UNIKASSEL
VERSITÄT

IMAT-MMS

Diplomarbeit

(2. Studienstufe)

Unterstützung suboptimaler Regelkreise durch eine zentrale Korrektursteuerung

Steffen Bretz
Matr.-Nr.: 47 31 32

8. März 2006

Universität Kassel
Fachbereich Maschinenbau
Institut für Mess- und Automatisierungstechnik
Fachgebiet Mensch-Maschine-Systeme
Prof. Dr.-Ing. Dr. h. c. Gunnar Johannsen
Dr.-Ing. Bernd-Burkhard Borys

Vorwort

Evolutionäre Algorithmen werden gerne für Optimierungsaufgaben mit sehr vielen Freiheitsgraden eingesetzt. Eine spezielle Konvergenzeigenschaft, daß nämlich der Rechenaufwand nach [Poh99a] nur mit der Wurzel der Anzahl der Unbekannten steigt, prädestiniert sie dafür. Die evolutionären Algorithmen haben aber auch noch eine weitere interessante Eigenschaft: Von der Zielfunktion wird nur verlangt, daß sie monoton ist — nichts weiter. Speziell wird, im Gegensatz zu gradientenbasierten Verfahren, keinerlei Ableitung von der Zielfunktion benötigt. Dadurch können evolutionäre Algorithmen auch in solchen Fällen eingesetzt werden, in denen Ableitungen der Zielfunktion nicht oder nur schwierig zu beschaffen sind.

Die evolutionären Algorithmen kommen deshalb mit so geringen Anforderungen an die Zielfunktion aus, weil nur absolute Bewertungen einzelner Punkte (hier Vektoren) im Lösungsraum durch die Zielfunktion vorgenommen werden. Dafür werden eine gewisse Anzahl Punkte gleichzeitig betrachtet. Im direkten Vergleich untereinander relativ günstig liegende Punkte werden für die weitere Rechnung übernommen, die anderen verworfen. Aus den Komponenten der übernommenen Punkte werden nun zufällig neue Punkte zusammengesetzt und ein wenig verschoben. Dann schließt sich der Kreis, indem diese neuen Punkte ebenfalls bewertet werden. Im Verlauf einer solchen Iteration konvergiert die Punktmenge in der Regel gegen ein Optimum. Oft kommt es gerade zu Beginn der Iteration zu schnellen Fortschritten.

In dieser Arbeit wird ein Verfahren vorgestellt, bei dem mit Hilfe von evolutionären Algorithmen verbessernde Eingriffe in laufenden Echtzeitsystemen vorgenommen werden. Was gut oder schlecht ist, wird zu diesem Zweck über die Zielfunktion für die Optimierung definiert. Da von der Zielfunktion letztlich das Verhalten des Gesamtsystems abhängt, sollte sie sorgfältig ausgewählt werden. Die Eingriffe in das System sind zeitlich begrenzte Steuertrajektorien. Sie werden zusätzlich zur permanent wirkenden Regelung auf das System aufgebracht. Um die Anzahl der zu optimierenden Variablen in Grenzen zu halten, werden die Steuertrajektorien durch wenige Parameter repräsentiert.

Da die Steuertrajektorien im voraus berechnet werden müssen, wird das Systemverhalten mittels eines Modells für eine gewisse, in der Zukunft liegende, Zeitspanne vorhergesagt. Wird die geforderte Qualität während dieser Zeitspanne unterschritten, kann so schon im Vorfeld ein Optimierungslauf des evolutionären Algorithmus durchgeführt werden. Allerdings ist die zur Verfügung stehende Rechenzeit von vornherein begrenzt. Daher ist es wesentlich, daß die mit evolutionären Algorithmen häufig assoziierte lange Rechenzeit nicht benötigt wird. Tatsächlich läßt sich unter Umständen mit wenig Rechenzeit auskommen. Erstens wird nur mit wenigen Variablen gerechnet, zweitens kommt es bei dem beschriebenen Verfahren — halbwegs gutmütige Systeme vorausgesetzt — gar nicht auf die letzte Nachkommastelle, sondern (ähnlich wie bei *Sliding-Mode*-Regelungen) mehr auf eine Tendenz an. Da evolutionäre Algorithmen aber gerade zu Beginn einer Iteration die größten Fortschritte in Richtung des Optimums machen, kann schon nach vergleichsweise wenigen Schritten eine deutliche Verbesserung der Gesamtsituation erreicht werden.

Gerade um eine schnelle Konvergenz zu erreichen, sind die spezielle Ausprägung und die Parameter des evolutionären Algorithmus mit Bedacht zu wählen. Dafür werden im Rahmen der Arbeit einige Experimente durchgeführt. Anhand der Ergebnisse der Experimente können konkrete Empfehlungen für eine günstige Konfiguration des evolutionären Algorithmus gegeben werden. Um es vorwegzunehmen: Zuviel Aufwand beim evolutionären Algorithmus zu treiben, lohnt sich nicht. Schon mit einfachen Konfigurationen können gute Ergebnisse erzielt werden. Die einzige Maßnahme, die sich bei den Experimenten tatsächlich als vorteilhaft herausstellte, war die Aufteilung der Gesamtpopulation (betrachtete Punktmenge im Lösungsraum) in mehrere Subpopulationen.

Schließlich wird noch ein Computerprogramm beschrieben, das die Arbeitsweise des vorgestellten Verfahrens am Bildschirm erlebbar macht. Die einzelnen Komponenten werden vom Programm während der Ausführung mit einigen wesentlichen Rechengrößen visualisiert. Der Betrachter erhält so einen besseren Eindruck vom Zusammenwirken der einzelnen Verfahrens-Teile.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Eine typische Situation in der Industrie ist, daß ein räumlich verteilter Produktionsprozess zentral überwacht und gesteuert wird. Zu diesem Zweck gibt es eigens eingerichtete Leitwarten mit einer Vielzahl von Anzeigen und Bedienelementen. Die aktuelle Situation wird mit den Anzeigen visualisiert. Fehler im System lösen Alarmmeldungen aus, auf die der Bediener selbst durch Eingriffe reagieren oder ein Wartungsteam benachrichtigen muß. Die Vielzahl der auftretenden Alarmmeldungen ist mittlerweile zu einem solchen Problem geworden, daß bereits Maßnahmen zum Alarmmanagement ergriffen worden sind. Diese zielen jedoch meist darauf ab, die Alarmmeldungen zu klassifizieren und nur die wichtigsten tatsächlich zu melden. Der hier vorgestellte Ansatz zielt dagegen darauf ab, einen gewissen Anteil der Fehlersituationen automatisch zu bearbeiten und zu beheben. Damit kann die Anzahl der Meldungen, um die sich der Bediener selbst kümmern muß, unter Umständen beträchtlich reduziert werden.

> „Eine durchgeführte Analyse von Honeywell über 300.000 Regelkreise weltweit ergab, dass 36 % der Regelkreise nicht arbeiten (open loop), für 10 % ist das Ergebnis „poor“, 22 % „fair“, 16 % „acceptable“ und nur 16 % sind „excellent“,“ ... [Pic05]

Dieses erschreckende Ergebnis läßt darauf schließen, daß wahrscheinlich viele der ausgelösten Alarme auf schlecht ausgelegte Regelkreise zurückzuführen sind.

In dieser Arbeit wird ein Verfahren vorgestellt, daß Eingriffe in das überwachte System bei Bedarf selbst ermittelt, ohne den Bediener zu behelligen. Nur wenn es dem Verfahren nicht gelingt, eine Verbesserung des Systemzustands herbeizuführen, braucht eine Alarmmeldung abgegeben zu werden.

Es ist dabei nicht das Ziel, alle auftretenden Fehler automatisch zu bearbeiten. Wegen der Vielzahl der möglichen Erscheinungen wird das auch gar nicht möglich sein. Speziell echte Defekte müssen in jedem Fall, zum Beispiel durch Reparatur oder Austausch von Komponenten, manuell behoben werden. Es geht daher vielmehr darum, die Bearbeitung einer bestimmten Klasse von Fehlersituationen zu automatisieren.

Der Bediener soll auch nicht allzusehr durch die Automatisierung verwöhnt werden, denn schließlich soll er ja auch im Training bleiben; etwa für den Fall, daß die automatische Bearbeitung nicht funktioniert. Sollte das in dieser Arbeit beschriebene Verfahren zu gut funktionieren, kann deshalb darüber nachgedacht werden, einen Teil der automatisch ermittelten Eingriffe nicht automatisch durchzuführen, sondern die Problembehebung dem Bediener zu überlassen. Die ermittelte Lösung könnte dabei trotzdem als Anregung mitgeteilt werden. Auch könnten erfolgreiche Eingriffe in einer Datenbank gespeichert werden und später sozusagen als Musterlösung dienen. Sie könnten außerdem als Startlösungen in die (ansonsten zufällige) Anfangspopulation des zur Optimierung eingesetzten evolutionären Algorithmus eingestreut werden.

2 Verfahren

Die Voraussetzung für eine Unterstützung durch das hier beschriebene System ist, daß von den zu überwachenden Prozessen Computer-Modelle existieren. Anhand dieser Modelle wird der Verlauf der Prozeßgrößen für die nähere Zukunft vorhergesagt. Die berechneten Werte dienen als Grundlage für weitere Entscheidungen und gegebenenfalls für Frühwarnungen.

Für jeden Prozeß wird neben dem realen Ablauf eine Berechung der Prozeßgrößen mittels eines Modells durchgeführt. Dabei wird immer schon etwas in die Zukunft gerechnet und somit eine Prognose über den zukünftigen Ablauf aufgestellt. Sollte der prognostizierte Verlauf einen Alarm auslösen, so wird versucht, mit einem automatisch generierten Steuereingriff den Prozeß so zu beeinflussen, daß der Alarm unterbleiben kann. Die Wirkung des Steuereingriffs selbst kann ebenfalls an dem Modell überprüft werden. Nur für den Fall, daß die automatische Steuerung nicht wie gewünscht funktioniert, wird der Anlagenführer mit einem Alarm behelligt.

Wenn der Steuereingriff anhand eines Modells des realen Prozesses optimiert wird, wird er wahrscheinlich sogar besser ausfallen als ein vom Anlagenführer aufgrund eines unscharfen mentalen Modells durchgeführter Eingriff.

Da bei einer geglückten automatischen Steuerung der Alarmzustand erst gar nicht erreicht wird, kann mit diesem Verfahren unter Umständen eine Verbesserung der Produktqualität erreicht werden.

Bei einem Prozeß mit wirkungsvollem eigenen Regler sollte eigentlich nie ein zusätzlicher Eingriff notwendig sein. Häufige Steureingriffe bei einem Prozeß sprechen daher für eine schlecht ausgelegte eigene Regelung. Es erscheint daher sinnvoll, die Eingriffe statistisch zu erfassen.

2.1 Prognose

Um genügend Zeit zur Berechnung eines Eingriffs durch die Zusatzsteuerung zu haben, muß der Zustand des überwachten Systems für einen bestimmten

Zeitraum im voraus bekannt sein. Das bedeutet, daß das überwachte System mit Hilfe eines Modells simuliert werden muß. Als Anfangsbedingungen für die Simulation werden reale Meßwerte verwendet. Im Laufe der Zeit werden sich die mit der Simulation berechneten Werte wegen der stets vorhandenen Ungenauigkeiten des Modells immer weiter von den tatsächlich eintretenden Werten entfernen. Daher muß die Simulation von Zeit zu Zeit (an den Synchronisationspunkten) mit dem realen System in Übereinstimmung gebracht werden.

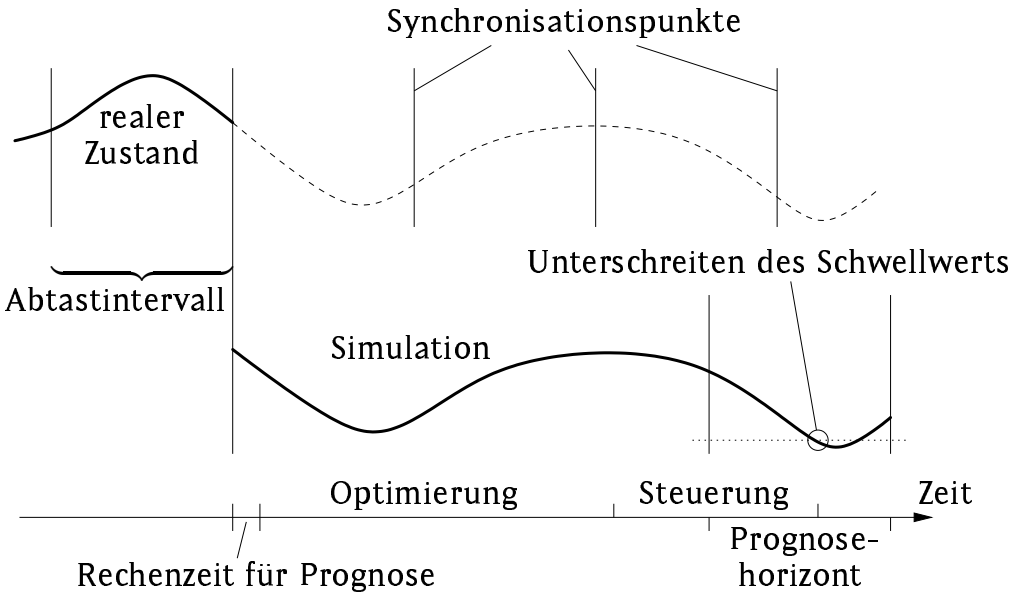

Abbildung 1: Prognose

Der Systemzustand wird während der Prognose laufend mit einer prozeßspezifischen Qualitätsfunktion bewertet. Sinkt die Qualität innerhalb des Prognosehorizonts unter einen Schwellwert, so wird eine Steuertrajektorie berechnet, die — zusätzlich zu der normalen Regelung — den Zustand in den erlaubten Bereich ziehen soll. Die Dauer der Steuerung kann im Voraus festgelegt sein, so daß der Startzeitpunkt der Zusatzsteuerung bekannt ist. Bis zu diesem Zeitpunkt kann jetzt die Steuertrajektorie optimiert werden.

Während der Rechenzeit des Optimierers kann die Prognose mit realen Meßwerten des fortgeschrittenen Zustands verbessert werden. Die Zielfunktion des Optimieres ist dann nicht mehr stationär. Solange sich die Prognose aber nur graduell ändert, sollte die Optimierung trotzdem konvergieren.

Um eine lückenlose Prognose des zukünfigen Verlaufs zu erreichen, muß der Prognosezeitraum genauso lang sein wie das Abtastintervall.

Prognosegüte

Um ein Maß für die Vertrauenswürdigkeit der Prognose zu bekommen, können die an den Synchronisationspunkten prognostizierten Werte mit den tatsächlich gemessenen Zuständen verglichen werden. Sind die Abweichungen groß, so kann kaum angenommen werden, daß die Übereinstimmung im weiteren Simulationsverlauf besser wird. Es liegt dann wahrscheinlich eine Störung vor, die dazu führt, daß das Modell nicht dem realen Prozeß entspricht. Es hat dann wenig Sinn, aufgrund der wahrscheinlich fehlerhaften Prognose einen Zusatzeingriff zu optimieren. Statt dessen sollte eine Meldung über die Prognoseprobleme abgegeben werden. Je nachdem, wie stark die Störung ist, wirkt diese sich unter Umständen nur gering auf kurzfristige Vorhersagen aus. In diesem Fall kann das System zumindest noch vorzeitig auf ein Verlassen des Toleranzbereichs durch den Prozeßzustand hinweisen.

Die Prognosegüte kann somit in drei Klassen eingeteilt werden:

- Prognose gut
- Prognose nur kurzfristig brauchbar
- Prognose unbrauchbar

Für den Bediener ist die Güteeinstufung der Prognose auf den ersten Blick unwichtig. Allerdings deutet eine schlechte Prognosequalität auf Probleme im Prozeß hin. Ein solcher Prozeß verdient eine erhöhte Aufmerksamkeit. Darüberhinaus kann bei völlig unbrauchbaren Prognosen keine Vorwarnung für ein Verlassen des Toleranzbereichs gegeben werden. Wenn dieser dann tatsächlich verlassen wird, muß unverzüglich gehandelt werden. Eine Meldung über schlechte Prognosewerte sollte daher den Bediener veranlassen, sich auf einen eventuellen Einsatz vorzubereiten.

Dauerhaft schlechte Prognosewerte deuten auf ein ungeeignetes Modell hin. In diesem Fall muß das Modell besser an den Prozeß angepaßt werden.

Werden dagegen anfangs gute Prognosewerte im Laufe des Betriebs immer schlechter, so kann das im Verschleiß der Anlage begründet sein. Nach [Jel05] beträgt die Halbwertszeit der Güte von Reglereinstellungen, bedingt durch sich ändernde Anlagenparameter, etwa sechs Monate. Die Prognosegüte kann daher auch als Indikator für den allgemeinen Anlagenzustand verwendet werden.

2.2 Zielfunktion

Mit der Zielfunktion wird eine Bewertung des Zustands nach erfolgtem Zusatzeingriff vorgenommen. In dem beschriebenen Verfahren wird dazu direkt der Wert der Qualitätsfunktion herangezogen. Im allgemeinen Fall können für evolutionäre Optimierungen auch andere Methoden angewendet werden. Als Beispiel soll hier das Pareto-Ranking[1] beschrieben werden. Hierbei handelt es sich um einen multikriteriellen (engl. *multiobjective*) Vergleich, bei dem eine Rangfolge in einer Menge von potentiellen Lösungen ermittelt wird.

Beim Pareto-Ranking dominiert von zwei potentiellen Lösungen diejenige, die in mindestens einem Kriterium besser und in keinem schlechter ist als die andere. Mit dieser Art des Vergleichs läßt sich eine Rangfolge aber nur bedingt und mit einigem Aufwand ermitteln.

Beispiel

Das Beispiel zeigt ein Pareto-Ranking für acht Vektoren. Ein Vektor dominiert einen anderen dann, wenn mindestens eine Komponente größer, aber keine kleiner ist als die entsprechende im anderen Vektor.

$$\underline{\mathbf{a}}\,\mathbf{dominiert}\,\underline{\mathbf{b}} \stackrel{\text{def}}{=} \forall i \mid a_i \geq b_i \wedge \exists i \mid a_i > b_i$$

Jeder Vektor, der von keinem anderen dominiert wird, hat den Rang eins. Für jeden Vektor, der einen Vektor $\underline{\mathbf{x}}$ dominiert, erhöht sich der Rang von $\underline{\mathbf{x}}$ um eins.

$$\mathbf{Rang}\,\underline{\mathbf{x}}_i \stackrel{\text{def}}{=} 1 + \sum_j \begin{cases} 1\,, & \text{wenn } \underline{\mathbf{x}}_j\,\mathbf{dominiert}\,\underline{\mathbf{x}}_i \\ 0 & \text{sonst} \end{cases}$$

[1] siehe auch [Poh99a]

Mit diesen Definitionen ergibt sich für acht zufällig gewählte Vektoren das folgende Bild:

$$\begin{array}{lcccccccc}
 & \begin{pmatrix}10\\4\\1\\6\\8\end{pmatrix} & \begin{pmatrix}7\\8\\4\\10\\0\end{pmatrix} & \begin{pmatrix}8\\1\\6\\2\\0\end{pmatrix} & \begin{pmatrix}0\\6\\3\\9\\4\end{pmatrix} & \begin{pmatrix}8\\7\\10\\10\\10\end{pmatrix} & \begin{pmatrix}9\\6\\2\\4\\5\end{pmatrix} & \begin{pmatrix}1\\0\\6\\3\\3\end{pmatrix} & \begin{pmatrix}8\\4\\10\\9\\7\end{pmatrix} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{Rang:} & 1 & 1 & 3 & 2 & 1 & 1 & 3 & 2
\end{array}$$

Je kleiner die Rangzahl ist, desto dominanter (und im Sinne einer Lösung besser) ist der zugehörige Vektor. An dem Ergebnis fällt auf, daß die acht Vektoren auf nur drei Rangstufen abgebildet werden. □

Charakteristisch für ein Pareto-Ranking ist, daß es immer nur auf eine geschlossene Menge von Vektoren anwendbar ist. Es gibt kein Maß, mit dem einzelne Vektoren bewertet werden könnten. Bei einer Anwendung in einem evolutionären Algorithmus müßte somit bei jeder Selektion die gesamte Population neu bewertet werden. Zwar müßten die einzelnen Vektoren nicht neu berechnet werden, aber die Rangfolge würde sich ja mit jedem hinzukommenden Nachkommen ändern.

Für das beschriebene Verfahren ist es dagegen notwendig, einen absoluten Maßstab zu besitzen. Ansonsten kann zum Beispiel keine Schwelle für die Auslösung des Optimiervorganges angegeben werden.

Neben dem Pareto-Ranking können auch Methoden der Fuzzy-Logik zur Bewertung herangezogen werden. In [Ber05] wird eine Methode beschrieben, bei der einfach auf die Defuzzifizierung verzichtet und die durch Inferenz gefundene Fuzzy-Variable als Bewertungszahl genommen wird. Da mit dieser Methode auch einzelnen Vektoren eine konkrete Zahl zugeordnet werden kann, ist sie ohne weiteres mit dem hier beschrieben Verfahren verwendbar.

2.3 Parametrisierung der Steuertrajektorie

Der zeitliche Verlauf der Steuertrajektorie muß zur Bearbeitung mit dem evolutionären Algorithmus geeignet parametrisiert werden. Auf der einen Seite sollen so wenige Parameter wie möglich gewählt werden, damit der Algorithmus möglichst effizient arbeitet. Auf der anderen Seite muß die Lösung der Aufgabe mit den zur Verfügung stehenden Parametern auch darstellbar sein. Eine Möglichkeit besteht darin, die Eingriffe durch Polynome darzustellen, so daß die Trajektorie mit wenigen Koeffizienten beschrieben werden kann. Dabei ist zu beachten, daß die geforderten Randbedingungen eingehalten werden. Am besten ist es, wenn schon die gewählte Parametrisierung die Einhaltung der Randbedingungen garantiert.

Als Rand- und Nebenbedingung wird hier gefordert, daß die Steuertrajektorie keine Sprünge aufweist; auch nicht am Anfang oder am Ende. Lagrange-Polynome besitzen die gewünschten Eigenschaften. Mit ihnen lassen sich die Trajektorien gut mit wenigen Parametern darstellen und Sprünge vermeiden. Abb. 2 zeigt zwei Lagrange-Polynome dritter Ordnung, mit denen sich eine einfache Steuertrajektorie durch Superposition zusammensetzen läßt.

$$
\begin{aligned}
p_1(x) &= \frac{27}{2}\, x\,(x - 2/3)\,(x - 1) \\
p_2(x) &= -\frac{27}{2}\, x\,(x - 1/3)\,(x - 1)
\end{aligned}
$$

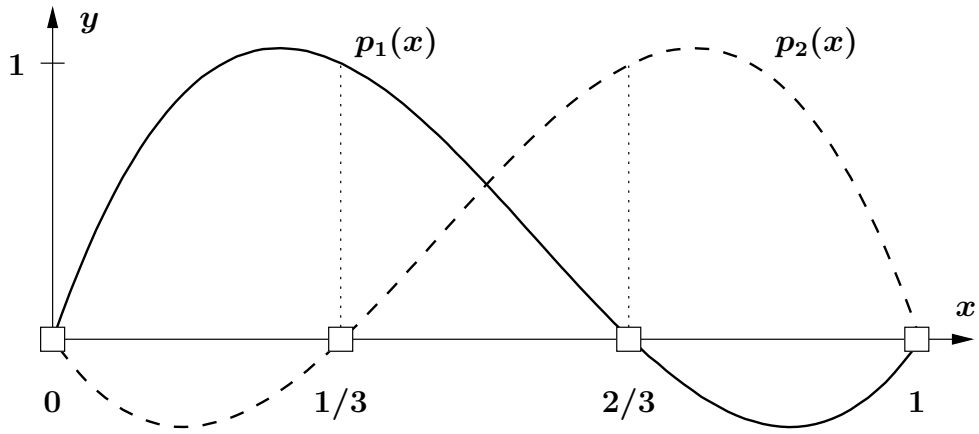

Abbildung 2: Lagrange-Polynome 3. Ordnung

3 Evolutionäre Optimierung

Ein evolutionärer Algorithmus besteht aus den folgenden Komponenten:

- Selektion
- Rekombination
- Mutation
- Bewertung

Zu Beginn muß der Algorithmus mit einer Anfangspopulation ausgestattet werden. Diese sollte nach Möglichkeit über den gesamten Lösungsraum verteilt sein, da durch eine einfache Rekombination nur Lösungen innerhalb dieses Raumes gefunden werden können. Der durch die Population aufgespannte Raum kann nur durch Mutation oder Rekombination mit Extrapolation erweitert werden. Aber auch dabei wird der Lösungsraum nur um einen angrenzenden Bereich erweitert.

Da die Individuen einer Population im Laufe der Optimierung dazu neigen, untereinander immer ähnlicher zu werden, besteht die Gefahr, daß der evolutionäre Algorithmus "einfriert". Wenn dies um das gesuchte globale Optimum geschieht, ist es nicht weiter schlimm. Es kann sich aber genausogut nur um ein lokales Optimum handeln. Um solch ein Einfrieren zu verhindern, gibt es unterschiedliche Strategien. Eine in [Poh99a] empfohlende Methode besteht darin, die Population in mehrere Subpopulationen aufzuteilen (regionales Modell) und diese für einige Generationen völlig getrennt voneinander zu behandeln. Nur gelegentlich können einige Individuen zwischen den Populationen migrieren und sich auf diese Weise vermischen.

Vom regionalen Modell unterscheidet sich das lokale Modell, indem es die Population zwar nicht aufteilt, aber räumliche Beziehungen zwischen den Individuen einführt. Nur benachbarte Individuen können hier miteinander rekombiniert werden. Ein gefundenes lokales Optimum kann sich dadurch nur schrittweise über die gesamte Population verbreiten. Konkurrieren mehrere Optima miteinander, können sich aneinandergrenzende Inseln bilden.

Ein völlig anderer Ansatz besteht darin, in jede Generation eine gewisse Anzahl vollkommen zufälliger Individuen (auch Monte-Carlo-Nachkommen genannt) einzustreuen. Diese müssen — wie die Anfangspopulation auch — über den gesamten Lösungsraum verteilt sein. Damit erhält eine Population, ähnlich wie bei der Migration des regionalen Modells, regelmäßig neue Impulse.

Außerdem erscheint es sinnvoll, die Lebenszeit der Individuen zu begrenzen. Durch die Selektion werden ja immer die besten Individuen für die Rekombination ausgewählt. Dadurch kann es geschehen, daß die Population irgendwann von der Elite aller vorangegangenen Generationen dominiert wird und neue Nachkommen kaum noch eine Chance haben. Alternativ kann auch ein Selektionsverfahren verwendet werden, daß auch für schwächere Individuen (in geringem Maße) durchlässig ist.

3.1 Selektionsverfahren

Je nach Wahl der Selektionsverfahrens und seiner Parameter lassen sich unterschiedliche Selektionsdrücke einstellen. Je höher der Selektionsdruck ist, umso stärker werden — im Sinne der Zielfunktion — bessere Individuen gegenüber schlechteren bei der Rekombination bevorzugt. Ein hoher Selektionsdruck begünstigt eine schnelle Konvergenz, bewirkt aber auch einen schnelleren Abbau der Vielfalt der Population (engl. *loss of diversity*). Als Selektionsverfahren zu nennen sind insbesondere:

- Abschneideselektion
- Turnierselektion
- Roulette-Selektion

Bei der Abschneide- und der Turnierselektion kommt es nur auf die Rangfolge der Individuen an, bei der Turnierselektion sogar nur auf der Rangfolge innerhalb der Turnierauswahl. Die Rangfolge wird dafür abhängig von den Zielfunktionswerten ermittelt. Die Roulette-Selektion dagegen wählt die Individuen mit einer ihrer Fitneß entsprechenden Wahrscheinlichkeit

aus. Diese Fitneß ist zweckmäßigerweise eine Funktion des Ranges. Der Zielfunktionswert selbst sollte nicht als Fitneß verwendet werden.

3.2 Rekombinationsverfahren

Das einfachste Rekombinationsverfahren besteht darin, die Nachkommen aus den einzelnen Komponenten der Eltern zusammenzusetzen. Die Entscheidung, welche Komponente von welchem Elter genommen wird, ist dabei zufällig. Abb. 3 zeigt ein Beispiel mit einer Population in einem zweidimensionalen Lösungsraum. Die Elterngeneration ist mit ausgefüllten Kreisen dargestellt. Die möglichen Nachkommen (nicht ausgefüllte Kreise) können bei der diskreten Rekombination nur Positionen auf einem bestimmten Gitter (punktierte Linien) einnehmen. Dieses Gitter ist durch die Lage der Eltern im Lösungsraum festgelegt. Die Zahl der möglichen Positionen der Nachkommen ist damit endlich und abzählbar. Nach einer Generation läßt sich die Anzahl der möglichen Positionen zu

$$\text{Anzahl Positionen} \;=\; \text{Anzahl Eltern}^{\,\text{Anzahl Dimensionen}}$$

berechnen. Diese Zahl steigt auch in folgenden Generationen nicht weiter an, da ja nach wie vor alle Individuen auf demselben Gitter liegen. Vielmehr kann sich diese Zahl sogar noch reduzieren, wenn einzelne Gitterlinien nach einer folgenden Selektion nicht mehr besetzt sind. Die Anzahl der möglichen Postionen wird dann irreversibel verringert. Daher müssen Strategien eingesetzt werden, mit denen diese Starrheit der diskreten Rekombination überwunden wird.

Bestehen die Komponenten aus Zahlen, so ist im Prinzip auch eine Inter- oder Extrapolation der beteiligten Eltern möglich. Damit wird die erreichbare Punktmenge von einer diskreten zu einer kontinuierlichen Menge erweitert. Mit jeder Generation entstehen dann neue Gitterlinien, so daß im Prinzip jeder Punkt des Lösungsraums erreicht werden kann. Ein anderer Ansatz besteht darin, die Nachkommen nachträglich zu modifizieren bzw. zu mutieren.

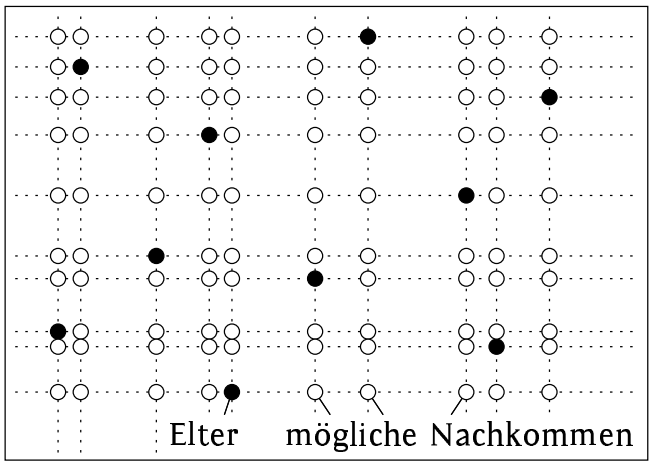

Abbildung 3: Diskrete zweidimensionale Rekombination

3.3 Mutation

Die erreichbare Punktmenge läßt sich nicht nur durch interpolierende Verfahren vergrößern. Auch durch eine der Rekombination nachfolgende normalverteilte Mutation kann eine endlich abzählbare Punktmenge auf unendlich viele Punkte erweitert werden. Durch die Mutation werden die Gitterpunkte der Nachkommen quasi verschmiert. Abb. 4 illustriert die Situation für dieselbe Population wie in Abb. 3. Die Nachkommen liegen dann nicht mehr unbedingt genau auf den Gitterpunkten, sondern können sich ein Stück davon entfernen. Dadurch entstehen mit jeder Generation neue Gitterlinien, und in der Folge sind auch die ursprünglich weniger wahrscheinlichen Positionen gut erreichbar.

Die Mutationsbreite oder Standardabweichung braucht nicht konstant gewählt zu werden, denn einerseits ist die günstigste Mutationsbreite abhängig von der Empfindlichkeit der mutierten Komponente, andererseits aber auch vom Fortschritt der Evolution. In der Nähe eines Optimums ist eine kleinere Mutationsbreite zweckmäßiger als in größerer Entfernung davon. Eine elegante Methode zur Lösung dieses Problems besteht darin, die Mutationsbreite mit zu vererben und dabei zu variieren. Erweist sich eine Mutationsbreite als vorteilhaft, so wird das zugehörige Individuum bei der Selektion bevorzugt. Damit kann der evolutionäre Algorithmus die Mutationsbreite automatisch einstellen.

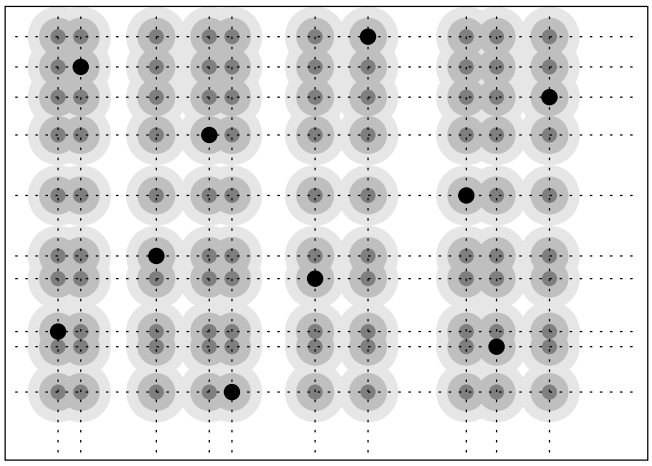

Abbildung 4: Diskrete Rekombination mit Mutation

Reparatur

Als Folge der Mutation (und auch durch eine Rekombination mit Extrapolation) können leicht Individuen entstehen, die außerhalb des zugelassenen Lösungsraums liegen. Daher ist anschließend eine Korrektur der betroffenen Individuen notwendig, indem die einzelnen Komponenten auf erlaubte Werte begrenzt werden.

4 Vergleich verschiedener evolutionärer Algorithmen

In diesem Kapitel soll das Konvergenzverhalten verschiedener Ausprägungen des evolutionären Algorithmus an einem interessanten Beispiel untersucht werden.

Die Optimierungsaufgabe besteht darin, ein in der stabilen Gleichgewichtslage hängendes Doppelpendel in die labile (stehende) Gleichgewichtslage zu überführen. Das Doppelpendel ist dazu an der Aufhängung mit einem Antrieb ausgestattet. Untereinander sind die beiden Stangen des Doppelpendels einfach mit einem Drehgelenk verbunden.

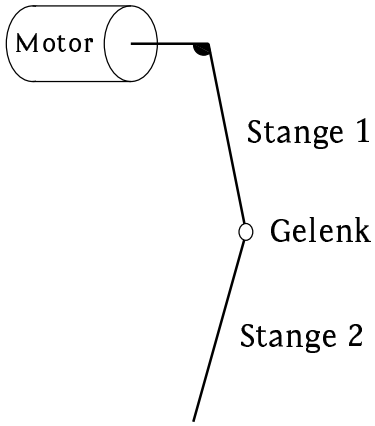

Abbildung 5: Anordnung des Doppelpendels

4.1 Kinematik des Doppelpendels

Das Doppelpendel wird durch die beiden Stablängen ℓ_1 und ℓ_2, die Massen m_1 und m_2 sowie die Trägheitsmomente Θ_1 und Θ_2 beschrieben. Für die beiden Winkel φ_1 und φ_2 sind zunächst die Bewegungsgleichungen gesucht. Einzige Eingangsgröße für die Bewegungsgleichung ist das vom Motor in der Aufhängung eingebrachte Drehmoment M_A.

Für die Berechnung der Bewegungsgleichung wird das Pendel zunächst wie in Abb. 7 freigeschnitten. Anschließend werden die Gleichgewichtsbedingungen für die beiden Stangen formuliert.

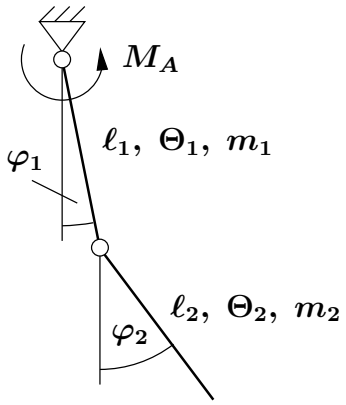

Abbildung 6: Doppelpendel

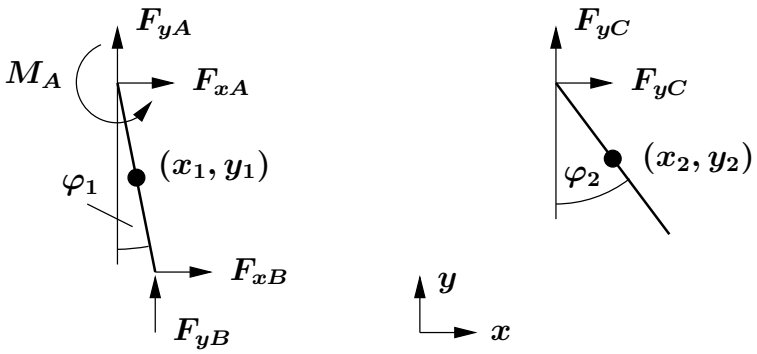

Abbildung 7: Freigeschnittenes Doppelpendel

Stange 1

$$\Theta_1 \ddot{\varphi}_1 = M_A + \left[\frac{\ell_1}{2}(F_{xB} - F_{xA})\cos\varphi_1 + \frac{\ell_1}{2}(F_{yB} - F_{yA})\sin\varphi_1\right] \quad (1)$$

$$m_1 \ddot{x}_1 = F_{xA} + F_{xB} \quad (2)$$

$$m_1 \ddot{y}_1 = F_{yA} + F_{yB} - m_1 g \quad (3)$$

Randbedingungen (Lager in A):

$$x_1 = \frac{\ell_1}{2}\sin\varphi_1 \quad (4)$$

$$y_1 = -\frac{\ell_1}{2}\cos\varphi_1 \quad (5)$$

Stange 2

$$\Theta_2\,\ddot{\varphi}_2 = -\frac{\ell_2}{2}\left(F_{xC}\cos\varphi_2 + F_{yC}\sin\varphi_2\right) \tag{6}$$

$$m_2\,\ddot{x}_2 = F_{xC} \tag{7}$$

$$m_2\,\ddot{y}_2 = F_{yC} - m_2\,g \tag{8}$$

Die gelenkige Verbindung zwischen den beiden Stangen wird durch die folgenden Kompatibilitätsbedingungen ausgedrückt:

$$x_2 = \ell_1\sin\varphi_1 + \frac{\ell_2}{2}\sin\varphi_2 \tag{9}$$

$$y_2 = -\ell_1\cos\varphi_1 - \frac{\ell_2}{2}\cos\varphi_2 \tag{10}$$

$$F_{xC} = -F_{xB} \tag{11}$$

$$F_{yC} = -F_{yB} \tag{12}$$

In (7) und (8) werden die zweiten Ableitungen von x_2 und y_2 benötigt.

$$\dot{x}_2 = \dot{\varphi}_1\,\ell_1\,\cos\varphi_1 + \dot{\varphi}_2\,\frac{\ell_2}{2}\,\cos\varphi_2 \tag{13}$$

$$\ddot{x}_2 = \ddot{\varphi}_1\,\ell_1\,\cos\varphi_1 - \dot{\varphi}_1^2\,\ell_1\,\sin\varphi_1 + \ddot{\varphi}_2\,\frac{\ell_2}{2}\,\cos\varphi_2 - \dot{\varphi}_2^2\,\frac{\ell_2}{2}\,\sin\varphi_2 \tag{14}$$

$$\dot{y}_2 = \dot{\varphi}_1\,\ell_1\,\sin\varphi_1 + \dot{\varphi}_2\,\frac{\ell_2}{2}\,\sin\varphi_2 \tag{15}$$

$$\ddot{y}_2 = \ddot{\varphi}_1\,\ell_1\,\sin\varphi_1 + \dot{\varphi}_1^2\,\ell_1\,\cos\varphi_1 + \ddot{\varphi}_2\,\frac{\ell_2}{2}\,\sin\varphi_2 + \dot{\varphi}_2^2\,\frac{\ell_2}{2}\,\cos\varphi_2 \tag{16}$$

Diese eingesetzt liefert dann mit (11) und (12)

$$\begin{aligned} F_{xB} &= -m_2\,\ddot{x}_2 \\ &= -m_2\,(\ddot{\varphi}_1\,\ell_1\,\cos\varphi_1 - \dot{\varphi}_1^2\,\ell_1\,\sin\varphi_1 \\ &\qquad + \ddot{\varphi}_2\,\frac{\ell_2}{2}\,\cos\varphi_2 - \dot{\varphi}_2^2\,\frac{\ell_2}{2}\,\sin\varphi_2) \end{aligned} \tag{17}$$

$$\begin{aligned} F_{yB} &= -m_2\,\ddot{y}_2 - m_2\,g \\ &= -m_2\,(\ddot{\varphi}_1\,\ell_1\,\sin\varphi_1 + \dot{\varphi}_1^2\,\ell_1\,\cos\varphi_1 \\ &\qquad + \ddot{\varphi}_2\,\frac{\ell_2}{2}\,\sin\varphi_2 + \dot{\varphi}_2^2\,\frac{\ell_2}{2}\,\cos\varphi_2) - m_2\,g\ . \end{aligned} \tag{18}$$

Für (2) und (3) werden noch die zweiten Ableitungen von x_1 und y_1 benötigt.

$$\dot{x}_1 = \dot{\varphi}_1\,\frac{\ell_1}{2}\,\cos\varphi_1 \tag{19}$$

$$\ddot{x}_1 = \ddot{\varphi}_1 \frac{\ell_1}{2} \cos\varphi_1 - \dot{\varphi}_1^2 \frac{\ell_1}{2} \sin\varphi_1 \tag{20}$$

$$\dot{y}_1 = \dot{\varphi}_1 \frac{\ell_1}{2} \sin\varphi_1 \tag{21}$$

$$\ddot{y}_1 = \ddot{\varphi}_1 \frac{\ell_1}{2} \sin\varphi_1 + \dot{\varphi}_1^2 \frac{\ell_1}{2} \cos\varphi_1 \tag{22}$$

Damit lassen sich nun die Kräftedifferenzen in (1) ausdrücken.

$$\begin{aligned} F_{xB} - F_{xA} &= 2\,F_{xB} - m_1\,\ddot{x}_1 \\ &= -2\,m_2 \left(\ddot{\varphi}_1\,\ell_1\,\cos\varphi_1 - \dot{\varphi}_1^2\,\ell_1\,\sin\varphi_1 \right. \\ &\qquad \left. + \ddot{\varphi}_2 \frac{\ell_2}{2} \cos\varphi_2 - \dot{\varphi}_2^2 \frac{\ell_2}{2} \sin\varphi_2 \right) \\ &\quad -m_1 \left(\ddot{\varphi}_1 \frac{\ell_1}{2} \cos\varphi_1 - \dot{\varphi}_1^2 \frac{\ell_1}{2} \sin\varphi_1 \right) \end{aligned} \tag{23}$$

$$\begin{aligned} F_{yB} - F_{yA} &= 2\,F_{yB} - m_1\,\ddot{y}_1 - m_1\,g \\ &= -2\,m_2 \left(\ddot{\varphi}_1\,\ell_1\,\sin\varphi_1 + \dot{\varphi}_1^2\,\ell_1\,\cos\varphi_1 \right. \\ &\qquad \left. + \ddot{\varphi}_2 \frac{\ell_2}{2} \sin\varphi_2 + \dot{\varphi}_2^2 \frac{\ell_2}{2} \cos\varphi_2 \right) \\ &\quad -m_1 \left(\ddot{\varphi}_1 \frac{\ell_1}{2} \sin\varphi_1 + \dot{\varphi}_1^2 \frac{\ell_1}{2} \cos\varphi_1 \right) \\ &\quad -2\,m_2\,g - m_1\,g \end{aligned} \tag{24}$$

(17) und (18) eingesetzt in (6) ergibt

$$\begin{aligned} \Theta_2\,\ddot{\varphi}_2 &= \frac{\ell_2}{2} \left(F_{xB} \cos\varphi_2 + F_{yB} \sin\varphi_2 \right) \\ &= -\frac{\ell_2\,m_2}{2} \left[\ddot{\varphi}_1\,\ell_1 \underbrace{\left(\cos\varphi_1\,\cos\varphi_2 + \sin\varphi_1\,\sin\varphi_2 \right)}_{\cos(\varphi_1 - \varphi_2)} \right. \\ &\qquad - \dot{\varphi}_1^2\,\ell_1 \underbrace{\left(\sin\varphi_1\,\cos\varphi_2 - \cos\varphi_1\,\sin\varphi_2 \right)}_{\sin(\varphi_1 - \varphi_2)} \\ &\qquad + \ddot{\varphi}_2 \frac{\ell_2}{2} \underbrace{\left(\cos^2\varphi_2 + \sin^2\varphi_2 \right)}_{1} \\ &\qquad \left. - \dot{\varphi}_2^2 \frac{\ell_2}{2} \underbrace{\left(\sin\varphi_2\,\cos\varphi_2 - \cos\varphi_2\,\sin\varphi_2 \right)}_{0} + g\,\sin\varphi_2 \right] \end{aligned}$$

$$= -\frac{\ell_2\, m_2}{2}\left[\ddot{\varphi}_1\,\ell_1\,\cos(\varphi_1-\varphi_2) - \dot{\varphi}_1^2\,\ell_1\,\sin(\varphi_1-\varphi_2) + \ddot{\varphi}_2\,\frac{\ell_2}{2} + g\,\sin\varphi_2\right] \tag{25}$$

und (23) sowie (24) eingesetzt in (1)

$$\begin{aligned}
\Theta_1\,\ddot{\varphi}_1 &= M_A + \frac{\ell_1}{2}\Bigg\{ - 2\,m_2\Bigg[\ddot{\varphi}_1\,\ell_1\,\underbrace{(\cos^2\varphi_1+\sin^2\varphi_1)}_{1} \\
&\qquad - \dot{\varphi}_1^2\,\ell_1\,\underbrace{(\sin\varphi_1\,\cos\varphi_1-\cos\varphi_1\,\sin\varphi_1)}_{0} \\
&\qquad + \ddot{\varphi}_2\,\frac{\ell_2}{2}\,\underbrace{(\cos\varphi_2\,\cos\varphi_1+\sin\varphi_2\,\sin\varphi_1)}_{\cos(\varphi_1-\varphi_2)} \\
&\qquad - \dot{\varphi}_2^2\,\frac{\ell_2}{2}\,\underbrace{(\sin\varphi_2\,\cos\varphi_1-\cos\varphi_2\,\sin\varphi_1)}_{-\sin(\varphi_1-\varphi_2)} \\
&\qquad + g\,\sin\varphi_1\Bigg] \\
&\quad - m_1\Bigg[\ddot{\varphi}_1\,\frac{\ell_1}{2}\,\underbrace{(\cos^2\varphi_1+\sin^2\varphi_1)}_{1} \\
&\qquad - \dot{\varphi}_1^2\,\frac{\ell_1}{2}\,\underbrace{(\sin\varphi_1\,\cos\varphi_1-\cos\varphi_1\,\sin\varphi_1)}_{0} \\
&\qquad + g\,\sin\varphi_1\Bigg]\Bigg\} \\
&= M_A - \frac{\ell_1}{2}\Bigg\{2\,m_2\left[\ddot{\varphi}_1\,\ell_1 + \ddot{\varphi}_2\,\frac{\ell_2}{2}\,\cos(\varphi_1-\varphi_2) + \dot{\varphi}_2^2\,\frac{\ell_2}{2}\,\sin(\varphi_1-\varphi_2) + g\,\sin\varphi_1\right] \\
&\qquad + m_1\left[\ddot{\varphi}_1\,\frac{\ell_1}{2} + g\,\sin\varphi_1\right]\Bigg\}\ .
\end{aligned} \tag{26}$$

Die letzten beiden Gleichungen (25) und (26) bilden nun ein implizites Differentialgleichungssystem zweiter Ordnung für die beiden Unbekannten φ_1 und φ_2.

$$\begin{aligned}
\ddot{\varphi}_1 &= \mathrm{f}_1(M_A,\varphi_1,\varphi_2,\dot{\varphi}_2,\ddot{\varphi}_1,\ddot{\varphi}_2) \\
\ddot{\varphi}_2 &= \mathrm{f}_2(\varphi_1,\varphi_2,\dot{\varphi}_1,\ddot{\varphi}_1,\ddot{\varphi}_2)
\end{aligned}$$

Für die numerische Integration wird jedoch eine explizite Form benötigt. Glücklicherweise sind die Terme mit den zweiten Ableitungen linear, so daß sich ein lineares Gleichungssystem für $\ddot{\varphi}_1$ und $\ddot{\varphi}_2$ angeben läßt:

$$\begin{aligned} 0 &= a_1\,\ddot{\varphi}_1 + a_2\,\ddot{\varphi}_2 + a_3 \\ 0 &= b_1\,\ddot{\varphi}_1 + b_2\,\ddot{\varphi}_2 + b_3 \end{aligned} \tag{27}$$

mit

$$\begin{aligned} a_1 &= -\ell_1^2\,m_2 - \Theta_1 - \frac{\ell_1^2}{4}\,m_1 \\ a_2 &= -\frac{m_2\,\ell_1\,\ell_2}{2}\,\cos(\varphi_2 - \varphi_1) \\ a_3 &= M_A + \frac{\ell_1\,\ell_2\,m_2}{2}\,\dot{\varphi}_2^2\,\sin(\varphi_2 - \varphi_1) - \ell_1\,m_2\,g\,\sin\varphi_1 - \frac{\ell_1\,m_1}{2}\,g\,\sin\varphi_1 \\ b_1 &= -\frac{\ell_1\,\ell_2\,m_2}{2}\,\cos(\varphi_2 - \varphi_1) \\ b_2 &= -\frac{\ell_2^2\,m_2}{4} - \Theta_2 \\ b_3 &= -\frac{\ell_1\,\ell_2\,m_2}{2}\,\dot{\varphi}_1^2\,\sin(\varphi_2 - \varphi_1) - \frac{\ell_2\,m_2\,g}{2}\,\sin\varphi_2 \end{aligned}$$

Die Lösung des Gleichungssystems (27) lautet in Matrizenform

$$\begin{aligned} \underline{\ddot{\varphi}} = \begin{pmatrix} \ddot{\varphi}_1 \\ \ddot{\varphi}_2 \end{pmatrix} &= \begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix}^{-1} \begin{pmatrix} -a_3 \\ -b_3 \end{pmatrix} \\ &= \frac{1}{a_1\,b_2 - a_2\,b_1} \begin{pmatrix} b_2 & -a_2 \\ -b_1 & a_1 \end{pmatrix} \begin{pmatrix} -a_3 \\ -b_3 \end{pmatrix} \end{aligned} \tag{28}$$

4.2 Parametrisierung der Steuertrajektorie

Es geht darum, das Doppelpendel aus der unteren Gleichgewichtslage in den oberen Totpunkt zu bringen. Als einzige Steuergröße kann ein Drehmoment an der Lagerung aufgebracht werden. Da das Doppelpendel ein Phasennichtminimalsystem ist, genügt es zum Aufrichten nicht, einfach ein Drehmoment in eine Richtung zu beaufschlagen. Vielmehr muß zunächst ausgeholt werden, um anschließend *beide* Stangen nach oben zu heben.

Im Hinblick auf die Phasennichtminimalität wird hier eine aus drei Lagrange-Polynomen zusammengesetzte Steuerkurve gewählt. Diese Steuerkurve hat

dann genau vier Freiheitsgrade bzw. Parameter: Die drei Wichtungsfaktoren w_1, w_2 und w_3 für die Polynome sowie die Dauer T des Ablaufs. Mit den Polynomen p_1 bis p_3 kann das Steuermoment M_A dann wie folgt dargestellt werden:

$$M_A(t) = \begin{cases} \sum_{i=1}^{3} w_i \, p_i(t/T) & \text{für} \quad 0 \leq t \leq T \\ 0 & \text{sonst} \end{cases} \tag{29}$$

Die freien Parameter sollen nun mit verschiedenen evolutionären Algorithmen bestimmt werden.

4.3 Zielfunktion

In der Zielfunktion muß zum Ausdruck kommen, daß die beiden Pendelstangen jeweils den Winkel π einnehmen und dabei stillstehen sollen. Dabei dürfen schlechte Werte bei einem Kriterium nicht durch gute bei einem anderen kompensiert werden können. Daher wird eine Maximum-Funktion verwendet, bei der das am schlechtesten erfüllte Kriterium den Funktionswert bestimmt. Außerdem wird eine Normierung vorgenommen, so daß der Zielfunktionswert immer im Intervall $[0, 1]$ liegt. Im optimalen Fall liefert die Zielfunktion den Wert null.

$$Z \stackrel{\text{def}}{=} 1 - \frac{1}{1 - \max(|\varphi_1 - \pi|, |\varphi_2 - \pi|, |\dot{\varphi}_1|, |\dot{\varphi}_2|)} \tag{30}$$

Zur Berechnung der Zielfunktion ist es notwendig, den Aufschwingvorgang mit der zu testenden Steuertrajektorie zu simulieren. Anschließend werden die so gefundenen Zustandsgrößen des Pendels in (30) eingesetzt. Die Berechnung des Zielfunktionswerts ist somit ziemlich aufwendig und nimmt den größten Teil der benötigten Rechenzeit ein. Daher wird dieser Teil der Optimierung nicht, wie der verwendete evolutionäre Algorithmus selbst, in einem m-Skript programmiert, sondern in Ada. Die Zielfunktion kann so in die Form einer `.oct`-Datei (einer vorkompilierten Octave-Bibliotheksfunktion) gebracht werden. Die Zielfunktion muß dann nicht interpretiert werden, sondern kann vom Prozessor direkt ausgeführt werden. Da die Erzeugung einer `.oct`-Datei aus einer Ada-Funktion recht kompliziert ist, wird die Vorgehensweise in Anhang A genauer beschrieben.

4.4 Versuchsdurchführung

Die Versuche wurden mit der GEATbx-Toolbox durchgeführt, die [Poh99a] beiliegt. Die Toolbox enthält einen weitgehend konfigurierbaren evolutionären Algorithmus für Matlab. Nach einigen kleineren Anpassungen ist diese Toolbox auch mit dem freien Programm Octave[2] anwendbar.

	Versuch				
Parameter	1	2	3	4	5
Populationen	4	1	4	1	1 (lokal)
Selektion	Turnierselektion (Turniergröße 4)		Abschneideselektion (Schwelle 0,4)		Halbstern

Tabelle 1: Versuchsbedingungen bei der Ausprägungsuntersuchung

Um vergleichbare Ergebnisse zu erhalten, sind die Parameter der Turnier- und der Abschneideselektion in den Versuchen so gewählt, daß sich etwa der gleiche Wert für die Selektionsintensität ergibt.

Neben drei Koeffizienten für ein Lagrange-Polynom wird auch noch die Dauer des Aufschwingvorgangs, also insgesamt vier Parameter, optimiert. In [Poh99a] wird empfohlen, für vier Freiheitsgrade 400 Generationen zu rechnen. In diesem Versuch werden jedoch mehr Generationen durchgerechnet, um einen besseren Überblick zu bekommen.

Die Streuung der Optimierungsläufe ist recht hoch. Um dennoch vergleichbare Ergebnisse zu bekommen, wurden mit jeder Konfiguration 11 Optimierungsläufe durchgerechnet. Die Verläufe der Zielfunktionswerte über die Generationen wurden anschließend innerhalb jeder Konfiguration gemittelt.

4.5 Versuchsergebnisse

Abb. 8 zeigt diese gemittelten Verläufe. Die beiden regionalen Konfigurationen scheinen deutliche Vorteile gegenüber den globalen und der lokalen Konfiguration zu haben.

[2] erhältlich unter http://www.octave.org/

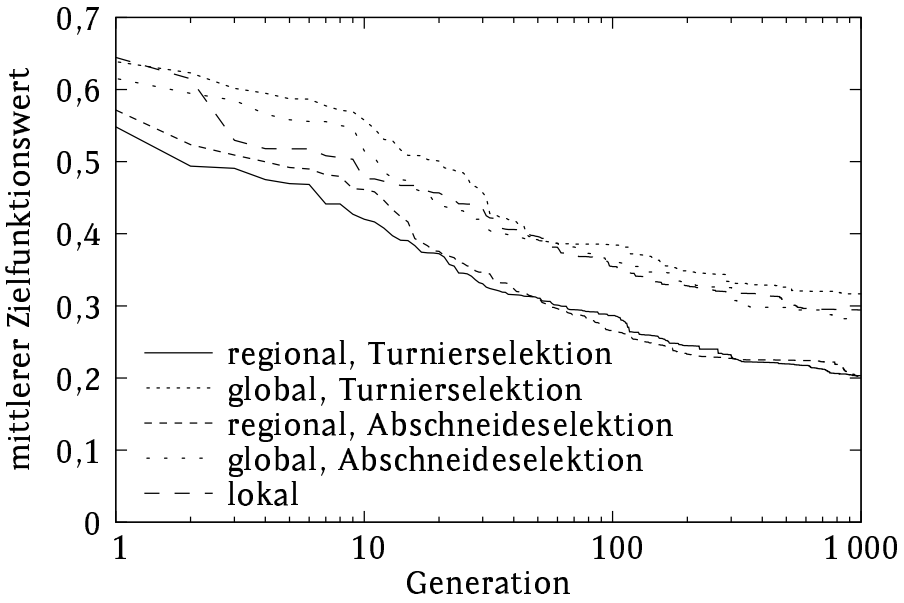

Abbildung 8: Konvergenz über Generationen

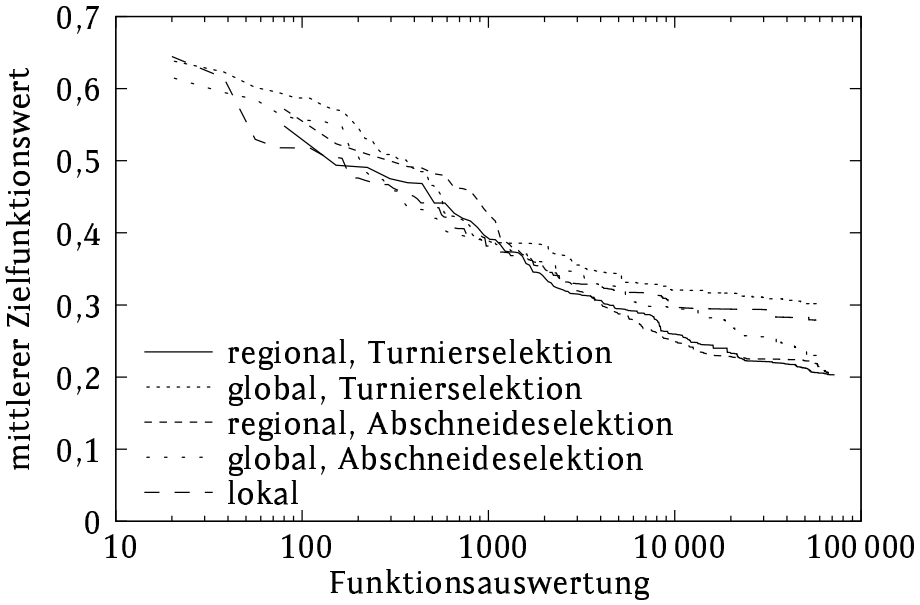

Abbildung 9: Konvergenz über Funktionsauswertungen

Dieses Bild relativiert sich jedoch, wenn der Zielfunktionswert nicht über die Anzahl der Generationen aufgetragen wird, sondern über die Anzahl der Zielfunktionsauswertungen (Abb. 9). Nach anfänglich vergleichbaren Ergebnissen scheinen das globale Modell mit Turnierselektion und — etwas später

— auch das lokale Modell vorzeitig zu konvergieren. Unter den anderen Varianten sind in diese Bild keine deutlichen Unterschiede im Konvergenzverhalten mehr zu erkennen.

Die dicht beieinanderliegenden Linien in Abb. 9 dürfen nicht darüber hinwegtäuschen, daß die Ergebnisse der einzelnen Optimierungsläufe sehr unterschiedlich ausgefallen sind. In Worten reicht die Spanne von fast gar keiner Konvergenz bis zu einem sehr guten Ergebnis. Abb. 10 zeigt die Bereiche, in denen die Lösungen nach 28 800 und 57 600 Funktionsauswertungen lagen, mit Zentralwert. Dabei entsprechen 28 800 Funktionsauswertungen 400 Generationen mit dem regionalem Populationsmodell.

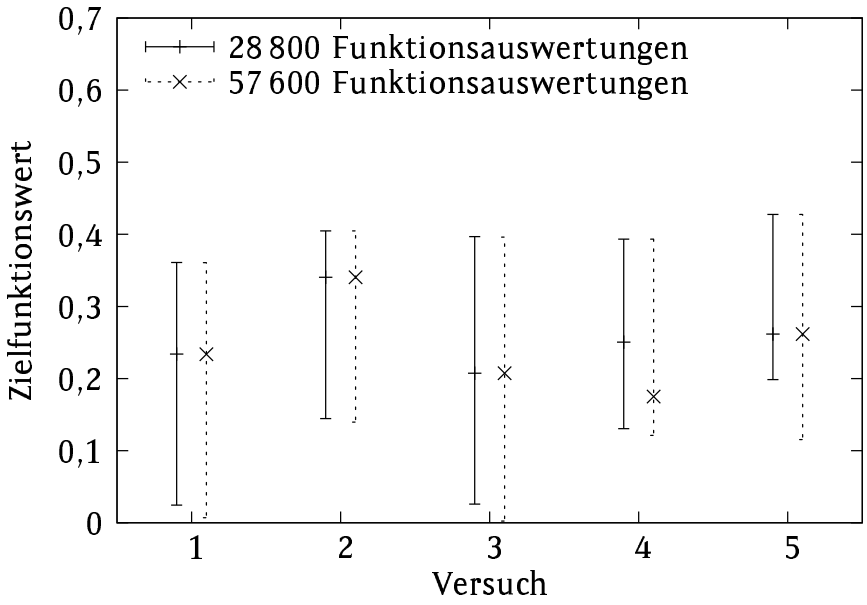

Abbildung 10: Ergebnisintervalle nach 28 800 und 57 600 Funktionsauswertungen

Das beste Ergebnis von allen lieferte die dritte Konfiguration (regionales Populationsmodell und Abschneideselektion). Abb. 11 zeigt den zeitlichen Verlauf der optimierten Steuertrajektorie und Abb. 12 den daraus resultierenden Verlauf des Aufschwingens.

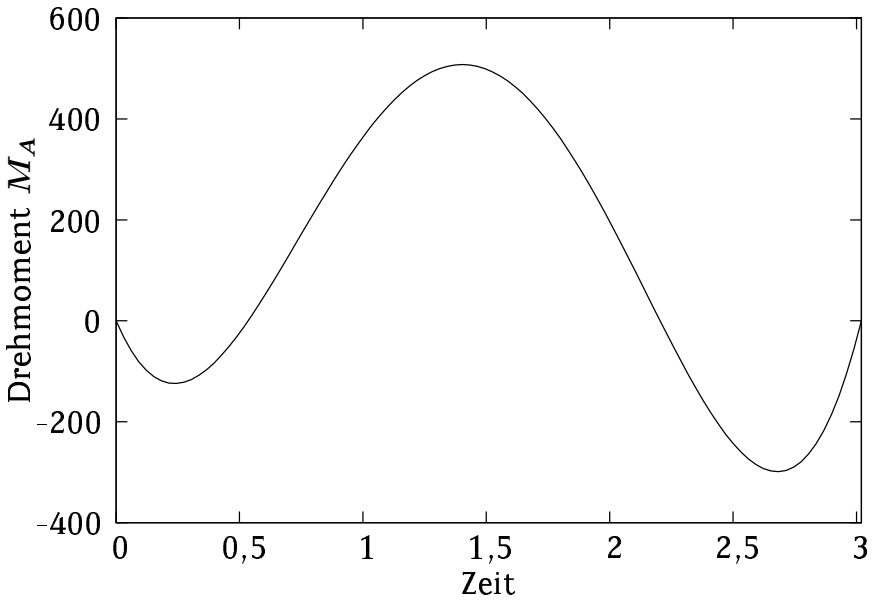

Abbildung 11: Beste Steuertrajektorie aller Optimierungsläufe

Schlußfolgerungen

Die Konvergenz eines evolutionären Algorithmus scheint im wesentlichen durch die Anzahl der Funktionsauswertungen bestimmt zu werden. Die spezielle Ausprägung des Algorithmus ist den Ergebnissen nach eher zweitrangig. Allerdings scheinen die globale Population mit Turnierselektion und die lokale Population systematisch etwas schlechter abzuschneiden als die anderen Konfigurationen. Ein sehr wesentlicher Faktor für die Güte des Ergebnisses ist jedoch bei allen Konfigurationen der Zufall.

Die besten Ergebnisse waren unter den Optimierungsläufen mit regionalen Populationen zu finden. Wer eine besonders gute Lösung finden will, sollte demnach eine solche Konfigurationen wählen und sich darauf einstellen, mehrere Optimierungsläufe zu rechnen.

Da die evolutionäre Optimierung in der vorliegenden Anwendung nach einer festen Zeitspanne, in der Regel lange bevor eine vorzeitige Konvergenz zu erwarten ist, abgebrochen wird, scheint hierfür jede der getesteten Ausprägungen geeignet zu sein.

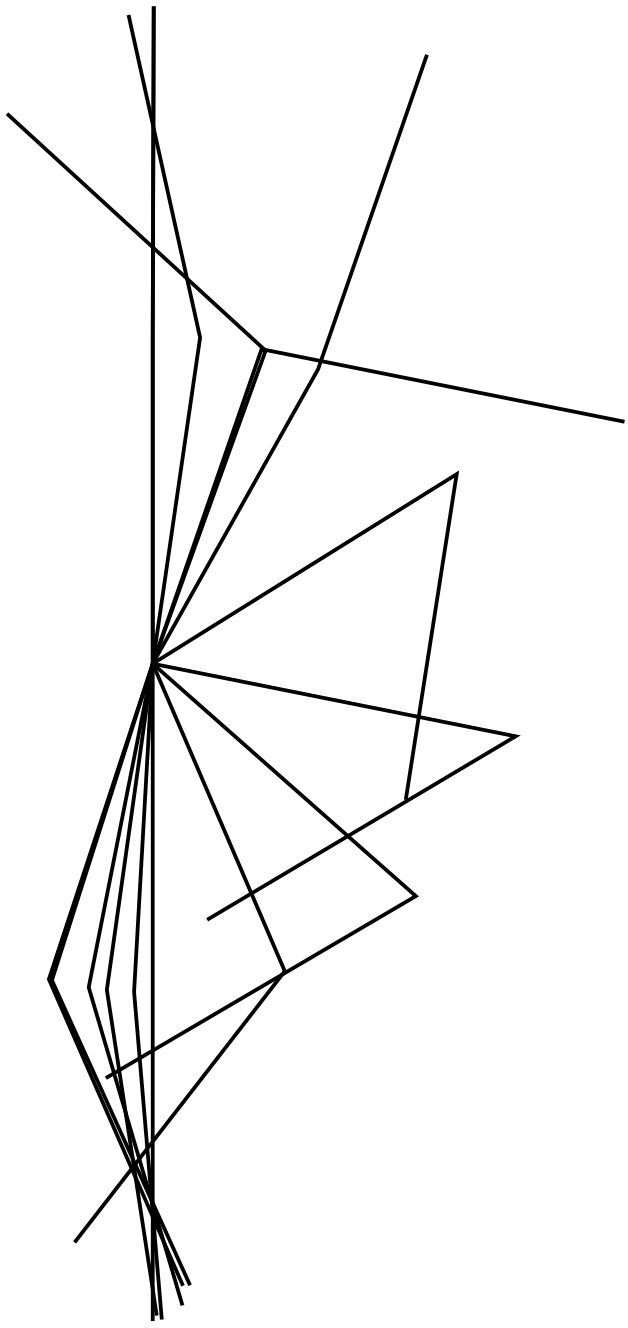

Abbildung 12: Aufschwingen des Doppelpendels

5 Vorversuch

Die Wirkung der Optimierung soll zunächst in einem "Trockenversuch" abgeschätzt werden. Hierzu sollen allerdings schon realistische Bedingungen geschaffen werden. Daher wird ein aus dem labilen Gleichgewicht fallendes Doppelpendel simuliert und für dieses Szenario eine Steuertrajektorie ermittelt, die den Zusammenbruch aufhalten soll. Bei dieser Gelegenheit wird auch der Einfluß des Selektionsdrucks, der Monte-Carlo-Nachkommen sowie der dynamischen Mutationsbreitenanpassung untersucht.

Versuchsablauf

- Inititialisieren des Doppelpendels mit zufälligen Werten nahe des labilen Gleichgewichts.
- Integration der Bewegungsgleichung bis ein unterer Grenzwert (0,7) für die Zielfunktion (Qualität des Zustands) unterschritten wird.
- Zurückgehen in der Zeit bis zum Start der noch zu ermittelnden Steuertrajektorie. Die Zustandswerte zu diesem Zeitpunkt sind die Anfangsbedingungen für die Optimierung.
- Optimieren der Steuertrajektorie.

5.1 Evolutionärer Algorithmus

Der verwendete evolutionäre Algorithmus arbeitet mit einer (globalen) Population, Abschneideselektion und adaptiver Mutationsbreite. Es handelt sich um eine modifizierte Umsetzung in Form einer Ada-Bibliothek des in [Blü91] beschriebenen Programms. Der Algorithmus kann über mehrere Parameter an die Aufgabenstellung angepaßt werden.

Die Schwelle $\boldsymbol{Trunc}$ der Abschneideselektion berechnet sich mit diesen Parametern zu

$$\boldsymbol{Trunc} = \frac{\mu}{\mu + \lambda + \chi} , \tag{31}$$

Parameter	Bedeutung
μ	Anzahl der Eltern je Generation
σ	Anzahl der Eltern je Nachkomme
λ	Anzahl der Nachkommen durch Rekombination je Generation
χ	Anzahl der Monte-Carlo-Nachkommen je Generation
ω	maximale Lebenszeit eines Individuums
ε	initiale Mutationsbreite

Tabelle 2: Parameter des evolutionären Algorithmus

wobei die Individuen mit erreichter maximaler Lebenszeit unabhängig von ihrer Fitneß automatisch hinter die Abschneideschwelle fallen. Wieviele Nachkommen je Elter erzeugt werden, wird vom $generation\ gap$ angegeben.

$$generation\ gap = \frac{\lambda + \chi}{\mu} \tag{32}$$

Die Wiedereinfügerate ist bei diesem Algorithmus gleich eins. Es stehen also der folgenden Selektion alle generierten Nachkommen zur Verfügung. Der Selektionsdruck ergibt sich aus dem Verhältnis der Größe der Gesamtpopulation zur Größe der von der Abschneideselektion ausgewählten neuen Elterngeneration.

$$Selektionsdruck = \frac{\mu + \lambda + \chi}{\mu} \tag{33}$$

Der Algorithmus arbeitet mit einer Mutationsrate von eins. Das heißt, daß grundsätzlich alle Variablen mutiert werden.

5.2 Einfluß des Selektionsdrucks

Es wurden mehrere Konfiguration gerechnet. Um aussagekräftige Mittelwerte bilden zu können, wurde jede Konfiguration mit acht verschiedenen Anfangsbedingungen und je fünf Optimierungsläufen, also 44 Durchläufe je Konfiguration, gerechnet. Abb. 14 zeigt die gemittelten Ergebnisse der Versuche. Alle Konfigurationen hatten eine Populationsgröße von 20 Individuen, unterschieden sich aber in den gewählten Selektionsdrücken.

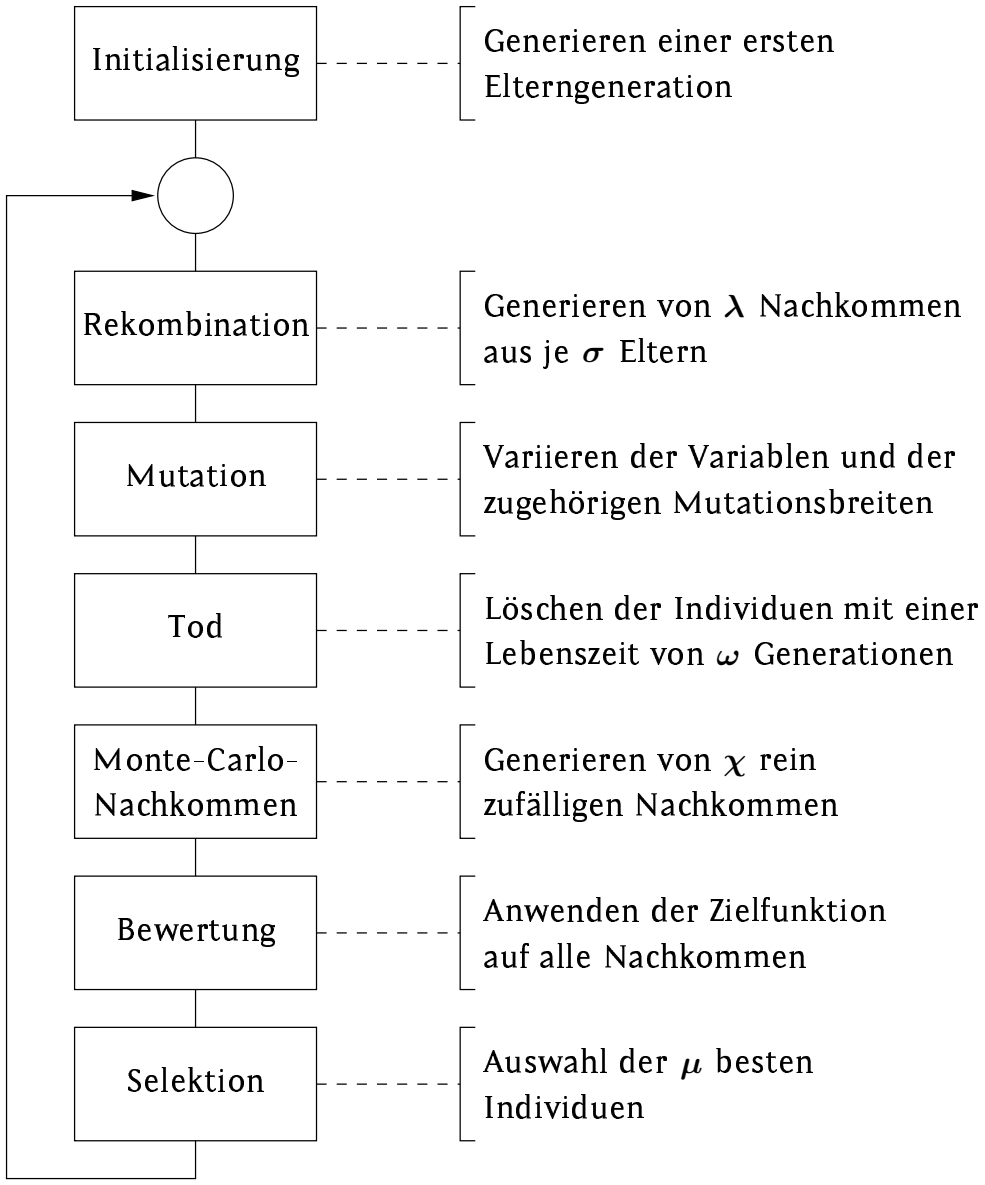

Abbildung 13: Hauptschleife des evolutionären Algorithmus

Dabei fällt eine Eigentümlichkeit auf: Bei etwa 200 Funktionsauswertungen schneiden sich die Kurven fast in einem Punkt. Da ist es offenbar egal, mit welchem Selektionsdruck der evolutionäre Algorithmus arbeitet. Vor diesem Punkt sind kleinere Selektionsdrücke vorteilhafter als größere, erst dahinter bewirkt ein größerer Selektionsdruck ein besseres Ergebnis.

Das Ergebnis nach tausend Funktionsauswertungen ist in Abb. 15 noch einmal über den Selektionsdruck aufgetragen. Dargestellt sind die Mittelwerte

Parameter	Versuch 1	2	3	4	5	6	7	8	9
Populationsgröße	20								
maximale Lebensdauer	3								
Selektionsdruck	2,0	2,2	2,5	2,9	3,3	4,0	5,0	6,7	10,0
Monte-Carlos	0								
Mutationsbreite	dynamisch								

Tabelle 3: Versuchsbedingungen der Selektionsdruckuntersuchung

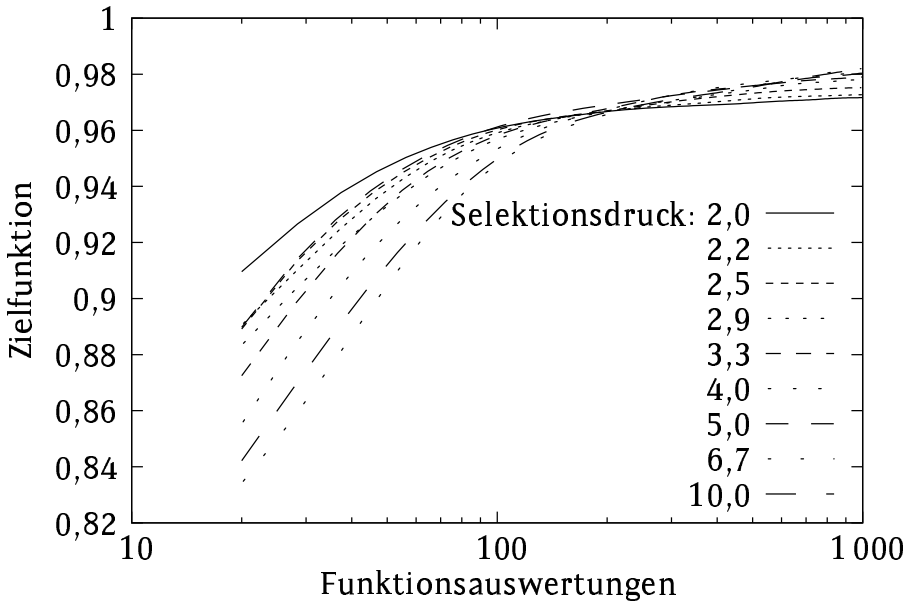

Abbildung 14: Konvergenz beim Vorversuch

und die einfache Standardabweichung nach beiden Seiten. Bis etwa zu einem Selektionsdruck von vier ist eine Verbesserung des Ergebnisses zu erkennen. Eine weitere Steigerung des Selektionsdrucks bewirkt keine nennenswerte Änderung mehr.

5.3 Einfluß der Monte-Carlo-Nachkkommen

In einem weiteren Experiment wurde untersucht, wie sich die Konvergenz verändert, wenn einzelne Nachkommen durch zufällige Monte-Carlo-Nach-

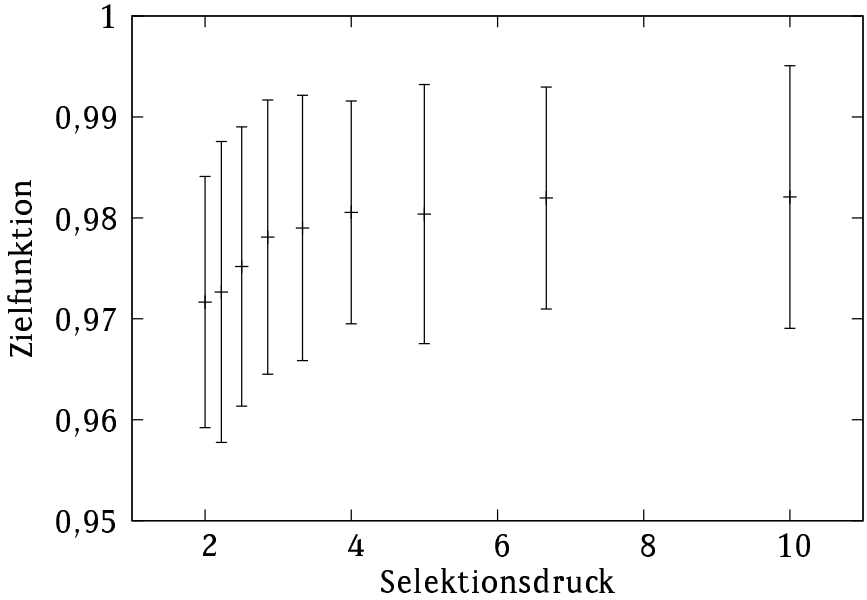

Abbildung 15: Einfluß des Selektionsdrucks auf das Ergebnis

kommen ersetzt werden. Zum Vergleich wird noch ein Optimierungslauf mit rein stochastischer Suche (nur Monte-Carlo-Nachkommen) durchgeführt. Abb. 16 zeigt wieder die erreichten Mittelwerte über die Funktionsauswertungen.

	Versuch				
Parameter	1	2	3	4	5
Populationsgröße	20				
maximale Lebensdauer	3				
Selektionsdruck	2	2	10	10	–
Monte-Carlos	0	1	0	1	10
Mutationsbreite	dynamisch				

Tabelle 4: Versuchsbedingungen beim Monto-Carlo-Experiment

Es zeigt sich, daß einzelne eingestreute Monte-Carlo-Nachkommen oberhalb von 200 Funktionsauswertungen wie eine Verringerung des Selektionsdrucks wirken. (Das Ergebnis nach tausend Funktionsauswertungen, einem Selektionsdruck von 10,0 und einem Monte-Carlo-Nachkommen ist ähnlich dem mit einem Selektionsdruck 2,5 ohne Monte-Carlos.) Werden mehr

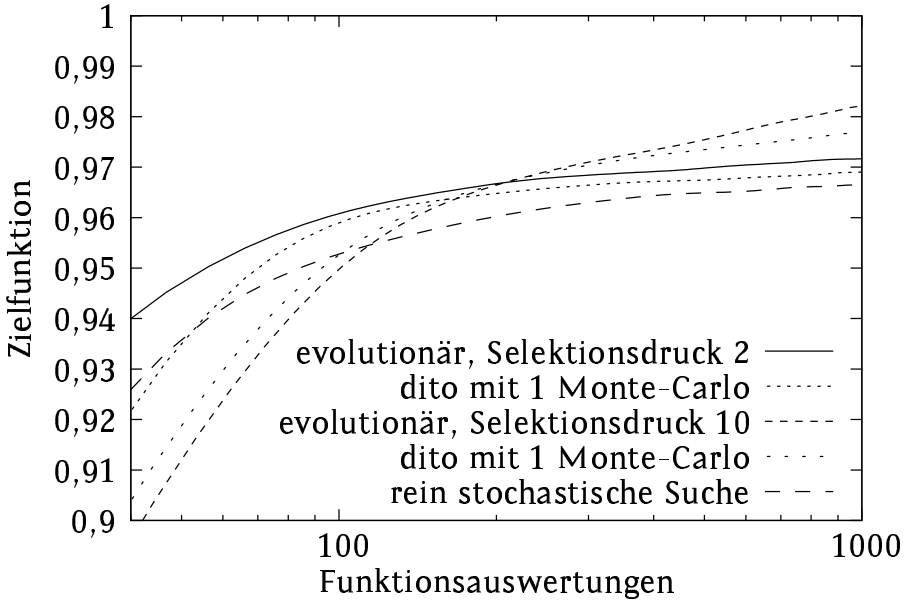

Abbildung 16: Vergleich mit stochastischer Suche

rekombinierte Nachkommen durch zufällige ersetzt, gleicht sich die Konvergenzkurve schnell jener der rein stochastischen Suche an.

5.4 Dynamische Mutationsbreite

Da sich eine dynamische Mutationsbreitenanpassung nur über einen längeren Optimierungslauf auswirken kann, wird im folgenden bis 10 000 Funktionsauswertungen gerechnet.

	Versuch	
Parameter	1	2
Populationsgröße	20	20
maximale Lebensdauer	3	3
Selektionsdruck	4	4
Monte-Carlos	0	0
Mutationsbreite	dynamisch	1,5

Tabelle 5: Versuchsbedingungen der Mutationsbreitenuntersuchung

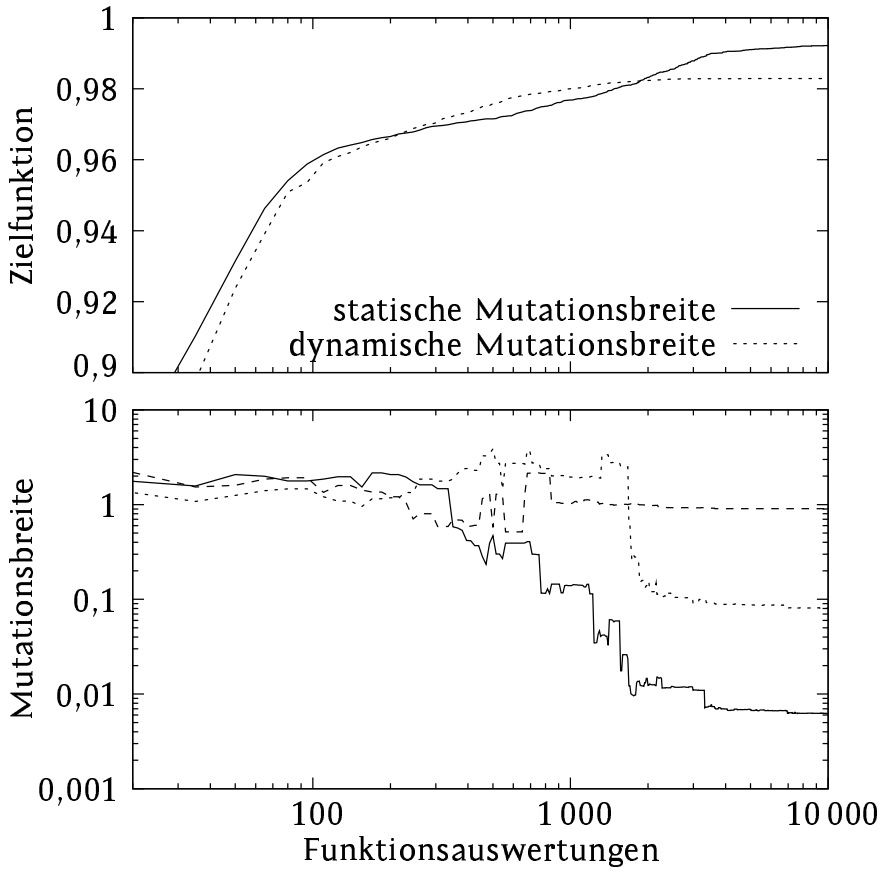

Abbildung 17: Dynamische Mutationsbreite

In Abb. 17 sind die Optimierungsläufe wieder gemittelt aufgetragen. Es ist deutlich zu erkennen, daß die Mutationsbreiten für zwei der drei Variablen ab etwa 330 bzw. 1 650 Funktionsauswertungen deutlich abnehmen. Das führt allerdings nicht zu höheren Zielfunktionswerten. Vielmehr scheint es dadurch zu einer vorzeitigen Konvergenz zu kommen, denn nach 1 650 Funktionsauswertungen ist keine merkliche Verbesserung mehr zu erkennen.

5.5 Anwendung im Steuerungsverfahren

In Abb. 18 ist der Verlauf der Qualitätsfunktion für ein Doppelpendel anfangs stehendes mit und ohne Steuerung über die Zeit aufgetragen. Die Simulation wurde mit dem in Kapitel 7 beschriebenen Programm durchgeführt. Wie zu sehen ist, kann die Steuerung den Fall des Pendels deutlich

verzögern. Gänzlich aufzuhalten vermag sie den Fall jedoch nicht. Da das Doppelpendel aber im Prinzip sogar aus der unteren Gleichgewichtslage aufgerichtet werden kann, liegt das möglicherweise an der fest vorgegebenen Steuerzeit oder auch an zu kurzen Optimierungszeiten. Für das Demonstrationsprogramm wird deshalb ein weniger kritischer Prozeß gewählt, der dann auch einen Endlosbetrieb des Programms ermöglicht.

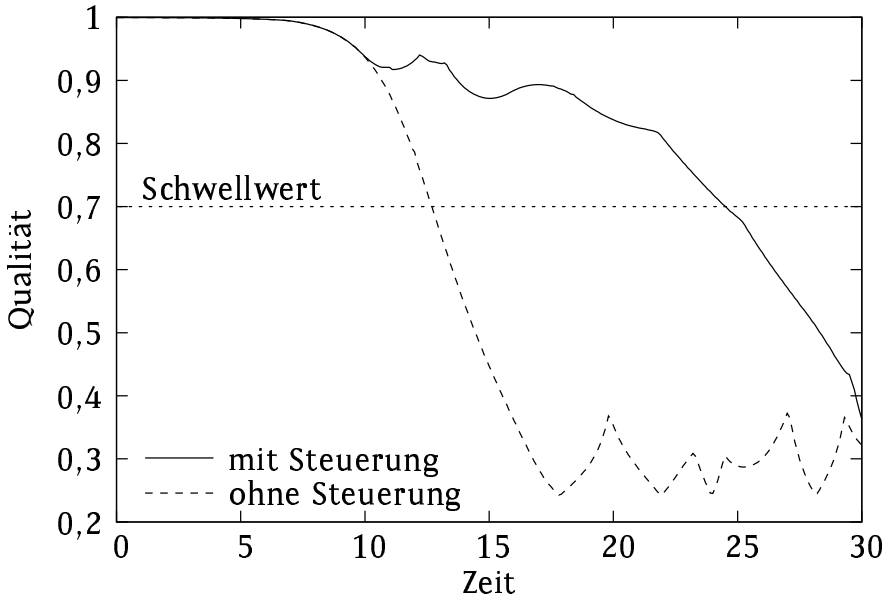

Abbildung 18: Doppelpendel mit und ohne Steuerung

6 Modellanwendung

Als Beispielanwendung soll ein Kran dienen, der eine Last befördert. Der Kran besteht aus einer Schiene und einer Laufkatze, an der an einem Seil die Last hängt. Die Last soll nun von Punkt A nach Punkt B transportiert werden.

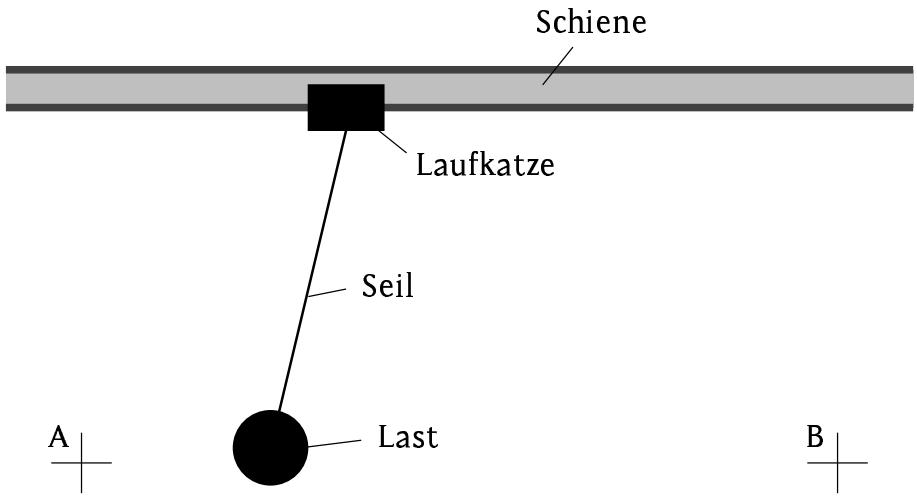

Abbildung 19: Kran

Der Kran besitzt selbst einen einfachen Regler für die Laufkatzengeschwindigkeit, der die Geschwindigkeit proportional zum Abstand vom Ziel einstellt und zusätzlich begrenzt.

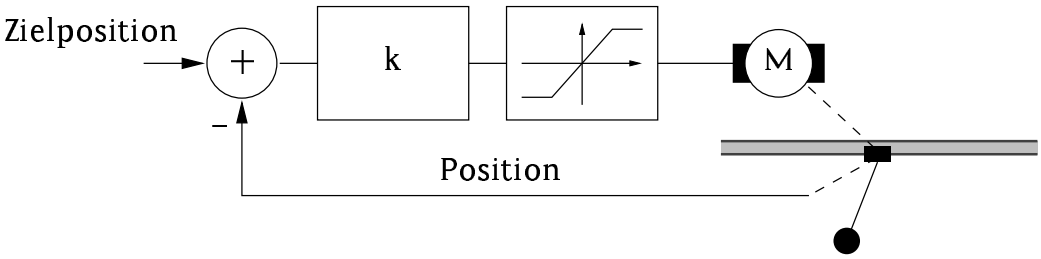

Abbildung 20: Regelkreis des Krans

Dieser einfache Regelkreis arbeitet keineswegs ideal: Er bewirkt große Schwingungsamplituden der Last und kann letztere höchstens zufällig am Zielort zum Stillstand bringen.

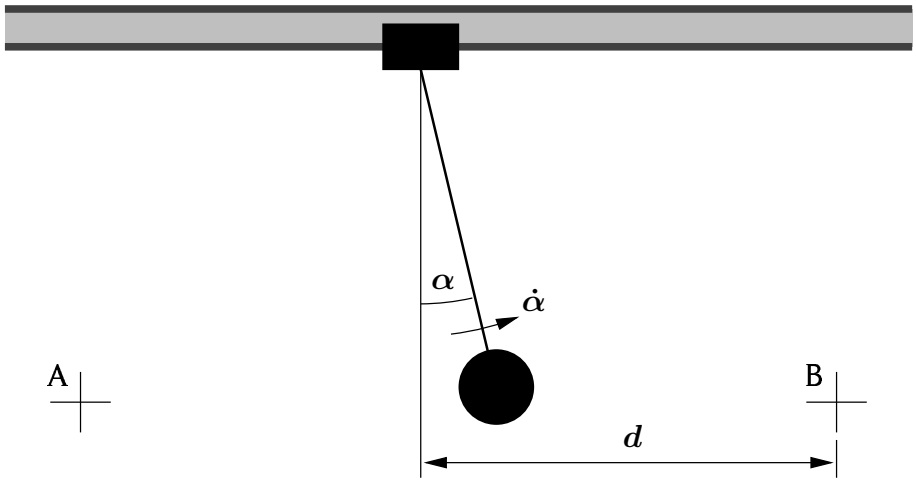

Abbildung 21: Beobachtete Zustandsgrößen des Krans

Die Aufgabe der Zusatzsteuerung besteht nun darin, die Last tatsächlich am Zielort zum Stillstand zu bringen. Um den Schwingungszustand des Seil-Last-Systems beschreiben zu können, wird dafür zusätzlich der Seilwinkel α und die Winkelgeschwindigkeit $\dot{\alpha}$ eingeführt.

Für die Optimierung der Steuertrajektorien wird noch eine Qualitäts- oder Zielfunktion benötigt, denn von dieser ist das Verhalten der Steuerung abhängig. Die Definition der Zielfunktion muß somit zum Ausdruck bringen, daß sowohl die Laufkatze über dem Ziel zum Stillstand kommen, als auch die Summe aus kinetischer und potentieller Schwingungsenergie der Last minimal werden muß.

Mit der Seillänge ℓ ist die kinetische

$$E_{\mathrm{kin}} = \frac{1}{2} m (\ell \dot{\alpha})^2 \tag{34}$$

und die potentielle Schwingungsenergie der Last

$$E_{\mathrm{pot}} = E_{\mathrm{pot},0} + g m \ell (1 - \cos \alpha) \quad . \tag{35}$$

Damit kann die Qualitätsfunktion wie folgt definiert werden:

$$Q(\alpha, \dot{\alpha}, d) \stackrel{\mathrm{def}}{=} \max \left\{ \begin{array}{l} \frac{1}{2} m (\ell \dot{\alpha})^2 + g m \ell (1 - \cos \alpha) \\ |d| \end{array} \right. \tag{36}$$

Das Optimum liegt für diese Zielfunktion bei null.

6.1 Kinematik des Krans

Die Berechnung der Bewegungsgleichung erfolgt hier nach dem Hamilton-Prinzip der kleinsten Wirkung, nach dem das Wirkungsintegral

$$\int_{t_1}^{t_2} L(t, \alpha, \dot{\alpha})\, dt \;=\; Extr. \tag{37}$$

extrem wird. Hierbei ist L die Lagrange-Funktion

$$L(t, \alpha, \dot{\alpha}) \;=\; E_{\mathrm{kin}}(t, \dot{\alpha}) - E_{\mathrm{pot}}(t, \alpha) \quad . \tag{38}$$

Das Wirkungsintegral wird extrem, wenn die Euler-Gleichung

$$\frac{\partial L}{\partial \alpha} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\alpha}} \;=\; 0 \tag{39}$$

erfüllt ist. Die potentielle Energie kann nach

$$E_{\mathrm{pot}} \;=\; \underbrace{E_{\mathrm{pot},0}}_{:=0} - g\, m\, \ell\, \cos\alpha \tag{40}$$

berechnet werden. Bei der kinetischen Energie ist hier nicht nur mit der reinen Schwingungsenergie zu rechnen, sondern auch mit dem translatorischen Anteil. Daher ist die Berechnung von E_{kin} hier etwas aufwendiger als für die Qualitätsfunktion.

$$E_{\mathrm{kin}} \;=\; \frac{1}{2}\, m\, v_m^2 \tag{41}$$

Die Geschwindigkeit v der Last hängt sowohl von der Winkelgeschwindigkeit $\dot{\alpha}$ als auch von der Laufkatzengeschwindigkeit $\dot{s}$ ab. Für die Position der Last gilt

$$\underline{\mathrm{x}}_m \;=\; \begin{pmatrix} s - \ell\, \sin\alpha \\ \ell\, \cos\alpha \end{pmatrix} \tag{42}$$

Die vektorielle Geschwindigkeit kann daraus durch eine zeitliche Ableitung gewonnen werden.

$$\begin{aligned} \underline{\mathrm{v}}_m \;&=\; \frac{d}{dt}\,\underline{\mathrm{x}}_m \\ &=\; \begin{pmatrix} \dot{s} - \ell\, \dot{\alpha}\, \cos\alpha \\ \ell\, \dot{\alpha}\, \sin\alpha \end{pmatrix} \end{aligned} \tag{43}$$

Damit kann die kinetische Energie berechnet werden.

$$\begin{aligned} E_{\text{kin}} &= \frac{1}{2} m \, \underline{\mathrm{v}}_m^{\mathrm{T}} \, \underline{\mathrm{v}}_m \\ &= \frac{1}{2} m \left[(\dot{s} - \ell \, \dot{\alpha} \, \cos\alpha)^2 + (\ell \, \dot{\alpha} \, \sin\alpha)^2 \right] \\ &= \frac{1}{2} m \, (\dot{s}^2 + \ell^2 \, \dot{\alpha}^2 - 2 \, \dot{s} \, \ell \, \dot{\alpha} \, \cos\alpha) \end{aligned} \tag{44}$$

Die Lagrange-Funktion kann jetzt direkt hingeschrieben werden:

$$L(t, \underline{\mathrm{x}}, \dot{\underline{\mathrm{x}}}) = \frac{1}{2} m \, (\dot{s}^2 + \ell^2 \, \dot{\alpha}^2 - 2 \, \dot{s} \, \ell \, \dot{\alpha} \, \cos\alpha) + g \, m \, \ell \, \cos\alpha \tag{45}$$

Für die Euler-Gleichung (39) sind nun einige Ableitungen der Lagrange-Funktion zu bilden.

$$\begin{aligned} \frac{\partial L}{\partial \alpha} &= m \, (\dot{s} \, \ell \, \dot{\alpha} \, \sin\alpha - g \, \ell \, \sin\alpha) \\ \frac{\partial L}{\partial \dot{\alpha}} &= m \, (\ell^2 \, \dot{\alpha} - \dot{s} \, \ell \, \cos\alpha) \\ \frac{d}{dt} \frac{\partial L}{\partial \dot{\alpha}} &= m \, (\ell^2 \ddot{\alpha} - \ddot{s} \, \ell \, \cos\alpha + \dot{s} \, \ell \, \dot{\alpha} \, \sin\alpha) \end{aligned}$$

Damit ist schließlich

$$\frac{\partial L}{\partial \alpha} - \frac{d}{dt} \frac{\partial L}{\partial \dot{\alpha}} = m \, (\ddot{s} \, \ell \, \cos\alpha - g \, \ell \, \sin\alpha - \ell^2 \ddot{\alpha}) = 0 \ , \tag{46}$$

woraus eine explizite Differentialgleichung für $\ddot{\alpha}$ folgt.

$$\ddot{\alpha} = \frac{1}{\ell} \, (\ddot{s} \, \cos\alpha - g \, \sin\alpha) \tag{47}$$

7 Demonstrationsprogramm

Die einzelnen Bausteine des Programms müssen weitgehend (quasi-) parallel ablaufen. Das läßt sich im Prinzip auf zweierlei Arten erreichen. Die erste Möglichkeit ist, die Programmteile wie bei einem Reißverschluß miteinander zu verzahnen. Diese Vorgehensweise führt allerdings zu kaum noch zu durchschauenden Programm-Kode. Die zweite Möglichkeit besteht darin, die Verzahnung vom Betriebssystem durchführen zu lassen. Dann können die einzelnen Programmteile ihrer Struktur entsprechend kodiert werden. Der letzteren Möglichkeit ist daher eindeutig der Vorzug zu geben. Eine Programmiersprache, die das in besonderer Weise unterstützt, ist Ada. Sie enthält spezielle Sprachkonstrukte für die Synchronisation von mehreren sogenannten *Tasks* sowie den Datenaustausch zwischen ihnen. Außerdem ist sie auch noch besonders für Echtzeitprogramme ausgelegt.

Für jeden zu überwachenden Prozeß gibt es in dem Programm einen Prädiktor, der die Simulation und ihre Synchronisierung mit dem realen Prozeß steuert. Dieser wird dazu in kleinen Zeitschritten mit einem Modell simuliert. Damit unterschiedliche Modelle von dem gleichen Prädiktor verwendet werden können, wird für alle Modelle eine einheitliche Schnittstelle definiert. Der Prädiktor verwendet von den Modellen nur diese Schnittstelle, wobei die Entscheidung, welches Modell benutzt wird, von der Ada-Laufzeitbibliothek getroffen wird (*runtime dispatching*).

Innerhalb des Prognosehorizonts werden die vorausberechneten Zustandswerte des jeweiligen Prozesses mittels einer prozeßspezifischen Qualitätsfunktion bewertet. Unterschreitet die Qualität einen Schwellwert, so wird ein Optimierer gestartet, der einen korrigierenden Steuereingriff berechnen soll. Der Optimierer benötigt allerdings als Anfangsbedingungen den Prozeßzustand einige Zeit vor dem Unterschreiten des Schwellwerts. Dieser Zustand muß also zunächst ermittelt werden. Im Programm wird das durch eine nochmalige Simulation ab dem letzten Synchronisationspunkt bis zum Startzeitpunkt des Steuereingriffs erreicht. Alternativ könnte auch während der ersten Simulation eine Historie geschrieben werden, aus der dann die benötigten Werte ablesbar wären.

Der Optimierer enthält im wesentlichen den evolutionären Algorithmus. Letzterer benötigt eine Zielfunktion zum Bewerten der einzelnen Individuen der Population. Ein Teil dieser Zielfunktion ist allgemein und daher Bestandteil des Optimierers: Für alle Zielfunktionen muß der Prozeß über den Zeitraum des Steuereingriffs einschließlich Steuerung simuliert werden. Am Ende des Steuereingriffs wird das erzielte Ergebnis mit der, schon vom Prädiktor verwendeten, Qualitätsfunktion bewertet. Der Wert der Zielfunktion für den evolutionären Algorithmus ist dann der von der Qualitätsfunktion gelieferte Wert. Im Prinzip wäre es auch möglich, den Verlauf der Zustandsgrößen während des Steuereingriffs mitzubewerten. Hierfür müßte aber in der Regel ein anderer Bewertungsmaßstab als für den Endwert angelegt werden, da während der Eingriffs auch Zustandsübergänge möglich sind. Es wäre also eine zweite Qualitätsfunktion notwendig. Im Programm wird auf diese Möglichkeit verzichtet.

Für den evolutionären Algorithmus kann im voraus keine bestimmte Anzahl durchzurechnender Generationen angegeben werden. Es steht vielmehr eine bestimmte Zeitspanne (nicht zu verwechseln mit CPU-Zeit) für die Optimierung zur Verfügung. Das letzte und beste Ergebnis des Algorithmus während dieser Zeitspanne ist auch das Endergebnis — sofern es besser als gar kein Eingriff ist. Nach Ablauf der zur Verfügung stehenden Rechenzeit wird die Optimierung abgebrochen und mit dem Ergebnis die Steuerung programmiert. Anschließend beginnt der Prädiktor wieder mit dem Vorhersagen des weiteren Zustandsverlaufs und ist dann auch wieder bereit, neue Optimierungen zu starten.

Neben den Komponenten Prädiktor und Optimierer gibt es noch die graphische Oberfläche, auf der die gerade stattfindenden Aktivitäten dargestellt werden. Jedes Programmmodul wird hier mit einem Fenster ähnlich denen der bekannten Fensteroberflächen moderner Betriebssysteme dargestellt. Die farbliche Intensität des Rahmens zeigt dabei die Aktivität des zugehörigen Moduls an: Bei einem blassen Rahmen ist das Modul inaktiv, bei einem leuchtenden Rahmen dagegen aktiv. Leuchtendes Rot zeigt einen Fehlerzustand an. Innerhalb des Rahmens erfolgt die Darstellung der wesentlichen Werte des Moduls. Nur für die Prozesse selbst muß die graphische Darstellung individuell programmiert werden; die anderen Programmmodule

kommen mit prozessunabhängigen Darstellungen aus. Da eine wesentliche Größe eines Prozesses auch die Bewertung des Zustands durch die Qualitätsfunktion ist, sollte auch der Wert dieser Funktion (z. B. in Form eines Balkens) dargestellt werden.

Die graphische Oberfläche wirft ganz eigene Probleme der Datenvorhaltung im Programm auf. Es müssen nicht nur die für den Algorithmus zu jedem Zeitpunkt wichtigen Daten gespeichert werden, sondern auch alle, die angezeigt werden. Um die Arbeit des Prädiktors in der Art wie in Abb. 1 zu darzustellen, muß z. B. einen Historie von Vorhersagewerten zur Verfügung stehen. Ebenso müssen für die Darstellung des Optimierungsfortschritts die Generationen, bei denen Verbesserungen des Ergebnisses aufgetreten sind, mit den zugehörigen Zielfunktionswerten für den gesamten bisherigen Optimierungslauf zur Verfügung stehen. Darüber hinaus hat die Anzeige ihren eigenen Takt, mit dem die Darstellungen aktualisiert werden. Die Daten werden also von der Oberfläche asynchron zum Programmablauf abgefragt. Das bedeutet, daß die Daten in speziellen Pufferspeichern abgelegt werden müssen, die vor Zugriffskonflikten geschützt sind.

Die graphische Oberfläche wird mit der Gtk-Bibliothek programmiert. Diese Bibliothek ist einerseits für verschiedene Betriebssysteme vorhanden und andererseits gibt es ein sogenanntes *Binding* für Ada.

7.1 Graphische Oberfläche

Die graphische Oberfläche dient der Visualisierung der Programmfunktion und soll den Ablauf des Programms verdeutlichen. Da das Programm vollständig von selbst abläuft, beschränkt sich die Oberfläche im wesentlichen auf eine reine Darstellung. Das einzige Bedienelement ist ein kleines Menü, mit dem die Aktualisierung der Graphiken ausgesetzt werden kann und zusätzliche Beschriftungen eingeblendet werden können.

7.1.1 Programmfenster

Das Hauptprogrammfenster besteht aus einem kleinen Menü, mit dem die wesentlichen Programmfunktionen bedient werden können, und einer großen Zeichenfläche. Die gesamte Visualisierung findet auf dieser Zeichenfläche statt. Die einzelnen Programmkomponenten werden dort — wie bei graphischen Fensteroberflächen — mit jeweils eigenen fensterartigen Blöcken gezeichnet. Untereinander werden diese Blöcke mit Pfeilen verbunden, die den Datenfluß andeuten. Bei Bedarf können zusätzliche Beschriftungen eingeblendet werden, die dann über die Zeichenfläche gelegt werden.

7.1.2 Komponentenfenster

Die Hauptkomponenten des Systems werden jeweils mit eigenen kleinen Fenstern im Hauptprogrammfenster dargestellt. Der äußere Rahmen ist dabei für alle Komponenten gleich. Er besteht aus einer schmalen farbigen Umrandung und einem Kopf, in den die Bezeichnung der Komponente geschrieben wird. Die Farbe der Umrandung einschließlich des Kopfes wird abhängig vom momentanen Zustand der Komponente gewählt.

Farbe	Bedeutung
hellblau	Die Komponente ist inaktiv bzw. in Wartestellung.
blau	Die Komponente ist aktiv.
rot	Es liegt eine Fehlersituation vor.

Tabelle 6: Rahmenfarben in der graphischen Oberfläche

Während des Programmablaufs wechselt somit die Farbigkeit der einzelnen Fensterumrandungen im Takt des Programms von einem hellen zu einem satten Blau und umgekehrt. Dadurch wird das Auge des Betrachters auf die gerade tätigen Programmteile gelenkt. Das Innere des Fensters wird abhängig von der darzustellenden Komponente gefüllt, wobei gerade inaktive Komponenten ebenfalls etwas blaß (d. h. mit grauen statt schwarzen Linien) gezeichnet werden.

Die einzelnen Komponenten erhalten für ihre Zeichenfunktionen jeweils eine eigene Zeichenfläche (Gtk-Pixmap). Dadurch sind sie vom globalen Koordinatensystem unabhängig und können im Fehlerfall auch nicht in Bereiche zeichnen, die nicht für sie vorgesehen sind. Die Inhalte der lokalen Zeichenflächen werden dann in die globale Zeichenfläche kopiert.

7.1.3 Prozeßdarstellung

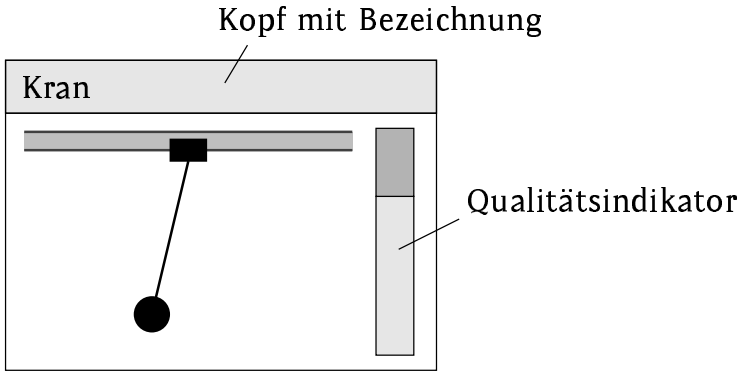

Abbildung 22: Darstellung des Krans

Die zu überwachenden Prozesse sollen charakteristisch visualisiert werden. Bei einfachen Prozessen wie dem Kran bietet sich eine stilisierte Darstellung an. Bei komplizierteren Prozessen sind gegebenenfalls weitergehende Abstraktionen notwendig, um das Fenster nicht zu überladen. Für den Kran eignet sich eine Darstellung wie in Abb. 19. Zusätzlich wird rechts neben dem Kran die aktuelle Bewertung durch die Qualitätsfunktion in Form eines Balkens angezeigt. Dieser Balken besteht aus einem blassgrünen und einem leuchtend roten Bereich. Je kleiner der von der Qualitätsfunktion gelieferte Wert ist, umso mehr verdrängt der leuchtend rote Bereich den unauffälligeren blassgrünen und lenkt so die Aufmerksamkeit des Betrachters auf sich.

7.1.4 Prädiktordarstellung

Der Prädiktor macht Voraussagen über den zukünftigen Verlauf der Zustandsgrößen des zugeordneten Prozesses. Im Prinzip könnten im Prädiktorfenster

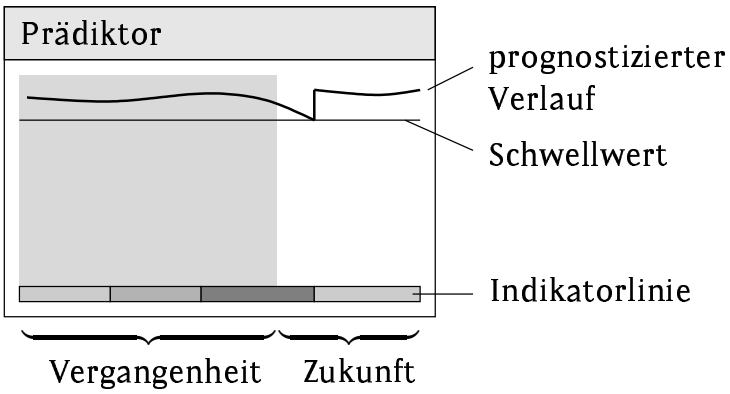

Abbildung 23: Darstellung des Prädiktors

diese vorausberechneten Größen in Form eines Funktionsplots gezeichnet werden. Da die Grafik nicht statisch ist, sondern sich im Laufe der Zeit immer wieder ändert, wird sie jedoch in der Praxis schon bei wenigen Zustandsgrößen so unübersichtlich, daß kaum noch etwas zu erkennen ist. Daher wird an Stelle der Zustandsgrößen nur die davon abgeleitete Qualität dargestellt. Der Graph wird dadurch auf eine einzige Kurve reduziert, deren Verlauf leicht mit dem Balken in der Prozeßdarstellung verglichen werden kann.

Die Qualitätskurve wird über ein Zeitintervall, das ein Stück weit in die Vergangenheit bis zum Ende des Prädiktionshorizonts reicht, gezeichnet. Damit der Betrachter erkennen kann, an welcher Stelle des Graphen sich die Gegenwart augenblicklich befindet, wird der bereits in der Verangenheit liegende Teil blau schattiert. Rechts von dem schattierten Teil befinden sich dann der prognostizierte zukünftige Verlauf des Qualitätswerts.

In den Graph wird außerdem noch als dünne Linie der Schwellwert eingezeichnet, bei deren Unterschreitung der Prädiktor den Optimierer startet. Bleiben die Qualitätswerte oberhalb dieser Linie, so wird der Prädiktor nur gelegentlich, nämlich an den Synchronisationspunkten, aktiv. Das ist dann an einer kurzen Farbveränderung des Rahmens und dem Weiterspringen des Prognosegraphen zu erkennen.

Da in dem Prädiktorfenster ein zeitlicher Verlauf angezeigt wird, ist es auch möglich, neben dem aktuellen Prädiktorzustand noch die Aktivitäten im be-

Farbe	Bedeutung
grün	Die Progose ist oberhalb des Schwellwerts. Alles ist im grünen Bereich.
orange	Es wurde eine Unterschreitung des Schwellwerts prognostiziert. Der Optimierer läuft.
rot	Der vom Optimierer berechnete Steuereingriff wird auf den Prozeß aufgebracht.

Tabelle 7: Farben der Indikatorlinie des Prädiktors

trachteten Zeitintervall darzustellen. Dies geschieht über eine farbige Indikatorlinie unterhalb des Graphen. Diese Linie kann drei Farben annehmen (siehe Tab. 7).

In Abb. 23 fällt ein Sprung in der prognostizierten Qualität auf. Dieser kommt dadurch zustande, daß am Ende des zuvor berechnetete Steuereingriffs der Wert der Qualitätsfunktion höher ist, als er ohne den Eingriff wäre.

7.1.5 Optimiererdarstellung

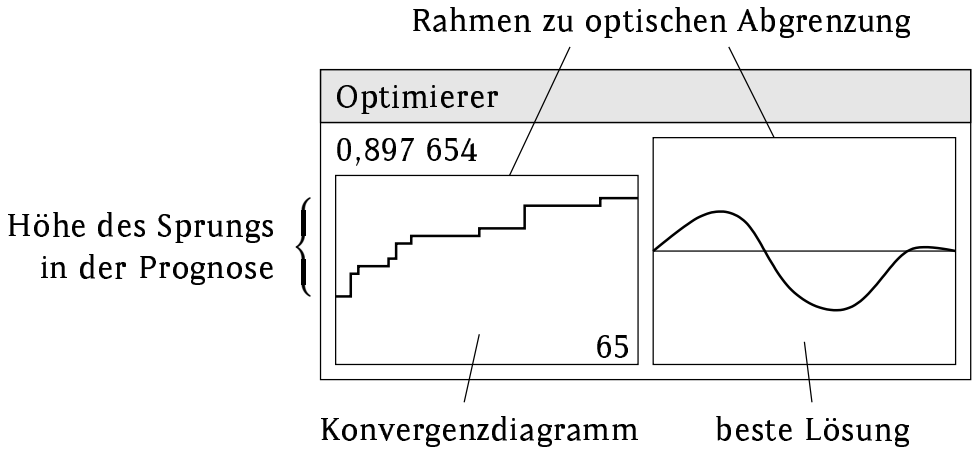

Abbildung 24: Darstellung des Optimierers

In dem Fenster des Optimierers weden die Fortschritte des evolutionären Algorithmus dargestellt. Dafür wird dieses Fenster horizontal in zwei Hälften unterteilt. In der linke Hälfte wird ein Konvergenzdiagramm gezeichnet,

in dem die erreichten Werte der Zielfunktion über die Generationen aufgetragen werden. Die Anzeige wird dahingehend normiert, daß die Abszisse immer den Bereich bis zur letzten Generation mit Verbesserung überstreicht. Bei jeder neuen Verbesserung wird der Graph dadurch nach links gestaucht und ein neues Stück rechts angefügt. Dadurch wird der Anzeigebereich unabhängig von der tatsächlich gerechneten Anzahl Generationen optimal genutzt. Über dem Graphen wird der beste Zielfunktionswert noch einmal als Zahl hingeschrieben. Unten rechts im Graph wird außerdem die Generation, in der die letzte Verbesserung stattgefunden hat, notiert.

In der rechten Hälfte wird die beste im Optimierungslauf gefundene Steuertrajektorie eingezeichnet. Diese Anzeige ändert sich je nach Optimierungsverlauf manchmal graduell, manchmal sprunghaft, zeitweilig aber auch überhaupt nicht.

7.1.6 Steuerungsdarstellung

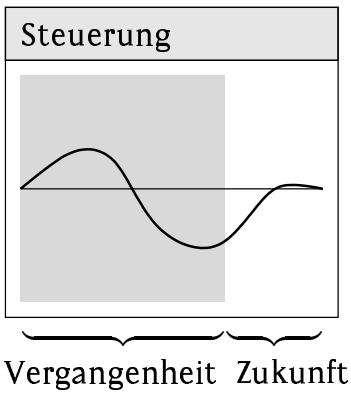

Abbildung 25: Darstellung der Steuerung

Die vom Optimierer ermittelte Steuertrajektorie wird von der Steuerung auf den Prozeß übertragen. Dieser Vorgang wird im Steuerungsfenster dargestellt. Dazu wird die Trajektorie als Graph über die Zeit gezeichnet. Außerdem wird die seit Beginn der Steuerung verstrichene Zeitspanne — wie schon in der Darstellung des Prädiktors — blau schattiert. Auf diese Weise kann der Betrachter den Vorgang der Übertragung verfolgen.

7.1.7 Verbindungen

Neben den Komponentendarstellungen gibt es auch noch Anzeigen, die auf der globalen Zeichenfläche dargestellt werden. Dazu gehören die Pfeilverbindungen, mit denen der Datenfluß angedeutet wird. Die Pfeilkomponenten zeichnen nur einen durch eine Punktfolge definierten Pfeil auf die globale Zeichenfläche.

7.1.8 Zusatzbeschriftungen

Die zusätzlich einblendbaren, gelb unterlegten Beschriftungen müssen, wie die Pfeile auch, auf der globale Zeichenfläche angebracht werden. Sie werden als letztes gezeichnet, damit sie immer im Vordergrund stehen.

Abb. 26 zeigt ein Bildschirmfoto des Demonstrationsprogramms. Die Beschriftungen auf gelbem Grund lassen sich ein- und ausblenden.

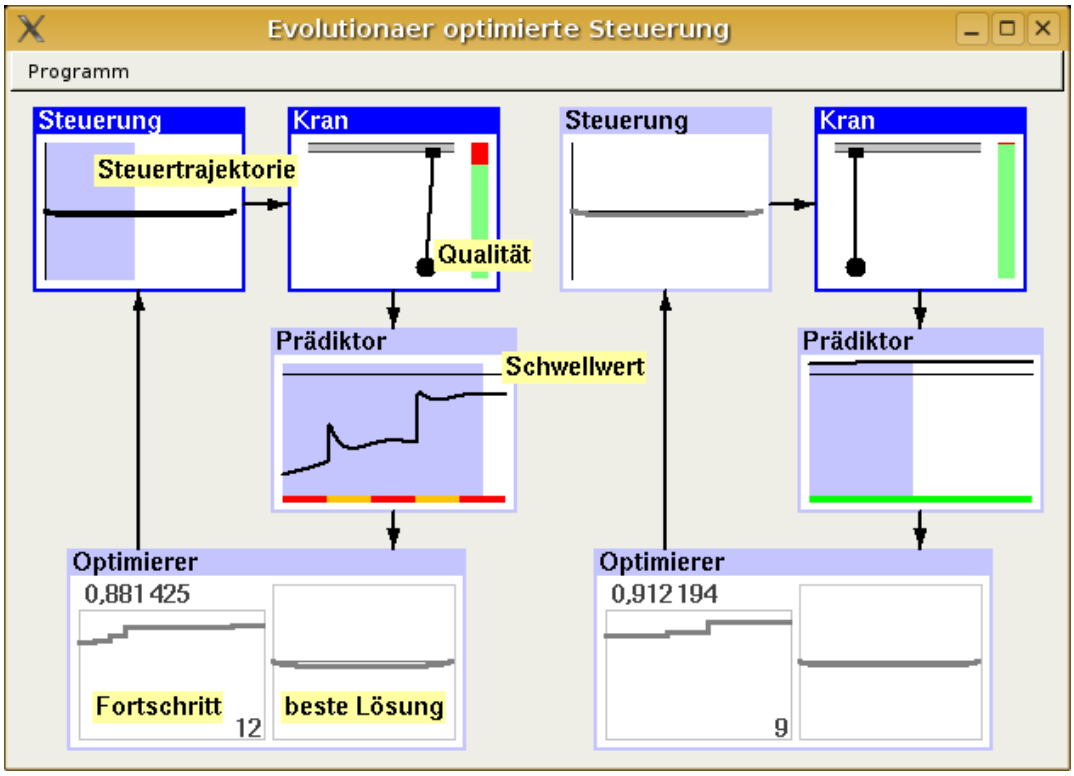

Abbildung 26: Bildschirmfoto

7.2 Programmstruktur

Abb. 27 zeigt die Abhängigkeiten der Hauptkomponenten des Programms. Ein Pfeil bedeutet darin „ist abhängig von“.

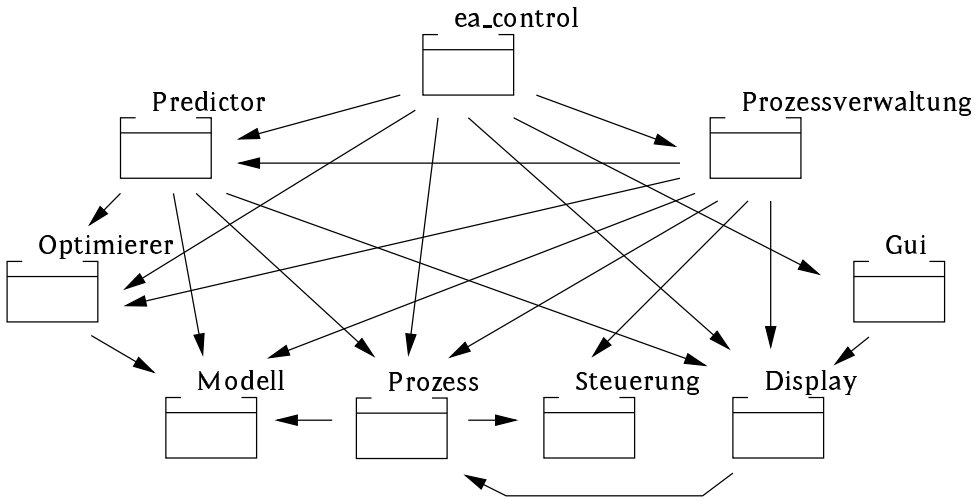

Abbildung 27: Programmstruktur

7.2.1 Wesentliche Programmfunktion

Im Hauptprogramm werden die Verbindungen zwischen den einzelnen Komponenten aufgebaut. Dafür werden einheitliche Schnittstellen für die die einzelnen Prozesse betreffenden Module benötigt. Das führt zu einem objektorientierten Ansatz. Im Programm werden für alle Prozesse dieselben Schnittstellen benutzt. Nur während der Initialisierung müssen — zum Erzeugen der einzelnen Objekte — die prozeßspezifischen Module direkt verwendet werden.

Nach dem Aufbau der Struktur startet das Hauptprogramm die Prädiktoren und den Task mit der Hauptschleife der graphischen Oberfläche. Die Prädiktoren sind dabei Objekte von Task-Typen. Dadurch ist es möglich, für jeden Prozeß einen eigenen Prädiktor anzulegen. Die parallel laufenden Prädiktoren übernehmen nun die Programmsteuerung. In Abb. 28 ist das zugehörige Ablaufdiagramm dargestellt. Die Prädiktoren fragen in regelmäßigen zeitlichen Abständen über Funktionen der Prozeßmodule deren Zustandsgrößen

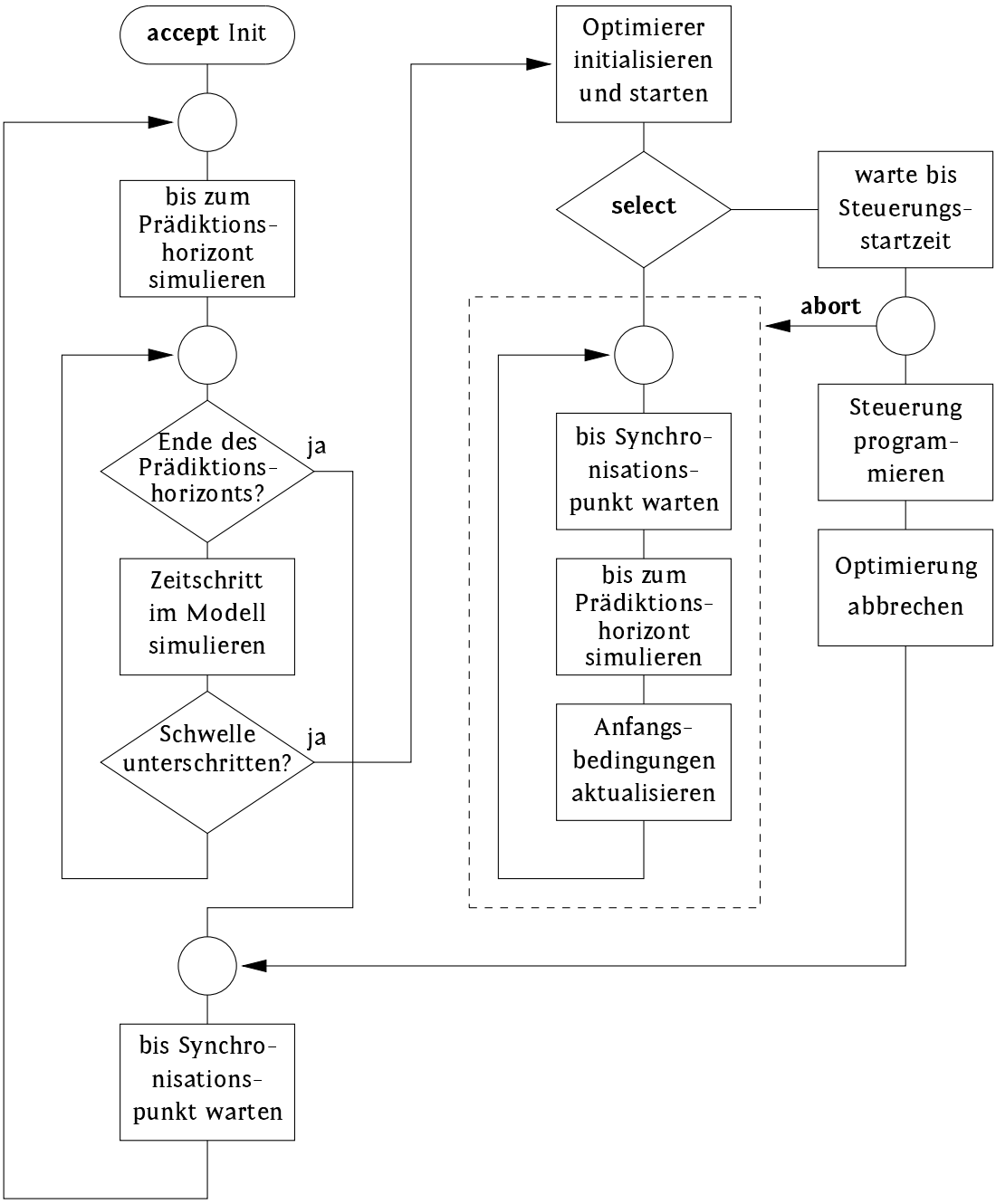

Abbildung 28: Ablaufdiagramm des Prädiktors

ab, die sie als Anfangswerte für die Simulationen verwenden. Die Simulationen werden daraufhin schrittweise mit Funktionen der prozeßspezifischen Modelle durchgeführt. Da der Zugriff auf diese Funktionen aber nur über die gemeinsamen Schnittstellen erfolgt, kann ein Prädiktor unterschliedlichste Prozesse überwachen. Unterschreitet während den Simulationen ei-

ne berechnete Qualität den prozeßspezifischen Schwellwert, so muß eine Steuertrajektorie berechnet werden. Dafür legt der betroffene Prädiktor ein Optimierer-Objekt an, initialisiert und startet es. Während der Optimierer läuft, führt der Prädiktor wieder in regelmäßigen Abständen Simulationen durch, um bessere Anfangsbedingungen für die Optimierung zu erhalten. Diese werden umgehend an den Optimierer weitergereicht. Daneben läuft im Prädiktor noch eine Uhr, die den Abbruch der Optimierung auslöst. Nach Ablauf der zur Verfügung stehenden Optimierungszeit wird das beste gefundene Ergebnis genommen und damit die zum Prozeß gehörende Steuerung programmiert. Das Optimierungsergebnis muß zu diesem Zweck asynchron zum Ablauf des Optimierers aus diesem ausgelesen werden können. Für diesen Datenaustausch wird ein Objekt eines geschützen Datentyps (*protected type*) verwendet. Bei solchen Objekten schließt ein Schreibzugriff jeden anderen Zugriff aus. Der Optimierer schreibt nun jedes verbesserte Ergebnis sofort in dieses Objekt, daß am Ende vom Prädiktor ausgelesen wird. Sollte während eines Schreibzugriffs ein Lesezugriff aufkommenden, so erfolgt dieser erst im Anschluß an den Schreibzugriff. Umgekehrt schließt ein Lesezugriff auch einen gleichzeitigen Schreibzugriff aus. Auf diese Weise ist sichergestellt, daß das gelesene Datum eines solchen geschützen Objekts keine undefinierten Zustände annehmen kann. (Geschützte Objekte werden daher ebenfalls benutzt, um Daten für die graphische Oberfläche vorzuhalten.) Sobald die Steuerung programmiert ist, wird der Optimierer gestoppt und die Schleife des Prädiktors beginnt wieder von vorne.

7.2.2 Graphische Oberfläche

Zu Beginn muß der Oberfläche mitgeteilt werden, welche Komponenten wo auf der Zeichenfläche dargestellt werden sollen. Zu diesem Zweck können die Komponenten einzeln nacheinander registriert werden. Dabei werden zwei Arten von Komponenten unterschieden: Die Funktionsblöcke, die Verbindungspfeile und die bei Bedarf einblendbaren Beschriftungen. Bei der Registrierung werden die Komponenten deshalb in zwei verschiedenen Liste gespeichert, die dann später abgearbeitet werden können.

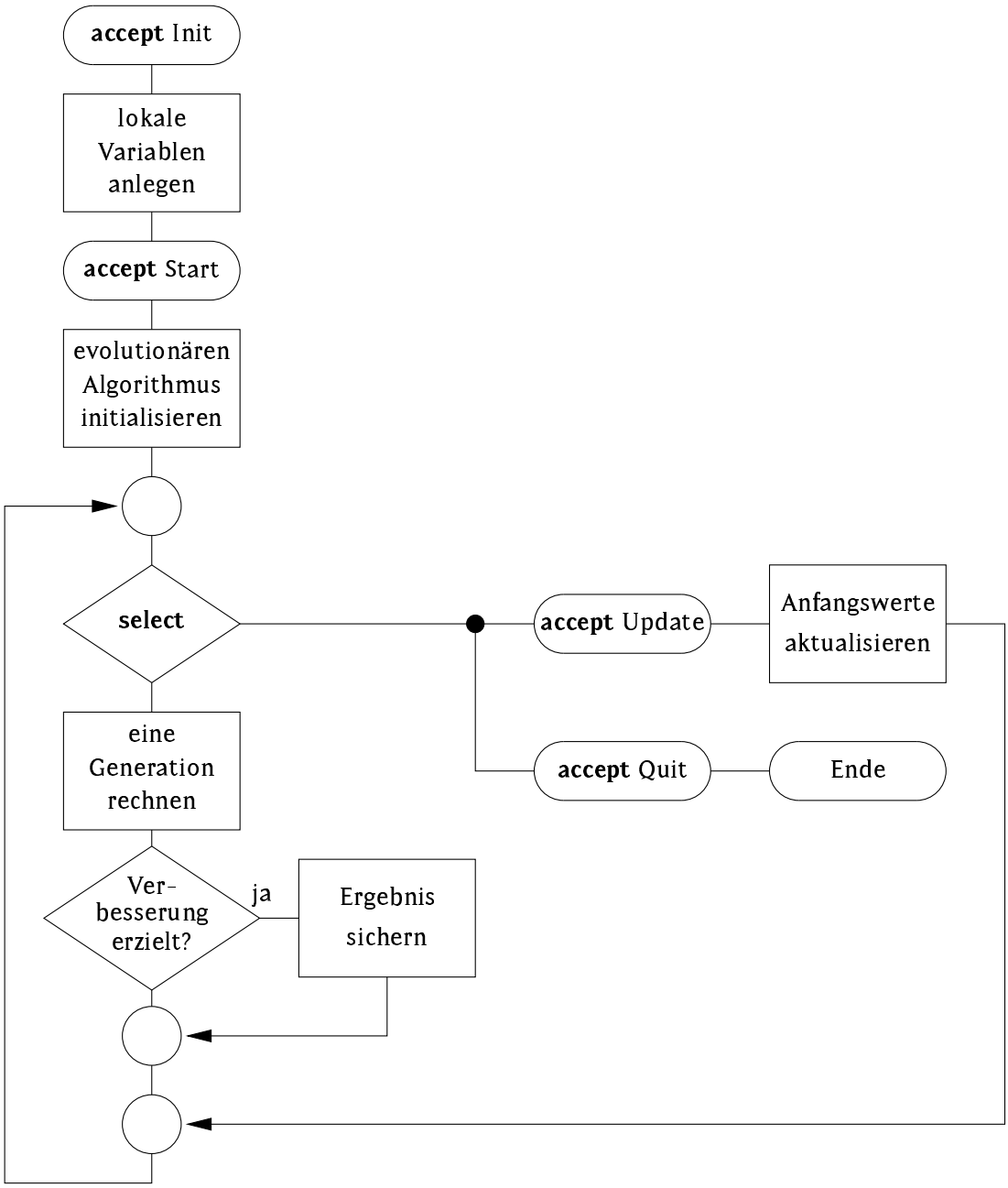

Abbildung 29: Ablaufdiagramm des Optimierers

Die graphische Oberfläche ist mit dem *Graphic Tool Kit* (Gtk) programmiert. Die Oberfläche benötigt im Programm einen eigenen Task, um unabhänig vom Ablauf des restlichen Programms zu sein. Die Gtk-Bibliothek hat nämlich ihre eigene Hauptschleife, die sich nur um die Reaktion auf Ereignisse kümmert. In der Gtk-Bibliothek ist alles ereignisorientiert. Was immer passieren soll, muß von einem Ereignis ausgelöst werden.

Das Verfahren der Zusatzsteuerung (speziell der Prädiktor und der Optimierer) würde zwar einige Ereignisse zur Aktualisierung der Darstellung liefern, für die Prozeßvisualisierung steht dagegen kein auslösendes Ereignis zur Verfügung. Deshalb wird eine konstante Aktualisierungsrate wie bei einem Film verwendet. Dazu muß in regelmäßigen zeitlichen Abständen ein Ereignis generiert werden, bei dessen Bearbeitung die Komponenten neu gezeichnet werden. Die Gtk-Bibliothek stellt speziell für diesen Zweck einen Timeout-Generator zur Verfügung. Dieser kann so programmiert werden, daß er regelmäßig ein beliebiges Ereignis auslöst. Der Generator wird daher so eingestellt, daß ein Ereignis zum Neuzeichnen generiert wird. Dann wird jedesmal die Zeichenroutine für die globale Zeichenfläche aufgerufen. Diese Zeichenroutine sorgt nun dafür, daß die einzelnen Fenster der Funktionsblöcke aktualisiert werden. Da die Komponenten in einer Liste gespeichert sind, braucht nur für jedes Element dieser Liste die entsprechende Komponenten-Zeichenroutine aufgerufen zu werden. Auf diese wird auch, wie schon die Prozeß- und Modellfunktionen, über eine einheitliche Schnittstelle zugegriffen. Damit ist die Zeichenroutine unabhängig von den konkreten Komponenten. Der Rahmen jeder Komponente wird anschließend in der Haupt-Zeichenroutine aktualisiert, da er ja immer gleich aussehen soll. Außerdem kann dann auch auf Fehlersituation in einer Komponenten-Zeichenroutine reagiert werden, indem der Rahmen eine andere Farbe erhält. Anschließend wird die zweite Liste mit den Zusatzkomponenten durchgegangen.

7.3 Einbinden von Prozessen

Um einen Prozess in das System einzubinden, müssen die abstrakten Spezifikationen in den Paketen Prozess, Modell und Display ausprogrammiert werden. Nach der Initialisierung kommuniziert das System ausschließlich über diese Schnittstellen mit den Prozessen. Die abstrakten Deklarationen sind im folgenden mit kurzen Erläuterungen aufgeführt.

package Prozess ist die Prozeßschnittstelle.

function Get_Modell (Item: Object) return Modell.Link liefert das zum

Prozeß gehörende Modell.

function Get_Steuerung (Item: Object) return Steuerung.Link liefert die zum Prozeß gehörende Steuerung.

function Get_Input_Dimension (Item: Object) return Natural liefert die Anzahl der Eingangsgrößen des Prozesses.

function Get_Zustand_Dimension (Item: Object) return Positive liefert die Anzahl der Zustandsgrößen des Prozesses.

function Get_Output_Dimension (Item: Object) return Natural liefert die Anzahl der Ausgangsgrößen des Prozesses.

function Control_DOF (Item: Object) return Natural liefert die Anzahl der Freiheitsgrade einer Steuertrajektorie.

function Control_Duration (Item: Object) return Time_Span liefert die Dauer eines Steuerungseingiffs.

function Get_Steuerung_Dimension (Item: Object) return Natural liefert die Anzahl der Steuereingänge.

procedure Set_Steuerung (Item: in out Object; Start_Time: Time; Dauer: Time_Span; Definition: Vector) übernimmt die Steuertrajektorie und gibt sie an die Steuerung weiter.

function Get_Input (Item: Object) return Vector liefert die Werte der Eingangsgrößen des Prozesses.

function Get_Zustand (Item: Object) return Vector liefert die Werte der Zustandsgrößen des Prozesses.

function Get_Output (Item: Object) return Vector liefert die Werte der Ausgangsgrößen des Prozesses.

package Modell ist die Modell- und Simulationsschnittstelle.

function Get_Input_Dimension (Item: Object) return Natural liefert die Anzahl der Eingangsgrößen des Modells.

function Get_Zustand_Dimension (Item: Object) return Positive liefert die Anzahl der Zustandsgrößen des Modells.

function Get_Output_Dimension (Item: Object) return Natural liefert die Anzahl der Ausgangsgrößen des Modells.

procedure Set_Input (Item: in out Object; Value: Vector) setzt die Eingangsgrößen des Simulation auf die angegebenen Werte.

function Get_Zustand (Item: Object) return Vector liefert die Werte der Zustandsgrößen der Simulation.

function Get_Output (Item: Object) return Vector liefert die Werte der Ausgangsgrößen des Simulation.

procedure Set_Time (Item: in out Object; Value: Time) setzt die aktuelle Zeit der Simulation.

function Get_Time (Item: Object) return Time liefert die aktuelle Zeit der Simulation.

procedure Time_Step (Item: in out Object; Delta_T: Time_Span) führt einen Zeitschritt der Simulation aus.

function Quality (Item: Object; State: Vector) return Float berechnet den Wert der Qualitätsfunktion für den angegebenen Zustand.

function Quality_Limit (Item: Object) return Float liefert den Schwellwert für die Qualität, bei dem ein Eingriff erfolgen soll.

procedure Set_Steuerung (Item: in out Object; Start_Time: Time; Dauer: Time_Span; Definition: Vector) übernimmt die angegebene Steuertrajektorie in die Simulation.

function Steuerung_Value (Item: Object; T: Time) return Vector liefert den Wert der Steuertrajektorie am angegebenen Zeitpunkt.

package Display ist die Schnittstelle für die graphische Anzeige.

procedure Draw (Item: Object; Background, Foreground: Gdk.Color.Gdk_Color; State: out Block_State) zeichnet die graphische Darstellung des Prozesses.

Abb. 30 zeigt die Situation bei einem realen Prozeß. der Prozeß selbst ist hier als ein regelungstechnisches Blockschaltbild stilisiert. Das bedeutet aber nicht, daß nur lineare Systeme unterstützt werden können. Der Steuerungskasten ist ein Gerät, daß die ermittelte Steuerungstrajektorie in Echtzeit

abspulen kann, nachdem sie vorher über die Steuerungsschnittstelle übertragen wurde. Da die diese Übertragung eine kurze Zeit vor dem eigentlichen Steuerungsbeginn stattfinden kann, braucht der Übertragungsweg zur Steuerung nicht echtzeitfähig zu sein.

Für einen simulierten Prozeß stützt sich das Prozeß-Modul auf das Modell, da es keinen wirklichen Prozeß im Hintergrund gibt. Das Prozeßmodul benötigt dann einen eigenen Task, der eine Simulation mit einer Kopie des Modells durchführt.

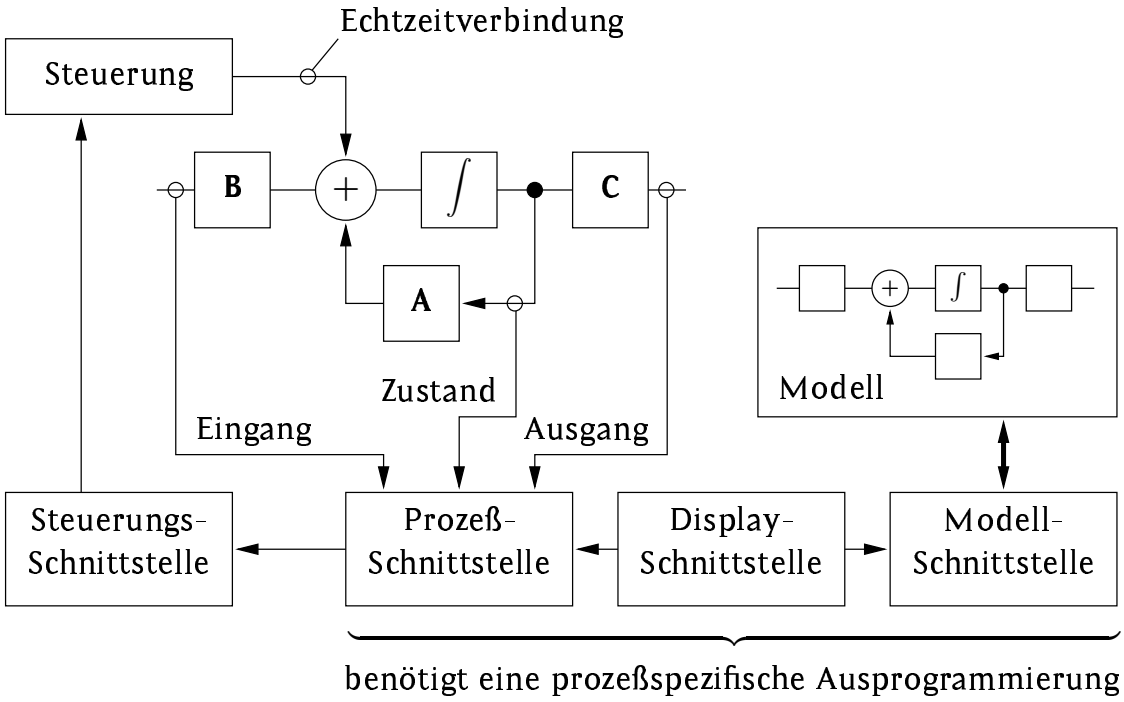

Abbildung 30: Schnittstelle bei einem realen Prozeß

8 Ausblick

Das Verfahren ließe sich sicherlich noch verbessern, indem die Dauer der Steuertrajektorien nicht konstant gewählt würde, sondern vom evolutionären Algorithmus mitoptimiert würde. Bei den Versuchen mit dem Doppelpendel wurde dies gemacht und es hat sich gezeigt, daß immer die gleiche Dauer herauskam. In unterschiedlichen Situationen kann die benötigte Zeit aber durchaus verschieden sein. In gewissen Grenzen ist das Verfahren in dieser Hinsicht ausbaubar. Allerdings wird immer ein zulässiges Intervall für die Steuerungsdauer vorgegeben werden müssen.

Das beschriebene Verfahren hängt sehr davon ab, ob für die zu unterstützenden Prozesse digitale Modelle vorhanden sind. Gerade in diesem Bereich gibt es zur Zeit durch die Einführung der sogenannten digitalen Fabrik eine Umwälzung. In Zukunft werden vermehrt standardisierte maschinenlesbare Beschreibungen für einzelne Komponenten wie auch für daraus zusammengesetzte Baugruppen Einzug halten. Dadurch wird der Einsatz des Verfahrens wesentlich erleichtert. Bei einheitlichen digitalen Prozeßbeschreibungen müssen die benötigten Modelle nicht von Hand programmiert werden, sondern können direkt von den digitalen Beschreibungen abgeleitet werden.

Ein ganz anderer Aspekt ist der, daß das Verfahren keine vorgefertigten Lösungen verwendet, sondern für aufkommende Probleme selbstständig individuelle Lösungen sucht. Das Verfahren arbeitet auch mit Systemen, für die keine vorgefertigten Handlungsanweisungen aufgestellt werden können. Ein Beispiel könnte das Gehen oder Laufen der jüngst populär gewordenen humanoiden Roboter sein, für die es kaum möglich sein dürfte, Bewegungsabläufe zur Fortbewegung in unterschiedlichstem und unvorhersehbarem Gelände fest zu programmieren. Das Verfahren könnte hierfür einen Ansatz zur Selbstprogrammierung bieten.

Gerade im Bereich autonomer Systeme kann das Verfahren vorteilhaft unterstützend eingesetzt werden. Es kann zwar nicht garantieren, eine Lösung zu finden oder zu verbessern, aber die Chancen stehen nicht schlecht. Auf das innewohnende Potential sollte daher nicht verzichtet werden.

A Verwenden von Ada-Funktionen in Octave

Einige der evolutionären Berechnungen wurden mit der GEATbx-Toolbox durchgeführt, die eigentlich für Matlab ausgelegt ist. Mit einigen kleineren Modifikationen ist sie jedoch auch mit dem freien Programm Octave zu verwenden. Für Oktave gibt es eine C++-Bibliothek, mit deren Hilfe man eigene Funktionen kompilieren kann. Da Ada ein Interface zu C besitzt, können im Prinzip auch Ada-Programme als Funktionen aus Oktave heraus aufgerufen werden. Das zu Octave gehörende spezielle Übersetzungsprogramm kann allerdings leider keine Ada-Quellen verarbeiten. Mit einigem Aufwand ist es aber trotzdem möglich.

Die Vorgehensweise soll hier an einem einfachen, aber vollständigen Beispiel gezeigt werden. Die zu importierende Funktion berechnet einfach das Quadrat einer Zahl.

Für die in Octave zu importierende Ada-Funktion muß nun ein Paket (hier ein Funktionsmodul) angelegt werden, da das Export-Pragma nur nach einer Spezifikation stehen darf. In der Paket-Spezifikation kann dann die Funktionsspezifikation nebst dem Export-Pragma untergebracht werden. Der Exportbezeichner der exportierten Funktion muß dabei einer bestimmten Namenskonvention genügen. Im Exportbezeichner tauchen neben der Länge des Namens auch noch die Typen der übergebenen Parameter auf. Im Beispiel wird an den eigentlichen Namen ein d angehängt, welches für *double* steht. Zeiger und Felder (*pointer*) werden mit einem vorangestellten P gekennzeichnet. Der Rückgabetyp taucht im Exportbezeichner nicht auf.

Datentyp	Schlüssel
double	d
double[]	Pd
int	i
void	v

Tabelle 8: Schlüssel für wichtige Argumenttypen in Octave

adasqr_fm.ads

```
1  with Interfaces.C;      use Interfaces.C;
2
3  package Adasqr_FM is
4
5     function Adasqr (Arg: Double) return Double;
6     pragma EXPORT (C, Adasqr, "_Z6adasqrd");
7     -- Konvention fuer Octave-Symbolnamen:
8     -- "_Z" & Namenslaenge & Name & Argumenttypen
9
10 end Adasqr_FM;
```

Die eigentliche Funktion ist dann im Paketrumpf definiert.

adasqr_fm.adb

```
1  package body Adasqr_FM is
2
3     function Adasqr (Arg: Double) return Double is
4     begin
5        return Arg ** 2;
6     end Adasqr;
7
8  end Adasqr_FM;
```

Die Ada-Funktionen kann in Octave nicht direkt importiert werden. Statt dessen ist ein Umweg über eine C++-Funktion (*Binding*) notwendig. Die Definition der Funktion in diesem Binding erfolgt über ein Makro namens `DEFUN_DLD`. Daher läßt sich diese Funktion kaum in Ada nachbilden. Außerdem müssen neben der eigentlichen Ada-Funktion auch noch die Ada-Initialisierungs- und Ada-Finalisierungsfunktion in der C++-Funktion aufgerufen werden.

adasqr.cc

```
1  #include <octave/oct.h>
2  #include "adasqr_fm.h"
3
```

```
4  DEFUN_DLD (adasqr, args, ,
5             "The 'Ada sqare function'.")
6  {
7    ColumnVector dx (1);
8    ColumnVector x (args(0).vector_value ());
9
10   adasqr_fm_init ();          // Ada — Initialisierung
11   dx(0) = adasqr (x(0));      // Ada — Funktionsaufruf
12   adasqr_fm_final ();         // Ada — Finalisierung
13
14   return octave_value (dx);
15 }
```

Diese Funktion benötigt noch die Spezifikationen der importierten Ada-Funktionen in der Header-Datei `adasqr_fm.h`. In dieser sind neben der eigentlichen Funktion auch die Initialisierungs- und die Finalisierungsfunktion aufgeführt.

adasqr_fm.h

```
1 extern void adasqr_fm_init (void);
2 extern void adasqr_fm_final (void);
3 extern double adasqr (double);
```

Die beiden Funktionen `adasqr_fm_init` und `adasqr_fm_final` werden vom Ada-Kompiler Gnat auf Anforderung automatisch generiert.

Schließlich muß, um die Funktion für Octave verfügbar zu machen, noch eine `.oct`-Datei erzeugt werden. Hierfür gibt es in der Octave-Distribution ein spezielles Skript namens `mkoctfile`. Vorab ist allerdings das Ada-Paket mit einem Ada-Kompiler zu übersetzen, da das Skript keine Ada-Dateien verarbeiten kann. Außerdem sind die Initialisierungs- und Finalisierungsfunktion zu generieren und zu übersetzen, damit diese im Binding aufgerufen werden können.

In der resultierenden Octave-Bibliothek muß laut der Gnat-Bedienungsanleitung die gesamte Ada-Laufzeitbibliothek statisch eingebunden sein.

Das heißt, daß die Objekt-Archive des Kompilers ausgepackt und die einzelnen Objekt-Dateien mit verlinkt werden müssen.

Anschließend kann mit dem Skript die C++-Funktion übersetzt und dabei gleich mit der Objekt-Datei der Ada-Funktion verlinkt werden. Beim Linken fällt auf, daß noch eine Funktion `__dummy` fehlt. Es scheint sich hierbei um eine wirkungslose Platzhalterfunktion zu handeln. Daher wird eine Funktion mit diesem Namen in `dummy.c` bereitgestellt und kompiliert.

dummy.c

```
1 void __dummy () {}
```

Im angebenen `Makefile` sind die einzelnen Schritte explizit aufgeführt. Es generiert eine Datei namens `adasqr.oct`. Der Name der `.oct`-Datei wird dabei von dem Übersetzungsskript `mkoctfile` aus dem ersten angegebenen Dateinamen abgeleitet.

Makefile

```
1  .PHONY: default clean distclean
2
3  default:
4          # siehe info gnat_ug: GNAT and Libraries ->
5          #    Creating an Ada Library to be Used in a Non-Ada Context
6          # (hierbei alles --- auch die Ada-Bibliotheken --- neu
7          # übersetzen und lokale Objekt-Dateien erzeugen)
8          gnatmake -c -a -f adasqr_fm
9          # generiere Initialisierungs- und Finalisierungsfunktionen
10         gnatbind -n -static -Ladasqr_fm_ adasqr_fm
11         # justiere Exportnamen
12         sed -i -e 's/"adasqr_fm_final"/"_Z15adasqr_fm_finalv"/' -e 's/"
               adasqr_fm_init"/"_Z14adasqr_fm_initv"/' b~adasqr_fm.ads
13         # und kompiliere
14         gnatmake -u b~adasqr_fm.adb
15         # erzeuge fehlendes "__dummy"
16         gcc -c dummy.c
17         # entpacke Ada-Laufzeitbibliothek
18         ar -x /usr/lib/gcc-lib/i486-linux/2.8.1/adalib/libgnat.a
```

```
19          ar -x /usr/lib/gcc-lib/i486-linux/2.8.1/adalib/libgnarl.a
20          # generiere .oct-Datei
21          mkoctfile -v adasqr.cc *.o
22
23  clean:
24          @- rm b~*.ad? *.o *.ali 2> /dev/null; true
25
26  distclean: clean
27          @- rm *.oct 2> /dev/null; true
```

Aus Octave heraus kann die Funktion nun, wie andere Funktionen auch, direkt aufgerufen werden:

```
1  octave:1> adasqr ([-1.4142136])
2  ans = 2.0000
3  octave:2>
```

B Glossar

Abschneideselektion. Selektionsverfahren, bei dem die Indiviuduen nach ihrem Rang sortiert werden und eine Anzahl bester ausgewählt wird (reihenfolgebasiert).

Ada. Echtzeitfähige Programmiersprache mit sehr guter Unterstützung von Multitasking-Anwendungen.

Binding. Schnittstelle zu einer Programmbibliothek in einer anderen Programmiersprache.

EA. Abk. für Evolutionäre Algorithmen.

Fitneß. Relative Selektionswahrscheinlichkeit eines Individuums gegenüber dem Durchschnitt bei der Roulette-Selektion.

Generation gap. Quotient aus der Anzahl der Nachkommen und der Populationsgröße.

Gtk. Graphic Tool Kit. Bibliothek für graphische Benutzeroberflächen, ursprünglich für das Bildbearbeitungsprogramm Gimp (Gnu Image Manipulation Program).

Lösungsraum. Mathematischer Raum, auf den die Lösungssuche und somit auch die Rekombination und Mutation beschränkt ist. Die erste Population eines evolutionären Algorithmus sollte gleichmäßig über den Lösungsraum verteilt sein. Der Lösungsraum sollte so klein wie möglich gewählt werden.

Loss of diversity. Verlust der Vielfalt innerhalb einer Population während der evolutionären Optimierung.

Monte-Carlo(-Nachkomme). Nachkomme, der nicht durch Rekombination gebildet wird, sondern zufällige Eigenschaften hat.

Mutation. Zufällige Veränderung der Variablen eines Individuums.

Mutationsrate. Relative Anzahl der Variablen eines Individuums, die mutiert werden.

Mutationsbreite. Standardabweichung der Mutation.

Open loop. Unterbrochene Rückführung in einem Regelkreis, wodurch der Regler unwirksam ist.

Prädiktor. Programmteil zur Vorhersage des Zustandsverlaufs eines Prozesses im Zusatzsteuerungsverfahren.

Qualitätsfunktion. Bewertungsfunktion für den Zustand eines Prozesses.

Rangfolge. In Abhängigkeit von der Zielfunktion sortierte Abfolge der Individuen.

Reinsertion rate. Relative Anzahl der Nachkommen, die in die Population übernommen werden.

Rekombination. Zufälliges Zusammenstellen eines Nachkommens aus den einzelnen Komponenten der Eltern. Neben der diskreten gibt es auch noch die inter- und extrapolierende Rekombination.

Roulette-Selektion. Die Individuen werden mit der ihrer Fitneß entsprechenden Wahrscheinlichkeit ausgewählt (fitneßproportional).

Selektion. Auswahl der Individuen einer Population, die für die Rekombination in Frage kommen. Die Selektion kann fitneß- oder rangfolgebasiert sein.

Selektionsdruck. Faktor der Erhöhung der Reproduktionswahrscheinlichkeit des besten Individuums gegenüber dem Durchschnitt.

Selektionsintensität. Erwartete durchschnittliche Fitneß nach der Selektion.

Task. Nebenläufiger Programmteil in Ada, wird in anderen Programmiersprachen auch *Thread* genannt.

Truncation selection. Siehe Abschneideselektion.

Turnierselektion. Selektion mit direktem Vergleich zweier oder mehrerer Individuen.

Zielfunktion. Funktion, deren Wert im Verlauf einer Optimierung minimiert oder maximiert wird.

Literatur

[Ber05] Th. Bernard. *Multikriterielle Optimierung eines chemischen Prozesses mit vielen Gütekriterien.* In Gesellschaft für Mess-und Automatisierungstechnik (Hrsg.), *GMA-Kongress 2005: Automation als interdisziplinäre Herausforderung*, Band 1883 der *VDI-Berichte*, S. 393 – 399. VDI/VDE, 2005.

[Blü91] Thomas Blümecke. *Wunder der Evolution. c't*, 1991, 12, S. 228 – 239.

[Duf97] S. Tucker Taft; Robert A. Duff (Hrsg.). *Ada 95 Reference Manual, Language and Standard Libraries, International Standard ISO/IEC 8652:1995(E)*, Band 1246 der *Lecture Notes in Computer Science*. Springer, 1997. ISBN 3-540-63144-5.

[Gro93] Werner Hauger; Walter Schnell; Dietmar Gross. *Technische Mechanik: Kinetik*, Band 3. Springer, Berlin, 4. Aufl., 1993. ISBN 3-540-56323-7.

[Jel05] M. Jelali. *Regelkreisüberwachung in der Metallindustrie: Anforderungen, Stand der Technik und Anwendungen.* In Gesellschaft für Mess-und Automatisierungstechnik (Hrsg.), *GMA-Kongress 2005: Automation als interdisziplinäre Herausforderung*, Band 1883 der *VDI-Berichte*, S. 429 – 439. VDI/VDE, 2005.

[Kru04] Ingrid Gerdes; Frank Klawonn; Rudolf Kruse. *Evolutionäre Algorithmen*. Vieweg, 2004. ISBN 3-528-05570-7.

[Mic01] Dipankar Dasgupta; Zbigniew Michalewicz (Hrsg.). *Evolutionary Algorithms in Engineering Applications*. Springer, 2001. ISBN 3-540-62021-4.

[Par04] Olivier Pauplin; Jean Louchet; Evelyne Lutton; Michel Parent. *Applying evolutionary optimization to robot obstacle avoidance. ISCIIA04*, 2004. http://arxiv.org/pdf/cs.AI/0510076.

[Pic05] G. Nöth; R. Pickhardt. *Anlagenoptimierung durch Performance Monitoring und Alarm Management.* In Gesellschaft für Mess-

und Automatisierungstechnik (Hrsg.), *GMA-Kongress 2005: Automation als interdisziplinäre Herausforderung*, Band 1883 der *VDI-Berichte*. VDI/VDE, 2005. (Lose-Blatt-Ergänzung).

[Poh99a] Hartmut Pohlheim. *Evolutionäre Algorithmen*. Springer, Berlin, 1999. ISBN 3-540-66413-0.

[Poh99b] Hartmut Pohlheim. *GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with Matlab*, 1999. http://www.geatbx.com/index.html.

[Sem91] I. N. Bronstein; K. A. Semendjajew. *Taschenbuch der Mathematik*. Teubner, Stuttgart, 25. Aufl., 1991. ISBN 3-8154-2000-8.

[Völ05] Anderas Völkel. *Evolutionäre Optimierung von Mensch-Maschine-Schnittstellen*. Logos Verlag, Berlin, 2005. ISBN 3-8325-0860-0.

[Wei02] Karsten Weicker. *Evolutionäre Algorithmen*. Teubner, 2002. ISBN 3-519-00362-7.

Index