbildung 3.3a, angedeutet durch die Pfeile, mit einer kleinen Mutationsstärke variiert, in Abbildung 3.3b hingegen mit größerer Varianz. Bliebe die Mutationsstärke nun immer groß, wäre es sehr schwierig, das Optimum



(a) Mutation mit kleiner Mutationsstärke (b) Mutation mit großer Mutationsstärke

Abbildung 3.3: Mutation eines Punktes x im zweidimensionalen Suchraum

zu erreichen, bei zu kleiner Mutation hingegen sind die Schritte in Richtung Optimum sehr klein und gegebenenfalls kann so ein lokales Optimum nicht mehr verlassen werden.

Eine Möglichkeit zur Einstellung der Mutationsstärke ist die so genannte 1/5–Regel [48]. Durch theoretische Analysen der $(1+1)$ Evolutionsstrategie wurde geschlussfolgert, dass in vielen Suchräumen bei einer Erfolgswahrscheinlichkeit von 1/5 die Fortschrittsgeschwindigkeit und somit die Mutationsstärke optimal eingestellt ist. Eine größere Erfolgswahrscheinlichkeit als 1/5 bedeutet, dass die Individuen nah am Optimum sind, die Mutationsstärke muss also verkleinert werden. Der umgekehrte Fall bedeutet, dass das Optimum weiter entfernt ist, daher muss die Mutationsstärke vergrößert werden.

Die 1/5–Regel ist gut geeignet zur Lösung von unimodalen Problemräumen. Bei Problemen mit mehreren (lokalen) Optima hingegen kann es passieren, dass durch das Herunterregeln der Mutationsstärke in einem lokalen Optimum dieses nicht mehr verlassen werden kann.

Einen wesentlich flexibleren Ansatz zur Anpassung der Mutationsstärke ist die so genannte (selbst–)adaptive Schrittweitenregelung [53]. Bei dieser Regelung werden die Strategieparamter (Mutationsstärken) der Individuen ebenfalls dem Evolutionsprozess unterworfen. Durch die Güte der entstandenen Individuen im Evolutionsprozess werden die so erzeugten Schrittweiten indirekt belohnt. Es sei hier erwähnt, dass diese Selbstanpassung bei Komma-Strategien prinzipiell besser arbeitet als bei Strategien, die nach der Plus-Selektion vorgehen.

Nachdem die grundlegenden Hintergründe der Mutation geklärt sind, wird auf die möglichen Realisierungen eingegangen.

Einfache Mutation

Bei der einfachen Mutation wird zur Mutation aller Objektvariablen **eine** Mutationsstärke σ benutzt. Für die Mutation der Objektparameter folgt daher

$$\widetilde{\mathbf{x}} := \mathbf{x} + \mathbf{z} \tag{3.5}$$

$$z^{(l)} \text{ sind unabhängig } \mathcal{N}(0, \sigma^2)\text{–verteilt.} \tag{3.6}$$

Bei einer Evolutionsstrategie mit (selbst–)adaptiver Schrittweitenregelung erfolgt eine Mutation $\widetilde{\sigma} = c\sigma$ der Strategievariablen σ mit einer logarithmisch-normalverteilten Variation c, d. h.

$$\ln c \text{ ist } \mathcal{N}(0, \tau^2)\text{–verteilt.} \tag{3.7}$$

Die Strategieparameter werden lognormalverteilt mutiert, da keine negative Schrittweiten auftreten dürfen.

Der Parameter τ gehört zu den exogenen Strategieparametern und lässt sich als Lernparameter interpretieren. Theoretische und praktische Untersuchungen haben gezeigt, dass τ von der Größenordnung $\tau = 1/\sqrt{N}$, bzw. $\tau = 1/\sqrt{2N}$ für stark multimodale Fitnesslandschaften sein sollte [4, 52].

Mutation mit n Einzelschrittweiten

Diese Art der Mutation benutzt zur Mutation jeder Objektvariablen eine eigene Mutationsstärke. In der Mutation $\widetilde{\mathbf{x}} := \mathbf{x} + \mathbf{z}$ ist dann komponentenweise $\widetilde{x^{(l)}} = x^{(l)} + z^{(l)}$ und

$$z^{(l)} \text{ sind unabhängig } \mathcal{N}(0, \sigma_l)\text{–verteilt.} \tag{3.8}$$

Durch die verschiedenen Mutationsschrittweiten für jede Koordinatenrichtung entsteht parallel zu den Koordinatenachsen ein Mutationselipsoid. Der Strategieparametervektor $\mathbf{s}$ entspricht dementsprechend $\mathbf{s} = (\sigma_1 \cdots \sigma_d)^T$.

Bei dieser Variante erfolgt die Selbstanpassung der Mutationsschrittweiten durch ein zufälliges $\widetilde{\mathbf{s}} = c_0(c_1\sigma_1 \cdots c_d\sigma_d)^T$. Dabei ist

$$\ln c_0 \; \mathcal{N}(0, \tau_0^2)\text{–verteilt} \tag{3.9}$$

sowie

$$\ln c_l \; \mathcal{N}(0, \tau^2)\text{–verteilt.} \tag{3.10}$$

Die Lernparameter sollten als

$$\tau_0 = \frac{c}{\sqrt{2N}}$$

und

$$\tau = \frac{c}{\sqrt{2\sqrt{N}}}$$

für große N gewählt werden [2]. Für eine $(10, 100)$–Strategie ist $c = 1$ eine vernünftige Wahl [53]. Der Parameter τ wirkt sich auf die Mutabilität der einzelnen Objektvariablen aus. Damit eine Ausdifferenzierung unterschiedlicher Mutationsschrittweiten nicht behindert wird, sollte in jedem Falle die Relation $\tau_0 < \tau$ eingehalten werden.

3.1.3 Rekombination

Durch die Rekombination wird ein Nachkomme aus ρ Elternindividuen erzeugt. Der Parameter ρ gibt somit die Größe einer Familie an und gehört ebenfalls zu den exogenen Strategieparametern. Für den Fall $\rho = 1$ wird das Elternindividuum geklont. Bei der Rekombination werden sowohl die Objektparameter als auch die Strategieparameter der Eltern miteinander rekombiniert. Unterschieden wird grundsätzlich zwischen *diskreter* (oder auch *dominanter*) und *intermediärer* Rekombination.

Sei ein Elternindividuum gegeben durch $\mathbf{e} = (e_1 \cdots e_k)^T$. Der Vektor $\mathbf{e}$ enthält sowohl die Objekt–, als auch die endogenen Strategieparameter. Die einzelnen Komponenten n_i des Nachkommenindividuums $\mathbf{n} = (n_1 \cdots n_k)^T$ werden bei der diskreten Rekombination bestimmt durch die zufällige Auswahl einer Komponente e_i aus allen ρ Eltern $\mathbf{e}_1, \ldots, \mathbf{e}_\rho$. Die intermediäre Rekombination hingegen bildet n_i aus dem arithmetischen Mittelwert der e_i aller Eltern $\mathbf{e}_1, \ldots, \mathbf{e}_\rho$.

Damit ergeben sich folgende Möglichkeiten zur Rekombination. Die Objekt– und Strategieparameter können auch durch unterschiedliche Varianten rekombiniert werden.

- Keine Rekombination
- Diskrete Rekombination von ρ Eltern
- Intermediäre Rekombination von ρ Eltern

3.2 Modellgestützte Evolutionsstrategien

Evolutionäre Algorithmen sind generell ein sehr mächtiges Werkzeug zur Optimierung. Erforderlich dazu sind allerdings gegebenenfalls sehr viele Auswertungen der Fitnessfunktion. Für jedes Individuum der Nachkommenpopulation muss die Fitness bestimmt werden. In einigen Fällen ist die Auswertung der Fitness nicht trivial. Sie kann z. B. sehr rechenaufwändig, nur experimentell bestimmbar oder auch im monetären Sinne teuer sein. Für die Optimierung der Parameter des Laufmodells eines Roboters beispielsweise muss der Roboter für die Bestimmung der Fitness den Lauf mit den entsprechenden Parametern ausprobieren. Dies ist einerseits sehr zeitaufwändig, andererseits auch sehr verschleißfördernd. Insbesondere Parameter, die in einem stolpernden Laufen oder gar in einem Umfallen resultieren, können den Roboter beschädigen.

Das Problem der vielen Fitnessauswertungen wird mit der Dimensionalität des Suchraumes noch verschärft. Für hochdimensionale Suchräume ist eine gute Exploration des Raumes nur mit einer hohen Zahl von Nachkommen λ möglich. Daher sind hier sehr viele Fitnessauswertungen zu erwarten, bis ein akzeptables oder globales Optimum erreicht ist.

Als Ausweg bieten sich modellgestützte Optimierungsmethoden an, die die Fitness von Individuen, basierend auf Meta-Modellen, schätzen. Wenn die Auswertung der Fitness mit dem Modell schneller und kostengünstiger zu realisieren ist, kann der Einsatz dieser Modelle eine Zeit- bzw. Kostenersparnis bedeuten. Da die eingesetzten Modelle aber in der Regel die tatsächliche Fitnessfunktion nur unzureichend approximieren können, muss eine Regelung benutzt werden, die möglichst ausschließt, dass die Strategie gegen ein falsches Optimum konvergiert. Abbildung 3.4 zeigt einen solchen Fall für eine eindimensionale Fitnessfunktion.

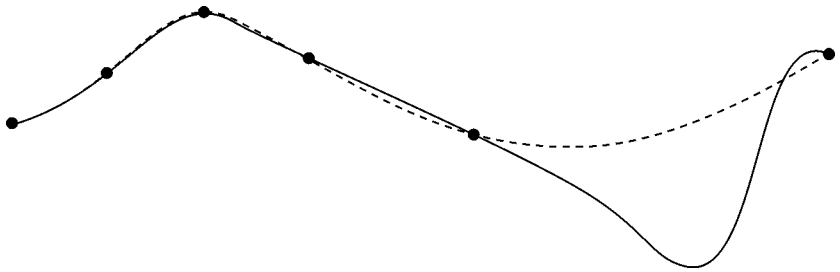


Abbildung 3.4: Die approximierte Funktion (gestrichelt) kann das reale Optimum der echten Fitnessfunktion (durchgehende Linie) nicht korrekt darstellen

Die reale Fitnessfunktion ist an den eingezeichneten sechs Stellen (mit einem Punkt gekennzeichnet) bekannt und wird durch die gestrichelt dargestellte Funktion approximiert. Die reale Fitnessfunktion ist mit der durchgehenden Linie dargestellt. Wenn die Fitnessauswertung nun nur basierend auf dem Modell erfolgt, konvergiert die Strategie gegen ein falsches Minimum. Andererseits führt das Modell aber schon in die richtige Richtung des realen globalen Minimums.

Um eine Konvergenz gegen ein falsches Optimum der approximierten Fitnessfunktion zu vermeiden bzw. das Risiko zu minimieren, müssen offensichtlich einige Individuen, basierend auf der realen Fitnessfunktion, ausgewertet werden. Die Aufgabe des so genannten *Modellmanagements* ist nun, die Individuen auszuwählen, deren Fitness von dem realen Problem bewertet werden müssen.

3.2.1 Modellmanagement

Prinzipiell wird zwischen generationenbasiertem [46] und individuenbasiertem Modellmanagement unterschieden. Bei dem generationsbasierten Management werden die Individuen der ersten Generation von der realen Fitnessfunktion bewertet und mit jeder Fitnessauswertung wird ein weiterer Punkt im Suchraum bekannt. Die folgenden Generationen werden

von dem Modell, basierend auf den bekannten Fitnesswerten, bestimmt, bis eine Konvergenz eintritt. Nachfolgend wird wieder eine Generation von der realen Fitnessfunktion bewertet, wodurch wiederum mehr Stützpunkte für das Modell gesammelt werden. Die folgenden Generationen werden schließlich wieder von dem nun besseren Modell bewertet. Dieses Verfahren wird abwechselnd durchgeführt, bis eine zufriedenstellende Lösung gefunden ist.

Bei dem individuenbasierten Modellmanagement hingegen werden einige Nachkommen von dem Modell, der Rest von der realen Fitnessfunktion bewertet. Vier Varianten dieser Steuerung werden im Folgenden miteinander verglichen und kurz erläutert [17].

Best Strategy

Bei der *Best Strategy* Regelung [31] wird die Fitness aller λ Nachkommen zuerst von dem Modell bewertet. Die $\lambda^* \leq \lambda$ Individuen mit der von dem Modell bewerteten besten Fitness werden noch einmal mit der realen Fitnessfunktion bewertet und nachdem diese neu gewonnen realen Fitnesswerte in das Modell integriert wurden, werden die übrigen $\lambda - \lambda^*$ Individuen noch einmal mit dem Modell bewertet. Dies geschieht, da davon ausgegangen wird, dass das Modell nun durch die Hinzunahme der weiteren Stützpunkte bessere Fitnessapproximationen liefern kann. Die besten μ Individuen werden schließlich aus allen λ Nachkommen als Eltern für die nächste Generation ausgewählt. Bei dieser Regelung kann eine Konvergenz gegen ein falsches Optimum nicht gänzlich ausgeschlossen werden, da die Selektion aus allen λ Nachkommen erfolgt, ohne Unterscheidung, ob das Individuum real oder durch das Fitnessmodell bewertet wurde.

Pre-Selection

Bei diesem Verfahren [66] (im Folgenden auch Vorselektion genannt) wird eine größere Anzahl $\lambda' > \lambda$ von Nachkommen erzeugt, die zunächst von dem Modell bewertet werden. Die λ besten Individuen werden anschließend mit der realen Fitnessfunktion bewertet und die hier neu gewonnen bekannten Stützpunkte im Suchraum in das Fitnessmodell integriert, wodurch von Generation zu Generation die Qualität des Modells verbessert wird. Die Auswahl der μ Eltern erfolgt nur aus den λ Individuen, die von der realen Fitnessfunktion bewertet wurden. Die erhoffte Beschleunigung der Konvergenz erfolgt, da durch eine größere Menge von λ' Nachkommen der Suchraum besser exploriert werden kann und die aussichtsreichsten Individuen von dem Modell vorselektiert worden sein sollten. Eine Konvergenz gegen ein falsches Optimum wird bei dieser Variante ausgeschlossen. Der Ablauf der Evolutionsstrategie mit Vorselektion ist zur Veranschaulichung in Abbildung 3.5 noch einmal visualisiert.

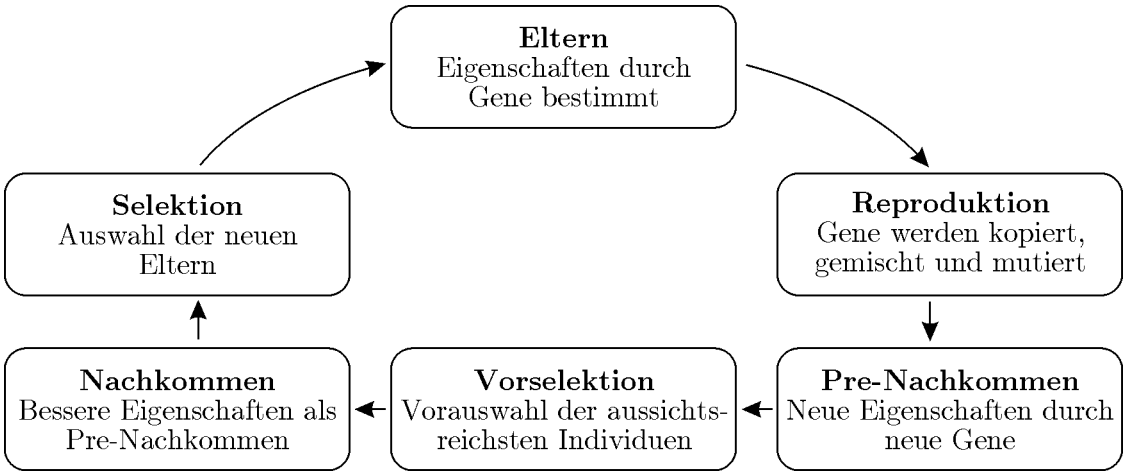


Abbildung 3.5: Evolutionskreislauf mit Vorselektion

Clustering Technique

Diese Variante [32] generiert ebenfalls λ Nachkommen, die anschließend mittels des k-means Algorithmus in λ^* viele Cluster eingeteilt werden. Die λ^* Individuen, die am nächsten zum Zentrum des jeweiligen Clusters liegen, werden von der realen Fitnessfunktion bewertet. Die ermittelten Fitnesswerte werden wiederum in das Modell zur Verbesserung der Qualität integriert und die Fitness der restlichen $\lambda - \lambda^*$ Individuen von dem Modell approximiert. Die μ Eltern der Folgegeneration werden aus allen λ Individuen selektiert.

Clustering Technique kombiniert mit Best Strategy

Dieses Modellmanagement stellt eine Variante der zuvor erläuterten Clustering Technique dar. Die λ Nachkommen werden ebenfalls in λ^* Cluster eingeteilt und basierend auf dem Modell bewertet. Anstatt der am nächsten zum Mittelpunkt des Clusters liegenden Individuen werden die λ^* Individuen mit der besten Fitness für eine reale Bewertung ausgewählt. Wie zuvor werden die neuen Stützpunkte in das Modell integriert und die $\lambda - \lambda^*$ Individuen neu bewertet. Die Selektion erfolgt ebenfalls aus allen λ Individuen.

Bewertung der Varianten

Die vier Modellmanagement Varianten wurden für die drei gängigen jeweils 10-dimensionalen Testfunktionen *Sphere*, *Rosenbrock* und *Ackley* verglichen [17]. Nach 1200 realen Fitnessauswertungen wurde die Optimierung der Zielfunktion jeweils abgebrochen. Zur Prädiktion der Fitness wurde ein Neuronales Netz nach [27] benutzt und nach Integration neuer Stützpunkte jeweils neu trainiert. Die modellgestützten Varianten konvergieren im Mittel schneller als die Standard Evolutionsstrategie. Bei über 20 Testläufen mit verschiedenen

zufällig gewählten Startpunkten variierten die Ergebnisse der Varianten *Best Strategy*, *Clustering Technique* und *Clustering Technique kombiniert mit Best Strategy* jedoch jeweils sehr stark. Sie waren in einigen Durchläufen dem Ergebnis der Standard Evolutionsstrategie unterlegen. Einzig das *Pre-Selection* Modellmanagement wies eine sehr geringe Varianz der gefundenen Optima auf und konnte nach 1200 Fitnessbewertungen fast immer ein besseres Ergebnis als die Standard Evolutionsstrategie finden. Aufbauend auf diesen Ergebnissen wird in dieser Arbeit im Folgenden das *Pre-Selection* Modellmanagement verwendet.

3.2.2 Modellierung des Suchraumes

Während des Optimierungsprozesses liefern die gemessenen (skalaren) Fitnesswerte Informationen über die Struktur des Suchraumes. Basierend auf den zuvor gemessenen Fitnesswerten soll das Modell in der Lage sein, Werte an nicht zuvor gemessenen Stellen vorherzusagen. Mathematisch gesehen muss das Modell die Funktion $y : \mathbb{R}^d \rightarrow \mathbb{R}$, basierend auf n bekannten (gemessenen) Funktionswerten $y_1 = y(\mathbf{x}_1), \ldots, y_n = y(\mathbf{x}_n)$ aus der Datenbasis $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ für jeden Punkt $\mathbf{x} = (x^{(1)} \cdots x^{(d)})^T \notin D$ approximieren können. Der von dem Modell geschätzte Funktionswert wird mit $\hat{y}(\mathbf{x})$ bezeichnet und soll möglichst nahe an dem realen (unbekannten) Funktionswert $y(\mathbf{x})$ liegen.

Prinzipiell wird zwischen lokalen und globalen Modellen unterschieden. Globale Modelle sind für den gesamten Suchraum gültig und können den Funktionswert jedes Punktes $\mathbf{x}$ schätzen. Sie müssen nur neu berechnet werden, wenn neue Stützpunkte zu der Datenbasis D hinzukommen. Bei der Fitnessprädiktion für Evolutionäre Strategien wäre dies nach jeder Generation der Fall, in der reale Fitnesswerte bestimmt wurden. Das größte Problem bei globalen Modellen liegt in der meist kleinen Anzahl von Stützpunkten in D bei verhältnismäßig großer Dimension des typischerweise multimodalen Suchraumes. Dies führt dann zu einer schlechten Approximation der zugrunde liegenden Fitnessfunktion.

Lokale Modelle sind nur für einen kleinen Bereich um einen Suchpunkt $\mathbf{x}$ gültig und müssen daher für jedes $\mathbf{x}$ (in Abhängigkeit der lokalen Umgebung) neu berechnet werden. Sie nutzen nur eine Teilmenge von D, da die nächsten Punkte die meiste Information über den Funktionswert von $\mathbf{x}$ liefern. Es sei noch einmal darauf hingewiesen, dass die Datenbasis D während des Evolutionsprozesses wächst. Jedes Individuum, dessen Fitness real bewertet wird, wird zu D hinzugefügt.

Nächster Nachbar Modell

Das einfachste hier behandelte Modell ist das *Nächste Nachbar Modell*. Der Funktionswert eines unbekannten Punktes $\mathbf{x}$ wird durch seinen nächsten Nachbar aus der Menge D der bekannten Punkte ermittelt. Verschiedene Distanzmaße sind denkbar, beispielsweise die

Euklidische Distanz, für die der Abstand Δ zweier Punkte $\mathbf{x}_i$ und $\mathbf{x}_j$ definiert ist durch

$$\Delta\left(\mathbf{x}_i, \mathbf{x}_j\right) = \sqrt{\sum_{l=1}^{d} \left(x_i^{(l)} - x_j^{(l)}\right)^2}. \tag{3.11}$$

Dieses Maß ist sinnvoll, wenn jede Dimension im Suchraum die gleiche Skalierung (d. h. Maßeinheit) hat und einen vergleichbaren Gradienten aufweist. Bei stark unterschiedlichen Maßeinheiten der Dimensionen würde andernfalls die Achse mit den größeren Werten das Abstandsmaß dominieren. In diesem Fall kann jede Dimension beispielsweise standardisiert werden, so dass die Werte je Dimension eine Standardabweichung von 1 haben und somit besser vergleichbar sind. Für die Abstandsberechnung muss dann jede Dimension $l = 1, \ldots, d$ mit dem Kehrwert der Standardabweichung $s^{(l)}$ der Werte $x_i^{(l)}$ für $i = 1, \ldots, n$ skaliert werden. Der Skalierungfaktor ist dann $1/s^{(l)}$.

$$\Delta\left(\mathbf{x}_i, \mathbf{x}_j\right) = \sqrt{\sum_{l=1}^{d} \left(\frac{x_i^{(l)} - x_j^{(l)}}{s^{(l)}}\right)^2}. \tag{3.12}$$

Der approximierte Funktionswert ist dann für dieses Modell gegeben durch

$$\hat{y}(\mathbf{x}) = y(\mathbf{x}_0) \quad \text{mit} \quad \mathbf{x}_0 = \arg\min_{i=1,\ldots,n} \left\{\Delta\left(\mathbf{x}_i, \mathbf{x}\right)\right\}. \tag{3.13}$$

Dieses Modell ist im Bezug auf die Rechenzeit sehr effizient. Es hat aber den Nachteil, dass es nicht zwischen mehreren Messwerten interpoliert und somit gegebenenfalls mehreren Individuen den gleichen Fitnesswert zuordnet, so dass der Selektionsoperator der Vorselektion ihre Fitness nicht unterscheiden kann.

Polynomielles Modell

Das polynomielle Modell verwendet ein Polynom, das an den ausgewählten Punkten in D die zugrunde liegende Funktion annähert. Dadurch wird eine kontinuierliche Funktion zur Fitnessprädiktion geschaffen. Zuerst wird das Prinzip der normalen Polynomregression erläutert und anschließend auf den gewichteten Fall der Polynomregression erweitert.

Ein d-dimensionales Polynom zweiten Grades ist definiert durch

$$p(\mathbf{x}) = a_0 + \sum_{l=1}^{d} a_l x^{(l)} + \sum_{l=1}^{d} \sum_{m=l}^{d} a_{lm} x^{(l)} x^{(m)} \tag{3.14}$$

$$= \begin{pmatrix} 1 & x^{(1)} & \cdots & x^{(d)} & x^{(1)}x^{(1)} & x^{(1)}x^{(2)} & \cdots & x^{(d)}x^{(d)} \end{pmatrix} \mathbf{a} \tag{3.15}$$

mit dem unbekannten Koeffizientenvektor

$$\mathbf{a} = \begin{pmatrix} a_0 & a_1 & \cdots & a_d & a_{11} & a_{12} & \cdots & a_{dd} \end{pmatrix}^T. \tag{3.16}$$

Das Problem der Regression ist nun, die Koeffizienten in $\mathbf{a}$ derart zu bestimmen, so dass $p(\mathbf{x})$ die tatsächliche Funktion $y(\mathbf{x})$ in allen zugrunde liegenden Punkten in D „so gut wie möglich“ approximiert. Dazu wird üblicherweise die quadratische Fehlerfunktion

$$Q(\mathbf{a}) = \sum_{i=1}^{n} \left(y\left(\mathbf{x}_i\right) - p\left(\mathbf{x}_i; \mathbf{a}\right) \right)^2 \tag{3.17}$$

bezüglich $\mathbf{a}$ minimiert.

Die bekannten Funktionswerte $y(\mathbf{x}_1), \ldots, y(\mathbf{x}_n)$ in D werden zu dem Vektor $\mathbf{y}$ zusammengefasst. Mit der Matrix

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & \cdots & x_1^{(d)} & x_1^{(1)}x_1^{(1)} & x_1^{(1)}x_1^{(2)} & \cdots & x_1^{(d)}x_1^{(d)} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n^{(1)} & \cdots & x_n^{(d)} & x_n^{(1)}x_n^{(1)} & x_n^{(1)}x_n^{(2)} & \cdots & x_n^{(d)}x_n^{(d)} \end{pmatrix} \tag{3.18}$$

gilt dann die Beobachtungsgleichung

$$\mathbf{y} = \mathbf{X}\mathbf{a} + \mathbf{v} \tag{3.19}$$

mit dem Fehlerterm $\mathbf{v}$, falls das Polynom die Funktionswerte in D nicht exakt approximieren kann.

Wie zuvor erwähnt, soll $Q\left(\mathbf{a}\right)$ minimiert werden und Gleichung 3.17 lässt sich in Matrixschreibweise wie folgt formulieren

$$Q(\mathbf{a}) = \sum_{i=1}^{n} \left(y\left(\mathbf{x}_i\right) - p\left(\mathbf{x}_i; \mathbf{a}\right) \right)^2 = \mathbf{v}^T\mathbf{v} = \left(\mathbf{y} - \mathbf{X}\mathbf{a}\right)^T \left(\mathbf{y} - \mathbf{X}\mathbf{a}\right). \tag{3.20}$$

Nullsetzen der partiellen Ableitung nach $\mathbf{a}$ ergibt schließlich für die Koeffizienten die Lösung

$$\hat{\mathbf{a}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{3.21}$$

und damit für den approximierten Funktionswert

$$\hat{y}(\mathbf{x}) = \begin{pmatrix} 1 & x^{(1)} & \cdots & x^{(d)} & x^{(1)}x^{(1)} & x^{(1)}x^{(2)} & \cdots & x^{(d)}x^{(d)} \end{pmatrix} \hat{\mathbf{a}}. \tag{3.22}$$

Mit dieser Methode lässt sich ein globales Modell aufstellen, wenn alle zur Verfügung stehenden Punkte aus D zur Bestimmung der Koeffizienten in $\mathbf{a}$ herangezogen werden. Ein multimodaler Suchraum lässt sich nur mit einem Polynom höheren Grades gut approximieren. Die Anzahl der nötigen Koeffizienten eines Polynom zweiten Grades der Dimension d beträgt bereits $(d+1)(d+2)/2$ und eignet sich doch nur zur Approximation einer unimodalen Funktion. Die zuvor beschriebene Methode der kleinsten Quadrate benötigt mindestens die gleiche Anzahl von Samples in D wie Koeffizienten existieren.

Das Problem, dass alle bekannten Stützstellen mit gleichem Gewicht in die Prädiktion $\hat{y}(\mathbf{x})$ eingehen, lässt sich mit einem lokalen Modell umgehen. Das obige globale Modell kann in ein lokales Modell überführt werden, indem der Einfluss der einzelnen Stützpunkte mit dem Abstand zu dem Punkt $\mathbf{x}$ gewichtet wird. Dadurch wird der Tatsache Rechnung getragen, dass Punkte nahe zu $\mathbf{x}$ mehr Informationen über seinen Funktionswert tragen als Punkte, die weiter entfernt sind. Alle Punkte aus D können dazu genutzt werden, oder nur eine Teilmenge von k Punkten, da weiter entfernte Punkte ohnehin ein Gewicht nahe 0 bekommen würden.

Die Gewichtung w_i ist eine Funktion der Distanz zwischen $\mathbf{x}$ und dem i-ten Punkt $\mathbf{x}_i$ der Datenbasis D. Wie in [25] vorgeschlagen, wird das Gewicht über die Gauß'sche Kernfunktion

$$w_i\left(\Delta\left(\mathbf{x}_i, \mathbf{x}\right)\right) = \exp\left(-\Delta^2\left(\mathbf{x}_i, \mathbf{x}\right)\right) \tag{3.23}$$

bestimmt. Die Distanz wird nach Gleichung 3.12 errechnet.

Für ein lokales Modell wird häufig ein Polynom ersten Grades benutzt, welches für einen d-dimensionalen Raum durch

$$p(\mathbf{x}) = a_0 + \sum_{l=1}^{d} a_l x^{(l)} = \begin{pmatrix} 1 & x^{(1)} & \cdots & x^{(d)} \end{pmatrix} \mathbf{a} \tag{3.24}$$

mit dem Koeffizientenvektor

$$\mathbf{a} = \begin{pmatrix} a_0 & a_1 & \cdots & a_d \end{pmatrix}^T. \tag{3.25}$$

definiert ist. Die Gewichte der k verwendeten Punkte werden in der Diagonalmatrix

$$\mathbf{W} = \begin{pmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & w_k \end{pmatrix} \tag{3.26}$$

abgelegt. Die Fehlerfunktion 3.20 ergibt sich in dem gewichteten Fall durch

$$Q(\mathbf{a}) = (\mathbf{y} - \mathbf{X} \cdot \mathbf{a})^T \mathbf{W} (\mathbf{y} - \mathbf{X} \cdot \mathbf{a}) \tag{3.27}$$

mit

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & \cdots & x_1^{(d)} \\ \vdots & \vdots & & \vdots \\ 1 & x_k^{(1)} & \cdots & x_k^{(d)} \end{pmatrix} \quad \text{und} \quad \mathbf{y} = \begin{pmatrix} y(x_1) \\ \vdots \\ y(x_k) \end{pmatrix}. \tag{3.28}$$

Die optimalen Koeffizienten ergeben sich dann durch

$$\hat{\mathbf{a}} = \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y} \tag{3.29}$$

und für die Approximation gilt

$$\hat{y}(\mathbf{x}) = \begin{pmatrix} 1 & x^{(1)} & \cdots & x^{(d)} \end{pmatrix} \hat{\mathbf{a}}. \tag{3.30}$$

Gauß'sches Prozessmodell

Ein weiteres verwendetes Modell zur Fitnessprädiktion ist der Gauß Prozess [45, 69]. Er trägt dem Umstand Rechnung, dass die Punkte $\mathbf{x}_1, \ldots, \mathbf{x}_n$, an denen y beobachtbar ist, und gegebenenfalls weitere Punkte $\mathbf{x}_{n+1}, \ldots, \mathbf{x}_{n+m}$, an denen y prognostiziert werden soll, nach einem bestimmten Zufallsprinzip ausgewählt werden. Darüberhinaus kann die Ermittlung der dazugehörigen Funktionswerte $y_1 = y(\mathbf{x}_1), \ldots, y_n = y(\mathbf{x}_n)$ sowie $y_{n+1} = y(\mathbf{x}_{n+1}), \ldots, y_{n+m} = y(\mathbf{x}_{n+m})$ (mess-) fehlerbehaftet sein. In diesem Sinne sind die Funktionswerte theoretisch als Zufallsvariable zu interpretieren, die sich nach einem bestimmten Zufallsprinzip, also nach einer bestimmten (statistischen) Verteilung realisieren.

Ist diese Verteilung die multivariate Normalverteilung, so wird vom Gauß'schen Prozessmodell gesprochen. Hier wird kurz auf die Grundlagen des Gauß Prozesses zu Prognosezwecken eingegangen. Ausführliche theoretische Details zum Gauß Prozess sind in [45, 61] gegeben.

Die n zu beobachtenden Funktionswerte werden zu dem Zufallsvektor $\mathbf{y} = (y_1 \cdots y_n)^T$ zusammengefasst. Die Aufgabe ist nun, eine Prognose für die m unbekannten Werte zu bestimmen, die analog in dem Zufallsvektor $\mathbf{y}_0 = (y_{n+1} \ldots y_{n+m})^T$ zusammengefasst sind. Unter der obigen Annahme der Normalverteilung ist die gemeinsame Verteilung von $\binom{\mathbf{y}}{\mathbf{y}_0}$ gegeben durch

$$\mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}_0\end{pmatrix}, \begin{pmatrix}\Sigma_{yy} & \Sigma_{yy_0}\\ \Sigma_{y_0y} & \Sigma_{y_0y_0}\end{pmatrix}\right) \tag{3.31}$$

mit Erwartungswerten $\boldsymbol{\mu} = \mathrm{E}(\mathbf{y})$, bzw. $\boldsymbol{\mu}_0 = \mathrm{E}(\mathbf{y}_0)$ und den Kovarianzmatrizen

$$\begin{aligned}
\Sigma_{yy} &= \mathrm{E}\left((\mathbf{y}-\boldsymbol{\mu})(\mathbf{y}-\boldsymbol{\mu})^T\right),\\
\Sigma_{y_0y} &= \mathrm{E}\left((\mathbf{y}_0-\boldsymbol{\mu}_0)(\mathbf{y}-\boldsymbol{\mu})^T\right) = \Sigma_{yy_0}^T \text{ und}\\
\Sigma_{y_0y_0} &= \mathrm{E}\left((\mathbf{y}_0-\boldsymbol{\mu}_0)(\mathbf{y}_0-\boldsymbol{\mu}_0)^T\right).
\end{aligned}$$

Für die Dichtefunktion von $\binom{\mathbf{y}}{\mathbf{y}_0}$ gilt

$$g(\mathbf{y}, \mathbf{y}_0) = \frac{1}{Z}\exp\left(-\frac{1}{2}\begin{pmatrix}\mathbf{y}-\boldsymbol{\mu}\\ \mathbf{y}_0-\boldsymbol{\mu}_0\end{pmatrix}^T\begin{pmatrix}\Sigma_{yy} & \Sigma_{yy_0}\\ \Sigma_{y_0y} & \Sigma_{y_0y_0}\end{pmatrix}^{-1}\begin{pmatrix}\mathbf{y}-\boldsymbol{\mu}\\ \mathbf{y}_0-\boldsymbol{\mu}_0\end{pmatrix}\right) \tag{3.32}$$

mit $Z = \left(\sqrt{2\pi}\right)^{n+m}\sqrt{\det\begin{pmatrix}\Sigma_{yy} & \Sigma_{yy_0}\\ \Sigma_{y_0y} & \Sigma_{y_0y_0}\end{pmatrix}}$.

Für die bedingte Dichte von $\mathbf{y}_0$ bei gegebenem $\mathbf{y}$ gilt definitionsgemäß nach dem Satz von Bayes $g(\mathbf{y}_0|\mathbf{y}) = \frac{g(\mathbf{y},\mathbf{y}_0)}{g(\mathbf{y})} = \frac{g(\mathbf{y}|\mathbf{y}_0)g(\mathbf{y}_0)}{g(\mathbf{y})}$. Einsetzen der Gleichungen für $g(\mathbf{y}, \mathbf{y}_0)$ und $g(\mathbf{y})$ ergibt dann

$$g(\mathbf{y}_0|\mathbf{y}) = \frac{\frac{1}{\left(\sqrt{2\pi}\right)^{n+m}\sqrt{\det\begin{pmatrix}\Sigma_{yy} & \Sigma_{yy_0}\\ \Sigma_{y_0y} & \Sigma_{y_0y_0}\end{pmatrix}}}\exp\left(-\frac{1}{2}\begin{pmatrix}\mathbf{y}-\boldsymbol{\mu}\\ \mathbf{y}_0-\boldsymbol{\mu}_0\end{pmatrix}^T\begin{pmatrix}\Sigma_{yy} & \Sigma_{yy_0}\\ \Sigma_{y_0y} & \Sigma_{y_0y_0}\end{pmatrix}^{-1}\begin{pmatrix}\mathbf{y}-\boldsymbol{\mu}\\ \mathbf{y}_0-\boldsymbol{\mu}_0\end{pmatrix}\right)}{\frac{1}{\left(\sqrt{2\pi}\right)^{n}\sqrt{\det\Sigma_{yy}}}\exp\left(-\frac{1}{2}\left(\mathbf{y}-\boldsymbol{\mu}\right)^T\Sigma_{yy}^{-1}\left(\mathbf{y}-\boldsymbol{\mu}\right)\right)} \tag{3.33}$$

Ausrechnen von Gleichung 3.33 unter Verwendung der Formel zur Inversion geteilter Matrizen ergibt schließlich

$$g(\mathbf{y}_0|\mathbf{y}) = \frac{1}{\left(\sqrt{2\pi}\right)^m \sqrt{\det \mathbf{P}_{\hat{\mathbf{y}}_0\hat{\mathbf{y}}_0}}} \exp\left(-\frac{1}{2}(\mathbf{y}_0 - \hat{\boldsymbol{\mu}}_0)^T \mathbf{P}^{-1}_{\hat{\mathbf{y}}_0\hat{\mathbf{y}}_0}(\mathbf{y}_0 - \hat{\boldsymbol{\mu}}_0)\right) \tag{3.34}$$

mit

$$\hat{\boldsymbol{\mu}}_0 = \boldsymbol{\mu}_0 + \Sigma_{y_0 y}\Sigma_{yy}^{-1}(\mathbf{y} - \boldsymbol{\mu}) \tag{3.35}$$

und

$$\mathbf{P}_{\hat{\mathbf{y}}_0\hat{\mathbf{y}}_0} = \Sigma_{y_0 y_0} - \Sigma_{y_0 y}\Sigma_{yy}^{-1}\Sigma_{y y_0}. \tag{3.36}$$

Die resultierende Prognose

$$\hat{\mathbf{y}}_0 = \boldsymbol{\mu}_0 + \Sigma_{y_0 y}\Sigma_{yy}^{-1}(\mathbf{y} - \boldsymbol{\mu}) \tag{3.37}$$

ist bei gegebenem $\mathbf{y}$ also wieder normalverteilt mit Erwartungswert $\hat{\boldsymbol{\mu}}_0$ und Kovarianzmatrix $\mathbf{P}_{\hat{\mathbf{y}}_0\hat{\mathbf{y}}_0}$.

Zur Bestimmung von $\hat{\mathbf{y}}_0$ müssen also die Erwartungswerte $\boldsymbol{\mu}$ und $\boldsymbol{\mu}_0$ und die Kovarianzmatrizen Σ_{yy}, $\Sigma_{y_0 y}$ und $\Sigma_{y_0 y_0}$ bekannt sein. Da dies aber in der Regel nicht der Fall ist, wird im Folgenden ein häufig genutztes Modell für die Struktur der Kovarianzmatrix beschrieben [45]. Die Erwartungswerte werden in der Praxis durch arithmetischen Mittelwerte geschätzt, bzw. durch vorherige Standardisierung auf Null gesetzt.

Eine häufige Wahl für die Kovarianzfunktion ist eine quadratisch exponentielle Struktur. Für die Einträge in den Kovarianzmatritzen gilt allgemein

$$\sigma_{ij} = \sigma^2 k_{ij}(\boldsymbol{\beta}) + \sigma_0^2 \delta_{ij} \tag{3.38}$$

mit

$$k_{ij}(\boldsymbol{\beta}) = \exp\left(-\sum_{l=1}^{d}(x_i^{(l)} - x_j^{(l)})^2 \beta_l\right) = k_{ji}(\boldsymbol{\beta}) \quad \text{für} \quad i,j = 1,\ldots,n+m. \tag{3.39}$$

Die Struktur der Kovarianzmatrix hat den Effekt, dass nahe beieinander liegende Eingaben zu einer ähnlichen Prädiktion führen. Die Parameter $\sigma^2, \sigma_0^2, \boldsymbol{\beta} = (\beta_1 \cdots \beta_d)^T$ der Kovarianzmatrix sind nun zu bestimmen. Eine Möglichkeit dazu bietet die Maximum Likelihood Methode, mit der die Dichtefunktion für $\mathbf{y}$ bezüglich der Parameter $\sigma^2, \sigma_0^2, \boldsymbol{\beta}$ maximiert wird.

Vergleich der Modelle

Die d-dimensionale multimodale Funktion

$$f(\mathbf{x}) = 20 + e - 20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{l=1}^{d} x^{(l)}x^{(l)}}\right) - \exp\left(\frac{1}{d}\sum_{l=1}^{d}\cos(2\pi x^{(l)})\right) \tag{3.40}$$

wird auch als Ackley-Funktion bezeichnet und häufig als Testfunktion für Optimierungsprobleme herangezogen. Die Testfunktion ist für den zweidimensionalen Fall im Bereich $-5 \leq x^{(l)} \leq 5$ in Abbildung 3.6 dargestellt. Das Minimum liegt bei $f(\mathbf{x}) = 0$ mit $\mathbf{x} = (0 \cdots 0)^T$.

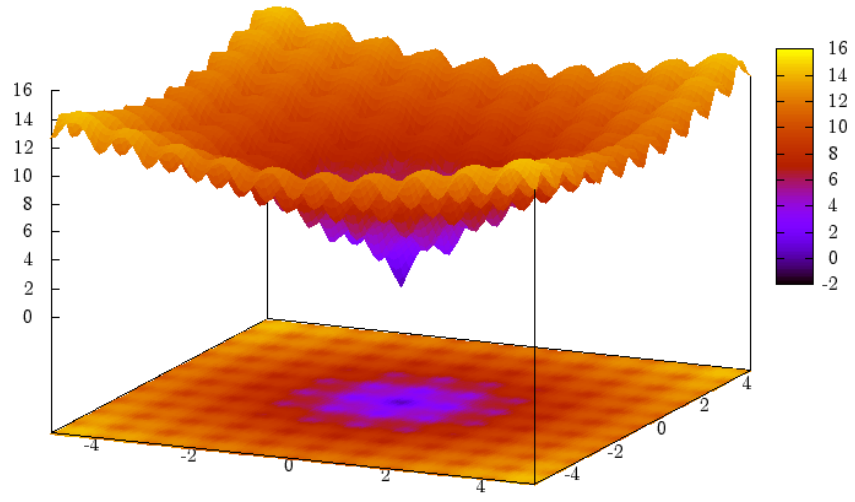


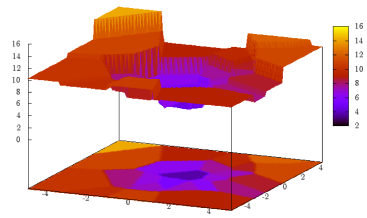
Abbildung 3.6: Die zweidimensionale Ackley Funktion

Approximationen der Ackley-Funktion mit verschiedenen Anzahlen von zufällig ausgewählten Stützstellen für die drei zuvor erläuterten lokalen Modelle

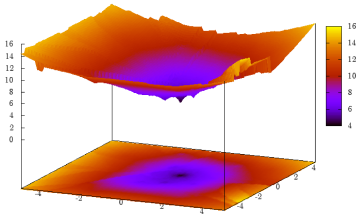
- Nächster Nachbar,
- abstandsgewichtet polynomiell und
- Gauß Prozess

sind in Abbildung 3.7 abgebildet. Für die Approximationen in der obersten Reihe standen 25 Stützpunkte zur Verfügung. Erwartungsgemäß enthält das Nächste Nachbar Modell (Abbildung 3.7a) große Plateaus mit gleicher Fitness. Das polynomielle Modell (Abbildung 3.7b) und der Gauß Prozess (Abbildung 3.7c) approximieren den Trend der Funktion für die geringe Anzahl von Stützpunkten hingegen schon recht gut, die kleinen Täler können hingegen nicht erfasst werden. Insbesondere der Gauß Prozess führt zu einer sehr glatten Funktion.

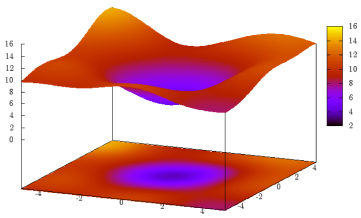
Ein ähnliches Bild stellt sich bei 100 Stützstellen dar: Während das Nächste Nachbar Modell nur Plateaus darstellen kann, ist die grobe Struktur gut von dem polynomiellen



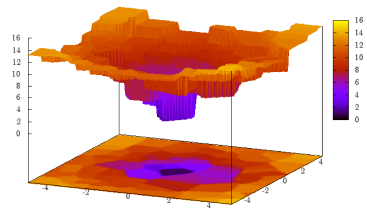
(a) Nächster Nachbar mit 25 Stützpunkten



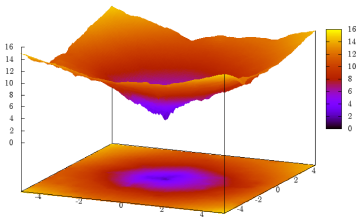
(b) Polynomiell mit 25 Stützpunkten



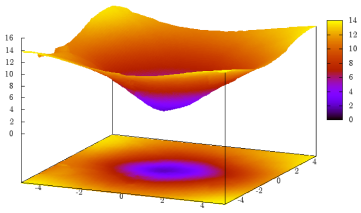
(c) Gauß Prozess mit 25 Stützpunkten



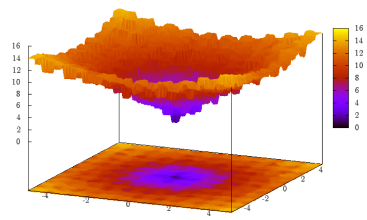
(d) Nächster Nachbar mit 100 Stützpunkten



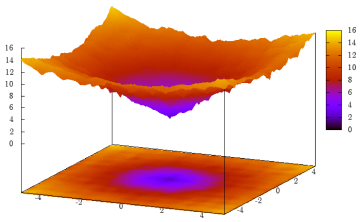
(e) Polynomiell mit 100 Stützpunkten



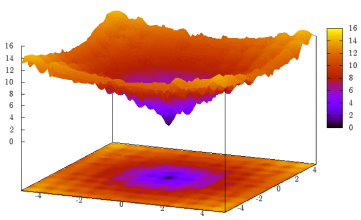
(f) Gauß Prozess mit 100 Stützpunkten



(g) Nächster Nachbar mit 1000 Stützpunkten



(h) Polynomiell mit 1000 Stützpunkten



(i) Gauß Prozess mit 1000 Stützpunkten

Abbildung 3.7: Approximation der Ackley Funktion mit verschiedenen Modellen und unterschiedlicher Anzahl der Stützstellen

Modell und dem Gauß Prozess wiedergegeben. Die Approximation mit dem Gauß Prozess ist abermals glatter und regelmäßiger als das polynomielle Modell, die Struktur der Täler ist noch nicht erfasst.

Bei 1000 Stützstellen (Abbildungen 3.7g - 3.7i) ist bei allen drei Modellen die Struktur der Täler der Ackley-Funktion erkennbar. Sie eignen sich also prinzipiell alle drei zur Approximation multimodaler Funktionen. Der Gauß Prozess scheint die Täler jedoch etwas besser modellieren zu können als das polynomielle Modell.

Welches der drei Modelle sich am besten zur Fitnessprädiktion eignet, kann aufgrund der kurzen Analyse nicht beurteilt werden. Zum einen kann es sinnvoll sein, wenn das Modell den Suchraum glättet und somit lokale Optima vermeiden kann, zum anderen kann durch diese Glättung auch das Erreichen des globalen Optimums verhindert werden. Prinzipiell

hängt die Qualität des Modelles auch von der Struktur des Suchraumes ab. Der Suchraum der Kugelfunktion $f(\mathbf{x}) = \sum_{l=1}^{d} x^{(l)} x^{(l)}$ z. B. könnte durch ein polynomielles Modell zweiten Grades exakt approximiert werden. Selbst das augenscheinlich schwächste Nächste Nachbar Modell kann im Falle der Treppenfunktion $f(\mathbf{x}) = \sum_{l=1}^{d} |x^{(l)}|$ den anderen Modellen überlegen sein. Prinzipiell kann festgestellt werden, dass sich das Modell möglichst flexibel an den Suchraum anpassen können muss. Eine weitere Bewertung der drei Modelle im Bezug auf die Fitnessprädiktion für das Laufenlernen eines Roboters wird später in Kapitel 5.3 vorgestellt.

3.2.3 Bewertung der Modellqualität

Theoretisch konvergiert die modellgestützte Evolutionsstrategie bereits schneller als die Standard Evolutionsstrategie, wenn die Vorselektion die Individuen besser als eine zufällige Selektion auswählt [66]. Ein Vorselektionsprozess, der hingegen schlechter als eine eine zufällige Auswahl selektiert, kann zu einer Verschlechterung der Konvergenzgeschwindigkeit führen oder sogar ein Erreichen von Optima verhindern, wenn diese real optimalen Individuen den Vorselektionsprozess nicht überleben.

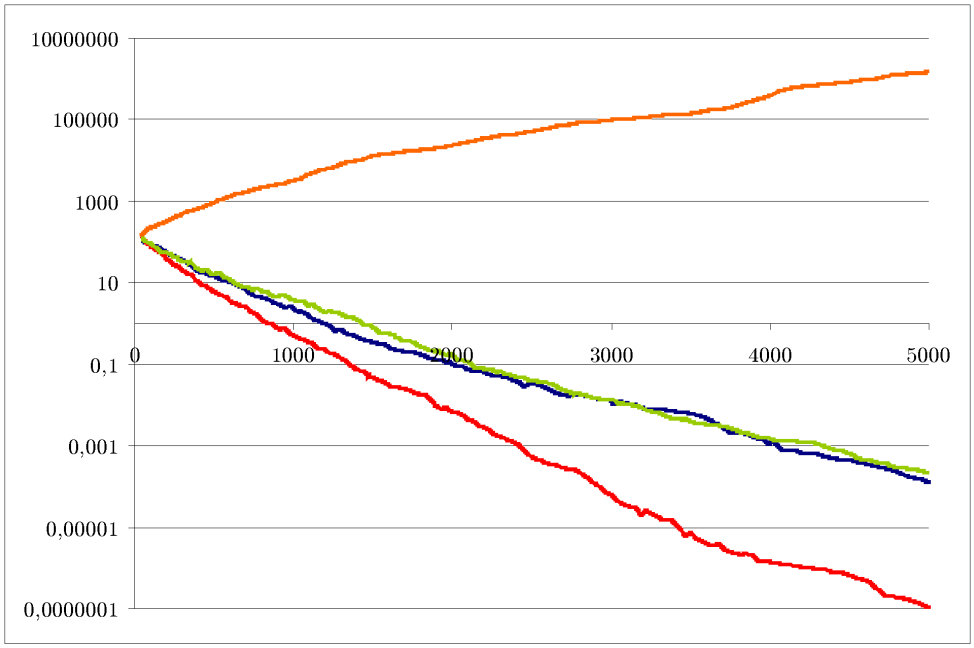


Abbildung 3.8: Vergleich von der Auswirkung unterschiedlicher Modellqualitäten auf die Optimierung der Kugelfunktion. Blau: Standard ES, Grün: Zufälliges Modell, Rot: Perfektes Modell, Orange: Schlechtes Modell

Der Einfluss der Modellqualität auf die Optimierung ist in Abbildung 3.8 veranschaulicht. Es wurde jeweils mit drei verschiedenen (künstlich erzeugten) Modellqualitäten eine Opti-

mierung der 20-dimensionalen Kugelfunktion

$$f(\mathbf{x}) = \sum_{l=1}^{20} x^{(l)} x^{(l)} \tag{3.41}$$

durchgeführt. Jede Optimierung wurde 15 mal mit zufällig initialisierten Startwerten durchgeführt. Nach 5000 Fitnessauswertungen wurde die Optimierung abgebrochen. Die dargestellten Kurven zeigen den Median der 15 Durchläufe. Die blaue Kurve zeigt das Ergebnis einer Standard Evolutionsstrategie mit $\mu = 5$ Eltern und $\lambda = 35$ Nachkommen ohne Modellunterstützung. Für die übrigen Optimierungen wurde das Prinzip der Vorselektion mit $\lambda_p = 2\lambda = 70$ genutzt. Die rote Kurve stellt den Verlauf der Optimierung mit einem perfekten Modell dar, das die Fitness immer korrekt vorhersagt. Offensichtlich konvergiert die Strategie wesentlich schneller als die Standard Strategie. In grün ist der Fitnessverlauf mit einem zufälligen Modell aufgezeichnet. Wie zu erwarten, ist die Konvergenz gleich schnell wie die der nicht modellgestützten Strategie. Die orange dargestellte Kurve zeigt den Verlauf bei einem denkbar schlechten Modell, das dadurch simuliert wurde, dass die Vorselektion immer die λ schlechtesten Individuen ausgewählt hat. Diese Kurve zeigt, dass im Falle eines (sehr) schlechten Modells die Strategie sogar divergieren kann.

Die Ergebnisse legen daher nahe, dass eine Steuerung vom Einfluss des Modells in Abhängigkeit der Modellqualität ratsam ist. Bei einem guten Modell kann dem Modell mehr getraut werden, während bei einem schlechten Modell das Vertrauen in das Modell zu reduzieren ist. Strategien zur Regelung des Modelleinflusses werden im nächsten Abschnitt behandelt. Zunächst wird darauf eingegangen, wie die Qualität eines Modells bestimmt werden kann.

Prinzipiell lässt sich die Qualität der Vorselektion über die Fähigkeit definieren, die real bewertet besten Individuen basierend auf der Prognose von dem Modell auszuwählen. Die numerische Differenz zwischen vorhergesagter und realer Fitness ist dabei von geringerem Interesse. Dies wird z. B. in Abbildung 3.7c deutlich: Das dort abgebildete Modell kann die tatsächliche Fitness für das globale Optimum von $y(\mathbf{x}_0) = 0$ an der Stelle $\mathbf{x}_0 = (0 \dots 0)^T$ nicht korrekt wiedergeben und liefert stattdessen einen Funktionswert von $\hat{y}(\mathbf{x}_0) \approx 4$. Wird dieses Modell aber zur Fitnessprädiktion für die zugrunde liegende Ackley-Funktion benutzt, bewertet es den Wert $\mathbf{x}_0$ mit der Fitness von ≈ 4, die (bei Minimierung) beste Fitness, die überhaupt von dem Modell in dem dargestellten Bereich prognostiziert werden kann. Daher wird dieses Individuum die Vorselektion trotz relativ schlechter absoluter Approximation höchstwahrscheinlich überleben und nachfolgend mit der realen besten (globalen) Fitness bewertet.

Die Bewertung der Vorselektion müsste nun eigentlich, basierend auf der Fähigkeit, die λ real besten Individuen aus den λ_p vielen Nachkommen auszuwählen, vorgenommen werden. Da die reale Fitness aber nur von den λ vielen Nachkommen bekannt ist, wird vorausgesetzt, dass die Vorselektion der λ Individuen aus allen λ_p Nachkommen mit der Selektion der μ Eltern aus den λ Individuen übereinstimmt, von denen sowohl die prognostizierte Fitness,

als auch die reale Fitness bekannt ist [30, 66]. Dazu wird überprüft, welche Individuen, basierend auf der vorhergesagten Fitness, zu Eltern selektiert worden wären, und in wie fern sich diese Auswahl mit den tatsächlich besten μ Individuen deckt. Das Qualitätsmaß nimmt dabei eine Gewichtung $(\lambda - i)$ der Individuen, basierend auf dem Ranking i ihrer realen Fitness vor, so dass die Qualität der Vorselektion als schlechter bewertet wird, wenn beispielsweise das real beste Individuum mit Gewichtung $(\lambda-1)$ nicht vorausgewählt wurde, als wenn das zweitbeste Individuum mit Gewichtung $(\lambda-2)$ nicht vorselektiert worden wäre. Die Qualität der Vorselektion errechnet sich schließlich durch die Summe der Gewichtungen

$$Q = \sum_{i=1}^{\mu} g_i(\lambda - i) \tag{3.42}$$

mit $g_i = 1$, falls das Individuum an der i-ten Stelle im realen Fitnessranking, basierend auf der vorhergesagten Fitness, ebenfalls selektiert worden wäre, andernfalls ist $g_i = 0$.

Offensichtlich gilt für die schlechtest mögliche Qualität $Q^{(min)} = 0$. Die beste mögliche Qualität errechnet sich durch

$$Q^{(max)} = \sum_{i=1}^{\mu}(\lambda - i) = \mu\lambda - \frac{\mu(\mu+1)}{2}. \tag{3.43}$$

Die Qualität einer zufälligen Vorselektion [66] ist berechenbar durch

$$Q^{(rand)} = \sum_{i=1}^{\mu} i \frac{\binom{\mu}{i}\binom{\lambda-\mu}{\mu-i}}{\binom{\lambda}{\mu}} \cdot \frac{1}{\mu}\sum_{i=1}^{\mu}(\lambda - i) = \frac{\mu^2}{\lambda}\frac{2\lambda - \mu - 1}{2}. \tag{3.44}$$

Die mit diesem Verfahren bestimmten Qualitätsmaße aus dem obigen Beispiel der Optimierung der 20-dimensionalen Kugelfunktion sind in Abbildung 3.9 als Mittelwerte von 10 Durchläufen für die verschiedenen Modellqualitäten dargestellt. Gemäß den Formeln für die Modellqualitäten ergeben sich für die Optimierung mit $\mu = 5$ und $\lambda = 35$ die Modellqualitäten $Q^{(min)} = 0$, $Q^{(rand)} \approx 22,85$ und $Q^{(max)} = 160$. Die maximale und minimale Modellqualität wird offensichtlich korrekt eingehalten. Die Qualität des zufälligen Modells (dargestellt in grün) schwankt wie erwartet um die theoretische zufällige Modellqualität von $Q^{(rand)} \approx 22,85$. Bei der arithmetischen Mittelung der Modellqualität von mehr (genau genommen unendlich vielen) Messdurchläufen mit dem zufälligen Modell müsste der Verlauf der Modellqualität der gestrichelt dargestellten Geraden entsprechen.

3.2.4 Regelung des Modelleinflusses

Wie zuvor motiviert, ist es ratsam, den Einfluss des Modells in Abhängigkeit von der aktuellen Modellqualität zu steuern. Die Qualität der Fitnessprädiktion kann sich im Verlauf der Optimierung ändern. Zum einen werden durch die realen Fitnessauswertungen mit

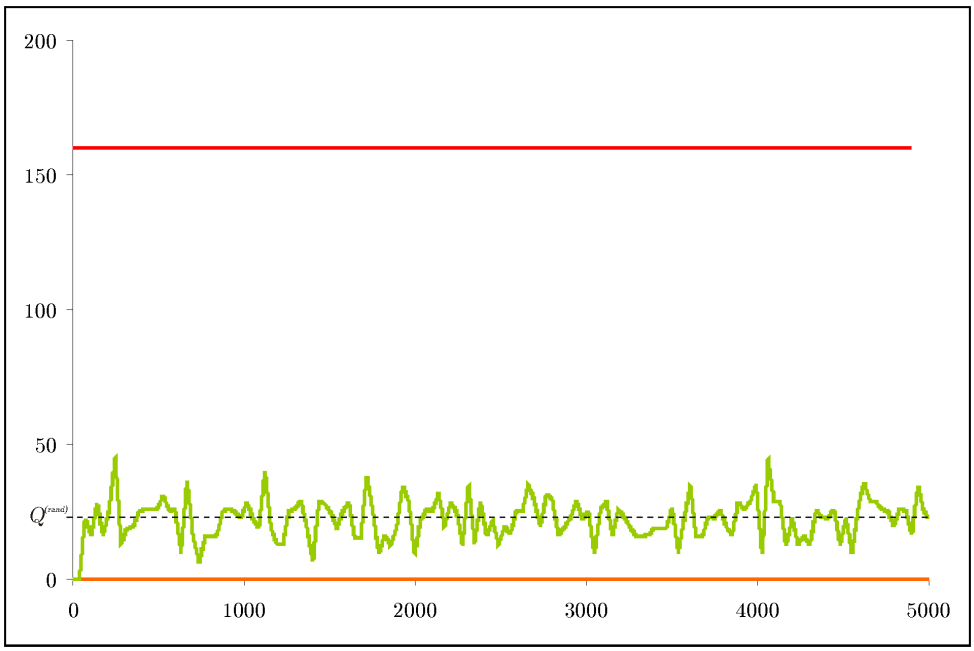


Abbildung 3.9: Modellqualitäten der Optimierung der Kugelfunktion. Grün: Zufälliges Modell, Rot: Perfektes Modell, Orange: Schlechtes Modell

jeder Generation mehr Stützpunkte zum Modell hinzugefügt, wodurch sich die Qualität verbessern müsste, zum anderen kann es aber auch passieren, dass die Evolution in Regionen des Suchraumes vorstößt, für die noch keine oder nur wenige nahe Stützstellen für die Approximation zur Verfügung stehen. Daher werden im Folgenden zwei Ansätze zur Regelung des Modelleinflusses erläutert.

Die Controlled Model Assisted Evolution Strategy

Bei der Controlled Model Assisted Evolution Strategy (C-MAES) [66] wird der Modelleinfluss durch die Anzahl der λ_p Individuen, die dem Vorselektionsprozess unterworfen werden, geregelt. Es wird exemplarisch für die jeweils 10 dimensionale Kugel– und Rosenbrockfunktion gezeigt, dass bei einem Modell, das besser als ein zufälliges Modell vorselektiert, mit steigendem λ_p ebenfalls die Konvergenzgeschwindigkeit steigt. Es wird geschlussfolgert, dass bei einer Modellqualität, die größer als die zufällige Modellqualität $Q^{(rand)}$ ist, ein größeres Vertrauen und damit eine schnellere Konvergenzgeschwindigkeit durch die Erhöhung von λ_p erzielt werden kann. Bei einer schlechteren Modellqualität muss der Einfluss des Modells wieder durch die Reduktion von λ_p gemindert werden. Nach jeder Generation t wird die Qualität $Q^{(t)}$ der Vorselektion bewertet und λ_p nach den folgenden Regeln angepasst.

Ist die aktuelle Qualität $Q^{(t)}$ des Modells besser als die Qualität $Q^{(rand)}$ eines zufälligen

Modells, wird die Anzahl der Nachkommen λ_p erhöht durch

$$\lambda_p^{(t+1)} = \lambda_p^{(t)} + \frac{Q^{(max)} - Q^{(t)}}{Q^{(max)} - Q^{(rand)}} \cdot \delta_{\lambda_p}, \tag{3.45}$$

ist die Qualität $Q^{(t)}$ der Vorselektion hingegen schlechter als die Qualität eines zufälligen Modells $Q^{(rand)}$, wird λ_p reduziert gemäß

$$\lambda_p^{(t+1)} = \lambda_p^{(t)} - \frac{Q^{(max)} - Q^{(t)}}{Q^{(rand)}} \cdot \delta_{\lambda_p}. \tag{3.46}$$

Der Faktor δ_{λ_p} wird als Anpassungsrate bezeichnet und bestimmt die Stärke der Änderung.

Das Verhalten dieser Regelung wurde für vier verschiedene Testfunktionen untersucht, indem nach einer bestimmten Anzahl von Fitnessauswertungen zu der von dem Modell approximierten Fitness ein zufälliger Rauschwert addiert und somit die Modellqualität gemindert wird [66]. Die Ergebnisse zeigen empirisch, dass durch diese Regelung eine Divergenz der Strategie vermieden wird, indem bei schlechter werdender Modellqualität die Anzahl λ_p reduziert wird.

Die λ-Controlled Model Assisted Evolution Strategy

Eine alternative Regelung des Modelleinflusses kann über die Anzahl der λ Nachkommen erfolgen, die von der realen Fitnessfunktion bewertet werden [25]. Diese Variante wird mit λ–Controlled Model Assisted Evolution Strategy (λ–CMAES) bezeichnet. Die Anzahl der λ_p Nachkommen, die von der Vorselektion bewertet werden, bleibt konstant.

Die Idee hinter dieser Regelung ist, dass bei einer guten Vorselektion nur ein geringerer Teil der Nachkommen von dem realen System bewertet werden muss. Angenommen, das Modell bewertet die μ real besten Nachkommen derart, dass sie die Vorselektion überleben, dann würde es ausreichen, nur diese μ vielen Individuen real zu bewerten, ohne dass diese geringe Anzahl von real bewerteten Individuen einen negativen Einfluss auf die Optimierung hat. Bei einer schlechten Modellqualität hingegen wäre eine Erhöhung der Nachkommen λ ratsam, um zu vermeiden, dass die real besten Individuen von dem Vorselektionsprozeß aussortiert werden.

Die Anzahl $\lambda^{(t+1)}$ der Nachkommen der nächsten Generation $t+1$ wird gemäß den folgenden Regeln gesteuert. Ist die Modellqualität $Q^{(t)}$ besser als ein zufälliges Modell $Q^{(rand)}$, wird der Einfluss des Modells erhöht, indem $\lambda^{(t+1)}$ reduziert wird durch

$$\lambda^{(t+1)} = \max\left\{\lambda^{(t)} - \frac{Q^{(max)} - Q^{(t)}}{Q^{(rand)}} \cdot \delta_\lambda, \mu\right\}. \tag{3.47}$$

Ist die gegenwärtige Modellqualität $Q^{(t)}$ schlechter als $Q^{(rand)}$, wird der Einfluss des Modells reduziert, indem die Anzahl der real zu bewertenden Individuen λ erhöht wird gemäß

$$\lambda^{(t+1)} = \min\left\{\lambda^{(t)} + \frac{Q^{(max)} - Q^{(t)}}{Q^{(max)} - Q^{(rand)}} \cdot \delta_\lambda, \lambda_p\right\}. \tag{3.48}$$

Eine Besonderheit dieser Regelung ist, dass die Bewertung des Qualität des Modells instabil wird für kleiner werdendes λ. Für $\lambda = \mu$ sind schließlich alle Individuen immer korrekt bewertet. Als Ausweg wird vorgeschlagen, im Falle von $\lambda < 2\mu$ die Qualität durch eine $\lambda/2$ aus λ Selektion zu bewerten. Die Formeln 3.43 und 3.44 für die Berechnung von $Q^{(max)}$ und $Q^{(rand)}$ müssen dann modifiziert werden, indem für den Fall $\lambda < 2\mu$ der Wert von μ ersetzt wird durch $\lambda/2$.

Diese Regelung wird in [25] im Bezug auf die Konvergenzgeschwindigkeit mit der C-MAES Regelung für die Optimierung der multimodalen Griewank Testfunktion verglichen. Die Anzahl der realen Fitnessbewertungen konnten im Vergleich zu der C-MAES nochmals drastisch reduziert werden.

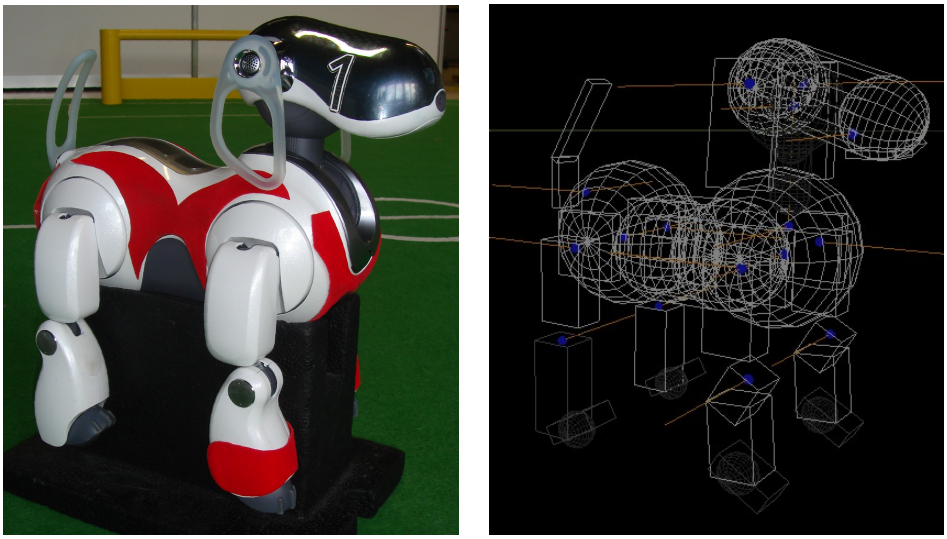
Kapitel 4

Parameteroptimierung einer dynamischen Robotersimulation

Bei der Arbeit mit Robotern sind realistische physikalische Simulatoren von hohem Nutzen für den Entwickler. Einerseits können mit Hilfe der Simulation Alternativen in Designfragen des Roboters evaluiert werden und somit die Entwicklung schneller erfolgen und Kosten eingespart werden. Andererseits werden die Entwicklung von Software und Algorithmen unterstützt, da dem Entwickler nicht zwangsläufig ein realer Roboter zur Verfügung stehen muss. Fehler in der Software können schneller aufgespürt werden, da durch die Simulation ein direktes Debuggen aus der Entwicklungsumgebung mit Breakpoints etc. ermöglicht wird. Bei sehr test- oder rechenaufwändigen Verfahren können weiterhin – beispielsweise in einem Rechencluster – mehrere Roboter parallel simuliert werden, wodurch die Zeit zur Evaluierung des Verfahrens drastisch reduziert werden kann und kein Risiko eingegangen wird, dass der Roboter beschädigt oder verschlissen wird. In dieser Arbeit wird später in Kapitel 5 die Performanz verschiedener Lernalgorithmen für das Laufenlernen von Robotern verglichen. Da ein fundierter Vergleich nur basierend auf zahlreichen Durchläufen der Optimierung erfolgen kann, wird in diesem Kapitel zunächst ein Simulatormodell des verwendeten Roboters angepasst, das anstelle des echten Roboters genutzt wird. Damit die Aussagekraft erhalten bleibt, muss der simulierte Roboter möglichst den echten Roboter, und somit die Struktur des Suchraumes des später zu optimierenden Problems gut approximieren.

Ein charakteristisches Merkmal aller Simulationen ist, dass sie die reale Welt nur approximieren können. Sie beinhalten immer ein Defizit, den so genannten *Reality Gap* [29]. Dieses Defizit beeinflusst alle Aspekte von Simulationen: Den Detailgrad des Roboters sowie die Eigenschaften der Sensoren und der Aktoren. Abbildung 4.1 zeigt den hier benutzten echten Aibo Roboter und das approximierte physikalische Modell.

Für viele Anwendungen ist die mangelnde Realitätstreue der Simulation nicht sonderlich problematisch, da dies in der Regel dazu führt, dass bei der Anwendung von Algorith-



(a) Realer Roboter (b) Physikalische Approximation

Abbildung 4.1: Der benutzte Aibo Roboter.

men, die in der Simulation entwickelt wurden, Parameter an den echten Roboter angepasst werden müssen. Es existieren aber auch verschiedene Verfahren, die in der Simulation entwickelt wurden, auf dem realen Roboter aber aufgrund der mangelnden Simulationsqualität versagen [42]. Speziell bei der Evaluierung von Laufbewegungen ist eine realistische physikalische Simulation vonnöten. Insbesondere bei Laufbewegungen, die durch maschinelle Lernverfahren optimiert wurden [22, 23, 49], ist zu beobachten, dass die Motoren in den Gelenken des Roboters in ihrem Grenzbereich betrieben werden (siehe Kapitel 2.1.2, Abbildung 2.6) und bestimmte Eigenschaften der Umwelt stark ausgenutzt werden. Bei den Aibo Robotern resultiert dies z. B. in einem Lauf auf den „Ellbögen“ der Vorderbeine, die hauptsächlich über den Untergrund rutschen, während der Vortrieb hauptsächlich von den Hinterbeinen erzeugt wird. Eine reale Simulation erfordert insbesondere bei laufenden Robotern mit vielen Freiheitsgraden eine korrekte Einstellung zahlreicher Parameter.

Im Folgenden wird ein Verfahren erläutert, das stufenweise die Lücke zwischen Simulation und Realität reduziert [20]. Das Verfahren wird auf den Aibo Roboter angewendet, ist aber nicht spezifisch für diesen Roboter und kann damit leicht auf andere Robotertypen übertragen werden. Als Grundlage wird der von der Universität Bremen entwickelte Simulator SimRobot genutzt. SimRobot wurde nicht speziell für eine Roboterplattform, sondern im Hinblick auf universelle Verwendbarkeit entwickelt. So existieren beispielsweise sowohl Simulationsmodelle für die kommerziell verfügbaren Roboter Sony Aibo und die humanoiden Roboter Bioloid der Firma Robotis und Nao der Firma Aldebaran als auch Modelle für Eigenentwicklungen der Universität Bremen, wie ein autonomer Rollstuhl oder Roboter mit omidirektionalem Antrieb für die Small-Size-League [36, 58].

4.1 Überblick über ähnliche Arbeiten

Simulatoren werden immer häufiger eingesetzt, um bestimmte Aufgaben eines Roboters maschinell zu erlernen. Die Qualität der Simulation ist hierbei besonders wichtig, da die erlernten Fähigkeiten auf dem realen System eingesetzt werden sollen. Zagal et al. beschreiben ein simulationsgestütztes Lernverfahren *Back to Reality* (BTR), das mittels Co-Evolution sowohl die primäre Zielfunktion als auch das verwendete Simulationsmodell optimiert [71, 73]. Das Verfahren wird für einen Aibo demonstriert, der lernt, einen Ball effizient zu kicken und im anderen Fall sein Laufmuster optimiert. Grundlegend besteht der BTR Algorithmus aus drei Lernstufen L1, L2 und L3 gemäß Abbildung 4.2. Zuerst wird in der ersten Lernstufe L1 die Zielfunktion, basierend auf der Simulation (in diesem Fall UCHLSIM [70, 72]) mit einem Genetischen Algorithmus, optimiert. Anschließend werden in Lernstufe L2 einige Individuen der letzten Generationen auf dem realen Roboter bewertet und für diese Individuen die Differenz $\Delta fitness$ von simulierter und realer Fitness bestimmt. In Lernschritt L3 werden nun die Parameter der physikalischen Simulation variiert, so dass $\Delta fitness$ minimiert wird und somit die Simulation realistischere Fitnessbewertungen liefern kann. Die drei Lernschritte werden wiederholt, bis auf dem realen Roboter eine zufriedenstellende Lösung gefunden wird.

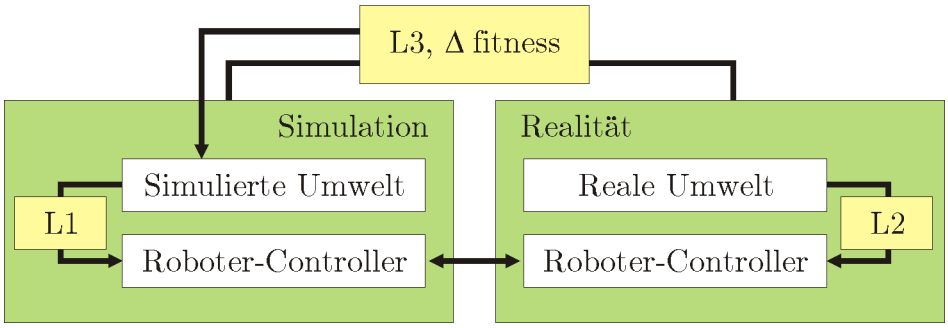


Abbildung 4.2: Drei Lernprozesse des Algorithmus *Back to Reality*.

Iocci et al. gehen einen anderen Weg und beschreiben, wie der Simulator USARSim [7] einen humanoiden Roboter bei der Laufoptimierung unterstützt [28]. Während des Lernens wird die Qualität der Aussagekraft der Simulation verbessert, ohne das Robotermodell [16] oder andere Simulationsparameter zu variieren. Der Ansatz bewertet während des Lernens die Individuen abwechselnd mit der Simulation und dem realen Roboter. Die Fitness, die von dem realen Roboter gemessen wurde, wird anschließend ebenfalls von dem Simulator bestimmt. Aus den Differenzen der Fitnessbewertungen von Realität und Simulation wird iterativ eine Korrekturfunktion errechnet, die die Differenz der simulierten und realen Fitness in Abhängigkeit des Punktes im Suchraum modelliert. Bei weiteren Fitnessbewertungen mit dem Simulator wird die Qualität der Fitnessbestimmung erhöht, indem zu der gemessenen Fitness das Ergebnis der Korrekturfunktion des entsprechenden Punktes addiert wird. Auf diese Weise wird die Aussagekraft des Simulators im Laufe des Optimie-

rungsprozesses verbessert. Es konnte die Anzahl der nötigen Fitnessauswertungen auf dem Roboter deutlich reduziert werden.

Die beiden erläuterten Ansätze erhöhen für die spezifischen Probleme zwar deutlich die Qualität der Fitnessbestimmung der Simulation, resultieren aber nicht in einer allgemein gültigen besseren physikalischen Simulation. Daher wird hier im Folgenden ein Verfahren vorgestellt, mit dem physikalische Parameter für die Simulation erlernt werden, die weitestgehend allgemeingültig sind.

4.2 Parameter der Simulation

Für jede Robotersimulation existieren zahlreiche Parameter, die die Realität und Performanz der Simulation beeinflussen. Der hier benutzte Simulator SimRobot basiert auf der frei verfügbaren *Open Dynamics Engine (ODE)* von Russell Smith [59]. ODE stellt eine effiziente physikalische Simulation von starren Körpern zur Verfügung und hat sich als Standard etabliert.

Die zur Verfügung stehenden Parameter der Simulation werden im Folgenden erläutert.

4.2.1 Parameter für die Motoren

SimRobot stellt ausschließlich rotatorische Motoren zur Verfügung. Jeder (rotatorische) Motor lässt sich insbesondere durch die Eigenschaften

- maximale Geschwindigkeit,
- maximales Drehmoment und
- Parameter des Positionsreglers

charakterisieren. Für kommerziell verfügbare Servomotoren werden immer die Werte für die maximale Geschwindigkeit und das maximale Drehmoment angegeben. Zum Anfahren und Einhalten der anzusteuernden Position des Motors sind Regler nötig, die die Kraft, mit der sich der Motor in Richtung der Sollposition dreht, abhängig von aktueller und Zielposition bestimmt. Der Regler soll sicherstellen, dass die Position möglichst schnell und gleichzeitig präzise angefahren wird. Am weitesten verbreitet ist bei Servomotoren der so genannte *PID Regler* [56]. Er besteht aus drei unabhängigen Reglern, die jeweils einen Anteil zur Gesamtregelung leisten. Der P-Anteil ist proportional zu der Regelabweichung, der I-Anteil proportional zu der aufsummierten (integrierten) Regelabweichung und der D-Anteil proportional zu der Änderung (Ableitung) der Regelabweichung. Die Gesamtregelung ergibt sich aus der Summe der einzelnen Anteile. Die Implementierung von SimRobot sieht für jeden Motor eine PID-Steuerung vor und stellt damit insgesamt fünf Parameter für die Beschreibung eines Motors zur Verfügung.

Die Parameter der Motoren haben augenscheinlich den größten Einfluss auf den Grad der Realität der Simulation. Die Angaben in Datenblättern zu den Motoren (sofern verfügbar) stellen einen guten Anfangswert für die Simulation dar, resultieren aufgrund der approximativen Simulation aber keinesfalls in den gleichen Bewegungen wie in der Realität.

4.2.2 Globale Simulationsparameter

Neben den Parametern für die Motoren existieren Parameter, die die Interaktion mit der Umwelt beeinflussen. Einige der hier erläuterten Parameter sind spezifisch für die ODE Physik-Engine, haben aber ähnliche Äquivalente in anderen Simulatoren.

Die zwei ODE Parameter *Error Reduction Parameter (ERP)* und *Constraint Force Mixing (CFM)* haben kein praktisches Gegenstück in der realen Welt. Sie dienen dazu, die Simulation mathematisch stabil zu halten. Durch numerische Fehler während der Simulation kann es passieren, dass sich Körperteile, die in einem Gelenk verbunden sind, voneinander entfernen und somit nicht mehr exakt in dem ursprünglichen Gelenk miteinander verbunden sind, was im Laufe der Simulation zu einem Auseinanderdriften der Teile führen kann. Dieser Effekt kann durch den Parameter ERP verhindert werden. Er sorgt dafür, dass in jedem Simulationsschritt eine Kraft auf die Körperteile wirkt, so dass sie wieder in das Gelenk zurückgezogen werden. Ein Wert von 0 schaltet diesen Mechanismus ab, ein Wert von 1 resultiert in einer Kraft, die dafür sorgt, dass die Gelenke im nächsten Simulationsschritt in jedem Fall wieder korrekt sitzen. Ein Wert von 1 ist aber aufgrund von zahlreichen internen Approximationen vom Entwickler nicht empfohlen. Der Parameter *CFM* bestimmt, in welchem Maß gewisse Beschränkungen verletzt werden dürfen. In der Regel sind die Teile des Roboters aus einem harten Material, so dass sie bei einer Kollision nicht ineinander eindringen können. In der Simulation kann es jedoch sinnvoll sein, ein Eindringen von Objekten bis zu einem gewissen Grad zu erlauben. Die Stärke der erlaubten Eindringung wird mit einem Wert von $CFM > 0$ festgelegt. In SimRobot sind die Parameter *ERP* und *CFM* global realisiert und damit für alle Gelenke des Roboters identisch.

Die Kollision von Objekten wird in ODE effizient durch ein temporäres Einfügen von Gelenken an dem Kontaktpunkt approximiert. Das Verhalten dieser temporären Gelenke kann in SimRobot durch 6 ODE Parameter beschrieben werden, die im Folgenden näher erläutert werden. Die hier beschriebenen Parameter gelten für Kollisionen von Roboterteilen mit dem Untergrund.

Der Parameter *bounce* beschreibt, wie stark der Untergrund federt. Ein Wert von 0 bedeutet, dass der Boden nicht federt. Durch den zugehörigen Parameter *bounce_vel* kann eine minimale Geschwindigkeit definiert werden, ab der die zuvor erwähnte Federung des Bodens stattfindet, unterhalb dieser definierten Kollisionsgeschwindigkeit federt der Boden nicht. Die Coulomb Reibung zwischen den Kontaktpunkten wird durch den Parameter *mu* bestimmt. Ein Wert von 0 resultiert in einem reibungslosen Kontakt, ein Wert von ∞ hingegen in einem Kontakt, der unter keinen Umständen rutscht. Ein zweiter Reibungs-

parameter *slip* bestimmt ein kraftabhängiges Rutschen (force-dependent-slip (FDS) im Original). Der Effekt FDS sorgt dafür, dass sich berührende Oberflächen mit einer Geschwindigkeit proportional zu der wirkenden Kraft auseinander bewegen. Angenommen, dass an einem Kontaktpunkt der Reibungskoeffizient *mu* unendlich ist und eine Kraft versucht, die Kontaktstellen auseinander zu drücken, bewegen sich – egal wie groß die Kraft ist – die Punkte nicht auseinander. Der FDS Koeffizient *slip* sorgt bei einem Wert von $slip > 0$ aber dafür, dass die Kontaktpunkte beim Wirken einer Kraft trotzdem mit konstanter Geschwindigkeit auseinander driften. Ein anschauliches Beispiel für diesen Effekt ist ein Wagen, dessen Reifen in Fahrtrichtung geradeaus zeigen und mit dem Untergrund einen unendlichen Reibungskoeffizienten haben. Steht der Wagen still, bewirkt eine Kraft senkrecht zu der Stellung der Räder keine Bewegung. Fährt der Wagen hingegen vorwärts und es wirkt eine Kraft senkrecht zur Fahrtrichtung, wird der Wagen in diese Richtung abgelenkt - dieser Effekt tritt beispielsweise bei schnell fahrenden Autos und Seitenwind auf.

Schlussendlich existieren bei den temporären Hilfsgelenken zur Kollisionsbehandlung noch die Parameter *soft_erp* und *soft_cfm*, die denselben Effekt wie die zuvor erläuterten globalen Parameter *ERP* und *CFM* haben. Mithilfe dieser Parameter kann beispielsweise eine Eindringtiefe der Roboterbeine in den Untergrund beim Laufen definiert werden.

4.3 Verfahren zum Lernen der Simulationsparameter

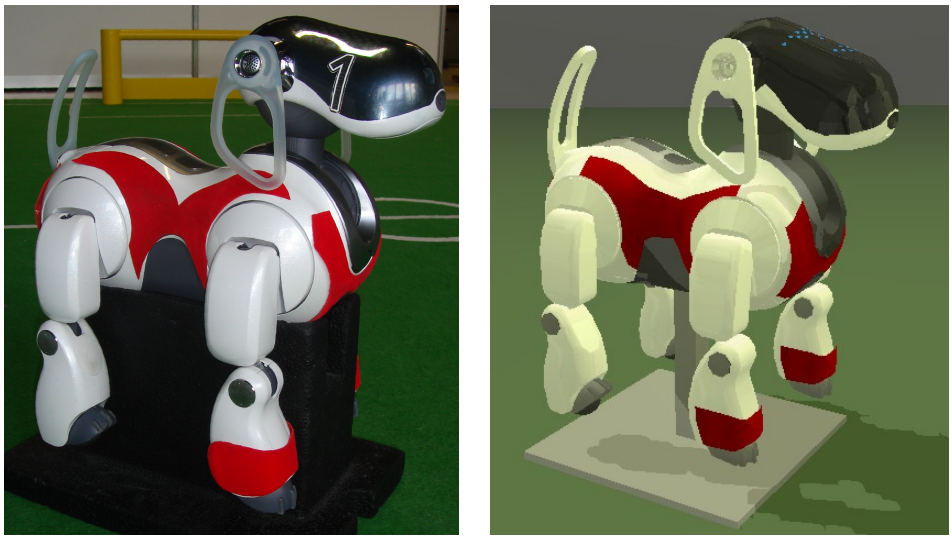
Für den Aibo Roboter werden im Folgenden die zuvor beschriebenen Parameter für die Beingelenke und die globalen Simulationsparameter gelernt. Ein Bein des Aibos besteht aus drei Gelenken und jedes Gelenk wird mittels fünf Parametern (siehe Kapitel 4.2.1) beschrieben. Da die Gelenke an den beiden Vorder- bzw. Hinterbeinen identisch sind, werden sie symmetrisch behandelt. Daher ergeben sich insgesamt 30 Parameter für die Beingelenke und 8 globale Parameter. Da einige Parameter unabhängig optimiert werden können, wird im Folgenden ein stufenweises Lernen der Parameter vorgestellt, um den Suchraum möglichst klein zu halten.

Zum Lernen der Parameter wurde die in Kapitel 3.1 beschriebene Evolutionsstrategie mit Selbstanpassung benutzt. Die Nachkommen wurden durch intermediäre Rekombination mit nachfolgender Mutation erzeugt. Für einige Parameter sind Wertebereiche definiert, auf die die entsprechenden Objektparameter gegebenenfalls geclippt werden. Die jeweils genutzte Fitnessfunktion wird zusammen mit der Lernstufe erläutert.

4.3.1 Maximale Motorgeschwindigkeit

Im ersten Lernschritt werden die maximalen Geschwindigkeiten der Motoren zusammen mit vorläufigen PID Werten gelernt. In diesem Schritt wurde ein Aufbau gewählt, der es ermög-

licht, die Auswirkung der maximal erreichbaren Geschwindigkeit zu maximieren und den Effekt der restlichen Parameter zu minimieren. Dazu wurde der Roboter, wie in Abbildung 4.3 gezeigt, auf einem Sockel platziert, so dass die Beine frei in der Luft hängen. Bei einer Bewegung der Beine wirkt nun – abgesehen von dem Gewicht der Beine und der nötigen Kraft, um die Massenträgheit bei einer Bewegung zu überwinden – keine Kraft. Die einzelnen Beingelenke des realen Roboters werden nun mit Bewegungssequenzen angesteuert, die es ermöglichen, Rückschlüsse auf die maximalen Geschwindigkeiten zu ziehen. Während der Bewegung zeichnet der Roboter alle 8 ms sowohl die Ansteuerung, als auch den aktuellen Sensorwert jedes einzelnen Gelenks auf. In der ersten Phase der Bewegungssequenz werden die Gelenke sinusförmig mit steigender Frequenz angesteuert, in der zweiten Phase wird eine Rechteckschwingung mit ebenfalls steigender Frequenz eingesetzt. Die Amplituden der Bewegung werden dabei jeweils auf 80% der maximalen Gelenkwinkel in positive und negative Richtung gesetzt. Es wird nicht die maximale Auslenkung genutzt, da hierbei der Motor an dem maximalen Winkel anschlagen würde und so die Charakteristik der PID Regelung kaum erfassbar wäre. Da das Gewicht der Beine bei diesem Roboter gering in Relation zu der maximalen Kraft der Motoren ist, wird die Bewegung hier hauptsächlich durch die maximalen Geschwindigkeiten der einzelnen Motoren bestimmt. Abbildung 4.4 zeigt einen Ausschnitt der Ansteuerung und der resultierenden Sensorkurve exemplarisch für das rechte vordere Schultergelenk. Die gesamte Bewegungssequenz in dieser Stufe betrug ca. 60 Sekunden.

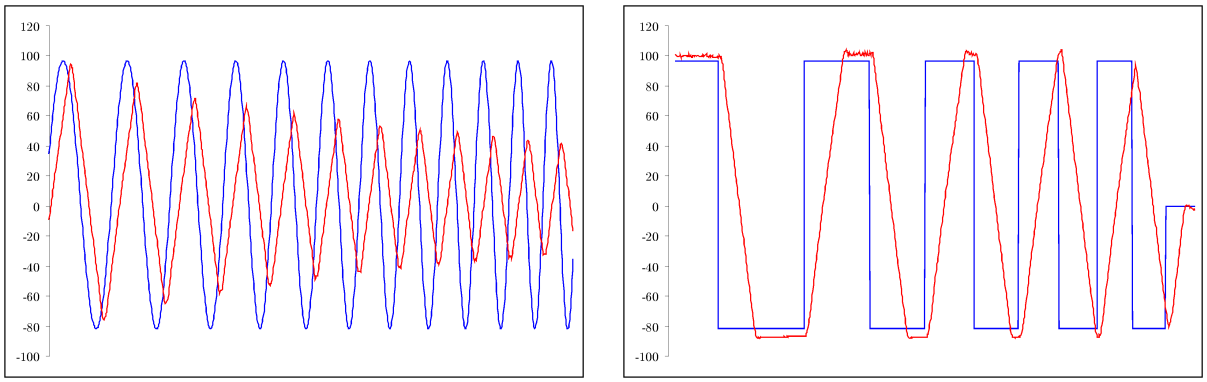


(a) Realer Roboter (b) Simulierter Roboter

Abbildung 4.3: Aufbau zum Lernen der maximalen Motorgeschwindigkeiten.

Nachdem die resultierende Bewegungssequenz des realen Roboters aufgezeichnet wurde, ist nun das Ziel dieser Lernstufe, die Bewegung des simulierten Roboters an die Bewegung des realen Roboters anzugleichen. Dazu werden die Gelenke des simulierten Roboters mit der gleichen Bewegungssequenz wie der echte Roboter zuvor angesteuert und seine Sensor-

kurve mit der des realen Roboters verglichen und ein Maß für die Ähnlichkeit der Kurven berechnet. Für jedes Gelenk wird für alle gemessenen Sensorwerte die quadrierte Differenz zwischen realer und simulierter Sensorkurve aufaddiert. Kleine Werte bedeuten also eine bessere Qualität der Simulation, daher wird dieses Ähnlichkeitsmaß als zu minimierende Fitness benutzt. Da der zeitliche Versatz der Sensorkurven nicht von Interesse ist, werden die simulierte und reale Sensorkurve bei der Fitnessberechnung in einem kleinen Fenster gegeneinander verschoben und der kleinste berechnete Wert als Fitness zugewiesen.



(a) Sinusförmige Ansteuerung

(b) Rechteckförmige Ansteuerung

Abbildung 4.4: Auszug der angesteuerten (blau) und der realen (rot) Bewegungssequenz zur Ermittlung der maximalen Motorgeschwindigkeiten.

Die in dieser Stufe ermittelten maximalen Geschwindigkeiten werden als endgültig betrachtet. Die ermittelten Werte für die PID Regler sind jedoch nur ein vorläufiges Zwischenergebnis, da sie abhängig von den maximalen Drehmomenten der Motoren sind, die erst in dem nächsten Lernschritt evaluiert werden.

4.3.2 Maximale Drehmomente

Der zweite Lernschritt wird nur als Zwischenschritt benutzt, um die maximalen Drehmomente und die PID Werte der Motoren als Startwerte für den letzten Lernschritt zu schätzen. In diesem Schritt werden die Ansteuerungs- und Sensorkurven der Gelenke des realen Roboters aufgenommen, während er Fußball spielt. Die Bewegungssequenz besteht daher sowohl aus Laufen in verschiedene Richtungen mit verschiedenen Geschwindigkeiten, als auch aus Schüssen des Balles oder Aufstehbewegungen. In dieser Lernstufe ist die Bewegungsphase ca. 3 Minuten lang. Erneut werden die angesteuerten Werte des realen Roboters auf dem simulierten Roboter abgespielt, der in dieser Lernstufe ebenfalls auf dem Boden steht. Die Fitness wird, wie in der vorherigen Stufe, aus den quadrierten Differenzen der realen und simulierten Sensordaten berechnet. Die resultierende Geschwindigkeit des Roboters wird noch ignoriert. Abbildung 4.5 zeigt exemplarisch die Bewegung des vorderen rechten Kniegelenkes. Da die in dieser Lernstufe ermittelten Werte zu einem gewissen Grad

von globalen Parametern, wie z. B. der Reibung mit dem Untergrund abhängen, werden sie als Startparameter für die nächste Lernstufe genutzt.

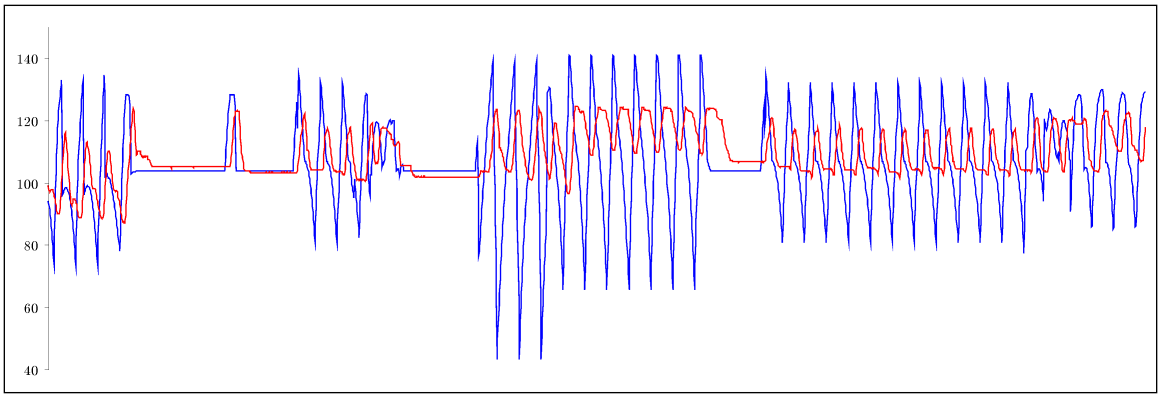


Abbildung 4.5: Auszug der angesteuerten (blau) und der realen (rot) Bewegungssequenz zur Ermittlung der (vorläufigen) maximalen Drehmomente.

4.3.3 Gesamtbewegung

Das Ziel der Optimierung soll sein, dass die Bewegung des simulierten Roboters möglichst der Bewegung des realen Roboters entspricht. Die Simulation soll möglichst allgemein gültig sein und nicht für eine bestimmte Laufart oder Laufrichtung optimiert werden. Daher wird in diesem Lernschritt der Unterschied der resultierenden Laufgeschwindigkeit zwischen realem und simuliertem Roboter minimiert.

Insgesamt wurden die Geschwindigkeiten von 60 verschiedenen Läufen mit verschiedenen Geschwindigkeiten auf dem realen Roboter vermessen. Diese 60 Läufe bestehen sowohl aus erprobten Laufbewegungen mit verschiedenen Geschwindigkeiten, als auch aus suboptimalen Läufen, bei denen der Roboter stark auf dem Boden rutscht oder stolpert. Auf dem realen Roboter wurden Läufe mit maximalen translatorischen Geschwindigkeiten von 480 mm/s und Rotationsgeschwindigkeiten von 220 °/s gemessen. Diese Auswahl wurde getroffen, damit der Lösungsraum möglichst gut abgedeckt wird und die optimierten Parameter nicht nur für gut funktionierende „glatte“ Laufbewegungen gültig sind. Die Geschwindigkeit des Roboters wird jeweils als Tripel gemessen, da sie aus einer translatorischen (x und y Richtung) und rotatorischen (Rotation r) Komponente besteht.

Jeder Lauf dauerte fünf Sekunden, nach der ersten Sekunde wurde die aktuelle Position des Roboters als Startposition P_S gespeichert und nach Ablauf der fünf Sekunden die Endposition P_E. Aus Start- und Endposition sowie der Zeitdifferenz von vier Sekunden kann nun die tatsächliche Geschwindigkeit des Roboters bestimmt werden. In der Simulation wurde die Roboterposition von einem *Supervisor* gemessen, der Zugriff auf alle Roboterdaten hat. Die Geschwindigkeit des realen Roboters wurde mit einem Deckenkamerasystem

vermessen, das mit der Formel für die Geschwindigkeitsberechnung später in Kapitel 5.2.2 genauer erläutert wird.

Die Fitnessberechnung in diesem Schritt erfolgte aus der quadrierten Differenz der realen und simulierten Robotergeschwindigkeit. Da die quadrierten Differenzen der einzelnen Komponenten der Geschwindigkeit aufsummiert wurden, wurde zuvor die Rotation mit einem Gewichtungsfaktor multipliziert, um sie in einen vergleichbaren numerischen Bereich wie die Translationsgeschwindigkeiten zu überführen.

In diesem Schritt werden alle acht zuvor erläuterten globalen Parameter optimiert, sowie die maximalen Drehmomente und PID Einstellungen der Motoren. Die Motorparameter wurden mit den Werten des letzten Optimierungsschrittes und einer geringen Mutationsstärke initialisiert.

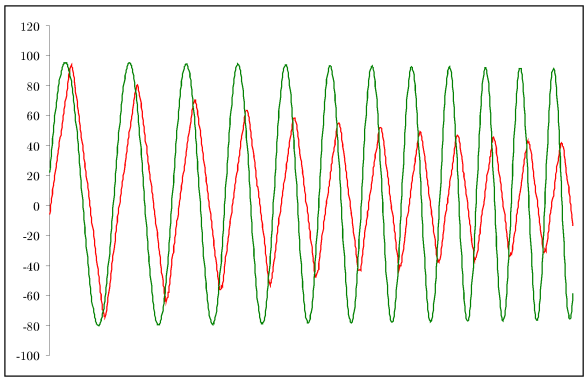
4.4 Ergebnisse der Lernschritte

Die Ergebnisse der drei einzelnen Lernschritte werden im Folgenden vorgestellt und bewertet. Die Fitness jedes Individuums der Evolutionsstrategie kann nur evaluiert werden, indem mit den Parametern, die das entsprechende Individuum beschreiben, eine Simulation durchgeführt wird und die Ergebnisse der Simulation mit denen des realen Roboters verglichen werden. Da eine Vielzahl von Fitnessauswertungen für die Optimierung erwartet wurde, wurden die Simulationen in einem Cluster mit 60 Rechnern durchgeführt. Jeder Rechner war mit einer Intel Pentium 4 CPU mit 2,4 GHz ausgerüstet und konnte damit die Simulation etwas schneller als in Echtzeit durchführen. Die einzelnen Rechner waren mit einer Serverapplikation verbunden, die die auszuwertenden Individuen an die verfügbaren Clients verteilt hat. Wenn ein Client bei der Simulation abgestürzt ist – was für manche Parameterkombinationen vorkommen konnte – wurde der Fitness dieses Indiviuums der schlechtmöglichste Wert zugewiesen, um auszuschließen, dass dieses Individuum einen Beitrag für die nächste Generation liefert.

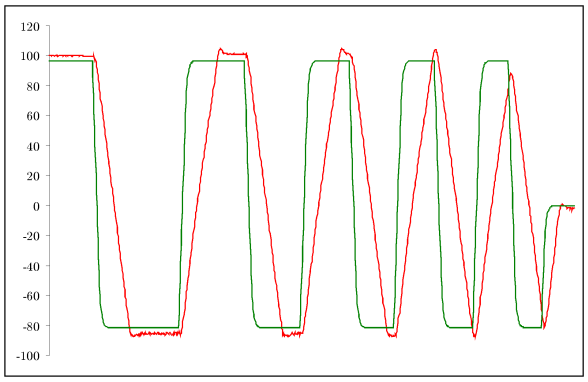
Jeder Lernschritt wird auch in Bezug auf die resultierende simulierte Robotergeschwindigkeit bewertet. Dies war zwar nicht das primäre Ziel der ersten beiden Lernschritte, wird aber hier dazu herangezogen, um zu überprüfen, ob die ersten beiden Lernschritte der Annäherung an das Gesamtziel zuträglich sind. Die 60 vermessenen Läufe wurden randomisiert in zwei Gruppen A und B aufgeteilt. Gruppe A bestand aus 40, Gruppe B aus 20 Läufen und Gruppe A+B dementsprechend aus allen 60 Läufen. In Tabelle 4.1 sind die Abweichungen zwischen der Geschwindigkeit des realen und des simulierten Roboters für alle Lernstufen aufgeführt. Für jede Laufrichtung (x vorwärts, y seitwärts, r Rotation) ist der Betrag der maximalen Abweichung und die Standardabweichung der Läufe der entsprechenden Gruppe angegeben.

In der ersten Lernstufe wurden die maximalen Geschwindigkeiten und eine vorläufige PID Einstellung für die Motoren gelernt. Dazu wurde eine $(5, 25)$ Evolutionsstrategie eingesetzt.

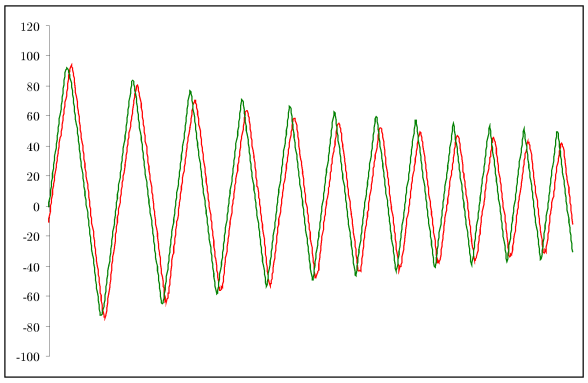
Bereits nach 80 Generationen wurde die Optimierung in dieser Stufe gestoppt, da eine klare Konvergenz der Fitness erfolgte. Abbildung 4.6 zeigt zwei Teile der Bewegungssequenz des rechten vorderen Schultergelenks, jeweils vor und nach der Optimierung. Die simulierte Bewegung (grün gezeichnet) kommt in beiden Ausschnitten der realen Bewegung sehr nahe. Die Optimierung der Geschwindigkeit zeigt auch einen eindeutigen Effekt auf die Laufbewegung. Alle maximalen Abweichungen und Standardabweichungen wurden signifikant reduziert. Die größte Abweichung mit den initialen Parametern betrug noch 1227,47 mm/s und wurde nach der ersten Lernstufe bereits auf 386,70 mm/s reduziert. Die Geschwindigkeiten aus Lernstufe 1 werden als endgültig betrachtet und dementsprechend in den folgenden Lernstufen nicht mehr variiert. Die gelernten PID Werte hingegen werden als Startparameter für Stufe 2 benutzt.



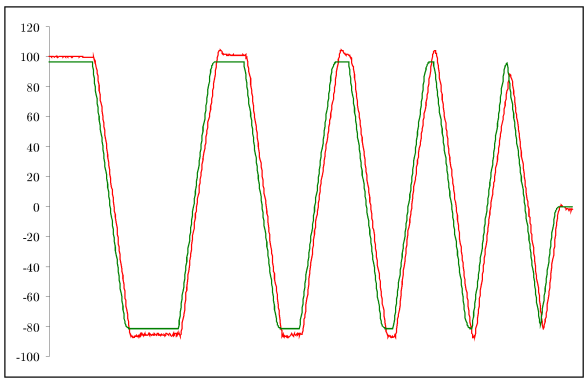
(a) Sinusförmige Ansteuerung vor Lernschritt 1



(b) Rechteckförmige Ansteuerung vor Lernschritt 1



(c) Sinusförmige Ansteuerung nach Lernschritt 1

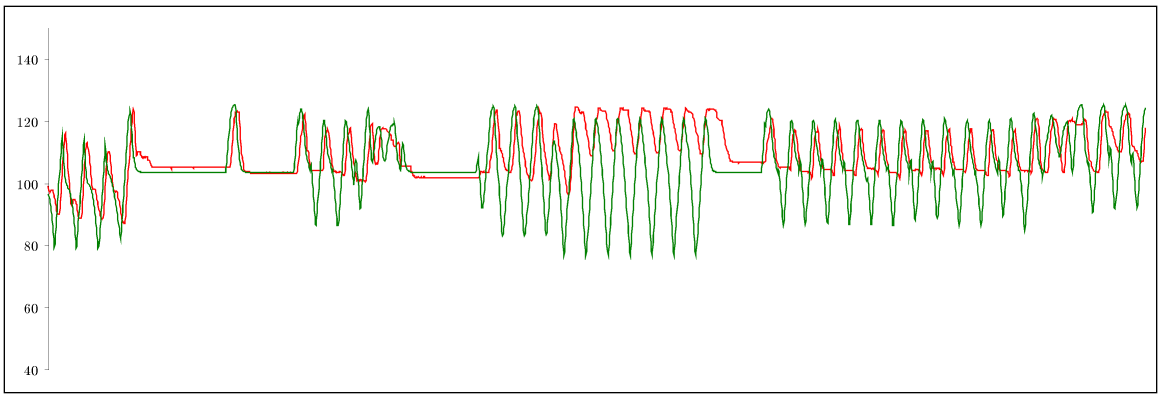


(d) Rechteckförmige Ansteuerung nach Lernschritt 1

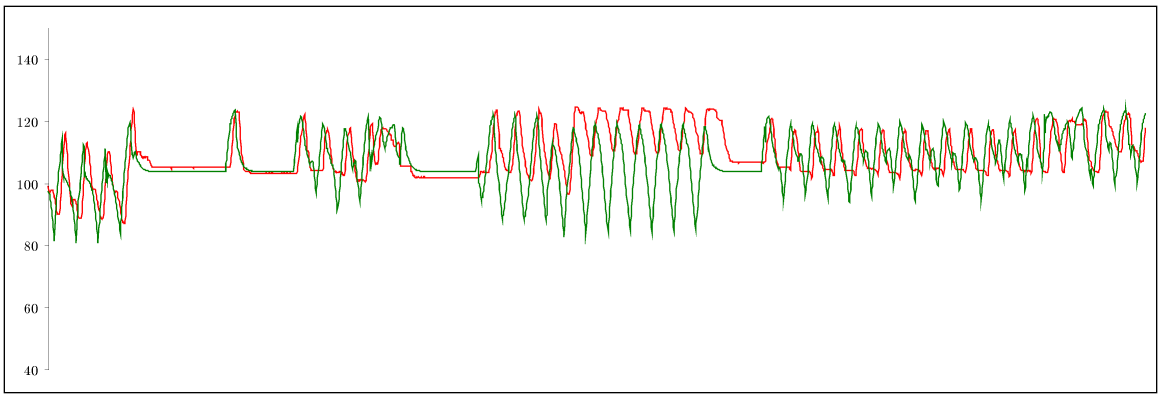
Abbildung 4.6: Auszug der simulierten (grün) und der realen (rot) Bewegungssequenz vor und nach dem ersten Lernschritt.

In der zweiten Lernstufe wurden vorläufige maximale Drehmomente und PID Werte der Motoren gelernt. Die Werte dienen nur als Intialisierungen für die folgende Lernstufe, da die maximalen Drehmomente beispielsweise von dem Reibungskoeffizient zwischen Roboter

und Boden abhängen. Für diesen Lernschritt wurde eine $(6, 30)$ Strategie gewählt, die nach 400 Generationen gestoppt wurde, da ab der 200. Generation keine signifikante Verbesserung der Fitness mehr eingetreten ist. Abbildung 4.7 zeigt einen Teil der benutzten Bewegungssequenz vor und nach diesem Lernschritt für das rechte Kniegelenk. Die Bewegung des simulierten Kniegelenks folgt dem realen Gelenk nach der Optimierung besser, wenn auch in einem Teil noch recht große Defizite auftreten. Die Ergebnisse dieses Lernschritts wirken sich ebenfalls auf die resultierenden Laufgeschwindigkeiten aus. Über alle verfügbaren 60 Läufe gesehen (Gruppe A+B) wurden alle maximalen Abweichungen zwischen realen und simulierten Läufen reduziert, die Standardabweichungen für Seitwärtslaufen und die Rotation wurden ebenfalls reduziert, die Standardabweichung für das Vorwärtslaufen hingegen stieg wieder von 80,57 mm/s auf 84,49 mm/s und wurde somit leicht erhöht.



(a) Bewegung vor Lernschritt 2



(b) Bewegung nach Lernschritt 2

Abbildung 4.7: Auszug der simulierten (grün) und der realen (rot) Bewegungssequenz des vorderen rechten Kniegelenks vor und nach dem zweiten Lernschritt.

Für die letzte Lernstufe wurde eine $(9, 50)$ Evolutionsstrategie gewählt, da insgesamt 32 Parameter gleichzeitig optimiert wurden (8 globale Parameter und die maximalen Drehmomente und PID Werte der 6 Motoren). Die maximalen Drehmomente und PID Werte wurden aus der vorherigen Optimierungsstufe übernommen und mit verhältnismäßig klei-

		Gruppe A		Gruppe B		Gruppe A+B	
		StdAbw	MaxAbw	StdAbw	MaxAbw	StdAbw	MaxAbw
Initial	x [mm/s]	144,51	574,16	273,33	1227,47	198,78	1227,47
	y [mm/s]	114,67	610,39	91,81	258,69	107,23	610,39
	r [°/s]	20,63	57,87	40,11	134,07	28,65	134,07
Stufe 1	x [mm/s]	92,12	386,70	53,81	131,25	80,57	386,70
	y [mm/s]	48,77	180,27	119,93	372,77	82,01	372,77
	r [°/s]	15,47	37,24	24,06	88,24	18,91	88,24
Stufe 2	x [mm/s]	89,78	368,90	72,65	264,94	84,49	368,90
	y [mm/s]	50,10	185,76	82,31	258,55	62,44	258,55
	r [°/s]	17,19	42,97	17,76	57,87	17,30	57,87
Stufe 3	x [mm/s]	40,00	113,47	55,55	183,40	45,48	183,40
	y [mm/s]	35,49	150,34	75,69	255,98	52,26	255,98
	r [°/s]	10,31	24,06	15,47	41,25	12,61	41,25
Webots	x [mm/s]	123,65	442,58	91,90	199,86	113,01	442,58
	y [mm/s]	147,80	421,33	218,06	502,51	173,35	502,51
	r [°/s]	44,12	178,76	42,97	129,49	43,54	178,76

Tabelle 4.1: Absolute maximale und Standardabweichungen der Differenzen der Geschwindigkeiten von realem und simuliertem Roboter.

nen Mutationsstärken initialisiert. Für das Lernen der Parameter wurden in diesem Schritt nur die 40 Läufe aus Gruppe A genutzt. Die Läufe in Gruppe B dienen in dieser Stufe nur als Kriterium, um abschließend zu prüfen, ob die Optimierung ein allgemeingültiges Ergebnis liefert, das ebenfalls auf die Läufe aus Gruppe B übertragbar ist. Die Evolution wurde in dieser Stufe nach 200 Generationen gestoppt. Erneut wurden durch diese Optimierungsstufe alle maximalen Abweichungen und Standardabweichungen klar reduziert - und dies sowohl in Gruppe A, als auch in der Kontrollgruppe B.

Eine Analyse der Läufe, die mit den gelernten Parametern zu den größten Abweichungen führen, hat eine Schwachstelle in dem genutzten Robotermodell aufgedeckt: Die Simulation des Aibos nutzt für alle Körperteile mit dem Untergrund nur einen Reibungskoeffizienten. Dies ist problematisch, da viele Laufbewegungen auf den vorderen Ellbögen laufen und die Beine quasi wie Ski nutzen, da die Reibung zwischen der Kunststoffummantelung der Beine und dem Boden sehr gering ist. Bei einigen Rückwärtsläufen hingegen drückt sich der Roboter mit den Gummiabdeckungen am Ende der Beine nach hinten ab und nutzt dabei die gute Haftung des Gummis auf dem Boden aus. Diese zwei unterschiedlichen Reibungskoeffizienten sind momentan in dem SimRobot Modell für den Aibo nicht vorgesehen.

Abschließend wird die Simulation mit den gelernten Parametern noch mit dem kommerziellen Robotersimulator *Webots* verglichen. Dazu wurde ein Interface implementiert, so dass die zuvor aufgezeichneten Läufe der Gruppen A und B in diesem Simulator abgespielt und vermessen werden konnten. In [26] wird dem Webots Simulator eine gute Realitäts-

treue bescheinigt, in dem ein Lauf eines realen und eines simulierten Roboters miteinander verglichen wird. Dieser Lauf ist aber mit 35 mm/s auf dem realen Roboter und 32,5 mm/s auf dem simulierten Roboter sehr weit von dem Limit des Möglichen (ca. 480 mm/s) entfernt und stößt damit keinesfalls an die physikalischen Grenzen der Motoren. Die letzte Zeile in Tabelle 4.1 zeigt die ermittelten Geschwindigkeitsdifferenzen. Während sich der Webots Simulator in einigen Abweichungen besser schlägt als SimRobot mit den initialen Parametern, wird er von SimRobot bereits nach der ersten Lernstufe in allen Bereichen übertroffen.

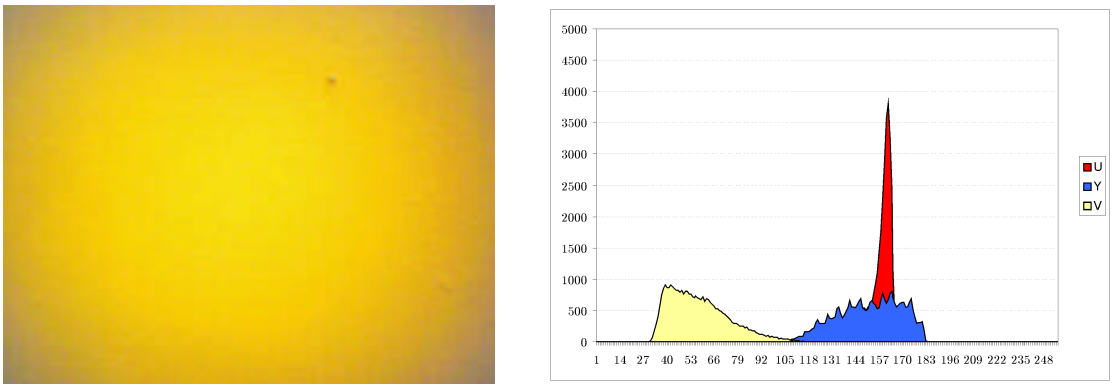
4.5 Mögliche Erweiterungen

Die zuletzt erwähnte Schwäche, dass momentan in SimRobot nur eine globale Reibung realisiert ist, lässt offenbar den Schluss zu, dass mit zwei oder mehr Reibungskoeffizienten die Simulation nochmals realistischer gestaltet werden kann. Weiterhin ist denkbar, in einer weiteren Lernstufe Gewichte oder Massenschwerpunkte von Körperteilen ebenfalls zu optimieren.

Kapitel 5

Optimierung von Laufmodellen mit der Evolutionsstrategie

Viele Algorithmen der Robotik enthalten parametrisierte Modelle. Die Einstellungen der Parameter haben typischerweise einen großen Einfluss auf die Qualität des Modells. Das Auffinden einer Parametereinstellung, die die Qualität des Modelles optimiert, ist häufig eine herausfordernde Aufgabe. Insbesondere, wenn das zugrunde liegende Problem nicht mathematisch beschrieben werden kann. Dann ist häufig die einzige Möglichkeit, Parametervariationen auszutesten und den Effekt so zu beobachten. Wie schon in Kapitel 3 motiviert, eignen sich Evolutionsstrategien sehr gut zur Optimierung dieser Probleme.



(a) Kamerabild (b) Histogramm

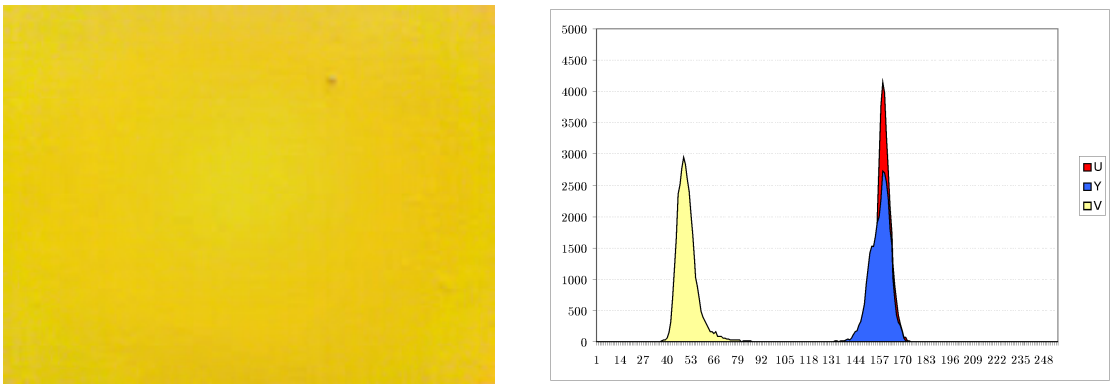
Abbildung 5.1: Kamerabild mit Farbverfälschungen

Ein gutes Beispiel für eine Anwendung dieser „Black Box“ Optimierung ist das Auffinden von Parametern für die Korrektur von Kamerabildern. Abbildung 5.1a zeigt das Kamerabild von einem Aibo ERS 7 Roboter, der auf eine gelbe Farbkarte blickt. Insbesondere in dem Randbereich weist das Gelb starke Verfärbungen in Richtung Blau. Dieser Effekt wird

als Vignetteneffekt bezeichnet und tritt insbesondere bei Kameras einfacher Bauart auf.

Da im Roboterfußball die wichtigsten Objekte am einfachsten über ihre Farbe erkannt werden können, muss eine Methode entwickelt werden, die die Farbe in dem Bild softwareseitig korrigiert. In Abbildung 5.1b ist das Histogramm der drei Farbkanäle Y,U und V des zu korrigierenden Bildes dargestellt. Ein perfektes, nicht farbverfälschtes Bild müsste in dem Histogramm für jeden Kanal einen scharfen Peak aufweisen, möglichst nur auf einem einzigen Farbwert. Wie das Histogramm andeutet, sind hier also insbesondere der Y und der V Kanal von der Störung betroffen.

In [40] wurde zur Lösung dieses Problems ein Modell aufgestellt, das die einzelnen Kanäle des Kamerabilds mit einer Korrekturfunktion überlagert und so das Gesamtbild korrigiert. Das Modell enthält 23 Parameter, die nur unzureichend von Hand einzustellen sind. Daher wurde die Optimierung mit einer Evolutionsstrategie durchgeführt. Als Fitness wurde die „Breite“ des Peaks im Histogramm des entsprechenden Farbkanals betrachtet. Je schmaler der Peak, umso besser ist die Fitness und umso weniger Farbverfälschungen weist das Kamerabild auf. Das Ergebnis dieser Optimierung ist in Abbildung 5.2 dargestellt. Für das menschliche Auge sind in dem korrigierten Kamerabild kaum noch Farbverfälschungen zu erkennen. Dementsprechend sind auch drei wesentlich schärfere Peaks in dem Histogramm zu erkennen.



(a) Kamerabild (b) Histogramm

Abbildung 5.2: Korrigiertes Kamerabild

In Kapitel 2 wurden, ebenso wie das Modell zur Farbkorrektur, parametrisierte Modelle entwickelt, die Bewegungen erzeugen können, mit denen der Roboter Laufbewegungen durchführen kann. Wie in diesem Beispiel der Farbkorrektur, ist der direkte Effekt der Parameter auf das Resultat, also die Qualität (wie z. B. die Geschwindigkeit) des Laufes, unbekannt. Daher werden im Folgenden ebenfalls Evolutionäre Strategien eingesetzt, um die Parameter der Laufmodelle für einen zweibeinigen und einen vierbeinigen Roboter zu optimieren.

5.1 Optimierung der Laufbewegung eines humanoiden Roboters

Das Laufmodell aus Kapitel 2.2 für den Kondo KHR-1 wird zuerst im Hinblick auf die resultierende Laufgeschwindigkeit optimiert. Eine Besonderheit weist diese Optimierung auf. Während die Wahl der Fitness als zu maximierende Laufgeschwindigkeit relativ einfach zu bestimmen ist, kann es bei diesem Roboter passieren, dass durch die Wahl der Laufparameter der Roboter umfällt. Individuen, die zu einem solchen Verhalten führen, werden mit einem Fitnesswert von Null bestraft und haben so kaum eine Chance, einen Beitrag für die nächste Generation zu leisten.

Bevor die eigentliche Optimierung des Laufmodells für den realen Roboter durchgeführt wird, werden in einem physikalischen Simulator dieses Roboters verschiedene Einstellungen, wie z. B. Populationsgröße oder Selektionsverfahren, für Evolutionsstrategien miteinander verglichen. Die vielversprechendste Strategie wird schließlich benutzt, um auf dem realen Roboter die Parameter zu optimieren.

Zusammenfassend werden noch einmal kurz die zu optimierenden Parameter mit sinnvollen Schranken zusammengefasst. Werden die angegebenen Schranken für einen Parameter durch die Mutation überschritten, wird der Parameter entsprechend auf den Wert der Schranke geclippt.

5.1.1 Zu optimierende Parameter des zweibeinigen Laufmodells

Insgesamt wurden für das Laufmodell 15 Parameter vorgesehen, die noch einmal zusammengefasst vorgestellt werden.

- **Schrittdauer**
  Die Schrittdauer t_{Step} bestimmt die Zeit für einen kompletten Schritt (beide Beine fahren ihre Trajektorie jeweils einmal komplett ab). In Millisekunden angegeben, sind sinnvolle Werte für die Schrittdauer im Bereich $700 \leq t_{Step} \leq 3000$ zu finden. Die untere Grenze von 700 ms ergibt sich aus der maximalen Servogeschwindigkeit und der Stabilität des Roboters bei zu hohen Servogeschwindigkeiten. Die obere Grenze von 3000 ms ist gewählt, da eine zu langsame Beinbewegung nur durch ein statisch stabiles Laufen erreicht werden kann, das wiederum nur zu sehr langsamen Laufgeschwindigkeiten führen kann.

- **Schritthöhe**
  Die maximale Höhe (in z-Richtung) der Halbellipse wird bestimmt durch den Parameter s_h. Der Parameter bestimmt also, wie hoch der Fuß in die Luft gehoben wird. Der minimale Wert wird auf 0, der maximale Wert auf 50 mm festgelegt, der sich durch die Länge der Beine ergibt. Es gilt also der Wertebereich $0 \leq s_h \leq 50$ mm.

- **Schrittlänge**
  Die maximale Länge der halbelliptischen Trajektorie (in x Richtung) wird durch den Parameter s_l bestimmt. Begrenzt durch die Beinlänge sind sinnvolle Grenzen gegeben durch $0 \leq s_l \leq 100$ mm.

- **Arm Amplitude in x Richtung**
  Die maximale Auslenkung der Arme in der x-z-Ebene wird über den Parameter $x_{arm_{max}}$ bestimmt. Als Wertebereich wird $0 \leq x_{arm_{max}} \leq 1$ festgelegt. Ein Wert von $x_{arm_{max}} = 0$ entspricht einem unbeweglichen Arm, $x_{arm_{max}} = 1$ hingegen lässt den Arm mit maximaler Amplitude nach vorne und hinten pendeln, so dass sich eine maximale Auslenkung von insgesamt 180° ergibt.

- **Phasenlage der Armbewegung in x Richtung**
  Die Phasenlage der Armbewegung in die Laufrichtung in Bezug auf die Beinbewegung wird über den Parameter $\varphi_{arm_{xz}}$ bestimmt. Theoretisch ist hier ein Wertebereich von $0 \leq \varphi_{arm_{xz}} \leq 1$ möglich. Es wird aber eine Phasenlage von $\varphi_{arm_{xz}} = 0,25$ festgelegt, so dass der linke Arm nach vorne pendelt, wenn sich das rechte Bein nach vorne bewegt, und umgekehrt. Eine Variation dieses Parameters macht wegen der symmetrischen Art des Laufens keinen Sinn.

- **Arm Amplitude in y Richtung**
  Die maximale Auslenkung der Arme in der y-z-Ebene wird über den Parameter $y_{arm_{max}}$ definiert. Ein Wertebereich von $0 \leq y_{arm_{max}} \leq 1$ ist möglich. Der Wert $y_{arm_{max}} = 0$ entspricht einem unbeweglichen, senkrecht nach unten gerichteten Arm (Armgelenk 2 um 90° nach unten geknickt), die Einstellung $y_{arm_{max}} = 1$ lässt den Arm mit maximaler Amplitude zur Seite bis hin zur Waagerechten schwingen.

- **Phasenlage der Armbewegung in y Richtung**
  Die Phasenlage der Armbewegung in der y-z-Ebene relativ zu der Beinbewegung wird über den Parameter $\varphi_{arm_{yz}}$ bestimmt. Die Phase wird ebenfalls konstant auf $\varphi_{arm_{yz}} = 0,5$ gesetzt, so dass der linke Arm nach oben pendelt, wenn sich das rechte Bein nach oben bewegt und umgekehrt. Eine Variation dieses Parameters macht wegen der symmetrischen Art des Laufens keinen Sinn.

- **Amplitude der Vorwärtsschwingung**
  Der Parameter $x_{body_{max}}$ definiert die maximale Auslenkung des Oberkörpers in x-Richtung. Der Wertebereich beträgt $0 \leq x_{body_{max}} \leq 1$. Ein Wert von $x_{body_{max}} = 0$ entspricht einem unbeweglichen, stets senkrecht aufgerichteten Oberkörper, $x_{body_{max}} = 1$ ließe den Oberkörper um 90° nach vorne schwingen.

- **Phasenlage der Vorwärtsschwingung**
  Die Phasenlage der Vorwärtsschwingung in Bezug auf die Beinbewegung wird bestimmt über den Parameter φ_{body_x}. Ein Wertebereich von $0 \leq \varphi_{body_x} \leq 1$ möglich, wird hier jedoch auf $\varphi_{body_x} = 0$ festgelegt, so dass der Oberkörper mit jedem einzelnen Schritt nach vorne wippt, um den Schritt in Laufrichtung mit seinem Schwung zu unterstützen.

- **Amplitude der Seitwärtsschwingung**
  Durch den Parameter $y_{body_{max}}$ wird die maximale Amplitude des Oberkörpers in y-Richtung festgelegt. Ein sinnvoller Wertebereich von $0 \leq y_{body_{max}} \leq 0,4$ ist durch die Ausmaße der Beine definiert. Die obere Grenze von $0,4$ ergibt sich, da bei Werten $y_{body_{max}} > 0,4$ die Beine mit dem Rumpf des Roboters kollidieren. Die Einstellung $y_{body_{max}} = 0$ entspricht wiederum einem unbeweglichen, senkrecht aufgerichteten Oberkörper.

- **Phasenlage der Seitwärtsschwingung**
  Der Parameter φ_{body_y} bestimmt die Phase der Seitwärtsschwingung in Relation zu der Beinbewegung. Ein Wertebereich von $0 \leq \varphi_{body_y} \leq 1$ ist zwar denkbar, wird hier aber konstant auf $\varphi_{body_y} = 0$ gesetzt, so dass sich der Oberkörper immer dann nach rechts verschiebt, wenn das linke Bein angehoben wird und umgekehrt. So verschiebt der Roboter den Schwerpunkt immer in Richtung des auf dem Boden stehenden Beines.

  Die folgenden vier Parameter wurden nicht in Kapitel 2.2.2 erläutert und haben nicht direkt mit der Generierung des Laufmusters zu tun, sondern bestimmen sozusagen die Referenzposition der Füße im Stillstand.

- **Abstand der Füße**
  Der Parameter d_y bestimmt, wie weit die Füße seitlich (in y-Richtung) im Stand voneinander entfernt sind. Jedes Bein wird also um $\frac{d_y}{2}$mm nach außen gespreizt. Für diesen Parameter wird ein Wertebereich von $8 \leq d_y \leq 50$ mm festgelegt. Werte, die kleiner als $d_y = 8$ mm betragen, verursachen ein Berühren der Füße, größere Werte als $d_y = 50$ mm sorgen dafür, dass der Roboter seine Beine so weit spreizt, dass ein Bein aufgrund der breitbeinigen Lage nicht mehr angehoben werden kann, weil das Gewicht des Roboters nicht mehr auf das andere Bein verlagert werden kann.

- **Einknicken der Beine**
  Mit dem Parameter d_z kann die Länge der Beine im Stehen eingestellt und somit der Roboterschwerpunkt gesenkt werden. Für einen Wert von $d_z = 0$ sind die Beine im Stillstand vollständig gestreckt. Für Werte von $d_z > 0$ wird die effektive Länge der Beine in z-Richtung durch ein Einknicken der Kniegelenke realisiert. Ein tieferer Schwerpunkt ist für ein stabileres und weniger schwankendes Laufen gegebenenfalls von Vorteil. Der Wertebereich wird auf $0 \leq d_z \leq (50 - s_h)$ mm mit der maximalen Schritthöhe s_h begrenzt, da bei größeren Werten als 50 mm die Servos in den Beinen so weit angewinkelt werden müssen, dass sie aneinander stoßen.

- **Fußneigung um die y-Achse**
  Der Parameter θ_{4_o} bestimmt einen festen Offset, der auf die Fußgelenkwinkel θ_4 addiert wird. Dies hat für positive Winkel $\theta_{4_o} > 0$ zur Folge, dass sich der gesamte Roboterkörper nach vorne neigt und für negative Werte entsprechend nach hinten. Dieser Wert kann nützlich sein, um den Roboter aufrecht stehen zu lassen, wenn beispielsweise der PDA an der Brust des Roboters befestigt wird, da das zusätzliche Gewicht den Roboterkörper nach vorne kippen lässt. Der Wertebereich wird auf

$-\frac{\pi}{2} \leq \theta_{4_o} \leq \frac{\pi}{2}$ festgelegt.

- **Fußneigung um die x-Achse**
  Dieser Wert θ_{5_o} wird zu den Fußgelenkswinkeln θ_5 addiert und neigt so den Roboter seitlich. Für positive Werte wird der Roboter nach links und für negative Werte nach rechts geneigt. Der Parameter kann Asymmetrien in der Gewichtsverteilung des Roboters ausgleichen und auch beim Seitswärtslaufen hilfreich sein. Auch hier ist der Wertebereich auf $-\frac{\pi}{2} \leq \theta_{5_o} \leq \frac{\pi}{2}$ begrenzt.

Durch das Festlegen der Phasenlagen ergeben sich für die Optimierung also insgesamt 11 Parameter, die zu bestimmen sind.

5.1.2 Bestimmung der Fitness

Die Bestimmung der Fitness eines Individuums wird durch Ausprobieren der Parameter, die das Individuum trägt, bestimmt. Dazu läuft der Roboter eine vorgegebene Distanz und misst die dafür benötigte Zeit. Somit kann direkt die Geschwindigkeit des Laufes errechnet werden und schließlich als zu maximierende Fitness zugewiesen werden. Fällt der Roboter beim Laufen um, wird dies über den integrierten Beschleunigungssensor erkannt und die Fitness des entsprechenden Individuums mit dem Wert 0 bestraft. Der Wert 0 stellt die schlechtmöglichste Fitness dar.

Damit der Roboter beim Loslaufen mit der maximalen Geschwindigkeit nicht stolpert, ist eine langsame Beschleunigung der Laufgeschwindigkeit implementiert. Der Roboter beginnt mit Treten auf der Stelle und vergrößert langsam die Schrittlänge bis hin zur durch die Parameter maximal vorgegebenen Schrittlänge. Erst wenn der Roboter die maximale Schrittlänge erreicht hat, wird mit der Messung begonnen. Beim Abbremsen des Roboters wird analog mit stetig kleiner werdender Schrittweite verfahren.

Die Bestimmung der Geschwindigkeit bzw. der Start- und Endposition im Simulator wird über ein so genanntes Orakel realisiert, das jederzeit die exakte Roboterposition kennt. Fällt der Roboter um, wird die Simulationsszene zurückgesetzt, so dass der Roboter sofort wieder an seiner Ausgangsposition steht und unmittelbar mit der Bestimmung der Fitness des nächsten Individuums begonnen werden kann.

Zur Bestimmung der Fitness des realen Roboters wurde ein Hilfsaufbau gefertigt, der in Abbildung 5.3 dargestellt ist. Zur Bestimmung der Geschwindigkeit wurden Lichtschranken an der Start- und Endposition der Laufstrecke angebracht. Der Roboter wurde für die Laufevolution über einen externen PC betrieben, an den die Lichtschranken angeschlossen sind. An dem Kopf des Roboters ist ein Drahtbügel angebracht, der wiederum um die oberen Streben des Aufbaus geführt ist. Fällt der Roboter bei dem Laufversuch um, bleibt er mit dem Bügel an der Strebe hängen und befindet sich somit in einer definierten Position, so dass er eine vordefinierte Aufstehbewegung ausführen kann, um selbständig

wieder aufzustehen und zurück zu der Ausgangsposition laufen zu können. Der Prozess des Laufenlernens kann also von dem Roboter vollständig autonom durchgeführt werden.

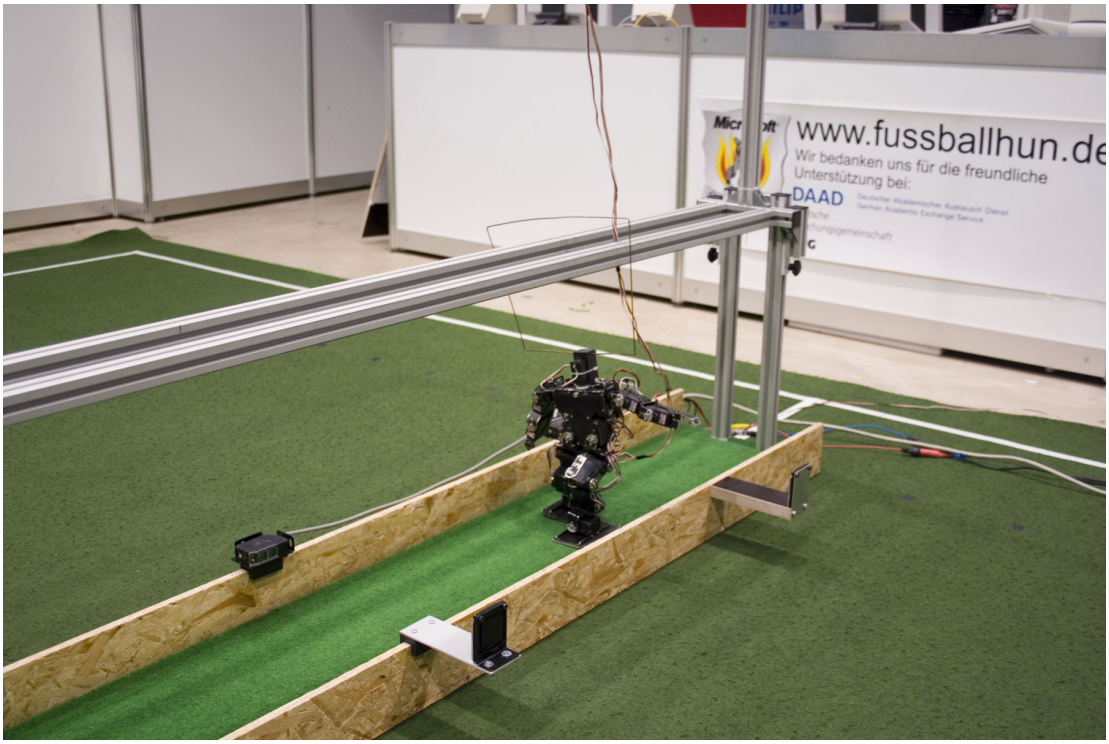


Abbildung 5.3: Gestell zum autonomen Laufenlernen eines zweibeinigen Roboters

5.1.3 Auswirkungen der Strategieparameter

Die exogenen Parameter der Evolutionsstrategie sind in Abhängigkeit der Dimension und der Struktur des Suchraumes zu wählen. In einem unimodalen Suchraum ist das Optimum sicherlich leichter zu finden, als in einem multimodalen, zerklüfteten Raum. Daher werden im Folgenden die Auswirkungen der Populationsgröße und des Selektionsoperators für die Optimierung des Laufmodells für den humanoiden Roboter untersucht. Diese Untersuchungen wurden zunächst in einem physikalischen Simulator durchgeführt.

Die $(1+1)$ Strategie

Zuerst wurde die einfache $(1+1)$ Evolutionsstrategie mit Anpassung der Mutationsstärke über die 1/5-Regel eingesetzt. Die Strategie hat einen Elter[1], der durch Mutation einen Nachkommen erzeugt. Hat der Nachkomme eine bessere Fitness als der Elter, ersetzt er

[1]Elter beschreibt die Einzahl von Eltern

den Elter, andernfalls verbleibt der Elter und erzeugt einen neuen Nachkommen. Die 1/5 Regelung wurde zuvor in Kapitel 3.1.2 erläutert.

Mit dieser Strategie war eine sinnvolle Optimierung kaum möglich. Es wurden zwar Läufe gefunden, die etwas besser als die Anfangskonfiguration waren, der Optimierungsprozess blieb aber sofort in einem lokalen Optimium hängen und regelte die Mutationsschrittweite herunter, was die Chance zum Verlassen des lokalen Optimums noch weiter reduziert hat. Insbesondere die Tatsache, dass Läufe, die in einem Umfallen des Roboters resultierten, mit der Fitness Null bewertet wurden, also immer mit der schlechtesten Fitness, führte dazu, dass die Strategie die Mutationsschrittweite aufgrund mangelnder Verbesserung heruntergeregelt hat.

Aufgrund der problematischen Schrittweitenregelung wurde testweise ein „Reset“ der Mutationsstärke auf ihren ursprünglichen Werte durchgeführt, wenn sie unter einen Schwellwert geregelt wurde. Diese Implementierung ist mit einem „Multi Start“ Ansatz vergleichbar, bei dem die Optimierung an mehreren Startpunkten im Suchraum neu gestartet wird. Abbildung 5.4 zeigt den Fitnessverlauf dieser Optimierung. Eine maximale Geschwindigkeit von ca. 110 mm/s wurde nach ungefähr 5000 Generationen (hier bei der (1+1) Strategie identisch mit Fitnessauswertungen) erreicht.

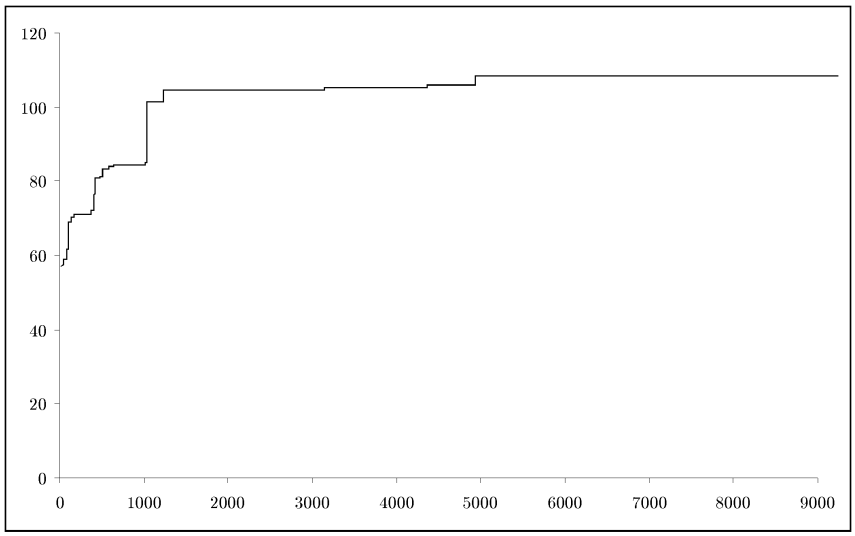


Abbildung 5.4: Laufoptimierung mit einer $(1+1)$ Evolutionsstrategie mit $\frac{1}{5}$ Schrittweitenregelung

Die $(1, 30)$ Strategie

Um zu vermeiden, dass die Strategie gegen ein lokales Optimum konvergiert, wurde als nächstes eine nicht elitäre Strategie ausgewählt, so dass der Elter der nächsten Generation nur aus den Nachkommen ausgewählt wird. Es wurde eine $(1, 30)$ Strategie gewählt, also

eine Populationsgröße von einem Elter und 30 Nachkommen. Die Nachkommen wurden durch Mutation des Elters erzeugt und die Mutationsschrittweiten durch Selbstanpassung geregelt. Abbildung 5.5 zeigt den typischen Fitnessverlauf dieser Strategie. Offensichtlich findet eine Optimierung statt, die jedoch nach ca. 170 Generationen einbricht. Dieses Phänomen war in mehreren Testläufen mit dieser Strategie an verschiedenen Stellen erkennbar. Eine Analyse hat ergeben, dass dieser Einbruch dadurch zustande kommt, dass innerhalb einer Generation alle Nachkommen nicht fähig waren zu laufen und somit mit einer Fitness von Null bewertet wurden. Der ausgewählte Elter der Nachkommengeneration war somit ebenfalls nicht in der Lage zu laufen und generierte ebenfalls nur Nachkommen, die nicht laufen konnten und somit mit der Fitness von Null bewertet wurden. Eine sinnvolle Selektion aus diesen gleich bewerteten Nachkommen ist dann nicht mehr möglich. Die beste Fitness in diesem Testdurchlauf lag bei reproduzierbaren 120 mm/s.

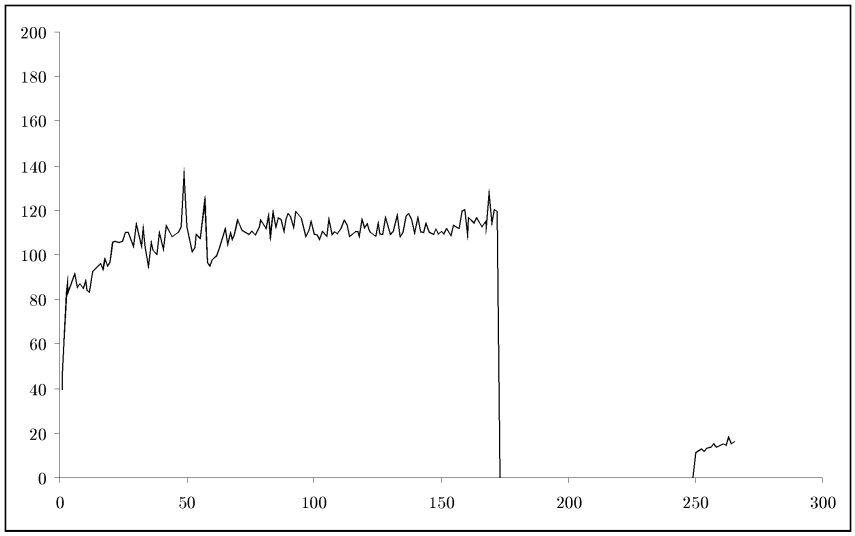


Abbildung 5.5: Laufoptimierung mit einer $(1, 30)$ Evolutionsstrategie mit Selbstanpassung

Die $(5, 30)$ Strategie

Zur besseren Exploration des Suchraumes wurde nun eine $(5, 30)$ Strategie mit Selbstanpassung gewählt, die die Nachkommen durch Rekombination zweier Eltern und anschließender Mutation erzeugt haben. Eine Anzahl von $\mu > 1$ Eltern ermöglicht eine größere Diversität der Nachkommen im Suchraum, da für $\mu = 1$ alle Nachkommen um den einen Elter herum gestreut sind. Abbildung 5.6 zeigt den deutlich steigenden Fitnesstrend mit dieser Optimierung. In allen Durchläufen mit dieser Strategie trat kein Einbruch der Fitness wie bei der $(1, 30)$ Strategie auf. Die größere Diversität der Nachkommen führte also dazu, dass immer genügend Nachkommen, die laufen konnten, zur Verfügung standen. Die beste Fitness, die mit dieser Einstellung der Strategieparameter gefunden wurde, liegt bei ca. 140 mm/s.

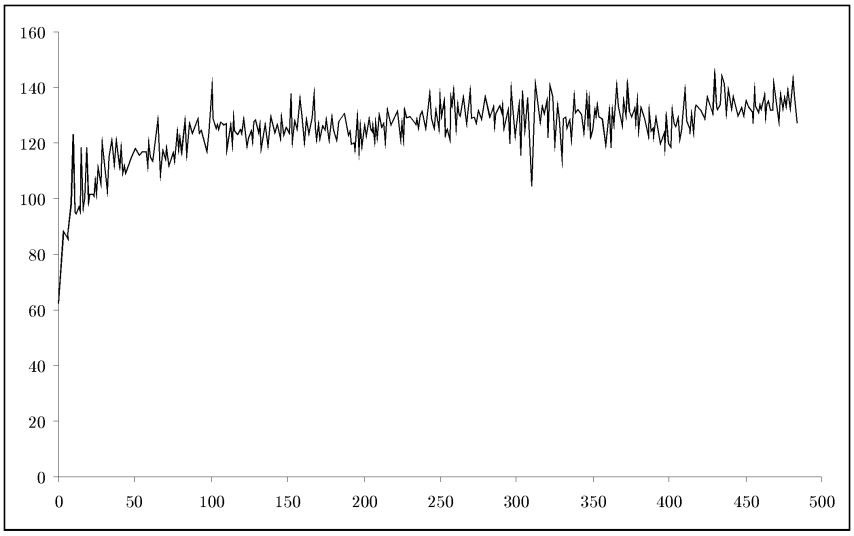


Abbildung 5.6: Laufoptimierung mit einer $(5, 30)$ Evolutionsstrategie mit Selbstanpassung

Die $(5 + 30)$ **Strategie mit maximaler Lebensdauer von** $\kappa = 3$

Zuletzt wird der Einfluss des Selektionsoperators bei gleicher Populationsgröße untersucht. Bei der $(5+30)$ Strategie werden die 5 Eltern aus den besten Nachkommen **und** den Eltern der Vorgängergeneration ausgewählt. Da bei reinen Plus-Strategien die Selbstanpassung nicht so gut funktioniert wie bei Komma-Strategien, wird hier eine maximale Lebensdauer von $\kappa = 3$ Generationen für die Eltern festgelegt. Auf diese Weise ist auch sichergestellt, dass Eltern, die durch Messfehler zu gut bewertet wurden, nach spätestens drei Generationen aussortiert werden und so den Optimierungsprozess nicht weiter gefährden können. In Abbildung 5.7 ist der Fitnessverlauf eines Optimierungsdurchgangs mit dieser Strategie dargestellt. Der teilweise treppenartige Verlauf zeigt deutlich, dass häufiger ein Elter über mehrere Generationen überlebt hat, bevor er aussortiert wurde. Es sind in dem Verlauf einige Peaks oberhalb von 200 mm/s zu erkennen, die sich jedoch bei späteren Überprüfungen als Messfehler herausgestellt haben, die tatsächlich höchstens bei 150 mm/s lagen. Diese Messfehler lassen darauf schließen, dass die maximale Lebensdauer κ nicht zu hoch gewählt werden sollte, da ansonsten gegebenenfalls irrtümlich zu gut bewertete Individuen die Optimierung dominieren können.

Insgesamt wurden mit der $(5 + 30)$ Strategie mit $\kappa = 3$ in mehreren Durchläufen die besten Resultate erzielt. Offenbar wurde die Optimierung durch die längere Existenz guter Individuen stabilisiert. Reproduzierbar wurden Geschwindigkeiten von mehr als 150 mm/s im Simulator erzielt.

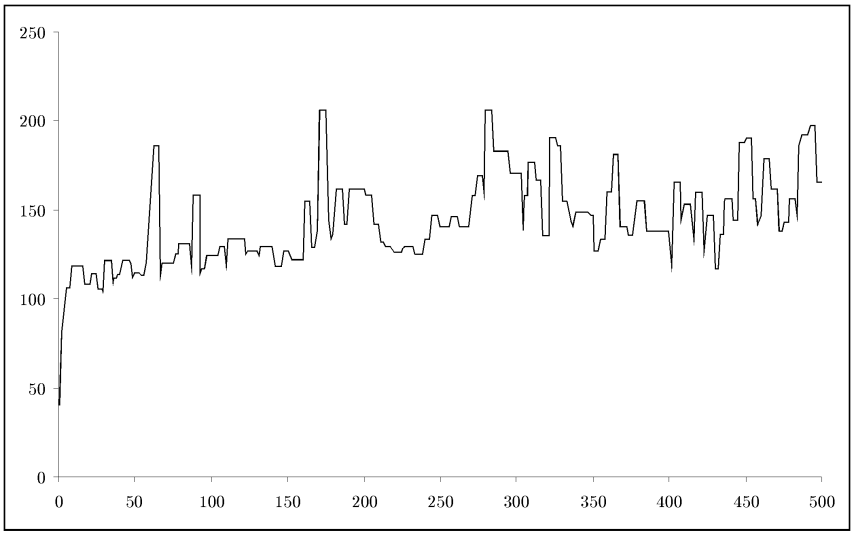


Abbildung 5.7: Laufoptimierung mit einer (5+30) Evolutionsstrategie mit Selbstanpassung und maximaler Lebensdauer $\kappa = 3$

5.1.4 Ergebnisse der Laufoptimierung auf dem realen Roboter

Da sich die $(5 + 30)$ Strategie mit $\kappa = 3$ in der Simulation am besten bewährt hat, werden diese Strategieeinstellungen für die Optimierung der Laufparameter auf dem realen Roboter eingesetzt. Der Roboter wurde aufgrund der begrenzten Laufzeit von einer externen Spannungsquelle gespeist. Wie schon erwähnt, erfolgte der komplette Evolutionsprozess völlig autonom. Wenn der Roboter hinfiel, richtete er sich wieder selbständig auf, bewerte dieses Individuum mit der Fitness 0 und lief zu der Ausgangsposition zurück. Abbildung 5.8 stellt den Fitnessverlauf der Optimierung dar.

Der schnellste Vorwärtslauf wurde nach 40 Generationen gefunden und erreicht eine Geschwindigkeit von 220 mm/s. Im Vergleich dazu wurde in [3] eine Laufgeschwindigkeit von 100 mm/s auf einem baugleichen Roboter erzielt. Es sei aber erwähnt, dass hier die 220 mm/s ohne Batterien und PDA an Bord des Roboters erreicht wurden und daher die erreichten maximalen Geschwindigkeiten nicht direkt vergleichbar sind.

5.2 Optimierung des Laufes eines vierbeinigen Roboters

In der RoboCup Four-Legged League, in der der Aibo ERS 7 als Roboter vorgeschrieben ist, ist es mittlerweile Stand der Technik, die Laufmusteroptimierung mittels Lernverfahren durchzuführen [9, 22, 34, 49], da die Anzahl der Parameter nicht per Hand einzustellen ist. Hier wird ebenfalls die resultierende Laufgeschwindigkeit als Fitness optimiert. Läufe, die

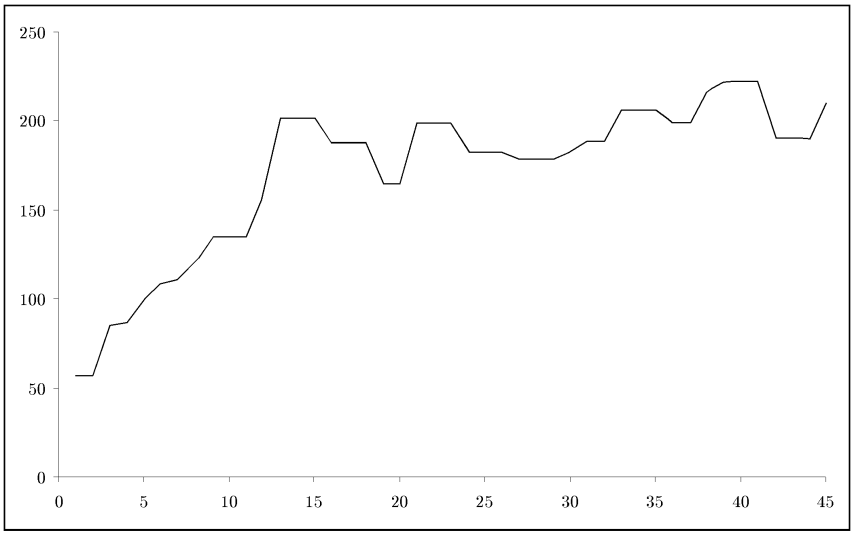


Abbildung 5.8: Fitnessverlauf der Laufevolution auf dem realen KHR-1

sich nicht in die gewünschte Richtung bewegen (z. B. eine Rotation beim Vorwärtslaufen durchführen) oder gar im Umfallen des Roboters resultieren, werden mit der schlechtmöglichsten Fitness 0 bestraft.

5.2.1 Parameter des Laufmodells

In Kapitel 2.1.2 wurde das hier genutzte Modell genauer beschrieben. Der Vollständigkeit halber werden die hier zu optimierenden Parameter noch einmal erläutert.

Die Beine bewegen sich, wie in Abbildung 5.9 gezeigt, entlang von dreidimensionalen Polygonen, die aus jeweils vier Stützpunkten $P_1, \ldots, P_4$ bestehen. Pro Stützpunkt ergeben sich also die drei Parameter $P^{(x)}, P^{(y)}, P^{(z)}$ zur Beschreibung der Position im Raum. Für die Stützpunkte des Polygons werden der Optimierung keine Grenzen vorgegeben, da die inverse Kinematik derart implementiert ist, dass sie automatisch bei nicht erreichbaren Punkten den nächsten erreichbaren Punkt ansteuert. Zusätzlich wird für jede der vier Kanten des Polygons eine relative Zeit $t_1, \ldots, t_4$ definiert, in der das Bein von dem Startpunkt zum Endpunkt der jeweiligen Strecke bewegt wird. Da die Zeiten relativ zu einer Gesamtzeit t_{step} definiert werden, gilt $t_1 + t_2 + t_3 + t_4 = 1$. Damit diese Vorgabe stets eingehalten wird, wird hier nur t_1, t_2 und t_3 optimiert, da sich t_4 automatisch durch $t_4 = 1 - t_1 - t_2 - t_3$ ergibt. Zusätzlich wurden noch für jede Zeit die sinnvolle Schranken $0 \leq t_i \leq 0,5$ definiert. Sollte eine Zeit durch Mutation außerhalb dieses Bereiches liegen, wird sie zurück auf die entsprechende Schranke geclippt. Für die Bewegung eines Beines entlang der Trajektorie ergeben sich damit die 12 Parameter für die Stützpunkte des Polygons, 3 relative Zeitparameter und die Gesamtzeit t_{step}, also insgesamt 16 Parameter.

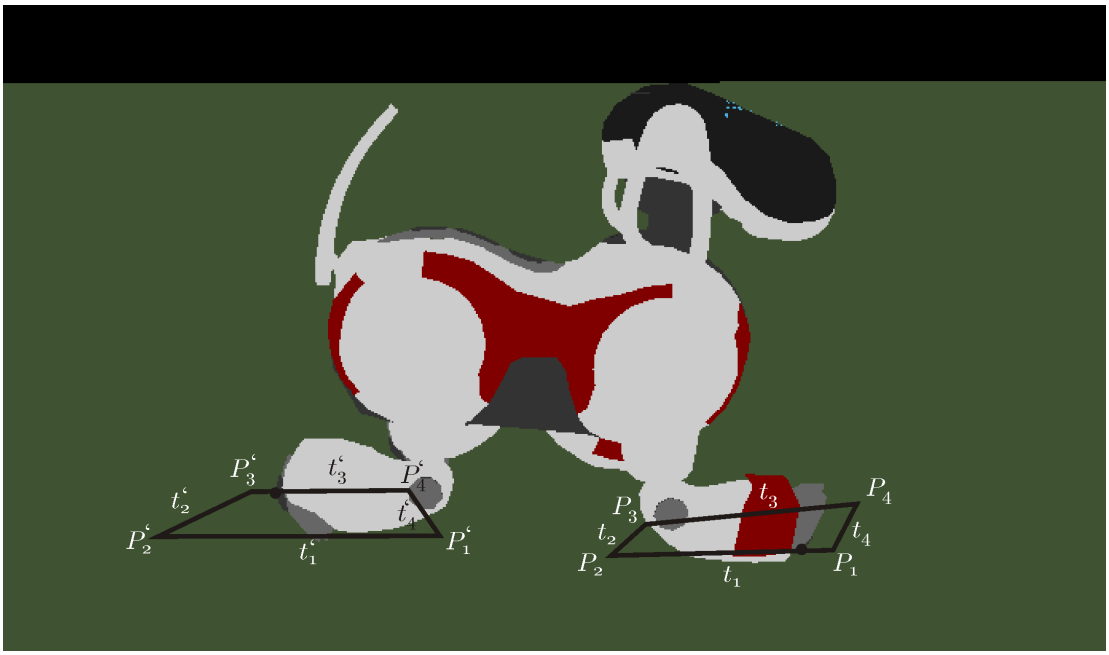


Abbildung 5.9: Laufbewegung des Aibo ERS 7

Aus Symmetriegründen sind die Trajektorien der beiden Vorderbeine und der beiden Hinterbeine identisch, jedoch aufgrund der Vorgabe des Laufmusters „Trab“ in der Phase um 0,5 verschoben. Die Trajektorien der Hinterbeine unterscheiden sich von den Trajektorien der Vorderbeine. Die Zeit für einen vollen Schritt t_{step} muss jedoch identisch sein. Daher ergeben sich für die gesamte Optimierung 31 Parameter.

5.2.2 Bestimmung der Laufgeschwindigkeit

Zur Bestimmung der Fitness wird auch für den Aibo Roboter die erreichte Laufgeschwindigkeit $\vec{v} = (v_x\, v_y\, v_r)^T$ in die entsprechende Richtung herangezogen. Die Indizes x und y deuten die Bewegungsrichtung in der Ebene an, r hingegen die Rotation des Roboters. Die Geschwindigkeit wird hier aus der Startposition $\vec{p_0} = (x_0\, y_0\, r_0)^T$ zur Zeit $t_0 = 0$, der Endposition $\vec{p_t} = (x_t\, y_t\, r_t)^T$ und der verstrichenen Zeit t errechnet. Es wird in der folgenden Rechnung davon ausgegangen, dass die Geschwindigkeit in dem betreffenden Zeitinterval konstant ist. Die Position $\vec{p}(t)$ zur Zeit t ist allgemein gegeben durch die Bewegungsglei-

chungen

$$r_t = r_0 + v_r t \tag{5.1}$$

$$x_t = x_0 + \int_0^t v_x \cos r(\tau) - v_y \sin r(\tau)\, \mathrm{d}\tau \tag{5.2}$$

$$y_t = y_0 + \int_0^t v_x \sin r(\tau) + v_y \cos r(\tau)\, \mathrm{d}\tau. \tag{5.3}$$

Die mittlere Rotationsgeschwindigkeit v_r zwischen Startpunkt $\vec{p}_0$ und Endpunkt $\vec{p}_t$ in der Zeit t ergibt sich durch einfaches Umstellen von Gleichung 5.1 zu

$$v_r = \frac{r_t - r_0}{t}. \tag{5.4}$$

Für die Auflösung der Gleichungen 5.2 und 5.3 nach den Geschwindigkeiten v_y und v_x werden die Gleichungen durch $z = x + iy$, bzw. $v_z = v_x + iv_y$ zunächst komplexwertig geschrieben.

Mit den Beziehungen

$$\begin{array}{llll} \mathrm{e}^{i\alpha} = \cos\alpha + i\sin\alpha & \mathrm{e}^{i\alpha} + \mathrm{e}^{-i\alpha} = 2\cos\alpha & |\,\mathrm{e}^{i\alpha}\,|^2 = \cos^2\alpha + \sin^2\alpha = 1 \\ \mathrm{e}^{-i\alpha} = \cos\alpha - i\sin\alpha & \mathrm{e}^{i\alpha} - \mathrm{e}^{-i\alpha} = 2i\sin\alpha & i^2 = -1 \end{array}$$

ist 5.2 und 5.3 äquivalent zu (5.2 + i5.3)

$$\begin{aligned} z_t - z_0 &= \int_0^t \left(v_x\ \mathrm{e}^{ir(\tau)} + i\, v_y\ \mathrm{e}^{ir(\tau)}\right) \mathrm{d}\tau = v_z \int_0^t \mathrm{e}^{ir(\tau)}\, \mathrm{d}\tau = v_z\ \mathrm{e}^{ir_0} \int_0^t \mathrm{e}^{iv_r\tau}\, \mathrm{d}\tau \\ &= v_z\ \mathrm{e}^{ir_0}\, \frac{1}{i\, v_r}\, \mathrm{e}^{iv_r\tau}\,\Big|_0^t = v_z\ \mathrm{e}^{ir_0}\, \frac{1}{i\, v_r}(\mathrm{e}^{iv_r t} - 1) = v_z\ \mathrm{e}^{ir_0}\, \frac{1}{i\, v_r}\, \mathrm{e}^{i\frac{v_r t}{2}} \Big(\underbrace{\mathrm{e}^{i\frac{v_r t}{2}} - \mathrm{e}^{-i\frac{v_r t}{2}}}_{2i\sin\frac{v_r t}{2}}\Big) \\ &= v_z\, \frac{2\sin\frac{v_r t}{2}}{v_r}\ \mathrm{e}^{i(r_0 + \frac{v_r t}{2})}\,. \end{aligned}$$

Für den Spezialfall $v_r = 0$ gilt entsprechend

$$z_t - z_0 = v_z\ \mathrm{e}^{ir_0}\, t.$$

Auflösen nach v_z ergibt

$$\begin{aligned} v_z &= \frac{v_r}{2\sin\frac{v_r t}{2}}\,(z_t - z_0)\ \mathrm{e}^{-i(r_0 + \frac{v_r t}{2})} \\ &= \frac{v_r}{2\sin\frac{v_r t}{2}} \big((x_t - x_0) + i\,(y_t - y_0)\big)\,\big(\cos(r_0 + \tfrac{v_r t}{2}) - i\,\sin(r_0 + \tfrac{v_r t}{2})\big)\,, \end{aligned}$$

bzw. für $v_r = 0$

$$v_z = \frac{(z_t - z_0)\ \mathrm{e}^{-ir_0}}{t}$$

Ausmultiplikation und Vergleich von Real- und Imaginärteil ergibt schließlich die gesuchten Geschwindigkeiten

$$v_x = \frac{v_r}{2\sin\frac{v_r t}{2}}\big((x_t - x_0)\,\cos(r_0 + \tfrac{v_r t}{2}) + (y_t - y_0)\,\sin(r_0 + \tfrac{v_r t}{2})\big) \tag{5.5}$$

$$v_y = \frac{v_r}{2\sin\frac{v_r t}{2}}\big(-(x_t - x_0)\,\sin(r_0 + \tfrac{v_r t}{2}) + (y_t - y_0)\,\cos(r_0 + \tfrac{v_r t}{2})\big) \tag{5.6}$$

und speziell für $v_r = 0$

$$v_x = \frac{1}{t}\big((x_t - x_0)\,\cos r_0 + (y_t - y_0)\,\sin r_0\big) \tag{5.7}$$

$$v_y = \frac{1}{t}\big(-(x_t - x_0)\,\sin r_0 + (y_t - y_0)\,\cos r_0\big). \tag{5.8}$$

Zur Bestimmung der Geschwindigkeiten sind nun die Start- und Endpositionen gefordert. Diese könnten zwar durch die Selbstlokalisierung des Roboters bestimmt werden, indem die Lokalisierung, wie im Roboterfußballspiel auch, autonom von dem Roboter durch erkannte Referenzobjekte erfolgt. Diese Methode ist jedoch einem gewissen Rauschen unterworfen und nutzt vor allem intensiv die Odometriedaten des Roboters [41]. Da die Laufgeschwindigkeit, also auch die Odometrie des Roboters unbekannt ist, können mit dieser Methode die exakten Start- und Endpunkte nicht präzise ermittelt werden.

Zur exakten Bestimmung der Roboterposition wurden daher zwei Kameras über dem Roboterspielfeld angebracht, die an einen PC angeschlossen sind. Ähnliche Aufbauten sind auch in der RoboCup Small-Size Liga zur Bestimmung der Roboterpositionen zu finden. Auf dem Rechner werden die Bilder verarbeitet und die Roboterposition extrahiert und über das WLAN direkt dem Roboter zur Verfügung gestellt. Der modulare Aufbau des Robotercodes erlaubt es, den *Self-Locator*, das Modul zur Bestimmung der Roboterposition, auf die von der Deckenkamera zur Verfügung gestellten Position umzuschalten. Damit der Roboter von der Kamera präzise erkannt werden kann und außerdem mehrere Roboter auf dem Feld unterscheidbar werden, wird auf dem Rücken des Roboters ein zusätzlicher Marker angebracht. Der Marker ist aus leichter Pappe gefertigt und hindert den Roboter bei seiner Bewegung nicht.

Messungen zur Präzision der Deckenkamera haben ergeben, dass die Positionierung eines stehenden Roboters im schlechtesten Fall, das heißt in den Ecken des Spielfeldes, mit einer Genauigkeit von ± 7 mm erkannt wird. Damit die Position des laufenden Roboters ebenfalls möglichst präzise erkannt und auch sichergestellt wird, dass im Falle einer kurzzeitigen Fehl- bzw. Nichterkennung die Position fortgeführt wird, wurde ein Kalman-Filter [68] zur Modellierung der Roboterposition implementiert.

Aufgrund der (durch die Gebäudedecke) begrenzten möglichen Höhe der Kamera wurden zwei Kameras über dem Spielfeld montiert und die Bilder auf dem PC separat ausgewertet. Im Überlappungsbereich der Kameras werden die Positionen der erkannten Objekte fusioniert. Weiterhin wurden die Kameras mit Weitwinkellisen ausgerüstet, was in einer

(a) Bild der Deckenkamera mit erkanntem Roboter

(b) Entzerrtes Bild der Deckenkamera

Abbildung 5.10: Spielfeld aus Sicht der Deckenkamera

Verzerrung des Kamerabildes resultiert. Abbildung 5.10a zeigt das verzerrte Bild einer Deckenkamera mit einem erkannten Roboter mit Marker.

Es wurden Algorithmen zur Perspektiv- und Linsenkorrektur implementiert, so dass es komfortabel möglich ist, in einem Bild durch Auswahl von vier Referenzpunkten die Kamera „virtuell“ über der Mitte des Spielfeldes zu platzieren. Abbildung 5.10b zeigt das entzerrte Bild einer Kamera mit den Referenzpunkten. Die Erkennung des Roboters erfolgt allerdings aus Effizienzgründen auf dem verzerrten Bild und die hier erkannte Position wird korrigiert.

5.2.3 Ergebnisse des Lernens

Wie in Kapitel 2.1.3 erläutert, können in der Laufsteuerung verschiedene Parametersätze für die unterschiedlichen Laufrichtungen genutzt werden. Hier wird nur auf die Ergebnisse der Optimierung des Vorwärtslaufes eingegangen, da die Ergebnisse der anderen Laufrichtungen analog sind.

Abbildung 5.11 zeigt den Fitnessverlauf der Optimierung für den Geradeauslauf. Zur Optimierung wurde, wie schon bei dem zweibeinigen Roboter, eine $(5+30)$ Evolutionsstrategie eingesetzt mit maximaler Lebensdauer von $\kappa = 3$ Generationen. Die Mutationsstärken wurden mit der automatischen Schrittweitenregelung angepasst. Als Startwerte für die Laufparameter der Optimierung wurde ein von Hand optimierter Lauf ausgewählt, der mit ca. 100 mm/s läuft. Die Optimierung erfolgte in diesem Fall relativ schnell. Nach nur ca. 25 Generationen wurde eine Laufgeschwindigkeit von ca. 460 mm/s erreicht. Diese schnel-

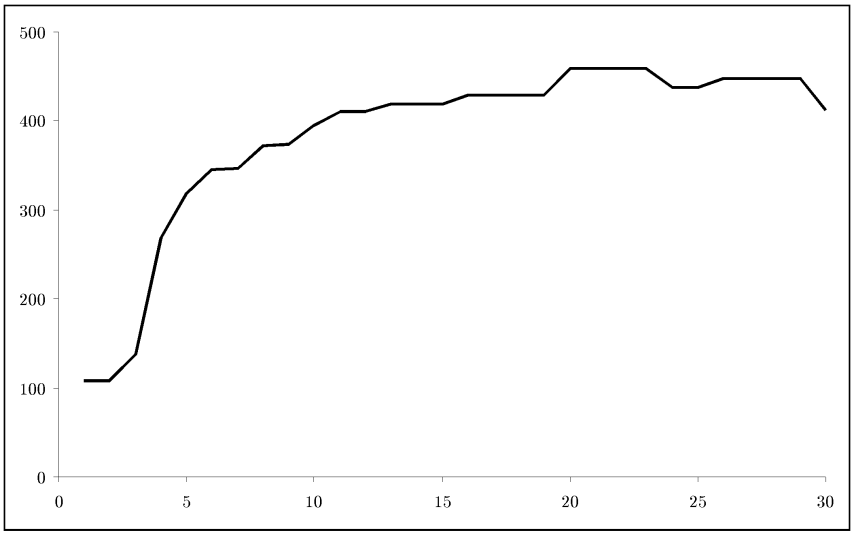


Abbildung 5.11: Fitnessverlauf Laufoptimierung des Aibos ERS 7 für gerades Vorwärtslaufen

le Konvergenz ist auf den Startwert zurückzuführen. Optimierungen mit anderen (weniger optimalen) Startwerten dauerten wesentlich länger. Außerdem ist zu erwähnen, dass es sich bei diesem gefundenen Lauf sicherlich nicht um ein globales Optimum handelt. Optimierungen mit anderen Startwerten führten häufig zu vergleichbaren Geschwindigkeiten. Die resultierenden Läufe unterscheiden sich jedoch sowohl in ihren Parametern, als auch optisch deutlich voneinander. Eine volle Exploration des 31-dimensionalen Suchraumes ist auf dem realen Roboter aufgrund der durchzuführenden realen Läufe wohl nicht möglich, da der Roboter bei dem Austesten der Geschwindigkeit der Laufparameter stark verschleißt. Insbesondere Parametersätze, die die Beine aneinanderschlagen lassen oder den Roboter stolpern lassen, können schnell zu Beschädigungen führen.

Erwähnenswert ist noch, dass die hier gefundenen Läufe alle den Roboter auf den „Ellbögen“ laufen lassen. Dies ist vor allem durch die Startparameter der Optimierung begründet und durchaus erwünscht, da die sich dadurch ergebende niedrigere Höhe des Roboterschwerpunktes zu einer besseren Stabilität führt. Eine Optimierung, bei der als Startwerte ein Lauf mit ausgestreckten Beinen ausgewählt wurde, erreichte eine Geschwindigkeit von 510 mm/s. Diese Geschwindigkeit wurde (nach Wissen des Autors) von keinem anderen RoboCup Team erreicht. Aufgrund der mangelnden Stabilität und Steuerbarkeit dieses Laufes wurde er später nur eingesetzt, wenn der Roboter im Roboterfußballspiel eine weite Distanz geradeaus (z. B. zum Ball) zu laufen hatte [18].

Tabelle 5.1 zeigt zusammenfassend die mit dieser Optimierung erreichten Laufgeschwindigkeiten in die verschiedenen Richtungen.

Vorwärts	Vorwärts (Sprint)	Rückwärts	Seitwärts	Rotation
460 mm/s	510 mm/s	400 mm/s	360 mm/s	200 °/s

Tabelle 5.1: Erreichte Laufgeschwindigkeiten des Aibo ERS 7

5.3 Modellgestütztes Lernen

Zur Reduktion des Zeitaufwandes des Lernens und vor allem zur Verminderung des zwangsläufig auftretenden Verschleißes des Roboters beim Auswerten der Fitness wird im Folgenden untersucht, ob modellgestützte Evolutionsstrategien sich eignen, um die Anzahl der realen Fitnessauswertungen zu reduzieren. Um die Aussagekraft der untersuchten Algorithmen zu erhöhen, wird jede Strategie mehrfach (mindestens 15 mal) mit unterschiedlichen zufälligen Initialisierungen der Startindividuen untersucht und die sich ergebenden Fitnessverläufe gemittelt dargestellt. Da es sich bei den Evolutionsstrategien um probabilistische Verfahren handelt, kann sich ansonsten durch den Zufall ergeben, dass eine in der Regel schlechtere Strategie in einem Durchlauf besser abschneidet als eine andere generell bessere Strategie.

Die Untersuchungen wurden ebenfalls für das Laufmodell des vierbeinigen Sony Aibo ERS 7 durchgeführt und die Ergebnisse hier präsentiert [21]. Aufgrund der hohen Anzahl von erwarteten Fitnessauswertungen wird hier auf den in Kapitel 4 erläuterten physikalischen Simulator zurückgegriffen. Eine Durchführung der Untersuchung auf einem realen Roboter wäre sehr zeitaufwändig und aufgrund des Verschleißes des Roboters ebenfalls sehr kostspielig. Die Auswertung der Fitness wurde parallel auf mehreren PCs eines Rechenclusters durchgeführt. Es wurden jeweils 30 PCs parallel zur Bewertung der Fitness einer Nachkommengeneration genutzt. Jeder dieser Rechner hat die Ergebnisse über das Netzwerk an einen Server übermittelt, auf dem die eigentliche Evolutionsstrategie ablief. Um sicherzustellen, dass die Selbstanpassung der Evolutionsstrategie optimal arbeitet, wurde für die folgenden Untersuchungen eine reine Komma-Strategie eingesetzt. Die Populationsgröße wird wie zuvor auf $(5, 30)$, also 5 Eltern und 30 Nachkommen, festgesetzt. Die Rekombination wurde intermediär durchgeführt. Bei den modellgestützten Evolutionsstrategien wurde durchweg das in Kapitel 3.2.1 beschriebene Modellmanagement *Pre-Selection* eingesetzt. Nach 7000 Fitnessauswertungen wurde die Optimierung jeweils beendet. Entgegen der sonst häufig verwendeten Auftragung der Fitness über die Anzahl der Generationen werden hier die Anzahl der Fitnessauswertungen herangezogen, da dies zum einen die Größe ist, die minimiert werden soll, und zum anderen aussagekräftiger ist, da zumindest bei den geregelten modellgestützten Varianten die Anzahl der Individuen pro Generation variieren kann.

Abbildung 5.12 zeigt den über 15 Verläufe gemittelten Fitnessverlauf der Standard Evolutionsstrategie ohne Modellunterstützung. Nach den 7000 Fitnessauswertungen scheint die Konvergenz noch nicht abgeschlossen zu sein.

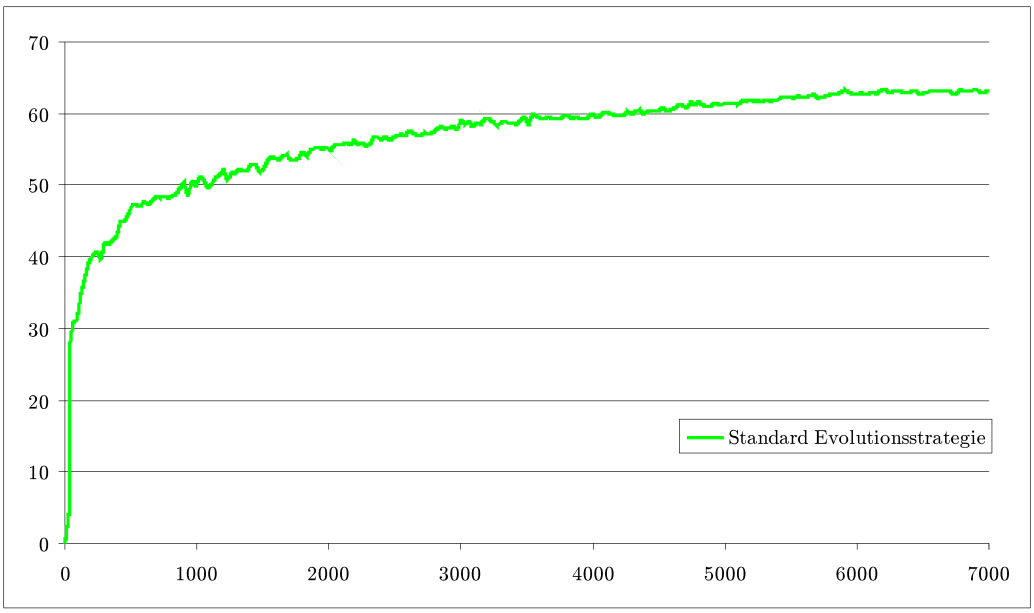


Abbildung 5.12: Gemittelter Fitnessverlauf der Standard Evolutionsstrategie

5.3.1 Optimierung mit der modellgestützten Evolutionsstrategie

Zunächst wird die Auswirkung der Modellunterstützung ohne Regelung des Modelleinflusses für $\lambda_p = 2\lambda = 60$ und unterschiedliche Modellierungen des Suchraumes untersucht. Es werden also zunächst die λ_p vielen Individuen von dem Modell bewertet und anschließend die besten λ vielen Individuen von dem (hier simulierten) Roboter bewertet. Wenn durch die Vorselektion die schlechtesten Individuen schon aussortiert bzw. im Umkehrschluss nur die besten Individuen von dem realen System bewertet werden, ist eine Beschleunigung der Konvergenz - und damit eine kleinere Anzahl von realen Fitnessauswertungen zu erwarten.

Als Modell werden zunächst die in Kapitel 3.2.2 erläuterten Modelle *Nächster Nachbar*, *gewichtetes polynomielles Modell* und der *Gauß Prozess* zur Fitnessprädiktion benutzt. Es ist zu erwarten, dass ein Modell, das den Suchraum besser approximieren kann, zu einer schnelleren Konvergenz der Optimierung führt. Für das lokale polynomielle Modell werden die 150 nächsten bekannten Punkte zu dem zu approximierenden Punkt mit dem Abstand gewichtet zur Modellbildung herangezogen. Der Gauß Prozess wird mittels der 2λ vielen zuletzt ermittelten Fitnesswerten trainiert [65].

Abbildung 5.13 zeigt die gemittelten Fitnessverläufe der modellgestützten Evolutionsstrategien mit drei unterschiedlichen Modellierungen des Suchraumes und der Standard Evolutionsstrategie im Vergleich. Offensichtlich konvergieren alle modellgestützten Strategien signifikant schneller zu dem Optimum von ungefähr 66 als die Standard Evolutionsstrategie. Alle modellgestützten Varianten haben das Optimum nach spätestens 5000 Fitnessauswertungen erreicht. Die Standard Strategie ohne Modellunterstützung hat dieses Optimum

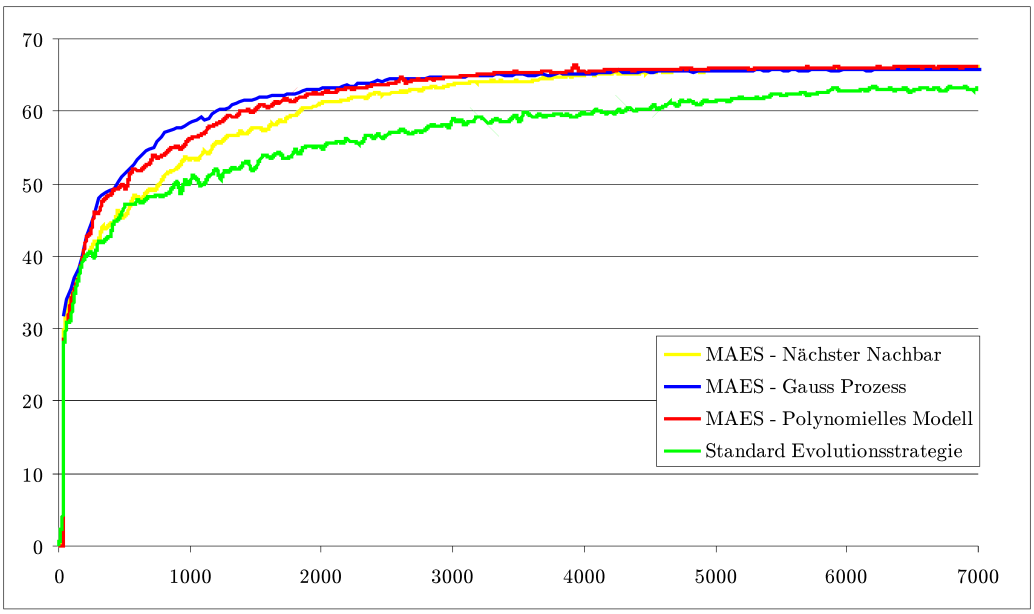


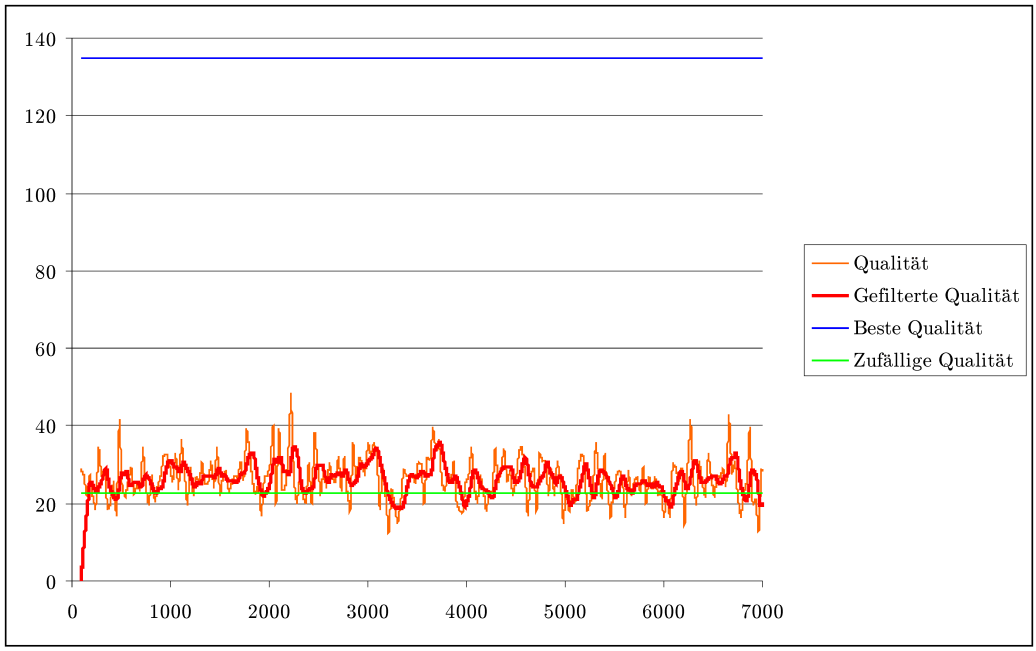
Abbildung 5.13: Gemittelte Fitnessverläufe der modellgestützten Evolutionsstrategien mit unterschiedlichen Modellierungen des Suchraumes im Vergleich mit der Standard Evolutionsstrategie

selbst nach 7000 Fitnessauswertungen noch nicht erreicht. Weiterhin ist in Abbildung 5.13 ersichtlich, dass sich die Fitnessverläufe der drei modellgestützten Varianten im Bezug auf die Konvergenzgeschwindigkeit leicht unterscheiden. Das Nächste Nachbar Modell konvergiert von den modellgestützten Varianten am langsamsten, gefolgt von dem polynomiellen Modell und dem Gauß Prozess. Dies lässt darauf schließen, dass der Gauß Prozess den Suchraum am besten approximieren kann, das polynomielle Modell etwas schlechter und das Nächste Nachbar Modell (vermutlich aufgrund seiner Unfähigkeit zu interpolieren) offensichtlich am schlechtesten.

Diese Vermutungen werden anhand des Verlaufs der Modellqualitäten während der Optimierung untermauert. Abbildung 5.14 zeigt die gemittelten Modellqualitäten während der Optimierung. Die Qualität ist in orange dargestellt, die rote Kurve ist ein gleitender Durchschnitt der Modellqualität über 90 Iterationen und veranschaulicht etwas besser den Verlauf. In grün ist die Qualität eines zufälligen Modells und in blau die Qualität eines perfekten Modells dargestellt. Auffällig ist, dass keines der Modelle annähernd an die maximal mögliche Qualität von 135 heranreicht. Weiterhin sind alle drei Modelle im gesamten Verlauf überwiegend besser als ein zufälliges Modell, das für die hier eingestellten Parameter eine Qualität von 22,5 aufweist. Untersuchungen der modellgestützten Evolutionsstrategie auf Testfunktionen, beispielsweise in [25, 66], haben ergeben, dass die Qualität in der Regel im Laufe der Optimierung mit zunehmender zur Verfügung stehender Stützpunkte zunimmt und teilweise sogar gegen die maximal mögliche Qualität strebt. Hier ist bei keinem der Modelle ein solches Verhalten zu erkennen. Das Nächste Nachbar Modell

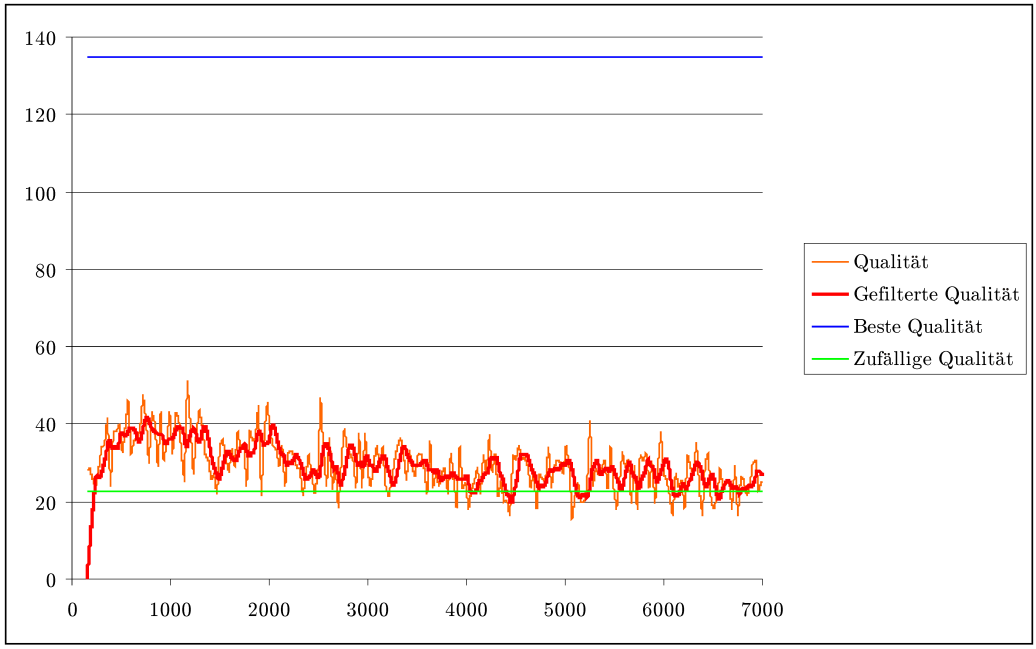
schwankt während der gesamten Optimierung relativ gleichmäßig um den Mittelwert von 26,31. Das polynomielle Modell und der Gauß Prozess hingegen zeigen insbesondere zu Beginn der Optimierung eine wesentlich bessere Modellqualität als zum Ende der Optimierung. Durchschnittlich liegt die Qualität des polynomiellen Modells bei 29,49 und die Qualität des der Prädiktion mit dem Gauß Prozess bei 31,39. Damit korreliert hier eindeutig die (durchschnittliche) Modellqualität mit der zuvor festgestellten unterschiedlichen Konvergenzgeschwindigkeiten der drei Modelle. Interessanterweise scheint sogar ein Modell, das numerisch gesehen nur geringfügig besser als ein zufälliges Modell ist, die Optimierung stark zu beschleunigen.

Die Tatsache, dass das polynomielle Modell und der Gauß Prozess am Anfang der Optimierung bessere Modellqualitäten als zum Ende hin haben, lässt sich eventuell damit begründen, dass am Anfang der Optimierung die Mutationsstärken größer als zum Ende hin sind und somit auch die Individuen untereinander eine große Varianz aufweisen und so „einfacher“ zwischen deutlich guten und deutlich schlechten Individuen unterschieden werden kann. Zum Ende des Optimierungsverlaufes werden die Mutationsstärken heruntergeregelt, so dass sich die Individuen nicht mehr stark voneinander unterscheiden und es so eventuell für das Modell schwierig ist, die Fitnesswerte im Feinen richtig zu schätzen. Anders ausgedrückt könnte auch gesagt werden, dass das Modell die grobe Steigung des Suchraumes besser wiedergeben kann als die kleinen (lokalen) Schwankungen.

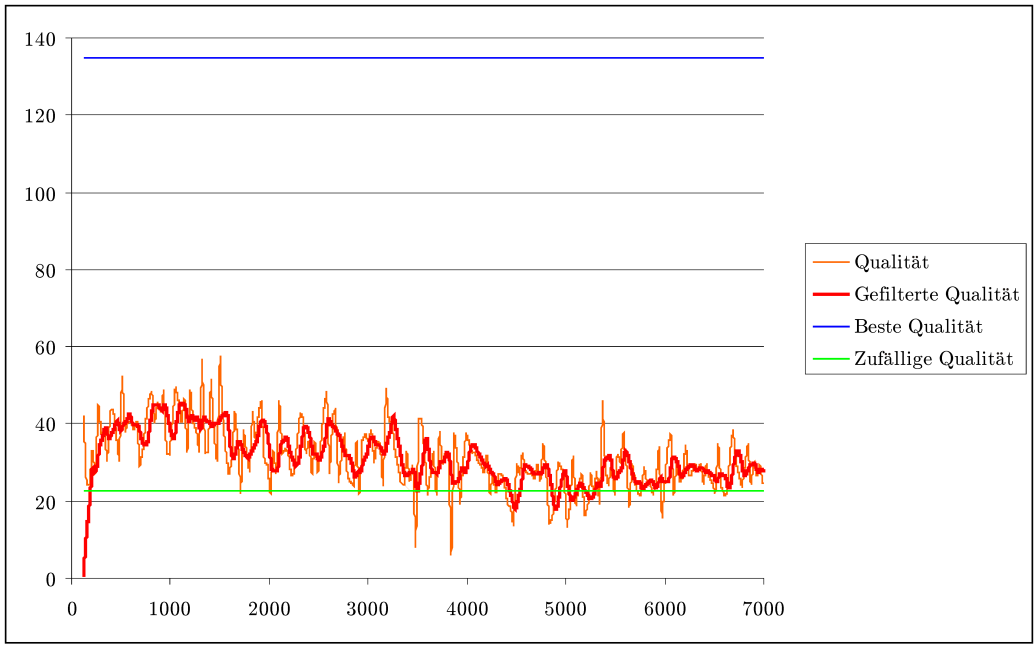


(a) Nächster Nachbar Modell

Abbildung 5.14: Modellqualitäten während der Optimierung für verschiedene Modelle des Suchraums



(b) Lokales polynomielles Modell



(c) Gauß Prozess

Abbildung 5.14: Modellqualitäten während der Optimierung für verschiedene Modelle des Suchraums (fortgesetzt)

5.3.2 Optimierung mit der geregelten modellgestützten Evolutionsstrategie

In dem nächsten Experiment wird untersucht, inwiefern eine Regelung des Modelleinflusses bei diesem Problem einen Einfluss auf die Konvergenzgeschwindigkeit hat. Es werden die CMAES (Controlled Modell Assisted Evolution Strategy) und die λ-CMAES (λ-Controlled Modell Assisted Evolution Strategy) verwendet. Als Modell für den Suchraum wird nur das polynomielle Modell verwendet, da es in der vorherigen Untersuchung dem Gauß Prozess nur leicht unterlegen war, aber eine drastisch kürzere Rechenzeit benötigt. Die Parameter zur Regelung des Modelleinflusses der CMAES aus Gleichung 3.45 und 3.46 werden auf $\delta_{\lambda_p} = 10$ und die Parameter der λ-CMAES aus Gleichung 3.47 und 3.48 auf $\delta_\lambda = \frac{\mu}{2} = 2,5$ gesetzt. Die restlichen Parameter der Evolutionsstrategie bleiben unverändert.

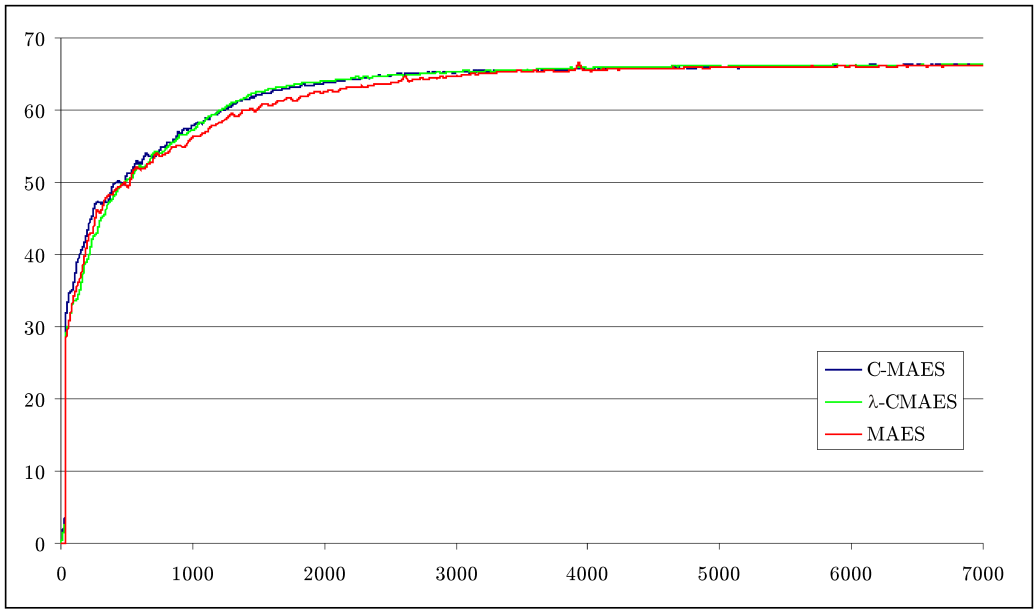


Abbildung 5.15: Gemittelte Fitnessverläufe der ungeregelten modellgestützten Evolutionsstrategie (MAES) und den geregelten Varianten C-MAES und λ-CMAES

Abbildung 5.15 zeigt die gemittelten Fitnessverläufe von 15 Optimierungsdurchläufen mit unterschiedlichen Startpunkten der beiden geregelten Varianten im Vergleich zu der ungeregelten Strategie mit $\lambda_p = 2\lambda = 60$. Sowohl die C-MAES als auch die λ-CMAES konvergieren etwas schneller als die ungeregelte MAES. Das erreichte Optimum stimmt in allen drei Fällen überein. Die etwas schnellere Konvergenz gegenüber der ungeregelten Strategie lässt sich durch die insbesondere am Anfang der Optimierung gute Modellqualität begründen, da die geregelten Strategien dann mehr Vertrauen in das Modell stecken. Eine Überlegenheit einer der beiden geregelten Varianten ist in diesem Fall nicht zu erkennen.

5.3.3 Bewertung der modellgestützten Optimierung

Wie hier gezeigt wurde, eignen sich modellgestützte Evolutionsstrategien sehr gut zur Reduktion der realen Fitnessauswertungen für die Optimierung von Laufmustern für Roboter. Während die Standard Evolutionsstrategie nach 7000 Fitnessauswertungen im Durchschnitt erst eine Fitness von ca. 63 erreicht hat, konnte diese Fitness von den modellgestützten Varianten im Durchschnitt bereits nach 2000 Fitnessauswertungen erreicht werden. Nach ca. 3000 Fitnessauswertungen haben die geregelten modellgestützten Evolutionsstrategien bereits das Optimum von 66 gefunden.

Die Tatsache, dass die Approximationsmodelle des Suchraums im Gegensatz zu den in [25, 66] beschriebenen relativ einfach strukturierten Suchräumen nicht bis an die theoretisch beste Qualität heranreichen, kann hier nicht begründet werden. Ob dies an der mangelnden Flexibilität der Modelle oder an einer zu geringen Anzahl von Stützstellen liegt, kann hier nicht beantwortet werden.

Kapitel 6

Zusammenfassung

In dieser Arbeit wurden Evolutionäre Strategien zum autonomen Lernen von Laufbewegungen für Roboter eingesetzt. Zuerst wurden Modelle für zwei verschiedene Robotertypen aufgestellt, mit Hilfe derer Laufbewegungen erzeugt werden können. Weiterhin erlauben die Modelle eine effiziente Steuerung der Laufrichtung und Laufgeschwindigkeit. Zuerst wurde die resultierende Laufbewegung eines vierbeinigen Roboters vom Typ Sony Aibo ERS 7 untersucht und aufbauend auf den Erkenntnissen ein möglichst flexibles Modell für die Laufmustergenerierung mit 31 Parametern erzeugt. Anschließend wurden für einen zweibeinigen Roboter vom Typ Kondo KHR-1 verschiedene Bewegungsformen der Beine untersucht und schließlich eine halb-elliptische Bahn der Fußbewegung als am aussichtsreichsten erkannt und zusammen mit Armbewegungen und Bewegungen des Oberkörpers des Roboters modelliert. Die Bewegung des zweibeinigen Roboters lässt sich über 15 Parameter beschreiben.

Die Parameter der Laufmodelle haben einen starken Einfluss auf die resultierenden Laufbewegungen beider Roboter. Die Einstellungen der Parameter von Hand durchzuführen, ist aufgrund der hohen Anzahl der Parameter, der unklaren direkten Auswirkung auf den Lauf und der gegenseitigen Abhängigkeit der Parameter sehr schwierig. Während es noch möglich ist, Parameter per Hand einzustellen, mit denen der Roboter langsam läuft, scheitert die Einstellung von Hand, wenn es darum geht, einen möglichst schnellen Lauf zu erstellen.

Zur Optimierung der Parameter der Laufmodelle wurden biologisch inspirierte Optimierverfahren, so genannte Evolutionäre Strategien, als Hilfsmittel eingesetzt, mit deren Hilfe die Roboter selbständig die Parameter ihrer Laufmodelle erlernten. Als zu optimierende Fitness wurde die resultierende Laufgeschwindigkeit gewählt. Zunächst wurden die Auswirkungen unterschiedlicher Einstellungen der Evolutionsstrategie am Beispiel des zweibeinigen Kondo KHR-1 in einer Robotersimulation untersucht. Anschließend wurden auf beiden realen Robotern Optimierungen der Laufgeschwindigkeiten durchgeführt. Sowohl für den Sony Aibo ERS 7 als auch für den Kondo KHR-1 wurden sehr hohe Laufgeschwindigkeiten

erlernt, die im RoboCup von kaum einem anderen Team erreicht wurden. Zum Erzielen von hohen Geschwindigkeiten in alle Laufrichtungen wurde eine Interpolation von Parametersätzen vorgeschlagen, so dass für jede Bewegungsrichtung ein möglichst optimaler Parametersatz zur Verfügung stand.

Ein weiterer Fokus wurde in dieser Arbeit auf die Reduktion der nötigen Trainingsläufe des Roboters gelegt. Schließlich muss der Roboter zur Bestimmung der Fitness eines Parametersatzes den Lauf mit diesen Einstellungen „ausprobieren“, was zum einen sehr zeitaufwändig und zum anderen sehr verschleißfördernd für den Roboter ist. Diese Reduktion der Trainingsläufe wurde erreicht durch eine Schätzung von Fitnesswerten, basierend auf zuvor ermittelten Fitnesswerten. Verfahren wurden vorgestellt, die die genutzte Evolutionsstrategie erweitern auf eine modellgestützte Strategie, die Schätzungen von Fitnesswerten in den Evolutionsdurchlauf integrieren. Für die Schätzung der Fitnesswerte wurden verschiedene Modelle vorgestellt und bezüglich ihrer Fähigkeit, den zugrundeliegenden Suchraum zu approximieren, miteinander verglichen. Darüberhinaus wurden Verfahren vorgestellt und untersucht, die den Einfluss des Modelles in Abhängigkeit der Qualität der Fitnessprädiktion regeln. Es wurde exemplarisch am Beispiel des Sony Aibo ERS 7 gezeigt, dass mithilfe der Modellunterstützung die Anzahl der nötigen Trainingsläufe drastisch reduziert werden kann.

Da insbesondere für die Analyse der modellgestützten Evolutionsstrategie zahlreiche Optimierungen durchgeführt werden mussten, um aussagekräftige Vergleiche zu erzielen, wurde ein Simulator für den Sony Aibo ERS 7 dahingehend optimiert, dass die Simulation der Bewegung möglichst realistisch erfolgt. Diese Optimierung wurde in mehreren Schritten durchgeführt und basierte ebenfalls auf dem Einsatz von Evolutionsstrategien. Es wurden Bewegungen von dem realen Roboter durchgeführt und die Bewegungen geloggt. Anschließend wurden diese Bewegungen in der Simulation durchgeführt und die Differenz zwischen der simulierten und der zuvor geloggten realen Bewegung minimiert. Die Optimierung hat zu einer deutlichen Verbesserung der Realitätstreue des Simulators geführt, so dass der optimierte Simulator im Hinblick auf die Simulation von Bewegungen sogar kommerziell verfügbare Simulationen geschlagen hat.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Albert, Amos: *Intelligente Bahnplanung und Regelung für einen autonomen, zweibeinigen Roboter*, Universität Hannover, Diss., 2001

[2] Bäck, Thomas ; Hammel, Ulrich ; Schwefel, Hans-Paul: Modelloptimierung mit evolutionären Algorithmen. In: Sydow, A. (Hrsg.) ; Lux, T. (Hrsg.) ; Schäfer, R. P. (Hrsg.): *Fortschritte in der Simulationstechnik, Band 6*. Braunschweig : Vieweg, 1993, S. 49–57

[3] Behnke, Sven ; Müller, Jürgen ; Schreiber, Michael: Using Handheld Computers To Control Humanoid Robots. In: *Proceedings of 1st International Conference on Dextrous Autonomous Robots and Humanoids (darh2005), Yverdon-les-Bains - Switzerland*, 2005, S. paper nr. 3.2

[4] Beyer, Hans-Georg: Toward a Theory of Evolution Strategies: Self-Adaptation. In: *Evolutionary Computation* 3 (1996), Nr. 3, S. 311–347

[5] Beyer, Hans-Georg ; Schwefel, Hans-Paul: Evolution strategies – A comprehensive introduction. In: *Natural Computing* 1 (2002), Nr. 1, S. 3–52

[6] Bredenfeld, Ansgar ; Burkhard, Hans-Dieter ; Riedmiller, Martin ; Rojas, Raúl: KI auf dem Fußballfeld. In: *c't Magazin für Computertechnik* 13 (2006), S. 102–109

[7] Carpin, Stefano ; Lewis, Michael ; Wang, Jijun ; Balakirsky, Stephen ; Scrapper, Chris: USARSim: A Robot Simulator for Research and Education. In: *ICRA*, IEEE, 2007, 1400-1405

[8] Cohen, David ; Ooi, Yao H. ; Vernaza, Paul ; Lee, Daniel D.: RoboCup 2004 Legged Soccer Team / University of Pennsylvania. 2004. – Forschungsbericht

[9] Düffert, Uwe ; Hoffmann, Jan: Reliable and Precise Gait Modeling for a Quadruped Robot. In: *RoboCup 2005: Robot Soccer World Cup IX*, Springer, 2006 (Lecture Notes in Artificial Intelligence)

[10] Fidelman, Peggy ; Stone, Peter: The Chin Pinch: A Case Study in Skill Learning on a Legged Robot. In: Lakemeyer, Gerhard (Hrsg.) ; Sklar, Elizabeth (Hrsg.) ; Sorenti, Domenico (Hrsg.) ; Takahashi, Tomoichi (Hrsg.): *RoboCup-2006: Robot*

Soccer World Cup X Bd. 4434, Springer Verlag, 2007 (Lecture Notes in Artificial Intelligence), S. 59–71

[11] Figliolini, Giorgio ; Ceccarelli, Marco: Climbing Stairs with EP-WAR2 Biped Robot. In: *ICRA*, IEEE, 2001. – ISBN 0–7803–6578–X, 4116-4221

[12] Fogel, L. J. ; Owens, A. J. ; Walsh, M. J.: *Artificial Intelligence through Simulated Evolution*. Willey, New York, 1966

[13] Goldberg, David E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA : Addison-Wesley, 1989

[14] Goswami, Ambarish: Postural Stability of Biped Robots and the Foot-Rotation Indicator (FRI) Point. In: *The International Journal of Robotics Research* 18 (1999), Nr. 6, S. 523–533

[15] Grasemann, Uli ; Stronger, Daniel ; Stone, Peter: A Neural Network-Based Approach to Robot Motion Control. In: Visser, Ubbo (Hrsg.) ; Ribeiro, Fernando (Hrsg.) ; Ohashi, Takeshi (Hrsg.) ; Dellaert, Frank (Hrsg.): *RoboCup-2007: Robot Soccer World Cup XI*, Springer, 2008 (Lecture Notes in Artificial Intelligence)

[16] Greggio, N. ; Silvestri, G. ; Antonello, S. ; Menegatti, E. ; Pagello, E.: A 3D Model of a Humanoid Robot for the USARSim Simulator. In: *Proceedings of the Workshop on Humanoid Soccer Robots of the IEEE-RAS International Conference on Humanoid Robots*, 2006. – ISBN 88–900426–2–1, S. 17–24

[17] Gräning, Lars ; Jin, Yaochu ; Sendhoff, Bernhard: Efficient evolutionary optimization using individual-based evolution control and neural networks: A comparative study. In: *ESANN*, 2005, 273-278

[18] Hebbel, Matthias ; Dahm, Ingo ; Fisseler, Denis ; Nisticò, Walter: Learning Fast Walking Patterns with Reliable Odometry Information for Four-Legged Robots. In: *ISMCR'05, 15th International Symposium on Measurement and Control in Robotics*, 2005

[19] Hebbel, Matthias ; Kosse, Ralf ; Nisticò, Walter: Modeling and Learning Walking Gaits of Biped Robots. In: *Proceedings of the Workshop on Humanoid Soccer Robots of the IEEE-RAS International Conference on Humanoid Robots*, 2006. – ISBN 88–900426–2–1, S. 40–48

[20] Hebbel, Matthias ; Laue, Tim: Automatic Parameter Optimization for a Dynamic Robot Simulation. In: *RoboCup 2008: Robot Soccer World Cup XII*, Springer, 2009 (Lecture Notes in Artificial Intelligence). – to appear

[21] *Kapitel* Learning to Walk with Model Assisted Evolution Strategies. In: Hebbel, Matthias ; Nistico, Walter: *Frontiers in Evolutionary Robotics*. Vienna, Austria : I-Tech Education and Publishing, 2008. – ISBN 978–3–902613–19–6, S. 209–220

[22] Hebbel, Matthias ; Nisticò, Walter ; Fisseler, Denis: Learning in a High Dimensional Space: Fast Omnidirectional Quadrupedal Locomotion. In: *RoboCup 2006: Robot Soccer World Cup X*, Springer, 2007 (Lecture Notes in Artificial Intelligence)

[23] Hemker, Th. ; Sakamoto, H. ; Stelzer, M. ; Stryk, O. von: Hardware-in-the-loop optimization of the walking speed of a humanoid robot. In: *CLAWAR 2006: 9th International Conference on Climbing and Walking Robots*. Brussels, Belgium, September 11-14 2006, S. 614–623

[24] Hengst, Bernhard ; Ibbotson, Darren ; Pham, Son B. ; Sammut, Claude: Omnidirectional Locomotion for Quadruped Robots. In: *RoboCup 2001: Robot Soccer World Cup V, A. Birk, S. Coradeschi, S. Tadokoro (Eds.)*, Springer, 2002 (Lecture Notes in Computer Science 2377), S. 368–373

[25] Hoffmann, Frank ; Hölemann, Sebastian: Controlled Model Assisted Evolution Strategy with Adaptive Preselection. In: *2006 IEEE International Symposium on Evolving Fuzzy Systems*, 2006, S. 182 – 187

[26] Hohl, L. ; Tellez, R. ; Michel, O. ; Ijspeert, A.J.: Aibo and Webots: Simulation, Wireless Remote Control and Controller Transfer. In: *Robotics and Autonomous Systems* 54 (2006), Nr. 6, 472–485. http://dx.doi.org/10.1016/j.robot.2006.02.006

[27] Hüsken, Michael ; Jin, Yaochu ; Sendhoff, Bernhard: Structure optimization of neural networks for evolutionary design optimization. In: *Soft Comput.* 9 (2005), Nr. 1, 21-28. http://dblp.uni-trier.de/db/journals/soco/soco9.html#HuskenJS05

[28] Iocchi, Luca ; Libera, Fabio D. ; Menegatti, Emanuele: Learning Humanoid Soccer Actions Interleaving Simulated and Real Data. In: Zhou, C. (Hrsg.) ; Pagello, E. (Hrsg.) ; Menegatti, E. (Hrsg.) ; Behnke, S. (Hrsg.): *Proceedings of the Second Workshop on Humanoid Soccer Robots in conjunction with the 2007 IEEE-RAS International Conference on Humanoid Robots*, 2007

[29] Jakobi, N. ; Husbands, P. ; Harvey, I.: Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics. In: *Lecture Notes in Computer Science* 929 (1995), S. 704–720

[30] Jin, Yaochu ; Hüsken, Michael ; Sendhoff, Bernard: Quality Measures for Approximate Models in Evolutionary Computation. In: Barry, Alwyn M. (Hrsg.): *GECCO 2003: Proceedings of the Bird of a FeatherWorkshops, Genetic and Evolutionary Computation Conference*. Chigaco : AAAI, 11 July 2003, S. 170–173

[31] Jin, Yaochu ; Olhofer, Markus ; Sendhoff, Bernhard: A framework for evolutionary optimization with approximate fitness functions. In: *IEEE Trans. Evolutionary Computation* 6 (2002), Nr. 5, 481–494. http://dblp.uni-trier.de/db/journals/tec/tec6.html#JinOS02

[32] Jin, Yaochu ; Sendhoff, Bernhard: Reducing Fitness Evaluations Using Clustering Techniques and Neural Network Ensembles. In: Deb, Kalyanmoy (Hrsg.) ; Poli,

Riccardo (Hrsg.) ; Banzhaf, Wolfgang (Hrsg.) ; Beyer, Hans-Georg (Hrsg.) ; Burke, Edmund K. (Hrsg.) ; Darwen, Paul J. (Hrsg.) ; Dasgupta, Dipankar (Hrsg.) ; Floreano, Dario (Hrsg.) ; Foster, James A. (Hrsg.) ; Harman, Mark (Hrsg.) ; Holland, Owen (Hrsg.) ; Lanzi, Pier L. (Hrsg.) ; Spector, Lee (Hrsg.) ; Tettamanzi, Andrea (Hrsg.) ; Thierens, Dirk (Hrsg.) ; Tyrrell, Andrew M. (Hrsg.): *GECCO (1)* Bd. 3102, Springer, 2004 (Lecture Notes in Computer Science). – ISBN 3–540–22344–4, 688-699

[33] Kitano, Hiroaki ; Asada, Minoru ; Kuniyoshi, Yasuo ; Noda, Itsuki ; Osawa, Eiichi: RoboCup: The Robot World Cup Initiative. In: *Proceedings of the First International Conference on Autonomous Agents (Agents'97), W. Lewis Johnson and Barbara Hayes-Roth (Eds.)*, ACM Press, 1997, S. 340–347

[34] Kohl, Nate ; Stone, Peter: Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004

[35] Koza, John R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992

[36] Laue, Tim ; Spiess, Kai ; Röfer, Thomas: SimRobot - A General Physical Robot Simulator and Its Application in RoboCup. In: Bredenfeld, A. (Hrsg.) ; Jacoff, A. (Hrsg.) ; Noda, I. (Hrsg.) ; Takahashi, Y. (Hrsg.): *RoboCup 2005: Robot Soccer World Cup IX*, Springer; http://www.springer.de/, 2006 (Lecture Notes in Artificial Intelligence 4020), S. 173–183

[37] Lipson, Hod ; Pollack, Jordan B.: Automatic design and manufacture of robotic lifeforms. In: *Nature* 406 (2000), 974–978. `http://www.nature.com/nature/journal/v406/n6799/abs/406974a0.html`

[38] Meredith, Michael ; Maddock, Steve: Real-Time Inverse Kinematics: The Return of the Jacobian / Department of Computer Science, University of Sheffield. 2004. (Research Memorandum CS-04-06). – Forschungsbericht

[39] Müller, Heiko ; Lauer, Martin ; Hafner, Roland ; Lange, Sascha ; Merke, Artur ; Riedmiller, Martin: Making a Robot Learn to Play Soccer Using Reward and Punishment. In: Hertzberg, Joachim (Hrsg.) ; Beetz, Michael (Hrsg.) ; Englert, Roman (Hrsg.): *KI* Bd. 4667, Springer, 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–74564–8, 220-234

[40] Nisticò, Walter ; Hebbel, Matthias: Real-Time Inter- and Intra- Camera Color Modeling and Calibration for Resource Constrained Robotic Platforms. In: Zaytoon, Janan (Hrsg.) ; Ferrier, Jean-Louis (Hrsg.) ; Andrade-Cetto, Juan (Hrsg.) ; Filipe, Joaquim (Hrsg.): *Proceedings of the 4th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, INSTICC Press, 2007, S. 378–383

[41] Nisticò, Walter ; Hebbel, Matthias: Temporal Smoothing Particle Filter for Vision Based Autonomous Mobile Robot Localization. In: *Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, INSTICC Press, 2008. – To appear

[42] Nolfi, S. ; Floreano, D. ; Miglino, O. ; Mondada, F.: How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics. In: Brooks, R. A. (Hrsg.) ; Maes, P. (Hrsg.): *4th International Workshop on Artificial Life*, MA: MIT Press, 1994. – R. A. Brooks and P. Maes (eds.)

[43] Nolfi, Stefano ; Floreano, Dario: *Evolutionary Robotics*. MIT Press, 2000 `http://mitpress.mit.edu/book-home.tcl?isbn=0262140705`

[44] Quinlan, Michael J. ; Henderson, Naomi ; Middleton, Richard H. ; Nicklin, Steven P. ; Fisher, Robin ; Knorn, Florian ; Chalup, Stephan K. ; King, Robert: The 2006 NUbots Team Report / The University of Newcastle, Australia. 2007. – Forschungsbericht

[45] Rasmussen, Carl E.: Gaussian Processes in Machine Learning. In: Bousquet, Olivier (Hrsg.) ; Luxburg, Ulrike von (Hrsg.) ; Rätsch, Gunnar (Hrsg.): *Advanced Lectures on Machine Learning* Bd. 3176. Heidelberg : Springer-Verlag, 2003 (Lecture Notes in Computer Science), S. 63–71

[46] Ratle, Alain: Accelerating the Convergence of Evolutionary Algorithms by Fitness Landscape Approximation. In: Eiben, A. E. (Hrsg.) ; Bäck, Thomas (Hrsg.) ; Schoenauer, Marc (Hrsg.) ; Schwefel, Hans-Paul (Hrsg.): *PPSN* Bd. 1498, Springer, 1998 (Lecture Notes in Computer Science). – ISBN 3–540–65078–4, 87-96

[47] Rechenberg, Ingo: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, TU Berlin, Diss., 1971

[48] Rechenberg, Ingo: *Evolutionsstrategie '94*. Frommann–Holzboog, Stuttgart, 1994

[49] Röfer, Thomas: Evolutionary Gait-Optimization Using a Fitness Function Based on Proprioception. In: *RoboCup 2004: Robot Soccer World Cup VIII*, Springer, 2005 (Lecture Notes in Artificial Intelligence 3276), S. 310–322

[50] Röfer, Thomas ; Laue, Tim ; Burkhard, Hans-Dieter ; Hoffmann, Jan ; Jüngel, Matthias ; Göhring, Daniel ; Lötzsch, Martin ; Spranger, Michael ; Altmeyer, Benjamin ; Goetzke, Verena ; Stryk, Oskar von ; Brunn, Ronnie ; Dassler, Marc ; Kunz, Michael ; Risler, Max ; Stelzer, Maximilian ; Thomas, Dirk ; Uhrig, Sebastian ; Schwiegelshohn, Uwe ; Dahm, Ingo ; Hebbel, Matthias ; Nisticò, Walter ; Schumann, Carsten ; Wachter, Michael: GermanTeam RoboCup 2004 / Universität Bremen, Humboldt-Universität zu Berlin, Technische Universität Darmstadt und Universität Dortmund. 2004. – Forschungsbericht

[51] Röfer, Thomas ; Laue, Tim ; Weber, Michael ; Burkhard, Hans-Dieter ; Jüngel, Matthias ; Göhring, Daniel ; Hoffmann, Jan ; Altmeyer, Benjamin ; Krause,

Thomas ; Spranger, Michael ; Stryk, Oskar von ; Brunn, Ronnie ; Dassler, Marc ; Kunz, Michael ; Oberlies, Tobias ; Risler, Max ; Schwiegelshohn, Uwe ; Hebbel, Matthias ; Nisticò, Walter ; Czarnetzki, Stefan ; Kerkhof, Thorsten ; Meyer, Matthias ; Rohde, Carsten ; Schmitz, Bastian ; Wachter, Michael ; Wegner, Tobias ; Zarges, Christine: GermanTeam RoboCup 2005 / Universität Bremen, Humboldt-Universität zu Berlin, Technische Universität Darmstadt und Universität Dortmund. 2005. – Forschungsbericht

[52] Schwefel, Hans-Paul: *Evolutionsstrategie und numerische Optimierung*, Technische Universität Berlin,Fachbereich Verfahrenstechnik, Dr.-Ing. Dissertation, 1975

[53] Schwefel, Hans-Paul: *Interdisciplinary Systems Research*. Bd. 26: *Numerische Optimierung von Computer–Modellenmittels der Evolutionsstrategie*. Basel : Birkhäuser, 1977. – ISBN 3–7643–0876–1

[54] Schwefel, Hans-Paul: Direct search for optimal parameters withinsimulation models. In: Conine, R. D. (Hrsg.) ; Katz, E. D. (Hrsg.) ; Melde, J. E. (Hrsg.): *Proc. Twelfth Annual Simulation Symp., Tampa FL*. Long Beach CA : IEEE Computer Society, 1979, S. 91–102

[55] Schwefel, Hans-Paul: *Evolution and Optimum Seeking*. New York : Wiley Interscience, 1995 (Sixth-Generation Computer Technology). – ISBN 0–471–57148–2

[56] Sellers, David: An Overview of Proportional plus Integral plus Derivative Control and Suggestions for Its Successful Application and Implementation / TxSpace at Texas A&M University Libraries (United States). 2001. – Forschungsbericht

[57] Siegwart, Roland ; Nourbakhsh, Illah R.: *Introduction to Autonomous Mobile Robots*. Bradford Book, 2004. – ISBN 026219502X

[58] Siems, U. ; Herwig, C. ; Röfer, T. ; Krieg-Brückner, B. (Hrsg.) ; Roth, G. (Hrsg.) ; Schwegler, H. (Hrsg.): *SimRobot, ein System zur Simulation sensorbestückter Agenten in einer dreidimensionalen Umwelt*. Zentrum für Kognitionswissenschaften. Universität Bremen, 1994 (ZKW Bericht 1/94)

[59] Smith, Russell: *Open Dynamics Engine - ODE*. 2007. – `www.ode.org`

[60] Sridharan, Mohan ; Stone, Peter: Color Learning on a Mobile Robot: Towards Full Autonomy under Changing Illumination. In: *The 20th International Joint Conference on Artificial Intelligence*, 2007, S. 2212–2217

[61] Stein, Michael L.: *Statistical Interpolation of Spatial Data: Some Theory for Kriging*. Springer, 1999 (Springer Series in Statistics)

[62] Stone, Peter ; Dresner, Kurt ; Fidelman, Peggy ; Jong, Nicholas K. ; Kohl, Nate ; Kuhlmann, Gregory ; Sridharan, Mohan ; Stronger, Daniel: The UT Austin Villa 2004 RoboCup Four-Legged Team: Coming of Age / The University of Texas at

Austin, Department of Computer Sciences, AI Laboratory. 2004 (UT-AI-TR-04-313). – Forschungsbericht

[63] Stone, Peter ; Sutton, Richard S. ; Kuhlmann, Gregory: Reinforcement Learning for RoboCup-Soccer Keepaway. In: *Adaptive Behavior* 13 (2005), Nr. 3, S. 165–188

[64] Stone, Peter ; Veloso, Manuela M.: Layered Learning. In: *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings* Bd. 1810, Springer, Berlin, 2000, 369–381

[65] Ulmer, H. ; Streichert, F. ; Zell, A.: *Evolution strategies assisted by gaussian processes with improved pre-selection criterion.* `citeseer.ist.psu.edu/ulmer03evolution.html`. Version: 2003

[66] Ulmer, Holger ; Streichert, Felix ; Zell, Andreas: Evolution Strategies with Controlled Model Assistance. In: *Proceedings of the 2004 IEEE Congress on Evolutionary Computation.* Portland, Oregon : IEEE Press, 20-23 June 2004. – ISBN 0–7803–8515–2, S. 1569–1576

[67] Vukobratović, Miomir ; Borovac, Branislav: Zero-moment point–Thirty five years of its life. In: *International Journal of Humanoid Robotics* 1 (2004), Nr. 1, S. 157–173

[68] Welch, Greg ; Bishop, Gary: An Introduction to the Kalman Filter / University of North Carolina at Chapel Hill. Version: 1995. `http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html`. Chapel Hill, NC, USA : University of North Carolina at Chapel Hill, 1995 (95-041). – Forschungsbericht

[69] Williams, Christopher K. I. ; Rasmussen, Carl E.: Gaussian Processes for Regression. In: Touretzky, David S. (Hrsg.) ; Mozer, Michael C. (Hrsg.) ; Hasselmo, Michael E. (Hrsg.): *Proc. Conf. Advances in Neural Information Processing Systems, NIPS* Bd. 8, MIT Press, 1995. – ISBN 0262201070

[70] Zagal, Juan C. ; Solar, Javier R.: UCHILSIM: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League. In: *RoboCup 2004: Robot Soccer World Cup VIII*, Springer, 2004 (Lecture Notes in Artificial Intelligence)

[71] Zagal, Juan C. ; Solar, Javier R. ; Vallejos, Paul: Back to Reality: Crossing the Reality Gap in Evolutionary Robotics. In: *Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles - IAV2004*, 2004

[72] Zagal, Juan C. ; Sarmiento, Iván ; Solar, Javier R.: An Application Interface for UCHILSIM and the Arrival of New Challenges. In: Bredenfeld, Ansgar (Hrsg.) ; Jacoff, Adam (Hrsg.) ; Noda, Itsuki (Hrsg.) ; Takahashi, Yasutake (Hrsg.): *RoboCup* Bd. 4020, Springer, 2005 (Lecture Notes in Computer Science). – ISBN 3–540–35437–9, 464-471

[73] Zagal, Juan C. ; Ruiz-Del-Solar, Javier: Combining Simulation and Reality in Evolutionary Robotics. In: *J. Intell. Robotics Syst.* 50 (2007), Nr. 1, S. 19–39. http://dx.doi.org/10.1007/s10846-007-9149-6. – DOI 10.1007/s10846–007–9149–6