Verteilte Online-Mehrziel-Parameter-Optimierung in mechatronischen Systemen

zur Erlangung des akademischen Grades eines
DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)
der Fakultät für Maschinenbau
der Universität Paderborn

genehmigte
Dissertation

von
Dipl.-Ing. Markus Deppe
aus Paderborn

Tag des Kolloquiums: 17. November 2006
Referent: Prof. Dr.-Ing. Joachim Lückel
Korreferent: Prof. Dr.-Ing. Ansgar Trächtler

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Mechatronik Laboratorium Paderborn (MLaP) der Universität Paderborn.

Dem Leiter des Fachgebiets, Herrn Prof. Dr.-Ing. Joachim Lückel, gilt mein besonderer Dank. Er gab mir die Gelegenheit, diese Arbeit durchzuführen und hat sie durch sein stetiges Interesse und seine wertvollen Anregungen maßgeblich begleitet und gefördert. Seinem Nachfolger Herrn Prof. Dr.-Ing. Ansgar Trächtler danke ich für die Übernahme des Korreferats.

Allen Mitarbeitern und Kollegen am MLaP danke ich für die kooperative und angenehme Arbeitsatmosphäre und die intensiven und anregenden Diskussionen. Mein besonderer Dank gilt Herrn Dr.-Ing. Rolf Naumann, Herrn Dr.-Ing. Rainer Rasche, Herrn Dipl.-Ing. Oliver Oberschelp und Herrn Dipl.-Ing. Norbert Neuendorf für die intensive Zusammenarbeit im SFB 376. Herrn MSc. Mauro Zanella, Herrn Dipl.-Ing. Michael Robrecht und Herrn Dr.-Ing. Ralf Stolpe danke ich für die gemeinsamen Arbeiten im Rahmen des SPP 1020. Darüber hinaus dürfen die verschiedenen Studien- und Diplomarbeiter sowie studentischen Hilfskräfte nicht unerwähnt bleiben, die mich mit ihren Arbeiten unterstützt haben.

Frau Annette Bökamp-Gros danke ich für die sorgfältige Durchsicht des Manuskripts.

Dank ganz anderer, aber nicht geringerer Art gilt meiner Frau Claudia, die mich über die Jahre unterstüzt und begleitet hat, und meinen Eltern, die mir diese Entwicklung erst ermöglichten.

Bad Lippspringe, im Januar 2007 Markus Deppe

Inhaltsverzeichnis

Notation

Abkürzungen

ABC	Active Body Control
ABS	Anti-Blockier-System
ACC	Adaptive Cruise Control
ACO	Ant Colony Optimization
AICC	Autonomous Intelligent Cruise Control
AMS	Autonomes Mechatronisches System
ANSI	American National Standards Institute
ASR	Antriebsschlupfregelung
CAMeL	Computer-Aided Mechatronics Laboratory
CAMeL-View	CAMeL-Virtual Engineering Workbench
CICC	Communicative Intelligent Cruise Control
DFG	Dataflow Graph und Deutsche Forschungsgemeinschaft
DFT	Diskrete Fourier-Transformation
DGL	Differentialgleichung
DOF	Degrees of Freedom
DSC	Dynamic System Code
ESP	Elektronisches Stabilitätsprogramm
FFT	Fast Fourier-Transformation
FPGA	Field Programmable Gate Array
FT	Fourier-Transformation
GPS	Global Positioning System
HILS	Hardware-in-the-Loop-Simulation
IAE	Integral Absolute Error
IEEE	Institute of Electrical and Electronics Engineers
INRIA	Institut National de Recherche en Informatique et Automatique
IPANEMA	Integration Platform for Networked Mechatronic Systems
IPI	Inter-Processor Interrupt
ISE	Integral Squared Error
iSIM	Intersection Simulator
ISR	Interrupt Service Routine
ITAE	Integral Time-weighted Absolute Error
ITSE	Integral Time-weighted Squared Error
KKT	Karush-Kuhn-Tucker
LIDAR	Light Detection and Ranging

LZI	Linear zeitinvariant
MFG	Mechatronische Funktionsgruppe
MFM	Mechatronisches Funktionsmodul
MLaP	Mechatronik Laboratorium Paderborn
MOPO	Multi-Objective Parameter Optimization
NBI	Normal Boundary Intersection
NBP	Neue Bahntechnik Paderborn
NOW	Networked Workstations
OCM	Operator-Controller-Modul
ODE	Ordinary Differential Equation
ODSL	Objective-Dynamic System Language
ODSS	Objective-Dynamic System Structure
OES	Optimierer-Evaluator-Simulator
OMM	Object-Oriented Mechatronic Model
PSO	Particle Swarm Optimization
RADAR	Radio Detection and Ranging
RMS	Rate Monotonic Scheduling
RTOS	Real-Time Operating System
SILS	Software-in-the-Loop-Simulation
TCP/IP	Transmission Control Protocol / Internet Protocol
VMS	Vernetztes Mechatronisches System
WCET	Worst-Case Execution Time

Formelzeichen

$\underline{A}$	Dynamikmatrix der linearen Zustandsraumdarstellung
A_k	Kolbenfläche für Hydraulikzylinder
$\underline{\alpha}$	Gewichtungsvektor
A_w	Angeströmte Fahrzeugstirnfläche
$\underline{B}$	Eingangsmatrix der linearen Zustandsraumdarstellung
β	Schwimmwinkel
$\underline{C}$	Ausgangsmatrix der linearen Zustandsraumdarstellung
c	Federsteifigkeit
c_w	Luftwiderstandsbeiwert
$\underline{D}$	Durchgriffsmatrix der linearen Zustandsraumdarstellung
d	Geschwindigkeitsproportionale Reibung
δ	Lenkwinkel
$\delta(x)$	Impulsfunktion
ΔT	Zeitintervall
$\underline{d}_k$	Suchrichtung
ε	Abweichung
$\underline{\Phi}$	Fundamentalmatrix
$\underline{g}$	Gradientenvektor
g_α	Skalares Funktional
η	Erfüllungsgrad einer Zielgröße
$\underline{H}$, $\underline{\tilde{H}}$	Hesse-Matrix, genäherte Hesse-Matrix
h	Integrationsschrittweite
$\underline{I}$	Einheitsmatrix
j	Imaginäre Einheit
L	Zielgrößenlimit
$\lambda_1, \ldots, \lambda_n$	Eigenwerte
λ_k	Schrittweite für die eindimensionale Minimumsuche
m	Masse

M	Mittelwert
n	Ganzzahliger Wert
n_p, n_z	Anzahl Parameter, Anzahl Zielgrößen
ω	Kreisfrequenz
$\underline{p}$, $\underline{p}^*$	Parametervektor, Pareto-optimaler Punkt im Parameterraum
p_{dyn}, p_{stat}	Dynamischer bzw. statischer Hydraulikdruck
P_H	Hydraulische Leistung
q_H	Hydraulischer Volumenstrom
$q_{i,k}$	Quadratische Ersatzzielfunktion
ρ	Luftdichte
$\underline{S}$	Empfindlichkeitsmatrix
s, $\dot{s}$, $\ddot{s}$, $\dddot{s}$	Weg, Geschwindigkeit, Beschleunigung, Ruck
t	Zeit
τ_{RT}	Echtzeitfaktor für die Online-Optimierung
T_S	Periodendauer (Zykluszeit)
$\underline{u}$	Eingangsvektor
Υ_k	Gewichtung der voherigen Suchrichtung im konj. Gradientenverfahren
U_P, $U_{P,max}$	Prozessorauslastung, theoretische maximale Prozessorauslastung
v	Geschwindigkeit
$\underline{x}$, $\dot{\underline{x}}$	Zustandsvektor, erste zeitliche Ableitung des Zustandsvektors
$\underline{y}$	Ausgangsvektor
ψ, $\dot{\psi}$	Gierwinkel, Gierwinkelgeschwindigkeit (Gierrate)
$\Im$	Menge der aktiven Zielgrößen
$\underline{z}$, $\underline{z}^*$	Zielgrößenvektor, effizienter Punkt im Zielgrößenraum

1 Einleitung

Die Mechatronik ist aus heutigen Kraftfahrzeugen nicht mehr wegzudenken. Längst sind Komponenten wie ABS (Anti-Blockier-System), ASR (Antriebsschlupfregelung), ESP (Elektronisches Stabilitätsprogramm) oder ABC (Active Body Control) serienmäßig verfügbar. Durch sie sind in den letzten Jahren große Fortschritte im Bereich der Fahrsicherheit und des Fahrkomforts erzielt worden.

Für den Straßenverkehr der Zukunft wird weltweit intensiv an der Entwicklung leistungsfähiger und intelligenter Verfahren zur gleichzeitigen Erhöhung der Kapazität des Straßennetzes sowie der Sicherheit gearbeitet. Zielsetzung ist die Ausschöpfung bzw. Erhöhung der Effizienz der vorhandenen Straßenkapazität ohne die Durchführung straßenbaulicher Maßnahmen. Dabei wird zwischen fest installierten infrastrukturellen Einrichtungen (z. B. Verkehrsleitsystemen) und im Fahrzeug integrierter Mechatronik unterschieden. Eine wichtige Anwendung ist das automatisierte Kolonnenfahren von Fahrzeugen, die dazu mit Abstandsregelsystemen ausgerüstet werden. Rein sensorgeführte Systeme werden mit AICC (Autonomous Intelligent Cruise Control [Dorißen, Höver 1996]) oder auch ACC (Adaptive Cruise Control) bezeichnet. Demgegenüber stehen CICC-Systeme (Communicative Intelligent Cruise Control [Mayr 2001]), bei denen zusätzlich zur Sensorinformation auch Datenkommunikation zwischen den Fahrzeugen stattfindet.

Durch Bildverarbeitung, RADAR- (Radio Detection and Ranging), LIDAR-Sensoren (Light Detection and Ranging), GPS (Global Positioning System) und Funknetzwerke etc. wird im Fahrzeug zukünftig eine Fülle von Informationen über die nähere und weitere Fahrumgebung vorliegen. Diese Informationen müssen intelligent verknüpft werden und dienen dann zur Unterstützung des Fahrers (komplexe Fahrerassistenzsysteme). Ein visionäres Fernziel ist das vollständig autonome mechatronische Fahren ohne menschlichen Fahrer. Abbildung 1-1 zeigt eine Prognose für die Entwicklung von Anwendungen in Straßenfahrzeugen von Fahrerassistenz-Systemen bis hin zu autonomen Fahrzeugen:

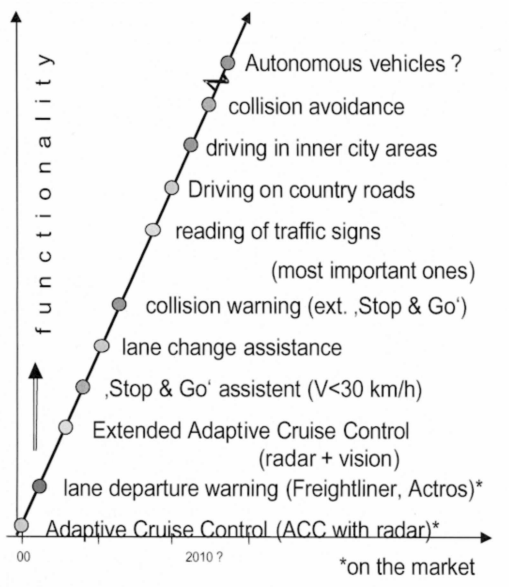

Abbildung 1-1: Fahrerassistenzsysteme in Straßenfahrzeugen [Vehicle Autonomous Systems 2002]

Als Vision für ein zukünftiges CICC-System ist die Idee für ein selbstorganisierendes Kreuzungsmanagement entstanden, das im DFG-Sonderforschungsbereich 376 "Massive Parallelität - Algorithmen, Methoden, Anwendungen" im Teilprojekt C1 ([Deppe, Rasche 2000/1], [Deppe, Rasche 2000/2], [Deppe at al. 2003/1], [Deppe at al. 2003/2]) als Anwendungsbeispiel dient. Ziel des Kreuzungsmanagements ist das vollständig dezentral organisierte Überqueren einer Straßenkreuzung. Dabei sind alle Fahrzeuge autonom gesteuert und sollen untereinander einen stillstands- und kollisionsfreien Ablauf selbstständig organisieren, indem wesentliche Anteile der verkehrsbegleitenden Informationsverarbeitung den zentralen Instanzen entzogen und in lokale, fahrzeugeigene Bereiche verlagert werden. Die dafür erforderliche lokale Rechenleistung stellen die beteiligten Kraftfahrzeuge selbst zur Verfügung, so dass ein massiv paralleles Rechnernetzwerk entsteht. Neben der Grundforderung nach Kollisionsfreiheit muß für die entstehenden Zielkonflikte bezüglich Mindestgeschwindigkeit, Durchsatz, Fahrkomfort und Energieverbrauch ein möglichst guter Kompromiss innerhalb harter Zeitschranken gefunden werden:

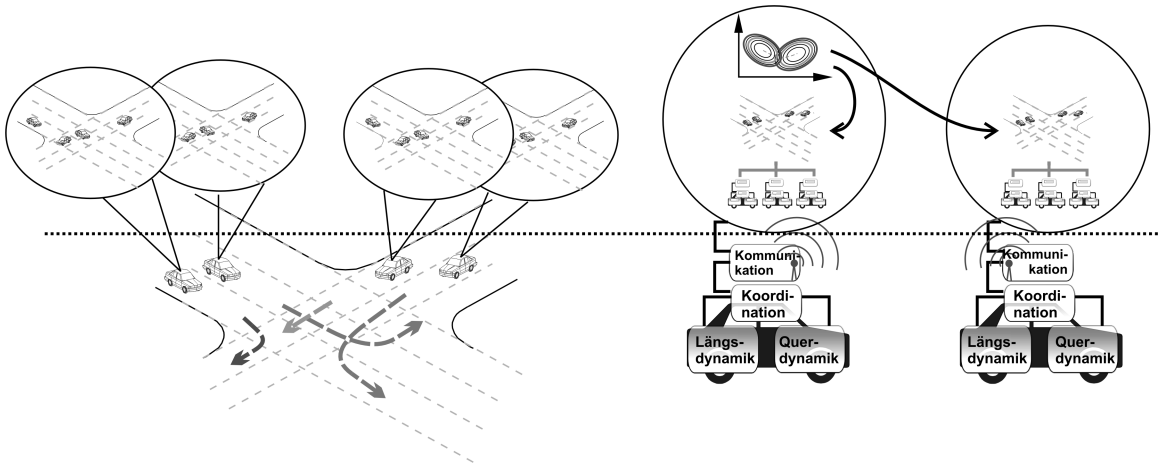

Abbildung 1-2: Selbstorganisierendes dezentrales Kreuzungsmanagement

Angestrebt ist nicht eine reale Umsetzung des Kreuzungsmanagements, so dass die Fragen bzgl. geeigneter Sensorik und Sicherheitstechnik auch nicht im Fokus stehen. Vielmehr wird nach Methoden gesucht, um das Problem von der konzeptionellen und algorithmischen Seite her zu lösen. Die wichtigste Methode für das Kreuzungsmanagement ist die Anwendung von onlinefähigen Mehrziel-Optimierungsverfahren auf der Basis von Modellen:

Mit Hilfe von Online-Mehrziel-Optimierungsverfahren lässt sich das Grundproblem der automatisierten Entscheidungsfindung in autonomen Systemen lösen. Neben physikalischen Zielkonflikten (z. B. schnelles Abbiegen und hoher Fahrkomfort) können auch die verschiedenen "Wünsche" der Einzelfahrzeuge (z. B. Zeit- oder Energieeffizienz) berücksichtigt werden. Dazu verfügt jedes Fahrzeug über Modelle von sich, von anderen Fahrzeugen und von der Kreuzung (vgl. Abbildung 1-2). Auf der Grundlage modellbasierter Vorhersagen ist dann eine Voraboptimierung der Überquerung möglich.

1.1 Motivation

Am selbstorganisierenden dezentralen Kreuzungsmanagement wird am MLaP (Mechatronik Laboratorium Paderborn) bereits seit einigen Jahren intensiv geforscht. Neben den Fahrzeugmo-

dellen, den Regelungssystemen zum Kolonnenfahren und den Verfahren zur Kollisionsvermeidung entstand dabei eine Umgebung für die Modellierung und Offline-Simulation von intelligenten mechatronischen Systemen mit hybrider (gemischt diskret-kontinuierlicher) Informationsverarbeitung [Naumann 2000]. Auch der systematische Entwurf von vernetzten mechatronischen Systemen unter dem Aspekt der Selbstoptimierung wurde untersucht [Rasche 2004].

Damit ergibt sich eine Anknüpfung an den DFG-Sonderforschungsbereich 614 "Selbstoptimierende Systeme des Maschinenbaus" [DFG-Antrag zum SFB 614 2001], der jedoch im Gegensatz zum Kreuzungsmanagement eine sehr weitreichende strukturelle Selbstoptimierung von Mechanik, Aktorik, Sensorik, Software und Hardware zur Laufzeit anstrebt. Insbesondere muss für die Selbstoptimierung ein vollständig automatisierter Ablauf erreicht werden, der mindestens einschließt: Erkennen der Notwendigkeit zur Optimierung, Identifikation von aktuellen Streckenmodellen, Definition von Optimierungszielen, Auswahl eines Optimierungsverfahrens, Konfiguration und Definition eines modellbasierten Optimierungsexperiments, Durchführung und Steuerung des Online-Optimierungsexperiments, Prüfen der Ergebnisse und Umsetzung auf das reale System. Damit wird deutlich, dass die in dieser Arbeit behandelte Online-Mehrziel-Optimierung nur als ein Teilproblem der Selbstoptimierung anzusehen ist.

Im Kreuzungsmanagement werden Strukturen (Vernetzung von Fahrzeugen zu Kolonnen) heuristisch gebildet, und die Optimierung modifiziert Parameter des vernetzten Systems. Die Gemeinsamkeiten bestehen in der konsequenten Hierarchisierung und Modularisierung der mechatronischen Systeme und der Nutzung von Online-Mehrziel-Optimierungsverfahren zur Laufzeit.

Als Anwendungsbeispiel im DFG-Sonderforschungsbereich 614 dient das Projekt NBP (Neue Bahntechnik Paderborn [Lückel 2000]). Hier wird das herkömmliche mechanische Tragen und Führen von Schienenfahrzeugen auf dem bestehenden Schienennetz mit dem fortschrittlichen verschleißfreien Linearantrieb kombiniert. Zusätzlich soll durch intelligente aktive Fahrwerkstechnik ein höherer Fahrkomfort erzielt werden. Wesentliches Element des neuen Verkehrssystems sind kleine, autonome Fahrzeuge, sogenannte Shuttles (Railcabs). Auch hier ergeben sich klare Parallelen zum Kreuzungsmanagement im Hinblick auf das autonome Fahren und die Kolonnenbildung von Shuttles durch Ein-/Ausfädeln an Weichen. Da eine vollständige Umsetzung des Kreuzungsmanagements in die Realität von vornherein nicht angedacht war, bietet die Neue Bahntechnik Paderborn die Plattform, um die Ergebnisse des Kreuzungsmanagements zukünftig praktisch anzuwenden.

1.2 Zielsetzung

Das originäre Anwendungsgebiet für Mehrziel-Optimierungsverfahren liegt im Bereich der Entwurfs- und Entwicklungsphasen für Systeme bzw. Produkte. Überträgt man nun die Anwendung der Verfahren auf die gesamte Lebensdauer von (vernetzten autonomen mechatronischen) Systemen, z. B. um die Fähigkeiten für ein selbstorganisierendes Kreuzungsmanagement in Fahrzeugen zu implementieren, so erwachsen ganz neue Randbedingungen. Diese ergeben sich auf ganz natürliche Art und Weise: Zum Einen gelten harte Echtzeitbedingungen aufgrund der Verknüpfung mit realen physikalischen Systemen, zum Anderen besitzt die Informationsverarbeitung aufgrund der Verknüpfung von Einzelfahrzeugen hochgradig verteilten Charakter.

Die Ziele der Arbeit gliedern sich im Einzelnen wie folgt auf:

1. Entwicklung eines Konzepts für die verteilte und echtzeitfähige Mehrziel-Optimierung, das sich konsequent an der modular-hierarchischen Struktur mechatronischer Systeme orientiert.
2. Weiterentwicklung der Laufzeitplattform IPANEMA zur Multitask-Realisierung von Online-Mehrziel-Optimierung auf Echtzeitsystemen.
3. Erweiterung der vorhandenen Mehrziel-Optimierungsalgorithmen von MOPO hinsichtlich Parallel- und Echtzeitverarbeitung.
4. Verfeinerung und Weiterentwicklung des vorhandenen Kreuzungsmanagements in Bezug auf Modelle, Verfahren zur Vorfahrtsbestimmung und Optimierungsexperimente.
5. Überprüfung der Ergebnisse an Anwendungsbeispielen.

1.3 Gliederung und Aufbau der Arbeit

Kapitel 2 skizziert die Grundlagen zu Entwurf, Rechnerabbildung und Hardware-in-the-Loop-Simulation für mechatronische Systeme. Ergänzend stellt Kapitel 3 die am MLaP entwickelten Konzepte zur modular-hierarchischen Strukturierung mechatronischer Systeme vor.

Danach geht Kapitel 4 speziell auf die Grundlagen für das Kernthema Mehrziel-Optimierung ein. Nach der Vorstellung von wichtigen Verfahren zur Mehrziel-Optimierung wird die MLaP-eigene Optimierungssoftware MOPO (Multi-Objective Parameter Optimization) beschrieben.

Aufbauend auf den Überblick zur Mechatronik und Mehrziel-Optimierung befassen sich die nachfolgenden Kapitel mit den eigenen Arbeitsergebnissen. Eine wichtige Basis für die Online-Optimierung ist die verteilte Echtzeitsimulation. Kapitel 5 beschreibt deren Konzept, die Erweiterung von IPANEMA um Möglichkeiten zur Multitask-Simulation und die Ergebnisse der Parallelisierung von Systemmodellen.

Das Konzept zur verteilten Online-Mehrziel-Optimierung wird in Kapitel 6 definiert. Dieses Konzept erlaubt es, vollständige Optimierungsanwendung inkl. Optimierer, Systembewertung, Reglern, Systemmodellen flexibel in eine verteilte Applikation zu integrieren. Es werden verschiedene Ebenen der Parallelität vorgeschlagen, die zu modular-hierarchisch verteilten Strukturen führen.

Kapitel 7 stellt drei Anwendungsbeispiele unterschiedlicher Komplexität vor. Durch die Beispiele werden die Einsatzmöglichkeiten der verteilten Online-Mehrziel-Optimierung für den Systementwurf, für Echtzeitanwendungen am Hardware-in-the-Loop-Prüfstand bis hin zu vernetzten mechatronischen Systemen beschrieben. Insbesondere der neueste Stand des Kreuzungsmanagements wird ausführlich dargestellt, bevor ein Beispiel für ein Mehrziel-Optimierungsexperiment der Kreuzung mit paralleler Verarbeitung gegeben wird.

Eine kurze Zusammenfassung und ein Ausblick auf mögliche weiterführende Arbeiten beschließen diese Arbeit in Kapitel 8.

Im Anhang finden sich weitergehende Detailinformationen zu den Verfahren zur Systembewertung, zu den Anwendungsbeispielen sowie das Literaturverzeichnis.

2 Mechatronik

Mit dem Ziel der Verbesserung mechanischer Systeme durch die Verwendung der Rechentechnik entstand eine neue Disziplin der Ingenieurwissenschaften, die Mechatronik. Der Begriff ist ein Kunstwort aus den beiden Anteilen **Mecha**nik und Elek**tronik** [Schweitzer 1989].

In der Mechatronik wird die Integration mechanischer Baugruppen und leistungsfähiger, digitaler Informationsverarbeitung auf besonders konsequente Weise von Beginn der Entwicklung bis zum Abschluss des Projekts durchgeführt. Dabei dienen als Verknüpfungselemente mit definierten Schnittstellen elektromagnetische, elektromechanische und elektrohydraulische Wandlersysteme für Sensorik (Ermittlung von Prozessinformationen) und Aktorik (Ansteuerung des technischen Prozesses).

Ein Ziel der Mechatronik ist es, das dynamische Verhalten eines technischen Systems zu verbessern. Dazu werden die Sensorinformationen durch Prozessoren verarbeitet, um eine für den jeweiligen Kontext "optimale" dynamische Reaktion mit Hilfe der Aktoren auszulösen. Am MLaP (Mechatronik Laboratorium Paderborn) wurde dazu folgende Definition formuliert:

> *Mechatronik ist die Wissenschaft von den **kontrollierten Bewegungsvorgängen** mechanischer Systeme mit Hilfe von Mikrorechnern.*

Die Zusammensetzung mechatronischer Systeme aus Komponenten der unterschiedlichen technischen Disziplinen Informationstechnik, Maschinenbau und Elektrotechnik führt in den meisten Fällen zu einem hohen Komplexitätsgrad des Systems und damit zu einem komplizierten und aufwendigen Entwurfsprozess.

Die Forschungsaktivitäten des MLaP konzentrieren sich auf die Entwicklungssystematik modularer und hochleistungsfähiger mechatronischer Systeme. Hier wird am Konzept der "Mechatronischen Komposition" geforscht ([Lückel et al. 2000], [Lückel et al. 2002/2], [Hahn et al. 1997], [Toepper 2002],). Die "Mechatronische Komposition" berücksichtigt zusammengefasst folgende Gesichtspunkte:

1. Konsequente wechselseitige Berücksichtigung von Konstruktionstechnik (Gestaltorientierung) und Mechatronik (Funktionsorientierung) von Anfang an (vgl. Kap. 2.1).
2. Vollständige Abbildung und Bearbeitung des zu entwickelnden Systems im Rechner als virtueller Prototyp (vgl. Kap. 2.2).
3. Ermöglichung der schrittweisen Realisierung von Prototypen durch Hardware-in-the-Loop-Techniken (vgl. Kap. 3) und ein Konzept zur modular-hierarchischen Strukturierung von mechatronischen Systemen (vgl. Kap. 3)
4. Bereitstellung einer umfangreichen Softwareunterstützung durch CAMeL (Computer-Aided Mechatronics Laboratory) mit einzelnen Werkzeugen zu Modellbildung, Simulation, Reglerentwurf, Mehrziel-Optimierung und Echtzeit-Simulation.

Der "Mechatronischen Komposition" kommt daher eine zentrale Bedeutung beim Entwurf mechatronischer Systeme zu. Abbildung 2-1 zeigt das Zusammenspiel der notwendigen Schritte

von der textuellen Aufgabenbeschreibung bis zum ersten Prototypen eines Systems:

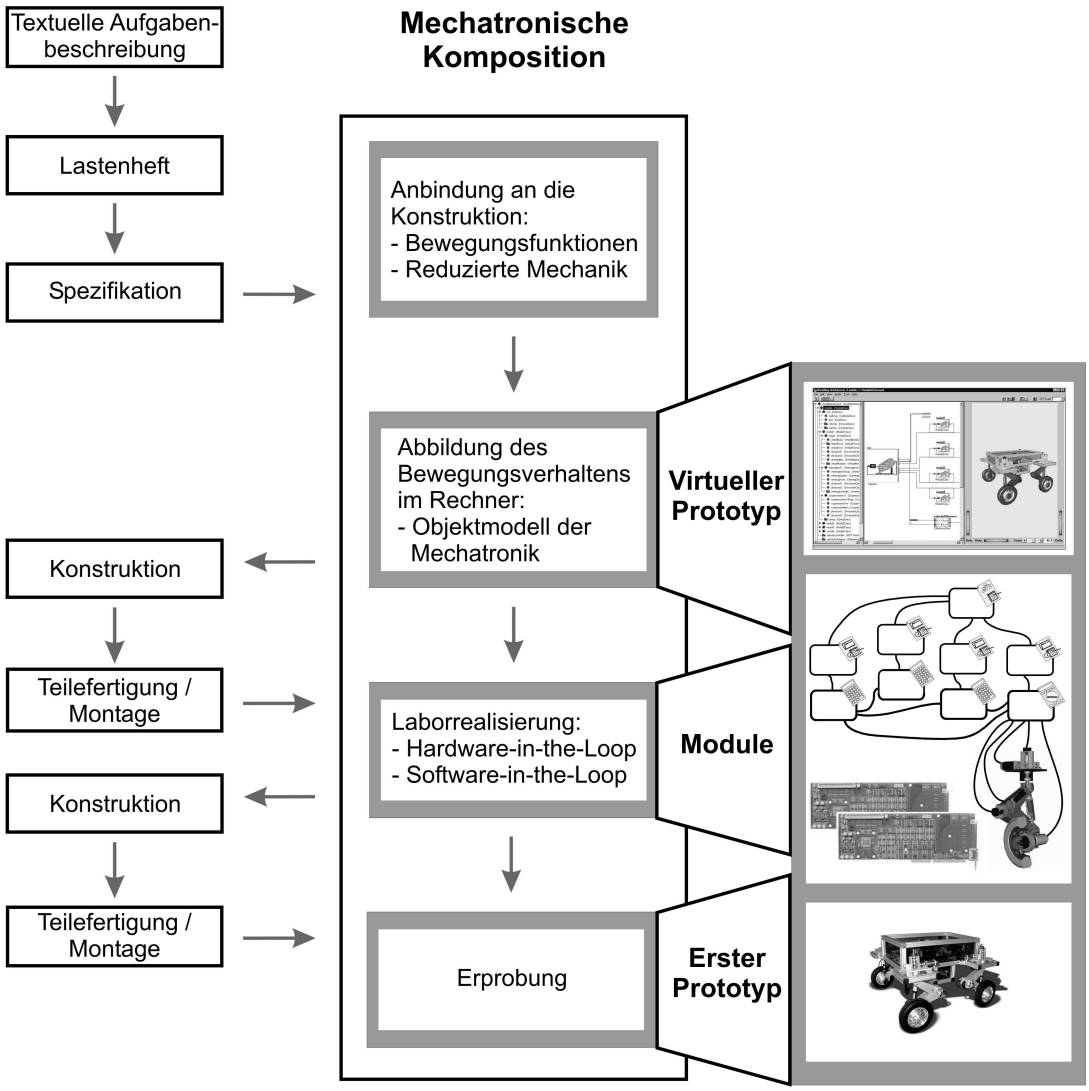

Abbildung 2-1: Entwurf mechatronischer Systeme

2.1 Anbindung an die Konstruktionstechnik

Wichtigste Grundforderung ist die frühzeitige Einbeziehung der Informations-/Regelungstechnik in die Produktentstehung. Oft wird die Mechatronik als Reparaturmöglichkeit eines bereits konstruierten und gebauten Prototypen "missbraucht". Der Gegensatz der gestaltorientierten Sichtweise der Konstruktionstechnik zu der funktionsorientierten Sichtweise (Bewegungsfunktionen) der Mechatronik erfordert ein geeignetes Konzept zur Einbindung der Konstruktionstechnik in den Mechatronikentwurf. Optimalerweise wird von Beginn des Entwurfs an eine wechselseitige Betrachtung von Bewegungsfunktionen und CAD-Lösungselementen zu deren Umsetzung erfolgen ([Lückel et al. 2002/2], [Wittler 2003]).

Für die softwareunterstützte Verknüpfung der Konstruktion mit der mechatronischen Komposition werden in [Koch 2005] die konzeptionellen Ansätze der Entwurfssystematik aufgegriffen und detailliert. Das Ziel ist hier die teilautomatisierte Integration mit Hilfe des Modellaustau-

sches. Neben der Unterteilung der Haupt- und der Teilbewegungsfunktionen in die Bereiche Kinematik, Dynamik und Mechatronik erfolgt von Anfang an eine hierarchische Strukturierung des Systems (vgl. Kap. 3), die als Bindeglied zwischen den Domänen dient.

Die Mechanik der CAD-Lösungselemente wird zunächst schematisch unter den Aspekten Funktion und Struktur ausgearbeitet. Mit Hilfe von reduzierten Modellen der CAD-Lösungselemente in Form von Starrkörper-Modellen mit reduzierter Anzahl von Freiheitsgraden kann deren Bewegungsverhalten analysiert werden. Die Analyseergebnisse fließen zurück in die schematische Ausarbeitung der Mechanik:

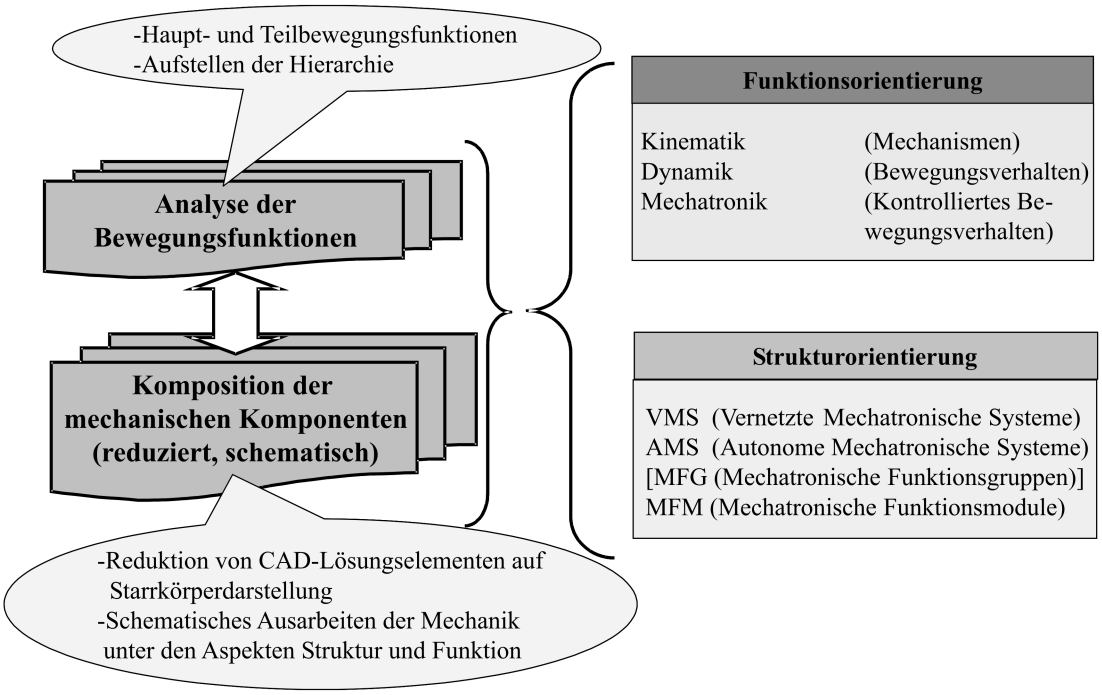

Abbildung 2-2: Anbindung an die Konstruktionstechnik

2.2 Rechnerabbildung mechatronischer Systeme

Die Komplexität mechatronischer Systeme kann durch die Verwendung verschiedener Systembeschreibungssprachen besser beherrscht werden. Die einzelnen Beschreibungssprachen befassen sich mit der Festlegung der Struktur und der Topologie des Systems, der mathematischen Beschreibung und schließlich der Festlegung verarbeitungsnaher Zusammenhänge.

Die Aufteilung in mehrere Ebenen schafft ein hierarchisches Systembeschreibungskonzept. Der Vorteil dieser Vorgehensweise liegt in der Reduktion der Information auf die in der jeweiligen Ebene beschriebenen Systemeigenschaften, was letztlich zu mehr Übersicht im Verlauf des mechatronischen Entwurfs führt.

Aufgrund der Erfahrungen im Bereich der Konzeption mechatronischer Systeme erfolgt die Systembeschreibung am MLaP auf drei Ebenen. Die drei Beschreibungsebenen werden im Rechner teilautomatisch erzeugt und synchronisiert. Diese Rechnerabbildung wird als Objektmodell der Mechatronik (OMM, vgl. Abbildung 2-3) bezeichnet [Hahn 1999]. Das OMM erlaubt eine ganzheitliche Abbildung verschiedenster Fachdisziplinen zur optimalen dynamischen Auslegung

des gesamten mechatronischen Systems im MLaP-Modellierungstool CAMeL-View ([iXtronics 2001/1], [iXtronics 2001/2]). Die drei Ebenen entsprechen den Gesichtspunkten Struktur, Verhalten und Verarbeitung:

1. **Struktur**:
   Die oberste Ebene beschreibt die topologische Struktur des mechatronischen Systems unter Verwendung der disziplinspezifischen Beschreibungsformen der einzelnen Komponenten sowie ihrer ungerichteten Verknüpfungen untereinander, durch die das mechatronische Gesamtsystem entsteht. Die einzelnen disziplinspezifischen Beschreibungsformen werden durch Verwendung objektorientierter Methoden zueinander kompatibel. Zur Beschreibung mechatronischer Systeme wurde eine erweiterbare, textuelle Systembeschreibungssprache entwickelt, die im Folgenden als ODSS (Objective-Dynamic System Structure) bezeichnet wird. Die Sprache wurde zur Beschreibung des topologischen Aufbaus von Systemen auf der physikalischen Ebene entwickelt und ermöglicht die Modellierung von Teilen und Aggregaten unterschiedlicher Disziplinen und deren Verbindungen.

2. **Verhalten**:
   Die mittlere, normierende Ebene vereinigt die Beschreibung des Systems zu einem mathematisch orientierten Formalismus auf der Basis von Differentialgleichungen in Zustandsraumdarstellung [Ludyk 1990]. Durch explizite Eingangs-/Ausgangsformulierung in Form sogenannter Blöcke lässt sich auch auf dieser Ebene der modulare und hierarchische Charakter des modellierten Systems nachvollziehen. Außerdem wird durch die modulare und hierarchische Modellbeschreibung der Aufbau selbst großer Systeme (in der Größenordnung von einigen hundert Differentialgleichungen) möglich. Der Name der dieser Ebene entsprechenden Beschreibungssprache lautet Objective-Dynamic System Language (ODSL). Diese Ebene kann mit Hilfe von Ableitungsformalismen automatisch aus ODSS erzeugt werden.

3. **Verarbeitung**:
   Die Orientierung der untersten Ebene ist bereits sehr stark auf die Verarbeitung im Digitalrechner ausgerichtet, ohne sich in die Abhängigkeit zu einer bestimmten Hardwarearchitektur zu begeben. Die Beschreibung des Systems auf dieser Ebene ist durch die Angabe der expliziten Auswertereihenfolge der Systemgleichungen gekennzeichnet. Entsprechend dem verarbeitungsnahen Charakter dieser Ebene heißt die verwendete Beschreibungssprache Dynamic System Code (DSC). Diese Beschreibungsebene wird automatisch aus ODSL erzeugt. Für die Ausführung von Simulationen erzeugt ein Codegenerator aus DSC effizienten plattformunabhängigen C-Code:

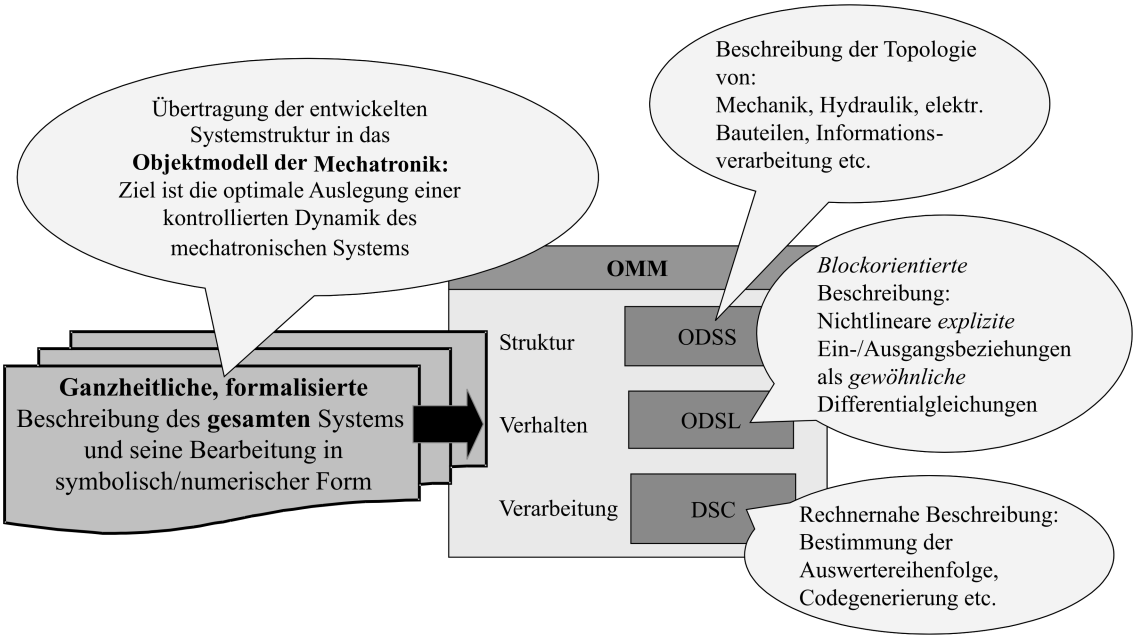

Abbildung 2-3: Objektmodell der Mechatronik zur Rechnerabbildung von mechatronischen Systemen

Verwendung von Graphen

Zur Beschreibung und Verarbeitung von mechatronischen Systemen werden am MLaP graphenbasierte Repräsentationen eingesetzt. Die Graphentheorie vereint die Vorteile der guten Umsetzbarkeit im Digitalrechner mit hoher Anschaulichkeit. Für jede der drei Ebenen des OMM existieren weitreichende Anwendungen der Graphentheorie [Hahn 1999]:

OMM Topologie-Ebene: Systemgraph

Knoten: Basiselemente und Hierarchieelemente

Kanten: Kopplung von Bauteil-Bauteil, System-Subsystem, Eltern-Kind

OMM Verhaltens-Ebene: Systemgraph

Knoten: Basiselemente und Hierarchieelemente

Kanten: Kopplung von System-System, System-Subsystem, Eltern-Kind

OMM Verarbeitungs-Ebene: Berechnungsgraph

Knoten: mathematische Funktionen

Kanten: Datenfluss, Reihenfolge von Funktionsaufrufen

Graphen bestehen aus einer Menge von Knoten und einer Menge von Kanten. Die Knoten repräsentieren diskrete Objekte und die Kanten die Beziehungen zwischen den Knoten. Zur graphischen Darstellung werden üblicherweise Knoten als Kreise und Kanten als Linien dargestellt. Wenn die Kanten Richtungsinformationen beinhalten werden sie als Pfeile dargestellt, man spricht dann von gerichteten Graphen im Gegensatz zu ungerichteten Graphen ([Walther 1984], [Nägler, Walter 1987]). Isolierte Teilgraphen innerhalb eines Graphen werden auch als Kompo-

nenten bezeichnet (siehe Abbildung 2-4). Extremer Sonderfall ist ein Graph ohne Kanten, in dem dann jeder Knoten eine Komponente darstellt. Um den Vernetzungsgrad in einem Graphen zu charakterisieren wird zwischen lichten und dichten Graphen unterschieden. In lichten Graphen ist die Kantenanzahl kleiner als die Knotenanzahl, in dichten Graphen ist dies genau umgekehrt.

Für gerichtete Graphen unterscheidet man zwischen Knoten mit ausschliesslich ausgehenden Kanten (Quellen) und ausschliesslich eingehenden Kanten (Senken). Das wichtigste Kriterium für einen gerichteten Graphen ist die Aussage, ob er kreisfrei ist oder nicht. Ein Graph ist dann kreisfrei, wenn es für keinen Knoten im Graphen einen Pfad gibt, der wieder zu ihm zurückführt. Auf kreisfreie gerichtete Graphen lässt sich die sog. topologische Sortierung anwenden. Sie hat zum Ziel eine Liste (sequentielle Reihenfolge) von Knoten so zu ermitteln, dass die Richtung aller Kanten auf den letzten Knoten der Liste weist. Ein weiterer wichtiger Algorithmus für kreisfreie gerichtete Graphen ist die Ermittlung der längsten oder kürzesten Pfade von einer Quelle zu einer Senke:

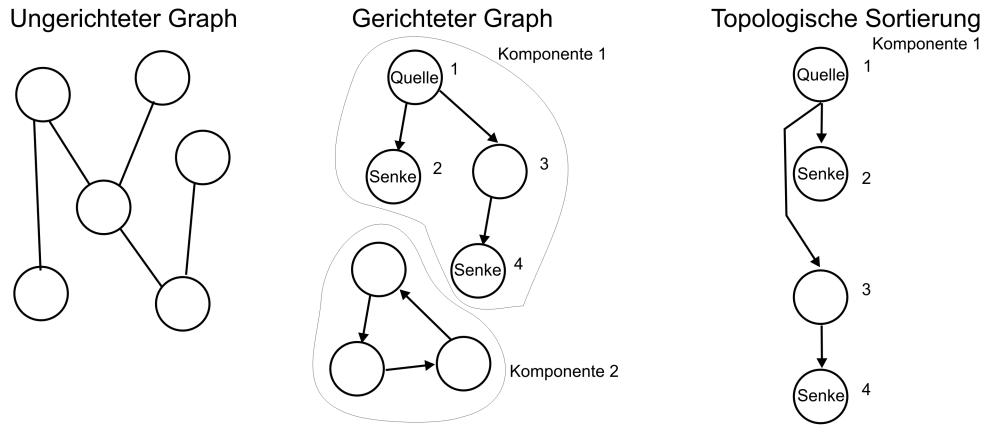

Abbildung 2-4: Gerichtete und ungerichtete Graphen

2.3 Hardware-in-the-Loop-Simulation

Besonderes Gewicht bei der Entwicklung mechatronischer Systeme haben die Modellierung, die Simulation und die Optimierung des zu entwickelnden Systems. Mit Hilfe dieser Techniken ist es möglich, zeitaufwendige Abläufe von der Werkshalle oder dem Versuchsfeld in den Rechner zu verlagern. Erklärtes Ziel dieser Vorgehensweise ist die Reduzierung der Durchlaufzeiten sowie eine bessere Reproduzierbarkeit der Ergebnisse.

Der Übergang vom rein simulierten zum vollständig physikalisch realisierten mechatronischen System ist, ingenieurtechnisch gesehen, ein sehr anspruchsvoller und fehlerträchtiger Schritt. Daher wird in zunehmendem Maße die Technik der sogenannten Hardware-in-the-Loop-Simulation (HILS) ([Castiglioni et al. 1992], [Wältermann 2000]) angewendet. Dieses Verfahren erlaubt einen schrittweisen Übergang vom rein simulierten zum vollständig realisierten mechatronischen System. Die Grundidee besteht in der Kopplung einer virtuellen, in Echtzeit simulierten Umgebung mit einem physikalisch aufgebauten Teilmodul als ein Aggregat des Gesamtsystems. Kräfte und Bewegungen an der mechanischen Schnittstelle des Gesamtsystems mit dem Teilmodul werden durch geregelte Aktoren nachgebildet. Die Reaktion des Teilmoduls wird über Sensoren

ermittelt und in den simulierten Teil zurückgeführt (Closed-Loop HILS).

In der Automobilindustrie ist die HILS inzwischen eine eingeführte und bewährte Vorgehensweise. Auch der Test von Steuergeräten mit Hilfe der HILS ist weit verbreitet [Hanselmann 1993]. Das zu testende Teilmodul des Gesamtsystems ist dann die Hard- und Software des Steuergerätes. In diesem Falle existieren eine Datenschnittstelle und eine elektrische Anbindung an die Umgebung.

2.4 Laufzeitplattform IPANEMA

Die Simulationsplattform IPANEMA (Integration Platform for Networked Mechatronic Systems) wurde zur Durchführung von verteilten HIL-Simulationen am MLaP entwickelt ([Honekamp 1998], [Stolpe 2004]). IPANEMA bietet Dienste in Form eines Software-Baukastensystems an, das unterschiedliche Klassen von informationstechnischen Prozessen enthält. Dabei existiert eine strikte Trennung zwischen weichen und harten Echtzeit-Bedingungen. Mit IPANEMA ist eine Abbildung der modular hierarchischen Modellstruktur auf eine ebenfalls modular-hierarchische Prozess- oder Taskstruktur möglich. Diese erleichtert den geforderten schrittweisen, d. h. modulweisen Übergang vom virtuellen Prototypen zum ersten Gesamtprototypen. IPANEMA bildet das Bindeglied zwischen der modular-hierarchischen Modellstruktur, die aus dem Strukturierungskonzept (siehe Kapitel 3.1) resultiert, und einer verteilten (modularen) Echtzeit-Simulation für HIL-Anwendungen:

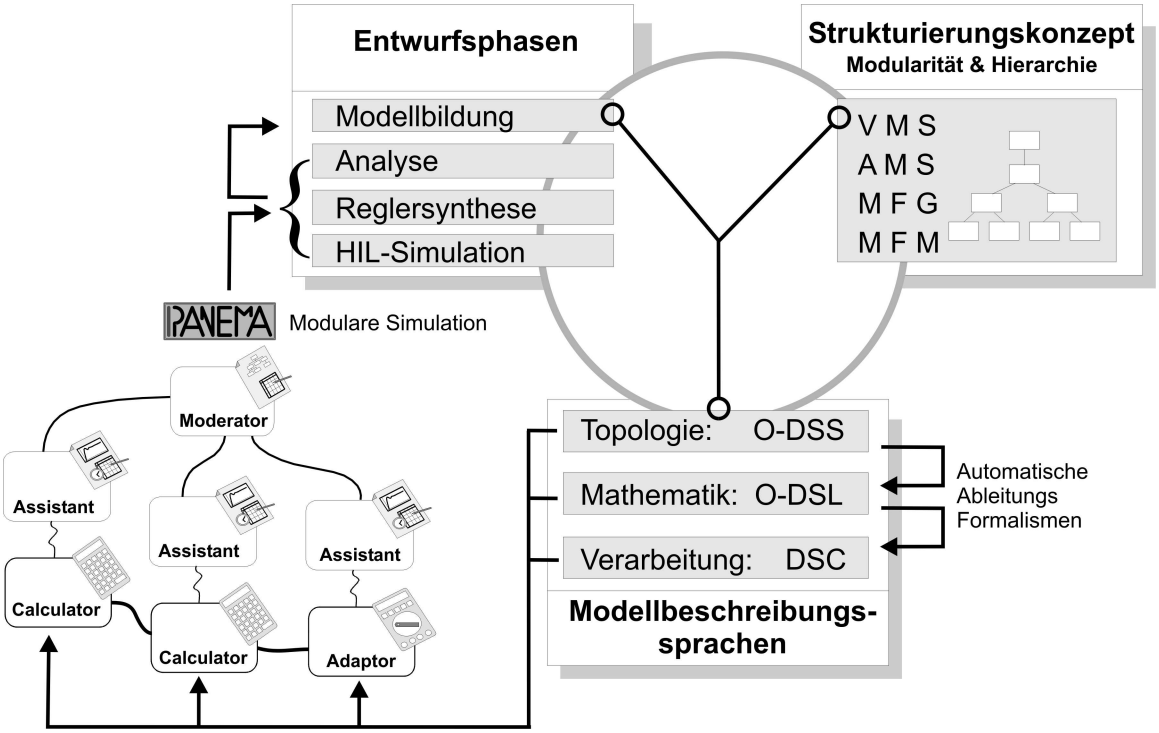

Abbildung 2-5: IPANEMA zur modularen Simulation des OMMs

IPANEMA wird darüber hinaus auch für verteile Offline-Simulationen auf vernetzten Workstations verwendet [Deppe 1997]. Die Architektur von IPANEMA basiert auf den folgenden Klassen:

1. **Moderator**:
   Implementiert das Anwenderprotokoll unter Herstellung von Netztransparenz, d. h. der Anwender braucht keine Kenntnis über den Aufenthaltsort von Variablen der verteilten Applikation zu haben.

2. **Assistant**:
   Trennt die Ebene der weichen Echtzeit (Anwenderseite, Moderator) von der harten Echtzeit über die Bereitstellung von Ringpuffern. Jeder Assistant ist mit dem Moderator verbunden.

3. **Calculator**:
   Beinhaltet die eigentliche periodisch bearbeitete, echtzeitkritische Anwendung. Jedem Calculator ist ein eigener Assistant zugeordnet.

4. **Adaptor**:
   Bildet die Daten-Schnittstelle zu Aktorik und Sensorik mit entsprechender Signalvorverarbeitung (Skalierung, Filterung etc.).

3 Strukturierung mechatronischer Systeme

3.1 Modular-Hierarchische Strukturierung

Die Aufteilung eines mechatronischen Systems in mechatronische Funktionsmodule (MFM) ist eine Form der Strukturierung des Systems, die sich am Aggregat-Gedanken des Maschinenbaus orientiert ([Lückel 1992], [Honekamp et al. 1997], [DFG-Antrag zum SFB 614 2001]). Hierbei findet eine Zuordnung von Bewegungsfunktionen des mechatronischen Systems zu Aktoren, Sensoren, Tragstrukturen und Informationsverarbeitung statt. Die Grundidee für die Strukturierung ist die Abbildung des Gesamtsystems im Rechner nach *physikalischen* Gesichtspunkten. Die Motivation für diese Strukturierung ist die Vereinfachung des Austausches von MFMs gegen andere mit gleicher oder ähnlicher Funktion, aber möglicherweise vollständig unterschiedlicher technischer Ausprägung.

Zur Strukturierung mechatronischer Systeme wurden neben der untersten Strukturebene der Mechatronischen Funktionsmodule (MFM) die drei höheren Ebenen Mechatronische Funktionsgruppe (MFG), Autonome Mechatronische Systeme (AMS) und Vernetzte Mechatronische Systeme (VMS) eingeführt. Ein Vorteil der Strukturierung in MFM, MFG, AMS und VMS liegt in der klaren Abgrenzung der einzelnen Funktionsmodule gegeneinander und der daraus resultierenden Schnittstellendefinition. Die einzelnen Funktionen können weitgehend unabhängig voneinander realisiert werden. Damit ist eine leichte Austauschbarkeit einzelner Bausteine möglich:

MFM: Ein mechatronisches Funktionsmodul ist eine lokale, austauschbare Kombination von Aktorik, Sensorik und Tragstrukturen mit digitaler Informationsverarbeitung. Ein MFM bekommt eine bestimmte Aufgabe innerhalb eines mechatronischen Gesamtsystems, insbesondere die Herstellung einer Solldynamik. Ein MFM verfügt über physikalische und informationstechnische Schnittstellen. MFMs können hierarchisch zu übergeordneten MFMs kombiniert werden.

MFG: Eine mechatronische Funktionsgruppe (MFG) ist einem oder mehreren MFMs oder MFGs ausschließlich informationstechnisch übergeordnet. Das MFG besitzt selbst keine Tragstruktur und keine Aktorik, dafür Informationsverarbeitung und ggf. Sensorik. Die Aufgabe eines MFG ist die Strukturierung der Informationsverarbeitung in MFM, AMS oder VMS. Beispiele sind aktive Fahrwerks-Systeme (z. B. Active Body Control), ESP-Systeme (Elektronisches Stabilitäts-Programm) etc.

AMS: Ein oder mehrere MFGs und MFMs bilden ein Autonomes Mechatronisches System (AMS). Dem AMS ist die zentrale mechanische Tragstruktur eines mechatronischen Systems zugeordnet. Die Informationsverarbeitung des AMS ist in Form von MFGs organisiert. Ein weiteres wichtiges Merkmal eines AMS ist das Vorhandensein einer eigenen Energiequelle. Diese stellt die Energie für die zugeordneten MFMs und damit für die Aktoren sowie die Sensoren zur Verfügung.

VMS: Durch die Kombination mehrerer AMSs kann die höchste Abstraktionsstufe durch die Bildung eines Vernetzten Mechatronischen Systems (VMS, engl. CMS: Cross-Linked Mechatronic System) erreicht werden. Ein VMS bietet ausschließlich eine Schnittstelle zur informationstechnischen Kopplung von AMSs. Weiterhin hat die Informationsverarbeitung im VMS hauptsächlich diskreten Charakter. VMS können in sich weiter hierarchisiert werden. Eine Fahrzeugkolonne wird z. B. als VMS0-Ebene bezeichnet, falls eine weitere übergeordnete Ebene, wie z. B. ein Kreuzungsmanagement (VMS1), existiert.

Wesentliches Kriterium für die Zuordnung der Teilsysteme eines mechatronischen Systems zu einer der vier Ebenen sind die in dem System enthaltenen Komponenten:

TABELLE 1. Zuordnung von Grundbausteinen zu Hierarchien mechatronischer Systeme

Hierarchie	Tragstruktur	Sensorik	Aktorik	Informations-verarbeitung
VMS		X		X
AMS	X	X		X
MFG		X		X
MFM	X	X	X	X

Am Beispiel des am MLaP entwickelten Fahrzeugmodells X-Mobile [Zanella et al. 2001] lässt sich das Strukturierungskonzept sehr anschaulich nachvollziehen (vgl. Abbildung 3-1). Beim X-Mobile handelt es sich um ein voll mechatronisches Fahrzeugmodell im Maßstab 1:8. Hier wurden Konzepte für ein aktives Fahrwerk, Einzelradantrieb und Einzelradlenkung umgesetzt. Jedes der vier Radmodule des X-Mobile besteht aus drei MFMs mit je einem Elektromotor als Aktor: Lenkung, Federung, Antrieb. Diese drei MFMs sind mechanisch gekoppelt und bilden das übergeordnete MFM Radmodul. Die Führungsgrößen der MFMs auf der untersten Ebene werden durch die Mechatronische Funktionsgruppe (MFG) auf der Basis der aktuellen Fahrsituation ermittelt. Die MFG-Ebene implementiert damit eine rein informationstechnische Kopplung der Lenkung, des Antriebs und des aktiven Fahrwerks. Die vier Radmodule werden durch den Fahrzeugaufbau auf der AMS-Ebene mechanisch gekoppelt. Hier findet sich auch die Informationsverarbeitung, die entweder autonom oder ferngesteuert über Wunsch-Fahrtrichtung und -Fahrbeschleunigung entscheidet. Verknüpft man mehrere Fahrzeuge z. B. mit Hilfe einer

Abstandssensorik zu einer Fahrzeugkolonne, so entsteht ein vernetztes mechatronisches System (VMS):

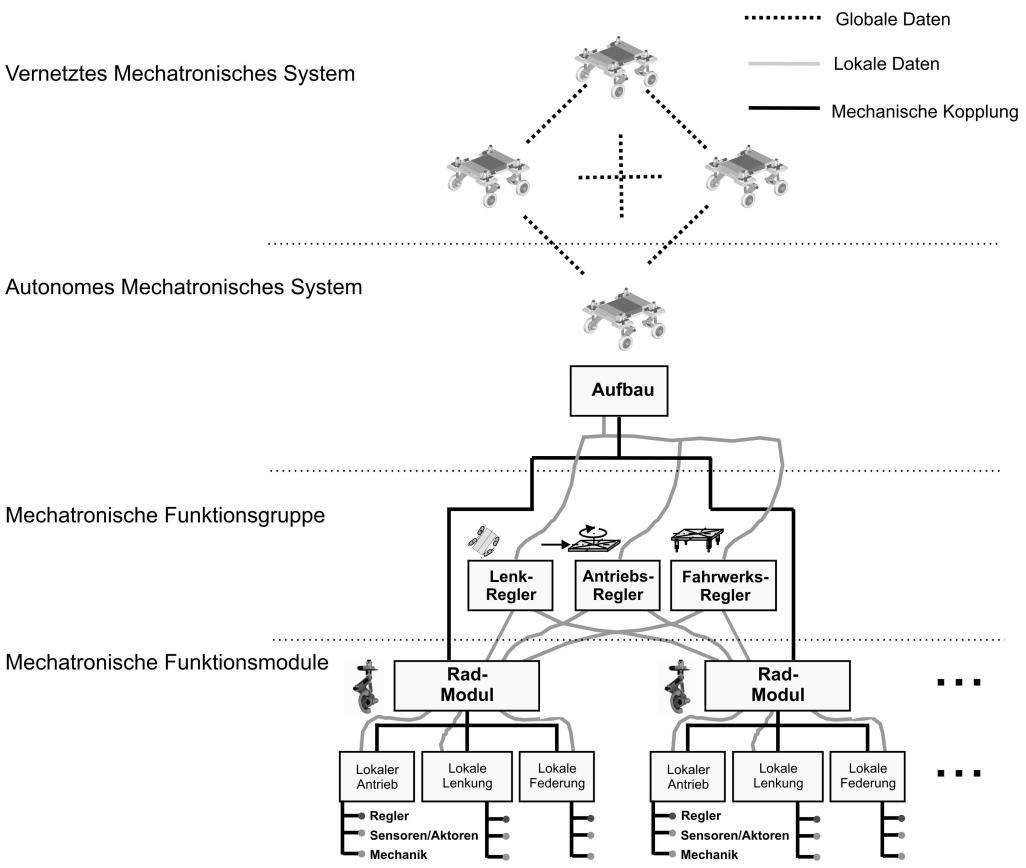

Abbildung 3-1: Strukturierungsebenen am Beispiel des X-Mobile

3.2 Operator-Controller-Modul

Für den Entwurf von übergeordneten Kontroll- und Optimierungsfunktionen des dynamischen Verhaltens mechatronischer Systeme wurde am MLaP das sogenannte Operator-Controller-Modul (OCM**)** konzipiert und weiterentwickelt [DFG-Antrag zum SFB 614 2001]. Das OCM lässt sich prinzipiell in zwei Bereiche einteilen: den "Operator" und den "Controller". Der Operator beinhaltet die diskreten Elemente der Informationsverarbeitung, wie Notfall-Routinen, Regler-Überwachung und die Optimierung. Der Controller umfasst die kontinuierlichen, regelnden Teile der Informationsverarbeitung. Während der Controller im Wesentlichen auf der "quasi-kontinuierlichen" digitalen Regelungstheorie basiert, fußt der Operator auf diskreter Logik. Implementierungstechniken umfassen z. B. Zustandsmaschinen und prozessbasierte Systeme wie z. B. Work-Flow-Diagramme.

Die Funktionen "Überwachung" und "Übertragung" verbinden Operator und Controller und formen mittels spezieller Routinen kontinuierliche Signale in diskrete Signale um und umgekehrt. Die Überwachung von Zustandsvariablen des zeitkontinuierlichen Reglers und der physikalischen Strecke ist eine weitere wichtige Aufgabe, welche die Stabilität des Gesamtsystems bei Parameterwechseln in den Reglern oder beim Austausch ganzer Reglerstrukturen sichert.

3.3 Erweitertes Streckenmodell

Grundlage für die modellbasierte Optimierung im OCM ist das sogenannte *erweiterte Streckenmodell* [Kasper 1985], das zusätzlich zum Modell der geregelten Strecke (Regelstrecke mit Controller) auch Modelle zur Anregung und zur Bewertung enthält. Damit liefert das erweiterte Streckenmodell Modelle der Realität für alle Elemente, die zur modellbasierten Optimierung herangezogen werden. Für jedes Element eines mechatronischen Systems (MFM, AMS, VMS) kann ein eigenes erweitertes Streckenmodell verwendet werden. Die nachfolgende Abbildung zeigt das zugehörige schematische Blockdiagramm für ein Element:

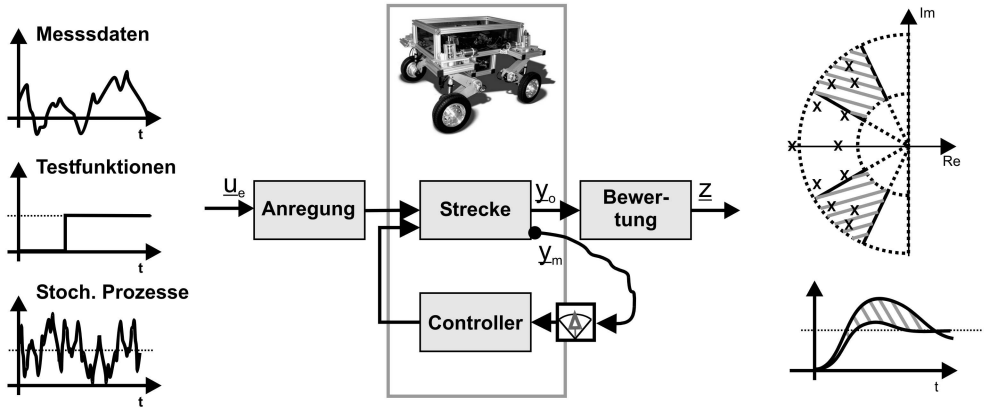

Abbildung 3-2: Erweitertes Streckenmodell mit Anregungs- und Bewertungsmodell

3.3.1 Anregungsmodell

Das Anregungsmodell dient als Bindeglied zwischen mathematisch formulierten Anregungsfunktionen und den physikalischen Eingängen des Streckenmodells. Es modelliert alle äusseren Eingänge in das System. Es wird zwischen Führungseingängen (Sollwertvorgaben) und Störeingängen (nicht beeinflussbaren Eingängen) unterschieden.

Nachfolgende Abbildung zeigt ein Beispiel für die mögliche Modellierung von Strassenprofil-Unebenheiten zur Anregung von (Vertikaldynamik-)Streckenmodellen. Am Anregungs-Eingang u_e wird weisses Rauschen eingespeist und durch das Anregungsmodell "Tiefpassfilter" in bandbegrenztes Rauschen mit angepasster Amplitude überführt. Der Ausgang z_s entspricht der modellierten vertikalen Strassenprofil-Auslenkung und wird als physikalisch interpretierbare Größe in

das Streckenmodell eingespeist. Je nach Anwendungsfall werden auf den Anregungsmodell-Eingang $\underline{u_e}$ Testfunktionen (Impuls, Sprung, Rampe, Sinus etc.), Messdaten oder stochastische Signale gegeben:

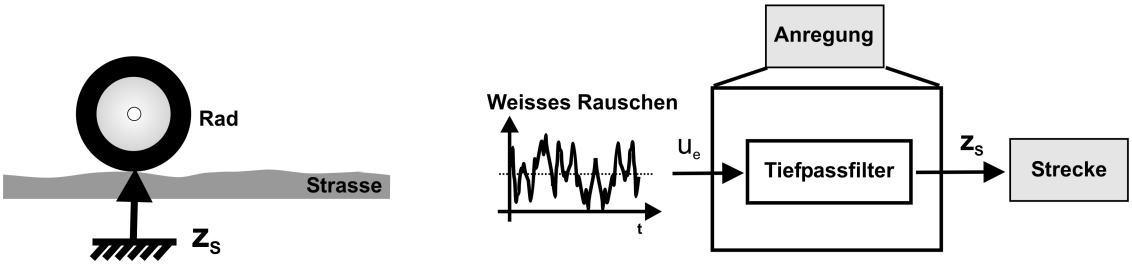

Abbildung 3-3: Beispiel für ein Anregungsmodell: Unebenheit eines Fahrbahnprofils

3.3.2 Bewertungsmodell

Die Aufgabe des Bewertungsmodells ist die Berechnung des Zielgrößenvektors $\underline{z}$. Die einzelnen Zielgrößen sollten Masszahlen zur Bewertung der folgenden wichtigen Eigenschaften mechatronischer Systeme sein:

- Stabilität
- Einschwingverhalten (Zeitkonstanten, Dämpfungen, Eigenfrequenzen)
- Führungs- und Störverhalten
- Robustheit gegen Parameteränderungen

Berechnung von Zielgrößen durch Zielfunktionen

Jede Zielgröße wird durch eine zugehörige Zielfunktion z_i berechnet. Für den hier betrachteten Fall der Parameter-Optimierung/-Variation ist jede Zielfunktion abhängig vom Strecken- und Controller-Parametervektor $\underline{p}$:

$$\underline{z}(\underline{p}) = \left[z_1(\underline{p}) \ \ldots \ z_{n_z}(\underline{p}) \right]^T \tag{3.1}$$

Durch direkte Zielfunktionen werden Systemeigenschaften inhärent bewertet und durch vergleichende Zielfunktionen Aussagen anhand von modelliertem Wunschverhalten abgeleitet.

Die für die direkten Zielfunktionen verwendeten Analyseverfahren unterscheiden sich für lineare und nichtlineare Systeme.

Vergleichende Zielfunktionen werden durch Integralfunktionen zur Berechnung von Fehlerflächen formuliert. Dabei wird der Fehler zwischen Soll- und Istkurven im Zeit- oder im Frequenzbereich integriert und gewichtet (vgl. Anhang Kap. 9.3).

Nachfolgend eine Übersicht über die prinzipielle Bildung von Zielfunktionen:

TABELLE 2. Möglichkeiten zur Bildung von Zielfunktionen

	Für lineare Systeme	**Für nichtlineare Systeme**
Direkte Zielfunktionen ohne Vergleichs-modelle	- Einhaltung von Begrenzungen - Kovarianzanalyse zur Bestimmung von mittleren Leistungen - Lage der Eigenwerte (Stabilität, Dämpfung, Eigenfrequenzen)	- Einhaltung von Begrenzungen - Stochastische Berechnungen (Mittelwerte, Varianzen etc.)
Vergleichende Zielfunktionen mit modelliertem Wunschverhalten	- Berechnung von zeitgewichteten Fehlerflächen oder Maximalabweichungen zwischen Ist- und Wunschverhalten auf der Basis von Zeitsignalen - Berechnung von frequenzgewichteten Fehlerflächen zwischen Ist- und Wunschverhalten auf der Basis von Spektren oder Frequenzkennlinien	

3.4 Verallgemeinerte Kaskade

Für den Entwurf von *hierarchischen* Regelsystemen gemäß dem VMS-/AMS-/MFG-/MFM-Strukturierungskonzept wird am MLaP das Prinzip der „verallgemeinerten Kaskade" ([Lückel et al. 2001], [Lückel et al. 2002/1]) vorgeschlagen. Die „verallgemeinerte Kaskade" kombiniert Elemente der dezentralen Regelung und der Kaskadenregelung.

Jedes mechatronische Element (MFM, MFG, AMS oderVMS) enthält eine separate Informationsverarbeitung. Dabei kann eine höhere Ebene mehrere Teilregler ansteuern, aber nicht mehrere höhere Ebenen dieselbe Subebene. Die Kommunikation zwischen den Ebenen erfolgt von oben nach unten.

Wie bei einer klassischen Kaskade [Föllinger 1994] nehmen die Zeitkonstanten der Regelkreise mit abnehmender Hierarchiestufe in der Struktur ab. Zudem verschiebt sich die Problematik in der Reglerauslegung von der Mannigfaltigkeit der Regelgrößen und Zustände hin zu Nichtlinearitäten im System. Entsprechend kommen auf hohen Ebenen Verfahren der Mehrgrößenregelung im Zeitbereich zum Einsatz, während auf den unteren Ebenen Techniken des Frequenzbereichs für Eingrößensysteme verwendet werden sollen. Da die so entstehende Struktur insbesondere durch ihre Hierarchie gekennzeichnet wird, bezeichnet man sie als „verallgemeinerte Kaskade“.

4 Grundlagen der Mehrziel-Parameter-Optimierung

Viele reale Optimierungsaufgaben erfordern es, mehrere Zielgrößen gleichzeitig zu minimieren bzw. zu maximieren. Dabei widerstreben üblicherweise die Zielgrößen einander, so dass niemals alle Zielgrößen mit einem Parametervektor gleich gut verbessert werden können. Daher kann die Lösung des Mehrziel-Optimierungsproblems nur ein möglichst guter Kompromiss sein. Die Mehrziel-Optimierung hat ihre Wurzeln im späten 19. Jahrhundert im Bereich der Wirtschaftswissenschaften durch die Arbeiten von Edgeworth[1] und Pareto[2].

4.1 Problemdefinition

Im Folgenden wird davon ausgegangen, dass Zielgrößen stets minimiert werden, da sich eine Maximierung durch Umkehrung des Vorzeichens als Minimierung beschreiben lässt. Sämtliche Forderungen und Wünsche, die vom Anwender an das Verhalten des zu optimierenden Systems zu richten sind, werden in dem n_z-dimensionalen Vektor der Zielfunktionen $\underline{z}$ zusammengefasst. Die zur Verfügung stehenden Entwurfsfreiheitsgrade legen den Raum der freien Parameter fest, zusammengefasst im Parametervektor $\underline{p}$ mit der Dimension n_p:

$$\underline{p} = \left[p_1\ p_2\ \cdots\ p_{n_p}\right]^T \tag{4.1}$$

Die Optimierungsaufgabe besteht nun darin, die Komponenten von $\underline{p}$ über den gesamten Verlauf der Optimierung so anzupassen, dass jede einzelne Komponente des Zielvektors $\underline{z}(\underline{p})$ am Ende der Optimierung möglichst minimale Werte mit einem Parametersatz $\underline{p}^*$ erreicht:

$$min(\underline{z}(\underline{p})) = \begin{bmatrix} z_1(\underline{p}^*) \\ z_2(\underline{p}^*) \\ \cdots \\ z_{n_z}(\underline{p}^*) \end{bmatrix} \tag{4.2}$$

Dabei sind für praktische Anwendungen Randbedingungen in Form von Ungleichungen und expliziten Variablengrenzen für $\underline{p}$ einzuhalten:

$$\underline{p} \in \underline{C} \tag{4.3}$$

$$\underline{C} = \{\underline{p} : \underline{h}(\underline{p}) \leq \underline{0},\ \underline{a} \leq \underline{p} \leq \underline{b}\} \tag{4.4}$$

Die skalare Definition von Optimalität greift bei Mehrziel-Problemen nicht mehr. Stattdessen wird üblicherweise die sogenannte Pareto-Optimalität herangezogen.

1. Francis Edgeworth - Irischer Wirtschaftswissenschaftler 1845-1926
2. Vilfredo Pareto - Italienischer Wirtschaftswissenschaftler 1848-1923

4.2 Pareto-Optimalität

Ein Punkt $\underline{p}^* \in \underline{C}$ wird Pareto-optimal [Pareto 1971] genannt, wenn es keinen anderen Punkt $\underline{p} \in \underline{C}$ gibt, der die Bedingung $z_i(\underline{p}) \leq z_i(\underline{p}^*)$ für alle $i \in \{1, 2, \ldots, n_z\}$ erfüllt. Pareto-optimale Punkte werden auch als *nicht-dominierte* Punkte bezeichnet. Die zu einem Pareto-optimalen Punkt im Parameterraum gehörenden Punkte $z_i(\underline{p}^*)$ des Zielgrößenraumes werden *effiziente* Punkte genannt.

Das Abweichen von einem Pareto-optimalen bzw. effizienten Punkt hat also immer eine Verschlechterung mindestens einer Zielgröße zur Folge. Da Punkte, die keine weitere Verbesserung zulassen, immer Elemente der Pareto-Menge sind, ist es das Ziel einer Optimierung von mehreren Zielgrößen, als Ergebnis stets einen Pareto-optimalen Punkt zu liefern. Typischerweise formen die Pareto-Punkte Linien bzw. Flächen im Parameter- bzw. Zielgrößenraum. Ganz allgemein spricht man von einer Pareto-Menge oder einem Pareto-Set:

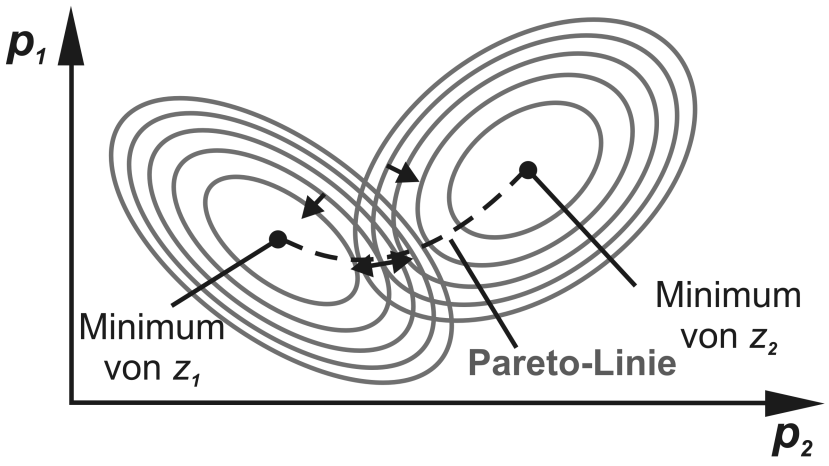

Abbildung 4-1: Höhenlinien zweier Zielgrößen im zweidimensionalen Parameterraum

Abbildung 4-1 zeigt den Verlauf der Höhenlinien zweier kontinuierlicher Zielfunktionen $z_1(p_1, p_2)$ und $z_2(p_1, p_2)$ in einem zweidimensionalen Parameterraum. Die Gradienten der Zielfunktionen stehen senkrecht auf den Höhenlinien. Die gestrichelte Linie beschreibt nun all die Punkte, bei denen die Gradienten der beiden Zielgrößen einen Winkel von 180° zueinander bilden. Diese entgegengesetzte Richtung der Gradienten hat zur Folge, dass man nicht beide Zielgrößen gleichzeitig weiter minimieren kann.

4.3 Karush-Kuhn-Tucker-Bedingung

Die notwendige Bedingung für Pareto-Optimalität von Kuhn und Tucker [Kuhn, Tucker 1951] bildet eine Verbindung zwischen skalarer und vektorieller Optimierung. Dazu wird zunächst die skalare Funktion g_α definiert:

$$g_\alpha(\underline{p}) = \sum_{i=1}^{n_z} \alpha_i \cdot z_i(\underline{p}) \tag{4.5}$$

$$\sum_{i=1}^{n_z} \alpha_i = 1 \tag{4.6}$$

Die Karush-Kuhn-Tucker-Bedingung erster Ordnung eines unbeschränkten Optimierungsproblems (keine Randbedingungen) für die Pareto-Optimalität eines Punkts $\underline{p}^*$ lautet:

$$\nabla g_\alpha(\underline{p}^*) = \sum_{i=1}^{n_z} \alpha_i \cdot \nabla z_i(\underline{p}^*) = 0 \tag{4.7}$$

In gewisser Weise basiert das Verfahren der gewichteten Summe (vgl. Kap. 4.6.1) auf diesen Ergebnissen von Kuhn und Tucker. Der Gewichtungsvektor $\underline{\alpha}$ kann dabei geometrisch als Koordinatensystem für die Suchrichtung interpretiert werden. Gleiche Werte für g_α bei konstantem Gewichtungsvektor $\underline{\alpha}$ liegen auf einer Geraden, die orthogonal zum Gewichtungsvektor verläuft [Hillermeier 2001] (vgl. Abbildung 4-2).

Allerdings erhält man durch die Suche der Minima von g_α nicht alle Pareto-Punkte des Mehrziel-Optimierungsproblems. Neben den Minima von g_α können auch dessen Sattelpunkte Pareto-optimale Punkte sein. Hierin liegt ein wesentlicher Unterschied zwischen der vektoriellen und der skalaren Optimierung [Hillermeier 2001].

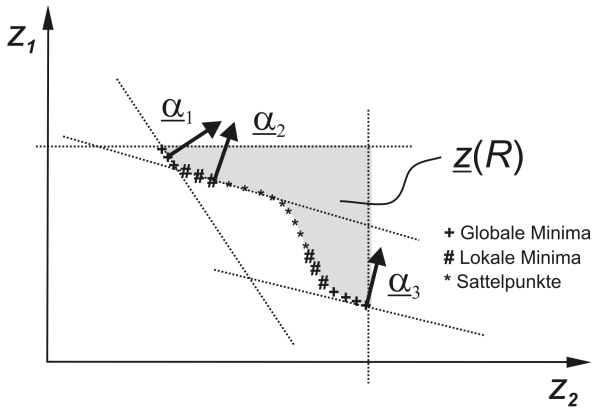

Abbildung 4-2: Geometrische Interpretation des Gewichtungsvektors $\underline{\alpha}$

4.4 Klassifizierung von Optimierungsaufgaben

Sowohl Ein- als auch Mehrziel-Optimierungsverfahren lassen sich hinsichtlich Zielsetzung, Zielfunktionseigenschaften und Algorithmik klassifizieren. Je nachdem, ob alle oder einige Paretooptimale Punkte eines Problems ermittelt werden sollen, werden globale und lokale Verfahren unterschieden. Im Bereich der Algorithmik unterscheidet man zwischen deterministischen und heuristischen Verfahren. Die Einbeziehung von Nebenbedingungen in das Optimierungsproblem

kann implizit oder explizit erfolgen:

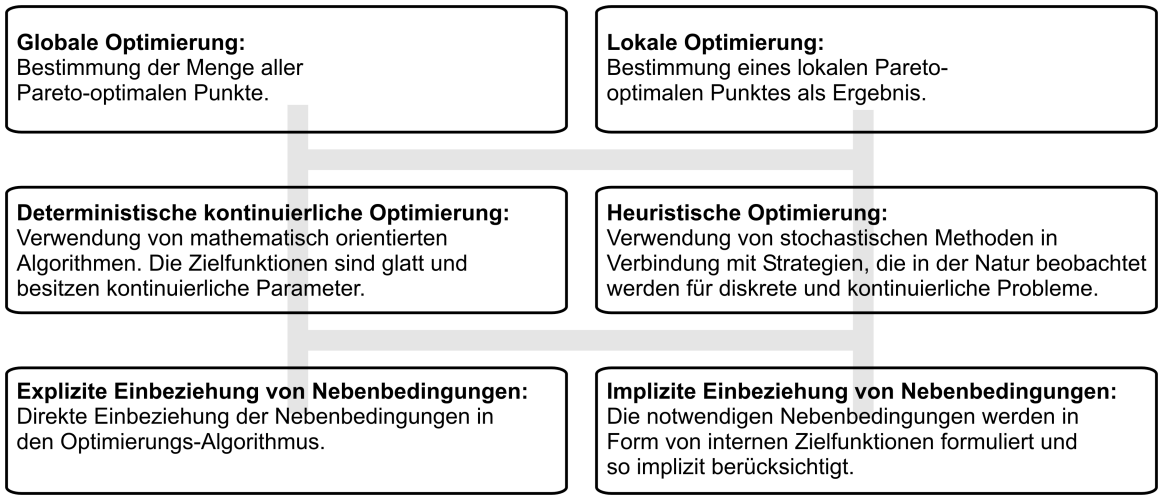

Abbildung 4-3: Verknüpfung verschiedener Eigenschaften von Optimierungsverfahren

4.5 Numerische Lösungsverfahren zur Minimierung von skalaren Funktionalen

Numerische Verfahren zur Minimumsuche haben daher für praktische Anwendungen eine hohe Relevanz. Die Minimierung von skalaren Funktionalen der Form $z(\underline{p})$ bildet für die Mehrziel-Optimierung eine wichtige Grundlage, da sie in Zwischenschritten auf die skalaren Elemente des Zielgrößenvektors $\underline{z}(\underline{p})$ angewendet werden.

Sehr häufig verwendet werden die folgenden fünf Verfahren, die in der Literatur ausführlich beschrieben sind [Entenmann 1976]:

1. **Eindimensionale Minimumsuche** für Funktionen $z(p)$ mit einem skalaren Parameter:
   Iteration von p so lange sich $z(p)$ in jedem Schritt verkleinert; die Zielfunktion wird nicht differenziert; Sonderfall des Koordinatensuchverfahrens.

2. **Koordinatensuchverfahren**:
   Jede Komponente von $\underline{p}$ wird als Koordinate aufgefasst und einzeln ausgelenkt; die Zielfunktion wird nicht differenziert. Die Konvergenz ist ungünstig und der Aufwand ist hoch, da in der Nähe des Optimums viele Zielfunktionsauswertungen notwendig sind.

3. **Gradientenverfahren**:
   Suche in Richtung des steilsten Abstiegs auf der Basis der ersten Ableitung der Zielfunktion. Das Verfahren hat lineare Konvergenz und ähnlich hohen Aufwand in der Nähe des Optimums wie das Koordinatensuchverfahren. Zusätzlicher Aufwand entsteht durch die Gradientenberechnung.

4. **Gradientenverfahren mit konjugierten Richtungen**:
   Gradientenverfahren mit Abfolgen von Suchrichtungen, die jeweils konjugiert zueinander sind (d. h. neue und alte Suchrichtung sind orthogonal bezüglich einer Matrix). Die Konvergenz ist quadratisch in der Nähe der Lösung. Der Aufwand ist mittel, da die Hesse-Matrix nicht bestimmt werden muß.

5. **Newton-Verfahren**:
   Bei diesem Verfahren wird neben der ersten Ableitung auch die zweite Ableitung der Zielfunktion herangezogen. Die Konvergenz ist quadratisch in der Nähe der Lösung. Der Aufwand ist hoch da Gradienten und die Hesse-Matrix numerisch ermittelt werden müssen.

4.6 Lösungsverfahren für Mehrziel-Optimierung

Nachfolgend werden zunächst deterministische Verfahren zur Mehrziel-Optimierung beschrieben (vgl. Abbildung 4-4):

4.6.1 Gewichtete Summe

Eine Standardtechnik zur Lösung von Mehrziel-Optimierungsproblemen ist die Reduktion auf eine einzelne Zielgröße durch Bildung einer positiv gewichteten Summe aus allen Zielfunktionen (vgl. Kap. 4.3). Neuere Verfahren verwenden dazu mehrere Sätze von Gewichtungsfaktoren [Das, Dennis 1996/2]. Es ist zwar leicht nachweisbar, dass die Minimierung der gewichteten Summe Pareto-optimale Punkte liefert, aber bei der Anwendung des Verfahrens gibt es große Nachteile:

1. Die Minimierung der gewichteten Summe liefert eine Hyperebene, von der i. A. nur ein Punkt zur Pareto-Menge gehört, den man aber zunächst nicht kennt. Erst durch die Verwendung mehrerer Sätze von Gewichtungsfaktoren und Bildung der Einhüllenden erhält man die Pareto-Menge.

2. Die Gewichtungsfaktoren sind für Anwendungen der Mechatronik physikalisch nicht interpretierbar und daher schwer beherrschbar.

3. Auch Sattelpunkte der gewichteten Summe können Pareto-optimale Punkte sein. Diese Punkte werden vom Verfahren nicht berücksichtigt.

4.6.2 Gewichtungsverfahren mit Lp-Metrik

Eine Variante der Gewichtungsverfahren ist die Verwendung einer gewichteten Vektornorm. Bei diesem Verfahren wird ein effizienter Punkt $z_i(\underline{p}^*)$ ausgewählt, an den man sich so gut wie möglich annähern möchte. Den Abstand zu diesem Punkt gewichtet man mit einer Vektornorm. Man erhält wiederum ein skalares Funktional mit Gewichtungsfaktoren. Der gewünschte effiziente Punkt, der Gewichtungsvektor und die Ordnung der verwendeten Vektornorm bilden die Parameter dieses Verfahrens. Leider kann keine universelle Aussage über die Bedingungen zur Variation der Verfahrensparameter zur Erzeugung effizienter Lösungen getroffen werden [Göpfert, Nehse 1990].

4.6.3 'Epsilon-constraint'-Methode

Diese Methode geht ursprünglich auf Marglin [Marglin 1967] und Haimes [Haimes 1973] zurück. Die Grundidee besteht in der Auswahl der wichtigsten Zielgröße, die minimiert werden soll. Für alle anderen Zielgrößen müssen nur obere Schranken als Vorgaben eingehalten werden. Diese Methode ist daher nur sinnvoll, wenn beim Optimierungsproblem eine Zielgröße klar im Vordergrund steht. Das Bestimmen von Schranken kann Schwierigkeiten bereiten. Bei mehr als zwei Zielgrößen kann die Pareto-Menge nicht mehr effizient bestimmt werden.

4.6.4 Multilevel Programming

Beim Multilevel Programming wird nur ein Punkt der Pareto-Fläche gesucht. Grundidee ist die Sortierung der Zielgrößen nach Prioritäten. Anschließend wird ein Satz von Parametervektoren gesucht, der die erste Zielgröße minimiert. Aus diesem Satz werden die Parametervektoren ausgewählt, die auch die zweite Zielgröße minimieren. Dieses Schema wird rekursiv bis zur letzten Zielgröße angewandt. Multilevel Programming ist nur dann sinnvoll anwendbar, wenn die Minimierung der wichtigsten Zielfunktionen im Vordergrund steht und kein Kompromiss für alle Zielfunktionen gefunden werden muss. Oft sind die Randbedingungen für die nachrangigen Ziele so eingeschränkt, dass sie gar nicht berücksichtigt werden können.

4.6.5 Normal-Boundary Intersection

Normal-Boundary Intersection (NBI) ist ein weiterer Vertreter der deterministischen Verfahren. NBI fasst die Zielfunktionen zu gewichteten Summen zusammen, deren Gewichtungen mit Hilfe einer geometrisch motivierten Methode ausgewählt werden [Das, Dennis 1996/1]. Durch Lösen dieser Gewichtungen mit Hilfe eines herkömmlichen Eingrößenverfahrens werden verschiedene Punkte der Paretomenge berechnet. Dabei wird eine gleichmäßige Verteilung der Punkte auf der Pareto-Fläche erreicht. NBI ist in der Lage, auch Probleme mit verrauschten oder diskontinuierlichen Pareto-Flächen zu lösen. Für mehr als zwei Zielfunktionen und nicht-konvexe Zielfunkti-

onsgebiete liefert das Verfahren jedoch nicht zwangsläufig Pareto-optimale Punkte als Ergebnis:

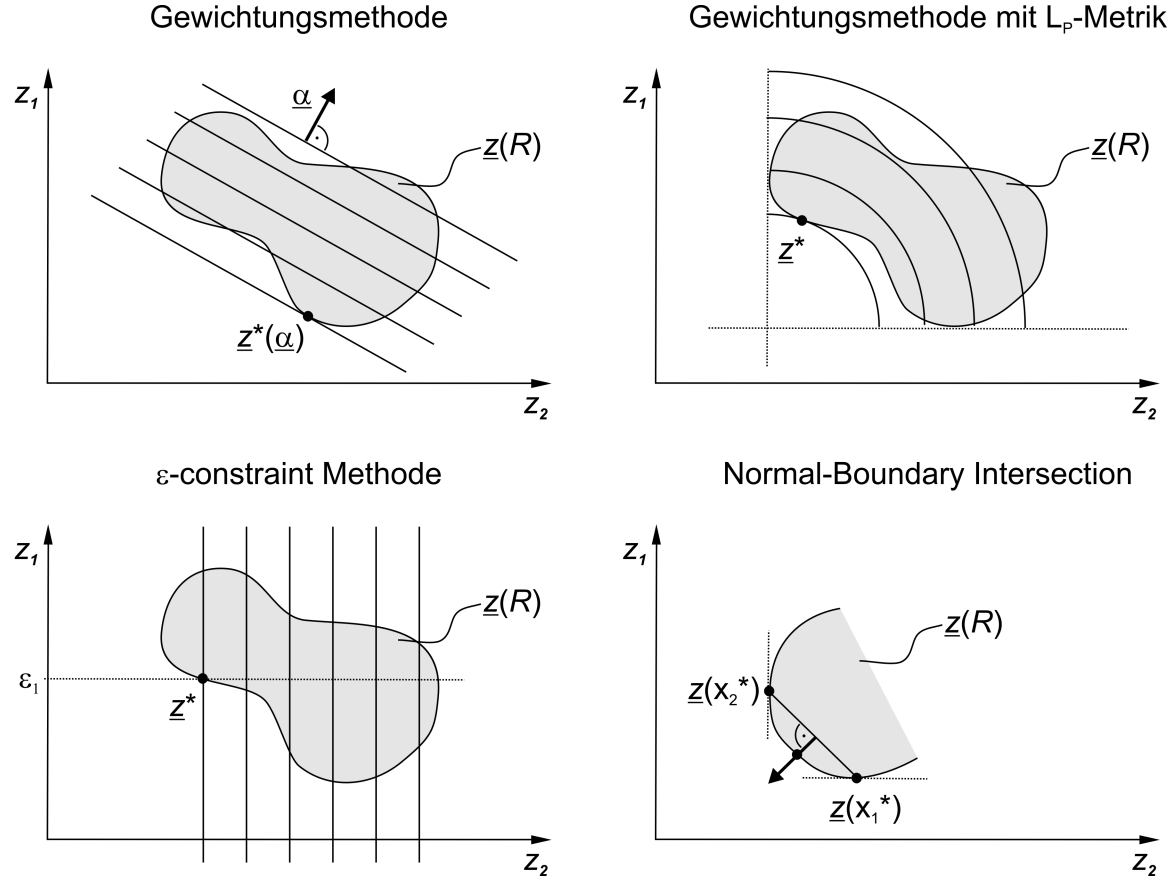

Abbildung 4-4: Geometrische Interpretation einiger Mehrziel-Optimierungsverfahren

4.6.6 Gütevektoroptimierung

Das Verfahren der Gütevektoroptimierung [Kreisselmeier, Steinhauser 1979] ist ein numerisches Verfahren zur simultanen Verkleinerung mehrerer Gütemaße (Zielfunktionen). Die Bezeichnung rührt daher, dass man mehrere skalare Gütemaße zu einem Vektor zusammenfassen kann, den man dann komponentenweise verkleinert (siehe auch [Föllinger 1994]).

Ausgehend von einem beliebigen Parametervektor $\underline{r}_0$ als Startpunkt wählt man die Vorgabewerte $C_i^{(0)}$ mit $C_i^{(0)} > \underline{z}_i(\underline{r}_0)$ (vgl. Abbildung 4-5). Durch Minimierung der nachfolgenden Maximumfunktion können sämtliche Gütemaße in neue engere Schranken $C_i^{(1)}$ eingeschlossen werden:

$$\alpha^{(0)}(\underline{r}) = max\left[\frac{z_1(\underline{r})}{C_1^{(0)}} \ \ldots \ \frac{z_{n_z}(\underline{r})}{C_{n_z}^{(0)}}\right] \tag{4.8}$$

Durch Iteration dieses Vorgehens mit $\underline{z}_i(\underline{r}_1) < C_i^{(1)} < C_i^{(0)}$ wird der Algorithmus weitergeführt, bis die Zielfunktionen hinreichend klein sind oder eine Verringerung der $C_i^{(k)}$ nicht mehr erreicht werden kann.

Für die praktische Durchführbarkeit wird die nicht differenzierbare Maximumfunktion (Gl. 4.8) durch eine analytische Funktion approximiert, so dass die Minimumsuche mit einem Gradienten-

verfahren ermöglicht wird:

$$\hat{\alpha}(\underline{r}) = \frac{1}{\rho} \cdot ln \sum_{i=1}^{n_z} e^{\rho \cdot \frac{z_i(\underline{r})}{C_i}} \tag{4.9}$$

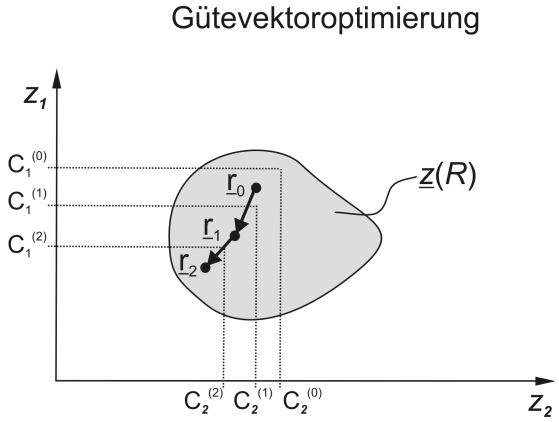

Abbildung 4-5: Geometrische Deutung der Gütevektoroptimierung

4.6.7 Homotopie-Verfahren

Das ursprüngliche Verfahren wird von Rakowska [Rakowska et al. 1991] beschrieben. Das klassische Homotopie-Verfahren ist eigentlich zur Lösung nichtlinearer Gleichungen gedacht [Allgower, Georg 1990]. Es basiert auf der Idee, eine Verbindung zwischen zwei Gleichungssystemen zu finden. Die Lösung des Gleichungssystems $\underline{H_0(y)} = \underline{0}$ ist dabei bekannt, während die Lösung von $\underline{G(y)} = \underline{0}$ gesucht wird. Eine Verbindung zwischen beiden kann z. B. durch $\underline{H(y,t)} = \underline{0}$ geschaffen werden (Homotopie-Parameter t):

$$\underline{H}(\underline{y}, t) = t \cdot \underline{G}(\underline{y}) + (1 - t) \cdot \underline{H}_0(\underline{y}) \tag{4.10}$$

Im Zusammenhang mit der Mehrziel-Optimierung wird dieses Verfahren zur Erzeugung von Nachbarschaften, ausgehend von Kandidaten für Pareto-Punkte, verwendet. Das ursprüngliche Verfahren ist auf zwei Zielfunktionen beschränkt und liefert nur diejenigen Pareto-optimalen Punkte, die Minima der konvexen Kombination der Zielfunktionen sind. Eine Erweiterung des Verfahrens [Hillermeier 2001] berücksichtigt eine beliebige Anzahl von Zielfunktionen. Es handelt sich um ein globales Verfahren, für das die Zielfunktionen zweimal stetig differenzierbar sein müssen.

4.6.8 Stochastische Verfahren

Mit Hilfe stochastischer Verfahren wird nach den globalen effizienten Lösungen eines Mehrziel-Optimierungsproblems gesucht. Die Anwendung dieser Verfahren ist nur dann sinnvoll, wenn die Dauer der Berechnungen nicht kritisch ist, da im Allgemeinen die Anzahl der Zielfunktionsauswertungen sehr hoch ist. Die Verfahren arbeiten mit iterativ veränderten Suchbereichen, in denen

die Zielfunktionen für stochastisch erzeugte Parametervektoren ausgewertet werden. Stochastische Verfahren sind für nicht differenzierbare Zielfunktionen geeignet; es gibt aber auch Varianten, die Gradienteninformationen einbeziehen [Timmel 1980].

4.6.9 Heuristische Verfahren

Heuristische Verfahren basieren oft auf Strategien, die in der Natur beobachtet werden. Hierbei sind insbesondere die auf genetischen Algorithmen basierenden Verfahren zu nennen. Auf eine "Population" werden genetische Operationen, etwa Rekombination, Mutation oder Selektion, angewendet. Es existieren eine Vielzahl von Verfahren, die nach diesem Prinzip aufgebaut sind [Fonseca, Fleming 1995].

Die beiden Verfahren Ant Colony Optimization (ACO) für diskrete Probleme [Dorigo, Gambardella 1997] und Particle Swarm Optimization (PSO) für kontinuierliche bzw. gemischt diskret-kontinuierliche Probleme [Coello, Lechunga 2002] gehören zu einer neuen Klasse von heuristischen Optimierungsverfahren, die auf dem kollektiven Verhalten von Schwärmen beruhen.

4.7 MLaP-Optimierungsverfahren MOPO

Am MLaP wird seit vielen Jahren an der Optimierungssoftware MOPO (Multi-Objective Parameter Optimization) gearbeitet ([Kasper 1985], [Lückel et al. 1985], [Kasper et al. 1990], [Jäker 1991]). Das Besondere an MOPO ist die konsequente Erhaltung der physikalisch interpretierbaren Einzelkriterien. Dieses Vorgehen wird z. B. auch im Verfahren der Gütevektoroptimierung ([Föllinger 1994], [Kreisselmeier, Steinhauser 1979]) angewandt. Zwischen MOPO und diesem Verfahren bestehen einige Ähnlichkeiten, jedoch ist MOPO für die praktische Anwendbarkeit besser geeignet (Verfahrenssteuerung/Konfiguration).

MOPO wird als instrumentelles Verfahren bezeichnet, bestehend aus kombinierten Optimierungsalgorithmen (Gradientenverfahren, eindimensionale Minimumsuche etc.), und wurde bislang für eine Vielzahl von Anwendungen im mechatronischen Umfeld erfolgreich eingesetzt. Bei MOPO handelt es sich um ein lokales, deterministisches Verfahren, das prinzipiell für eine Online-Verarbeitung geeignet ist.

Das MOPO-Verfahren wurde am MLaP im Rahmen des DFG-Sonderforschungsbereich 376 "Massive Parallelität - Algorithmen, Methoden, Anwendungen" im Team des Teilprojekts C1 beständig weiterentwickelt. Wichtiger Meilenstein ist hier die Überarbeitung des Gradientenverfahrens (Formulierung des lokalen Problems als quadratisches Optimierungsproblem) und die gleichzeitige Umsetzung des MOPO-Quelltextes von ADA nach ANSI-C ([Münch 2001], [Rasche 2004]). Weitere bedeutende Verbesserungen waren die Implementierung eines Quasi-Newton Verfahrens, die Eliminierung von Startwertabhängigkeiten und die Definition von sog. Erfüllungsgraden [Münch 2003].

Gleichzeitig wurde an der Untersuchung des MOPO-Laufzeitverhaltens im Hinblick auf Echtzeitverarbeitung [Deppe et al. 2001/2] und an der Unterstützung der parallelen Systembewertung durch MOPO gearbeitet [Deppe et al. 2003/1].

In den folgenden Kapiteln wird zunächst die bestehende MOPO-Implementierung nach [Münch

2003] im Hinblick auf die Eigenschaften von Zielfunktionen (Zielgrößen, Erfüllungsgrade), die Anwendung von Parameterschranken und die mathematischen Grundlagen von Gradienten- und Quasi-Newton-Verfahren dargelegt. Verknüpft mit diesen Grundlagen können dann darauf aufbauend die im Rahmen dieser Arbeit sehr wichtigen Aspekte des MOPO-Laufzeitverhaltens und der MOPO-Parallelisierung erklärt werden.

4.7.1 Eigenschaften von Zielfunktionen

Die Zielfunktionen sollten für lokale Optimierungsverfahren günstigerweise als glatte und konvexe Funktionen formuliert werden, da prinzipbedingt nicht zwischen lokalen und globalen Minima unterschieden werden kann. Bei nicht konvexen Optimierungs-Problemen kann MOPO in einem lokalen Minimum oder am Parameterrand hängen bleiben. D.h. je nach Wahl des Optimierungs-Startpunktes werden verschiedene lokale oder globale Minima als Lösung gefunden, da das Verfahren im ersten gefundenen Optimum endet.

Grundsätzlich wird davon ausgegangen, dass eine Verkleinerung einer Zielfunktion einer Verbesserung entspricht. Dabei kann bei der Formulierung einer Zielfunktion ein Maximierungsproblem durch Vorzeichenumkehr in ein Minimierungsproblem umgewandelt werden. Damit eine Zielfunktion nur ein Minimum besitzt, muss sie konvex sein:

> *Eine in (a,b) differenzierbare Funktion f heißt in (a,b) konvex von unten, wenn für alle x_1, x_2 aus (a,b) mit x_1 ungleich x_2 gilt* [Bronstein, Semendjajew 1989]*:*

$$f(x_2) \geq f(x_1) + f(x_1)' \cdot (x_2 - x_1) \tag{4.11}$$

4.7.2 Skalierung

Für Zielgrößen und Parameter werden vom Anwender obere und untere Schranken für deren Wertebereich vorgegeben. Diese Schranken (Limits) müssen für Parameter strikt eingehalten werden. Diese Forderung geht als Nebenbedingung in das Optimierungsverfahren mit ein. Für Zielgrößen sind das untere Limit als die bestmögliche und das obere Limit als mindeste Erfüllung des Optimierungsziels anzusehen.

Intern werden sämtliche Berechnungen auf der Basis von skalierten Größen durchgeführt. Diese Skalierung hat einen erheblichen Einfluss auf die Genauigkeit der numerischen Berechnungen und somit auf das Konvergenzverhalten bzw. das Ergebnis der Optimierung [Münch 2003]:

- Parameter werden intern linear auf das Intervall *[-1...+1]* skaliert, wobei *-1* der unteren Parameterschranke und *+1* der oberen Parameterschranke entsprechen.
- Für Zielgrößen hat es sich als sinnvoll herausgestellt, das Intervall *[0...+1]* zur skalierten Abbildung der unteren und der oberen Zielgrößenschranken (Limits) zu verwenden. Für Zielgrößenwerte oberhalb des oberen Limits treten hier Werte größer als *+1* auf. Unterhalb des unteren Limits sind die Werte negativ.

4.7.3 Erfüllungsgrad von Zielgrößen

Für jede Zielgröße lässt sich bei bekannten oberen und unteren Limits der Erfüllungsgrad der Zielgröße definieren; er entspricht dem intern verwendeten skalierten Wert einer Zielgröße [Münch 2003]:

$$\eta_i = \frac{z_i(\underline{p}_k) - L_{i,\,Lower}}{L_{i,\,Upper} - L_{i,\,Lower}} \tag{4.12}$$

Durch die Skalierung werden die Zielgrößen in ihrem Erfüllungsgrad direkt untereinander vergleichbar, so dass eine gemeinsame Darstellung, z. B. in Form eines Kreisdiagramms ("Radar-Plot"), eine Übersicht über den Erfüllungsgrad aller Zielgrößen erlaubt:

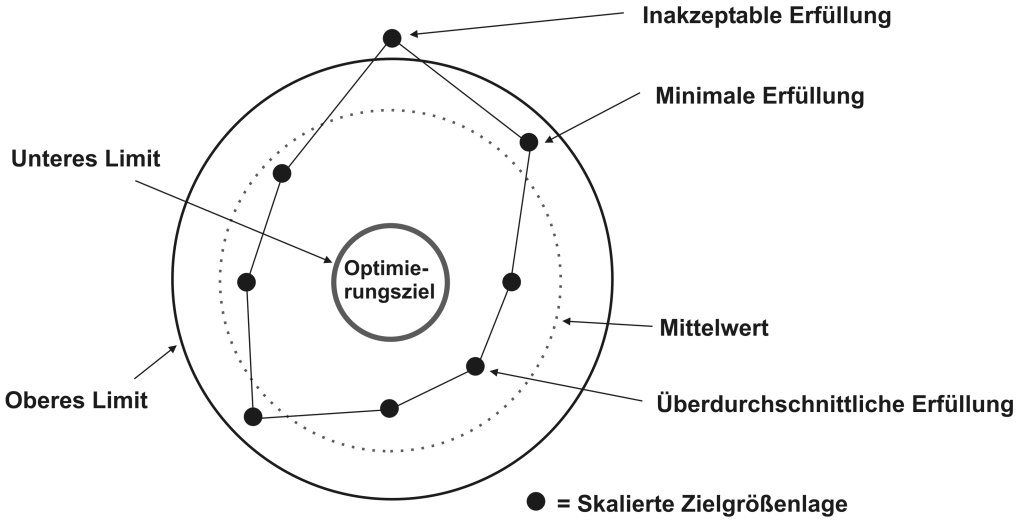

Abbildung 4-6: Kreisdiagramm zur Darstellung der Erfüllungsgrade

4.7.4 Begrenzung von Parametern

Die vorgegebenen Parametergrenzen sind im Zuge der Optimierung unbedingt einzuhalten, da in ihnen Wissen über physikalisch unrealistische Werte oder über instabile Systemzustände etc. enthalten ist. Die Parametergrenzen können prinzipiell explizit im Optimierungsalgorithmus oder implizit in Form von zusätzlichen Zielfunktionen sichergestellt werden.

Für das Gradientenverfahren werden die Parametergrenzen implizit berücksichtigt. Sobald ein Parameter seinen Wertebereich verlassen hat, wird eine zusätzliche Zielfunktion hinzugefügt. Deren Gradient wird so konstruiert, dass die gewünschte Verkleinerung/Vergrößerung des Parameters durch die Minimierung der Parameter-Zielfunktion im Laufe der nächsten Optimierungsschritte erfolgt.

Auch für das Quasi-Newton-Verfahren werden die Parametergrenzen implizit berücksichtigt. Für jeden Parameter existiert eine skalare quadratische Zielfunktion $q_{p,i}$ [Münch 2003]:

$$q_{p,\,i}(s_i) = g_{p,\,i} \cdot s_i + \frac{1}{2} \cdot h_{p,\,i} \cdot s_i^{\,2} \tag{4.13}$$

Für diese Zielfunktion werden als obere und untere Zielgrößen-Limits ($L_{i,Lower}$, $L_{i,Upper}$) die Parametergrenzen gewählt. Die Parameter $g_{p,i}$ und $h_{p,i}$ werden so bestimmt, dass beim Einsetzen der Parametergrenzen für s_i genau diese Schranken erreicht werden. Solange sich die Zielfunktionswerte innerhalb der Schranken befinden, wird die zugehörige Zielfunktion in der Optimierung nicht berücksichtigt, sie ist passiv. Sobald die Parametergrenzen über- oder unterschritten werden, wird die Zielfunktion aktiv einbezogen. Das Optimierungsverfahren wird dann versuchen, diese Zielfunktion zu minimieren, so dass der zugehörige Parameter wieder in seine vorgegebenen Grenzen verschoben wird.

4.8 MOPO-Gradientenverfahren

In Anlehnung an bewährte skalare Optimierungsmethoden arbeitet auch das MOPO-Gradientenverfahren in zwei Schritten [Münch 2003]:

1. Lösen eines lokalen Problems durch Berechnung einer Suchrichtung $\underline{d}_k$, ausgehend von einem Startparametervektor $\underline{p}_0$ oder dem letzten Optimierungsschritt $\underline{p}_k$
2. Lösen eines globalen Problems durch Berechnung eines neuen Parametervektors: $\underline{p}_{k+1} = \underline{p}_k + \lambda_k \cdot \underline{d}_k$

Dies kann man sich anhand der bereits bekannten Skizzen von zwei Zielgrößen im zweidimensionalen Parameterraum folgendermaßen vorstellen:

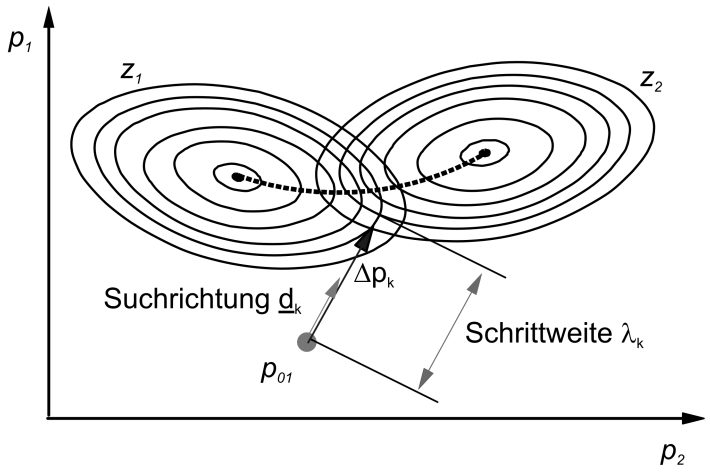

Abbildung 4-7: Lokales und globales Teilproblem beim Gradientenverfahren

Ausgehend vom Startparametervektor $\underline{p}_{01}$, wird zunächst die Suchrichtung $\underline{d}_k$ bestimmt, die eine größtmögliche Verbesserung beider Zielgrößen verspricht (lokales Problem). In dieser Suchrichtung $\underline{d}_k$ wird nun das λ_k gesucht, das die maximale Verbesserung beider Zielgrößen bewirkt.

Man stelle sich vor, dass ausgehend von $\underline{p}_{01}$ in Richtung $\underline{d}_k$, der Weg für z_1 und z_2 zunächst ins Tal führt. Es gibt jedoch einen Punkt, an dem der Weg (immer in Richtung $\underline{d}_k$) zumindest für eine Zielgröße wieder ansteigt, wenn unterstellt wird, dass die Minima der Zielfunktionen nicht im Unendlichen liegen. Dieser Punkt wird genau durch das Ende des Differenzvektors $\Delta\underline{p}_k$ gekennzeichnet. An dieser Stelle ist der *k*-te Optimierungsschritt beendet. Es ist nun ein neues lokales Teilproblem zu lösen, d. h. eine Bestimmung einer neuen Suchrichtung $\underline{d}_{k+1}$, woran sich wieder das globale Teilproblem zur Bestimmung von λ_{k+1} anschließt etc., bis ein Pareto-optimaler

Punkt erreicht ist.

4.8.1 Numerische Gradientenberechnung

Da die Zielfunktionen im Allgemeinen nicht analytisch vorliegen, werden die benötigten partiellen Ableitungen numerisch berechnet. Dazu müssen die Zielfunktionen glatt, d. h. einmal stetig differenzierbar sein. Für die Berechnung der partiellen Ableitung wird der rechtsseitige (Vorwärts-) Differenzenquotient jeder Zielgröße für jeden Parameter gebildet:

$$\frac{\partial z_i}{\partial p_k} \approx \frac{z_i(p_k + \Delta p) - z_i(p_k)}{\Delta p} \tag{4.14}$$

Diese Berechnung wird für alle Zielgrößen und Parameter durchgeführt. Da die Parameter skaliert sind und somit den gleichen Wertebereich haben, können alle Parameter mit demselben Δp ausgelenkt werden.

Um die Genauigkeit zu steigern, kann auch eine Differentiation 2. Ordnung verwendet werden. Allerdings ist hierfür die doppelte Anzahl an Zielfunktionsauswertungen notwendig. Für die Berechnung der partiellen Ableitung wird vom Parameterwert p_k nach rechts und links um Δp ausgelenkt. Es ergibt sich der sogenannte zentrale Differenzenquotient:

$$\frac{\partial z_i}{\partial p_k} \approx \frac{z_i(p_k + \Delta p) - z_i(p_k - \Delta p)}{2 \cdot \Delta p} \tag{4.15}$$

Mittels dieser Ableitungen kann für jede Zielgröße z_i der Gradientenvektor $\underline{g_i(p)}$ gebildet werden:

$$\underline{g}_i(\underline{p}) = \nabla z_i(\underline{p}) = \left[\frac{\partial z_i}{\partial p_1} \; \frac{\partial z_i}{\partial p_2} \; \dots \; \frac{\partial z_i}{\partial p_{n_p}}\right]^T \tag{4.16}$$

Die Gradientenvektoren aller Zielfunktionen werden in der sogenannten Empfindlichkeitsmatrix $\underline{S}$ spaltenweise angeordnet:

$$\underline{S} = [\underline{g}_1, \underline{g}_2, \dots, \underline{g}_{n_z}] = \begin{bmatrix} \frac{\partial z_1}{\partial p_1} & \frac{\partial z_2}{\partial p_1} & \dots & \frac{\partial z_{n_z}}{\partial p_1} \\ \dots & \dots & \dots & \dots \\ \frac{\partial z_1}{\partial p_{n_p}} & \frac{\partial z_2}{\partial p_{n_p}} & \dots & \frac{\partial z_{n_z}}{\partial p_{n_p}} \end{bmatrix} \tag{4.17}$$

4.8.2 Lokales Problem

Das sogenannte lokale Problem besteht aus der Bestimmung einer Suchrichtung $\underline{d}_k$, die es erlaubt eine möglichste steile Abstiegsrichtung für jede einzelne aktive Zielgröße (vgl. Kap. 4.8.4) zu gewährleisten (zum Vergleich: Für skalare Optimierungsprobleme wählt man im Gradientenver-

fahren $\underline{d}_k = -\underline{g}(\underline{p})$). Die Bestimmung einer Suchrichtung $\underline{d}_k$ für die Mehrziel-Optimierung soll im folgenden als ein skalares quadratisches Minimierungsproblem formuliert werden. Dazu betrachtet man zunächst die Winkel γ_i zwischen der Suchrichtung $\underline{d}_k$ und den einzelnen (aktiven) Gradienten $\underline{g}_i(\underline{p})$ über ihr Skalarprodukt:

$$cos\gamma_i = \frac{-\underline{d}_k \cdot \underline{g}_i(\underline{p})}{|\underline{d}_k| \cdot |\underline{g}_i(\underline{p})|} \tag{4.18}$$

Wenn es gelingt den maximalen Winkel $\gamma_{i,\,max}$ möglichst klein zu halten, so hat man eine gut balancierte Suchrichtung $\underline{d}_k$ gefunden. Als Rand- und Abbruchbedingung gilt, dass die γ_i im Intervall *[0° ... 90°[* liegen müssen. Die Suche nach einem möglichst kleinen maximalen Winkel lässt sich dann als skalares quadratisches Minimierungsproblem formulieren:

$$Min\left\{\frac{1}{(cos\gamma_{i,\,max})^2}\right\} \tag{4.19}$$

Der Ansatz zur Bestimmung der Suchrichtung $\underline{d}_k$ ist im dreidimensionalen Parameterraum sehr anschaulich zu beschreiben [Münch 2001]: Gesucht ist der Kegel, der den kleinsten Winkel $\gamma_{i,\,max}$ zwischen der Mantelfläche und $-\underline{d}_k$ aufweist, wobei alle Gradienten im Inneren des Kegels oder auf seiner Oberfläche liegen sollten. Ist ein solcher Kegel gefunden, gewährleistet dies, dass alle Gradienten maximal im Winkel $\gamma_{i,\,max}$ zu $-\underline{d}_k$ liegen. Der Gradient $\underline{g}_4$ sei für das Beispiel passiv und muß nicht einbezogen werden:

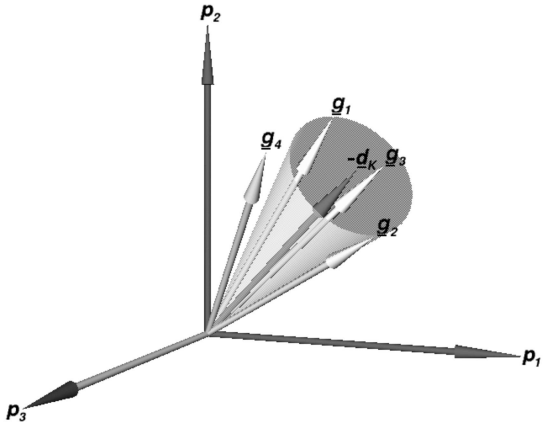

Abbildung 4-8: Gradienten im 3D-Parameterraum und Suchrichtung $\underline{d}_k$

Da lediglich die Richtung von $\underline{d}_k$ gesucht ist, wird der Betrag von $\underline{d}_k$ festgelegt zu:

$$|\underline{d}_k| = \frac{1}{cos\gamma_{i,\,max}} \tag{4.20}$$

Dann gilt auch:

$$\frac{1}{(\cos\gamma_{i,\,max})^2} = \left|\underline{d}_k\right|^2 \tag{4.21}$$

Jetzt kann das Minimierungsproblem aus Gl. 4.19 auch folgendermaßen notiert werden:

$$Min\{\left|\underline{d}_k\right|^2\} \tag{4.22}$$

Dabei muss die Nebenbedingung gelten, dass alle Winkel γ_i kleiner oder gleich $\gamma_{i,\,max}$ sind.

Quadratische Probleme mit Nebenbedingungen in dieser Form lassen sich durch das Standardverfahren *Active-Set-Methode* [Gill, Murray 1978] lösen. In diesem Verfahren wird die Lösung des Optimierungsproblems auf das iterative Lösen von Gleichungssystemen variabler Ordnung zurückgeführt. Dieses Verfahren eignet sich für eine echtzeitfähige Implementierung.

4.8.3 Globales Problem

Bei bekannter Suchrichtung muss das sog. globale, nichtlineare Problem gelöst werden. Hier soll die optimale Schrittweite für das Fortschreiten in die Richtung $\underline{d}_k$ ermittelt werden, so dass sich eine möglichst große Verbesserung aller Zielgrößen ergibt:

$$\underline{p}_{k+1} = \underline{p}_k + \lambda_k \cdot \underline{d}_k \tag{4.23}$$

Dazu wird eine iterative eindimensionale Minimumsuche (Liniensuche) verwendet [Münch 2003]. Es wird solange iteriert, bis sich eine Zielgröße verschlechtert. Tritt bereits beim ersten Schritt der Minimumsuche eine Verschlechterung einer Zielgröße auf, so wird die Iteration von dort aus rückwärts weitergeführt. Beim Vorwärtsschreiten wird die Schrittweite in jedem Schritt verdoppelt, beim Rückwärtsschreiten in jedem Schritt halbiert. Mit Hilfe der letzten zulässigen Zielfunktionswerte wird abschließend für jede Zielgröße eine quadratische Interpolation (vgl. Abbildung 4-9) durchgeführt. Als Ergebnis erhält man eine Schrittweite λ_i für jede einzelne Zielgröße, die anhand der Lage des Minimums der interpolierten Zielfunktion ausgewählt wird. Als globale Schrittweite wird der kleinste λ_i-Wert ausgewählt:

$$\lambda_k = min(\lambda_i) \tag{4.24}$$

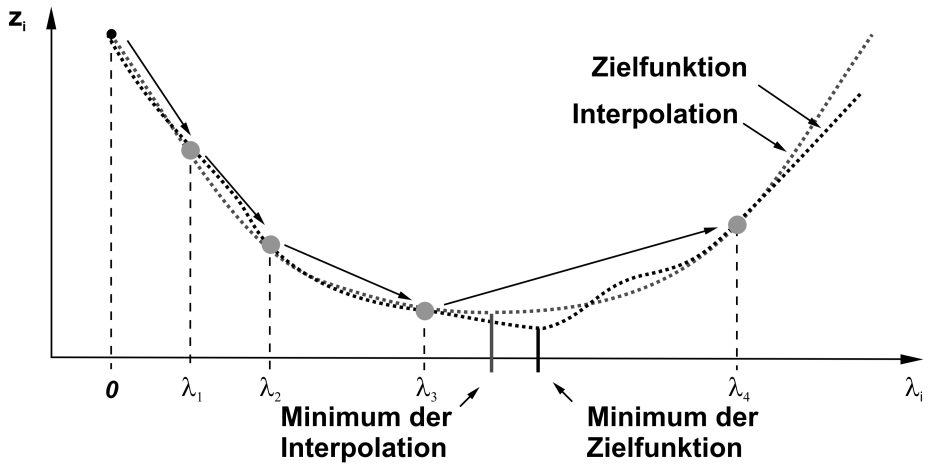

Abbildung 4-9: Minimumsuche und quadratische Interpolation für eine Zielfunktion

4.8.4 Aktive und passive Zielgrößen

Im MOPO-Gradientenverfahren wird zwischen aktiven und passiven Zielgrößen unterschieden. Durch diese Einteilung wird das Verfahren derart gesteuert, dass Zielgrößen unterhalb ihres unteren Limits (passive Zielgrößen) sich verschlechtern dürfen. Dagegen müssen sich die Zielgrößen oberhalb ihres unteren Limits (aktive Zielgrößen) in jedem Optimierungsschritt verbessern. Die Einteilung in aktiv und passiv erfolgt nach jedem Optimierungsschritt neu. Die Einteilung der Zielgrößen hat für den Optimierungsalgorithmus folgende Auswirkungen:

- Nur aktive Zielgrößen werden im lokalen Problem zur Suchrichtungsbestimmung verwendet.
- Aktive Zielgrößen dürfen sich im globalen Problem nicht verschlechtern.
- Passive Zielgrößen dürfen im globalen Problem ihr unteres Limit nicht überschreiten.

4.9 MOPO Quasi-Newton-Verfahren

Zur Verbesserung des bisher verwendeten Gradientenverfahrens wird ein Quasi-Newton-Verfahren [Broyden 1967] herangezogen. Die Vorteile des Verfahrens liegen in der geringeren Anzahl von Zielfunktionsauswertungen, da durch die quadratische Approximation große Schrittweiten zur Minimumsuche entstehen. Das macht das Verfahren auch robuster gegen numerische Ungenauigkeiten oder verrauschte Zielgrößen. Ein Nachteil ist im erhöhten Rechenaufwand pro Optimierungsschritt zu sehen. Üblicherweise ist jedoch der Rechenaufwand des Verfahrens im Vergleich zu dem der Zielfunktions-Auswertung vernachlässigbar.

Beim Quasi-Newton-Verfahren wird jede einzelne Zielfunktion z_i an den Stellen $\underline{p}_k$ durch eine eigene quadratische Ersatzzielfunktion $\tilde{q}_{i,k}$ approximiert, indem neben dem Gradientenvektor auch die geschätzte Hesse-Matrix genutzt wird:

$$\tilde{q}_{i,k}(\underline{s}_k) = z_i(\underline{p}_k) + \underline{g}_i(\underline{p}_k)^T \cdot \underline{s}_k + \frac{1}{2} \cdot \underline{s}_k^T \cdot \underline{H}_i(\underline{p}_k) \cdot \underline{s}_k \tag{4.25}$$

Dabei ist allerdings die Bestimmung der Hesse-Matrix $\underline{H}_i(\underline{p}_k)$ mit größerem Aufwand verbunden als nur die beschriebene numerische Bestimmung des Gradientenvektors $\underline{g}_i(\underline{p}_k)$. Daher wird die Hesse-Matrix mit Hilfe eines Schätzverfahrens durch die Matrix $\tilde{\underline{H}}_{i,k}$ angenähert [Münch 2003]. Dazu wird ein iteratives Aufdatierungsverfahren nach Broyden-Fletcher-Goldfarb-Shannon ([Fletcher 1980], [Bronstein, Semendjajew 1989]) verwendet:

$$\tilde{\underline{H}}_{i,k+1} = \tilde{\underline{H}}_{i,k} - \frac{\tilde{\underline{H}}_{i,k} \cdot \Delta\underline{p}_k \cdot (\tilde{\underline{H}}_{i,k} \cdot \Delta\underline{p}_k)^T}{\Delta\underline{p}^T_k \cdot \tilde{\underline{H}}_{i,k} \cdot \Delta\underline{p}_k} + \frac{\Delta\underline{g}_{i,k} \cdot \Delta\underline{g}^T_{i,k}}{\Delta\underline{g}^T_{i,k} \cdot \Delta\underline{p}_k} \tag{4.26}$$

$$\Delta\underline{p}_k = \underline{p}_{k+1} - \underline{p}_k \tag{4.27}$$

$$\Delta\underline{g}_{i,k} = \underline{g}_i(\underline{p}_{k+1}) - \underline{g}_i(\underline{p}_k) \tag{4.28}$$

Als Bedingung muss $\tilde{\underline{H}}_{i,k}$ positiv definit sein, so dass nur konvexe und glatte Zielfunktionen auftreten. Gestartet werden kann mit einer beliebigen positiv definiten Matrix, z. B. der Einheitsmatrix. Damit $\tilde{\underline{H}}_{i,k+1}$ positiv definit bleibt, muss erfüllt sein:

$$\Delta\underline{g}^T_{i,k} \cdot \Delta\underline{p}_k > 0 \tag{4.29}$$

Falls diese Bedingung nicht zutrifft, wird der zugehörige Aufdatierungsschritt übersprungen.

4.9.1 Quadratisches Ersatzproblem

Mit der Kenntnis des Gradienten und der Näherung der Hesse-Matrix lässt sich ein quadratisches Ersatzproblem formulieren [Münch 2003]:

$$q_{i,k}(\underline{s}_k) = z_i(\underline{p}_k) + \underline{g}_i(\underline{p}_k)^T \cdot \underline{s}_k + \frac{1}{2} \cdot \underline{s}^T_k \cdot \tilde{\underline{H}}_i(\underline{p}_k) \cdot \underline{s}_k \tag{4.30}$$

$$Min\{\underline{q}_k(\underline{s}_k)\} \tag{4.31}$$

Dieses Ersatzproblem hat die Eigenschaft, dass die realen Zielfunktionen in der Umgebung von $\underline{p}_k$ quadratisch approximiert werden. Durch Lösen dieses Ersatzproblems erhält man eine Abschätzung für die Lage des Optimums des realen Optimierungsproblems.

Als Lösungsverfahren für das Ersatzproblem kann das oben beschriebene Gradientenverfahren genutzt werden, das nun die Ersatzzielfunktionen verwendet, um über den lokalen und den globalen Schritt den nächsten Parametervektor (s. Gl. 4.23) zu bestimmen. Günstiger ist aber ein Verfahren, das die Vorteile der symbolisch beschriebenen Ersatzzielfunktionen ausnutzen kann.

Unter der Annahme, dass die Nebenbedingungen erfüllt sind, kann ein Pareto-optimaler Punkt $\underline{s}_k^*$ des Ersatzproblems durch das skalare Funktional $g_\alpha(\underline{s}_k)$ folgendermaßen beschrieben werden:

$$g_\alpha(\underline{s}_k) = \sum_{i=1}^{n_z} \alpha_{i,k} \cdot q_{i,k}(\underline{s}_k) \tag{4.32}$$

$$\sum_{i=1}^{n_z} \alpha_{i,k} = 1 \tag{4.33}$$

$\underline{s}_k$* ist dann ein sog. Karush-Kuhn-Tucker (KKT-) Punkt des skalaren Funktionals $g_\alpha(\underline{s}_k)$. Die KKT-Bedingung 1. Ordnung (ohne Nebenbedingungen) als Notwendigkeit für einen Pareto-optimalen Punkt des quadratischen Optimierungs-Ersatzproblems lautet:

$$\nabla g_\alpha(\underline{s}_k^*) = \sum_{i=1}^{n_z} \alpha_{i,k} \cdot \nabla q_{i,k}(\underline{s}_k^*) = 0 \tag{4.34}$$

Die Variablen $\alpha_{i,k}$ geben hier an, welcher Punkt der Paretomenge berechnet wird. Sie spannen auf der Paretomenge ein Koordinatensystem auf (vgl. Kap. 4.3). Gl. 4.34 soll nun nach $\underline{s}_k$ aufgelöst werden. Zunächst wird daher Gl. 4.32 nach $\underline{s}_k$ differenziert:

$$\nabla_{\underline{s}_k}(g_\alpha(\underline{s}_k)) = \sum_{i=1}^{n_z} \alpha_{i,k} \cdot (\underline{g}_i(\underline{p}_k)^T + \tilde{\underline{H}}_i(\underline{p}_k) \cdot \underline{s}_k) \tag{4.35}$$

Mit Gl. 4.34 ergibt sich daraus als Notwendigkeit für einen Pareto-optimalen Punkt:

$$\sum_{i=1}^{n_z} \alpha_{i,k} \cdot \underline{g}_i(\underline{p}_k)^T + \sum_{i=1}^{n_z} \alpha_{i,k} \cdot \tilde{\underline{H}}_i(\underline{p}_k) \cdot \underline{s}_k^* = 0 \tag{4.36}$$

Diese Gleichung wird nach $\underline{s}_k$* aufgelöst:

$$\underline{s}_k(\underline{\alpha})^* = -\underline{m}_k(\underline{\alpha})^{-1} \cdot \sum_{i=1}^{n_z} \alpha_{i,k} \cdot \underline{g}_i(\underline{p}_k)^T \tag{4.37}$$

$$\underline{m}_k(\underline{\alpha}) = \left(\sum_{i=1}^{n_z} \alpha_{i,k} \cdot \tilde{\underline{H}}_i(\underline{p}_k) \right) \tag{4.38}$$

Diese Ausdrücke beschreiben die Lage eines Paretopunktes $\underline{s}_k$* in Abhängigkeit der Faktoren $\alpha_{i,k}$ für das quadratische Ersatzproblem in der Umgebung des Parametervektors $\underline{p}_k$. Bei Gl. 4.37 zusammen mit Gl. 4.38 handelt es sich um ein lineares Gleichungssystem, das mit Standardverfahren gelöst werden kann.

Zur Invertierung von $\underline{m}_k(\underline{\alpha})$ wird in MOPO die Cholesky-Zerlegung angewandt, die nur für symmetrische, positiv definite Matrizen durchführbar ist. Die Matrix $\underline{m}_k(\underline{\alpha})$ ist dann positiv definit, wenn alle $\tilde{\underline{H}}_i(\underline{p}_k)$ positiv definit sind und alle $\alpha_{i,k}$ größer oder gleich Null sind. Die Bedingung, dass $\tilde{\underline{H}}_i(\underline{p}_k)$ positiv definit ist, wird bereits für das BFGS-Verfahren vorausgesetzt (vgl. Kap. 4.9, Gl. 4.29) und wird dadurch erzwungen, dass notfalls der BFGS Aufdatierungsschritt übersprungen wird.

4.9.2 Definition des Optimierungsziels

Zur Ermittlung eines geeigneten Optimums werden zwei Forderungen gestellt [Münch 2003]:

1. Das Optimierungsergebnis soll ein Pareto-Optimum sein.

2. Die Erfüllungsgrade aller Zielgrößen sollen den selben Wert anstreben.

Insbesondere die zweite Forderung hat einen sehr großen Einfluss auf die resultierenden Optimierungsergebnisse und die implementierte Algorithmik des Verfahrens. Insgesamt ergibt sich eine Balancierung der Zielgrößen, die sich in der Praxis bewährt hat:

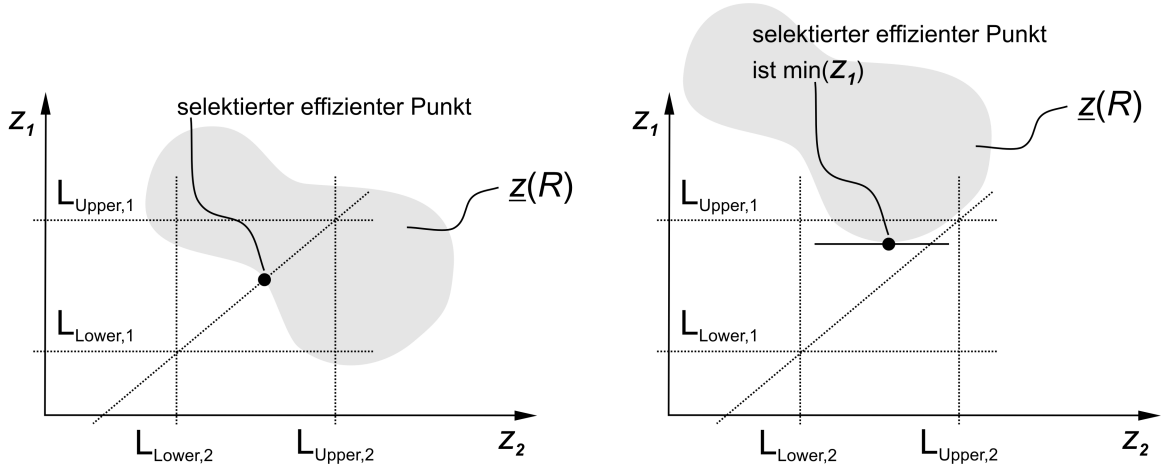

Abbildung 4-10: Verwendung der Zielgrößenlimits zur Ermittlung von Durchstoßpunkten

Beide Forderungen lassen sich graphisch sehr gut veranschaulichen (vgl. Abbildung 4-10). Durch die Konstruktion einer Geraden durch die Schnittpunkte der Begrenzungsgeraden der oberen und der unteren Limits wird ein effizienter Punkt im Zielgrößenraum (Durchstoßpunkt) ausgewählt.

Falls ein solcher Durchstoßpunkt nicht existiert (vgl. Abbildung 4-10 rechter Teil), wird die im gesamten Bereich schlechtere Zielgröße (hier Zielgröße z_1) allein berücksichtigt und die bessere Zielgröße (hier Zielgröße z_2) als passiv gesetzt.

Die praktische Umsetzung zur Balancierung der Zielgrößen im allgemeinen Mehrgrößenfall erfolgt durch die Unterscheidung von sogenannten aktiven und passiven Zielgrößen.

4.9.3 Aktive und passive Zielgrößen für das Quasi-Newton-Verfahren

Eine Pareto-Optimalität wird durch Gl. 4.37 sichergestellt. Die Forderung nach gleich guter Erfüllung aller Zielgrößen bedingt eine Unterscheidung zwischen aktiven und passiven Zielgrößen. Die Definition für aktive und passive Zielgrößen entspricht nicht der Definition, wie sie für das Gradientenverfahren verwendet wird.

Damit eine Zielgröße passiv ist, müssen zwei Bedingungen erfüllt sein: Der zugehörige Faktor $\alpha_{i,k}$ muss gleich 0 und der zugehörige Erfüllungsgrad besser als der Durchschnitt sein. Damit ergibt sich die Menge der aktiven Zielgrößen zu:

$$\Im = \langle i|\alpha_{i,k} > 0\rangle \cup \langle i|\eta_i(\underline{\alpha}_k) > \bar{\eta}(\underline{\alpha}_k)\rangle \tag{4.39}$$

Der durchschnittliche Erfüllungsgrad aller Zielgrößen lautet:

$$\bar{\eta}(\underline{\alpha}_k) = \frac{1}{|\Im|} \cdot \sum_{i \in \Im} \eta_i(\underline{\alpha}_k) \tag{4.40}$$

Aufgrund der gegenseitigen Abhängigkeit beider Gleichungen wird ein Iterationsverfahren zur Lösung verwendet [Münch 2003]. Nach der Lösung beider Gleichungen stehen die aktiven und die passiven Zielgrößen fest, und es lässt sich für jede Zielgröße mit Hilfe von Gl. 4.12 und Gl. 4.40 ein aktuelles Optimierungsziel abschätzen:

$$\bar{q}_{i,\,k}(\underline{\alpha}_k) = \bar{\eta}(\underline{\alpha}_k) \cdot (L_{i,\,Upper} - L_{i,\,Lower}) + L_{i,\,Lower} \tag{4.41}$$

4.9.4 Lösen des quadratischen Ersatzproblems

Um die optimalen Faktoren $\alpha_{i,\,k}$ zu finden, wird die folgende skalare quadratische Funktion mit Hilfe eines konjugierten Gradientenverfahrens (Eingrößenfall) minimiert:

$$r_k(\underline{\alpha}_k) = \sum_{i \in \Im} \left(q_{i,\,k}(\underline{\alpha}_k) - \bar{q}_{i,\,k}(\underline{\alpha}_k)\right)^2 \tag{4.42}$$

Für das konjugierte Gradientenverfahren wird die partielle Ableitung von r_k nach $\underline{\alpha}_k$ benötigt. Diese kann vorab symbolisch bestimmt werden und lässt sich in zwei Teilableitungen aufteilen:

$$\frac{\partial r_k}{\partial \underline{\alpha}_k} = \frac{\partial r_k}{\partial \underline{s}_k} \cdot \frac{\partial \underline{s}_k}{\partial \underline{\alpha}_k} \tag{4.43}$$

Durch Ableiten von Gl. 4.42 nach $\underline{s}_k$ ergibt sich:

$$\frac{\partial r_k}{\partial \underline{s}_k} = \sum_{i \in \Im} 2 \cdot (q_{i,\,k} - \bar{q}_{i,\,k}) \cdot \left(\frac{\partial q_{i,\,k}}{\partial \underline{s}_k} - \frac{\partial \bar{q}_{i,\,k}}{\partial \underline{s}_k}\right) \tag{4.44}$$

Die partielle Ableitung der Zielfunktion $q_{i,k}$ nach $\underline{s}_k$ ergibt sich durch Differentation von Gl. 4.30:

$$\frac{\partial q_{i,\,k}}{\partial \underline{s}_k} = \underline{g}_i(\underline{p}_k)^T + \underline{\tilde{H}}_i(\underline{p}_k) \cdot \underline{s}_k \tag{4.45}$$

Die partielle Ableitung des Optimierungsziels $\bar{q}_{i,\,k}$ nach $\underline{s}_k$ ergibt sich durch Differenzieren und Einsetzen aus Gl. 4.12, Gl. 4.40 und Gl. 4.41:

$$\frac{\partial \bar{q}_{i,\,k}}{\partial \underline{s}_k} = \frac{L_{i,\,Upper} - L_{i,\,Lower}}{|\Im|} \cdot \sum_{j \in \Im} \frac{1}{L_{j,\,Upper} - L_{j,\,Lower}} \cdot \frac{\partial q_{j,\,k}}{\partial \underline{s}_k} \tag{4.46}$$

Die fehlende partielle Ableitung von $\underline{s}_k$ nach $\underline{\alpha}_k$ zur Vervollständigung von Gl. 4.43 ist recht aufwendig herzuleiten. Eine Beschreibung findet sich in [Münch 2003]. Hier sei nur das Ergebnis dargestellt:

$$\frac{\partial \underline{s}_k}{\partial \underline{\alpha}_k} = -\left(\sum_{i=1}^{n_z} \alpha_{i,k} \cdot \tilde{\underline{H}}_i(\underline{p}_k)\right)^{-1} \cdot (\underline{g}_i(\underline{p}_k)^T + \tilde{\underline{H}}_i(\underline{p}_k) \cdot \underline{s}_k) \tag{4.47}$$

4.9.5 Konjugiertes Gradientenverfahren

Zur Minimierung von Gl. 4.42 wird ein konjugiertes Gradientenverfahren verwendet. Es basiert ebenso wie das reine Gradientenverfahren auf einem lokalen und einem globalen Schritt:

1. Lösen eines *lokalen* Problems, d. h. Berechnung einer Suchrichtung $\underline{d}_{k+1}$ anhand des Parametervektors $\underline{\alpha}_{k+1}$:

$$\underline{d}_{k+1} = -\nabla r(\underline{\alpha}_{k+1}) + \Upsilon_k \cdot \underline{d}_k \tag{4.48}$$

2. Lösung eines *globalen* Problems, d. h. Berechnung eines neuen Parametervektors $\underline{\alpha}_{k+1}$ aus der Schrittweite λ_k durch eindimensionale Minimumsuche (Liniensuche):

$$\underline{\alpha}_{k+1} = \underline{\alpha}_k + \lambda_k \cdot \underline{d}_k \tag{4.49}$$

Für die Bestimmung der Suchrichtung $\underline{d}_{k+1}$ und der Gewichtung der vorherigen Suchrichtung Υ_k wird die oben hergeleitete partielle Ableitung von r_k nach $\underline{\alpha}_k$ verwendet (s. Gl. 4.43). Die Gewichtung der vorherigen Suchrichtung wird nach der Polak-Ribiere Variante [Papageorgiou 1996] des konjugierten Gradientenverfahrens berechnet:

$$\Upsilon_k = \frac{[\nabla r(\underline{\alpha}_{k+1}) - \nabla r(\underline{\alpha}_k)]^T \cdot \nabla r(\underline{\alpha}_{k+1})}{\nabla r(\underline{\alpha}_k)^T \cdot \nabla r(\underline{\alpha}_k)} \tag{4.50}$$

4.9.6 Zusammenfassung

Das MOPO-Quasi-Newton-Verfahren ist in zwei Varianten verfügbar [Münch 2003]. Die Unterschiede liegen in der Art der Lösung des quadratischen Ersatzproblems (s. Gl. 4.30). Dieses Problem kann entweder mit dem beschriebenen MOPO-Gradientenverfahren gelöst werden oder mit dem erläuterten Ansatz. Dieser nutzt die Tatsache aus, dass die quadratischen Ersatzzielfunktionen in symbolischer Form vorliegen. An dieser Stelle soll nun kurz zusammengefasst werden, warum dieser Ansatz trotz der Verwendung der KKT-Bedingung 1. Ordnung für Pareto-Optimalität (s. Gl. 4.34) nichts mit dem Optimierungsverfahren der "gewichteten Summe" gemeinsam hat. Das Verfahren der "gewichteten Summe" steht im Widerspruch zum obersten Grundsatz für die Mehrziel-Optimierung mit MOPO, nämlich der Erhaltung von physikalisch interpretierbaren Einzelkriterien auch in allen internen Abläufen. Diesem Grundsatz wird durch die folgenden Aspekte Rechnung getragen:

- Pro Zielgröße wird in jedem Optimierungsschritt eine eigene neue quadratische Ersatzzielfunktion ermittelt.
- Die Variablen $\alpha_{i,k}$ (Gewichtungen) werden zunächst nur genutzt, um auf der Paretomenge ein Koordinatensystem zu definieren; es wird nicht versucht $g_\alpha(\underline{s}_k)$ zu minimieren.

- In jedem Optimierungsschritt werden aktive und passive Zielgrößen ermittelt.
- Für jede einzelne Zielgröße wird ein eigenes Optimierungsziel (Erfüllungsgrad) abgeschätzt.
- Zur optimalen Balancierung der Zielgrößen (Sub-Optimierungsproblem) werden die Variablen $\alpha_{i,k}$ mit Hilfe eines konjugierten Gradientenverfahrens bestimmt.

5 Verteilte Echtzeitsimulation

5.1 Differentialgleichungen zur Regler- und Modellbeschreibung

In der Regelungstechnik wird als grundlegende mathematische Beschreibung die Zustandsraumdarstellung mit den Eingängen $\underline{u}(t)$, den Ausgängen $\underline{y}(t)$, den Zuständen $\underline{x}(t)$ und den zeitinvarianten Parametern $\underline{p}$ verwendet. Bei der Zustandsraumdarstellung handelt es sich um zeitinvariante, explizite, nichtlineare Differentialgleichungen 1. Ordnung mit Anfangsbedingungen:

$$\dot{\underline{x}}(t) = \underline{f}(\underline{x}(t), \underline{u}(t), \underline{p}) \text{ mit } \underline{x}_0 = \underline{x}(t_0) \tag{5.1}$$

$$\underline{y}(t) = \underline{g}(\underline{x}(t), \underline{u}(t), \underline{p}) \tag{5.2}$$

Eine besonders wichtige Stellung nehmen die zeitinvarianten **linearen** Systeme (LZI) in Zustandsraumdarstellung ein, da für sie eine umfassende und geschlossene Systematik zur linearen Analyse und zum Reglerentwurf existiert ([Föllinger 1994], [Ludyk 1990], [Kwakernaak, Sivan 1972]):

$$\dot{\underline{x}}(t) = \underline{A}(\underline{p}) \cdot \underline{x}(t) + \underline{B}(\underline{p}) \cdot \underline{u}(t) \tag{5.3}$$

$$\underline{y}(t) = \underline{C}(\underline{p}) \cdot \underline{x}(t) + \underline{D}(\underline{p}) \cdot \underline{u}(t) \tag{5.4}$$

Die Zustandsraumdarstellung ist die Grundlage der mittleren, normierenden Verhaltensebene des am MLaP entwickelten Objektmodells der Mechatronik (OMM), da sie zahlreiche Vorteile für die Analyse, den Reglerentwurf, die Modell-Strukturierung und die numerische Echtzeitverarbeitung bietet.

Andere Ansätze (z. B. in DIVA, SPICE, ADAMS und SIMPACK) benutzen Differential-Algebraische Gleichungen (DAEs) zur Beschreibung von physikalischen Systemen. Die Modellbildung führt normalerweise auf diese Form, wenn man die Modellgleichungen in ihrer ursprünglichen, den Prozess beschreibenden Form belässt. Daher werden diese DAE-Systeme auch als Deskriptorsysteme (describere (lat.) = beschreiben) bezeichnet. Hier als Besipiel eine semi-explizite Darstellung. Man bezeichnet $\underline{J}$ als Deskriptormatrix und $\underline{a}$ als algebraischen Zustandsvektor:

$$\underline{J}(\underline{x}, \underline{a}, \underline{u}, t) \cdot \dot{\underline{x}} = \underline{f}(\underline{x}, \underline{a}, \underline{u}, t) \tag{5.5}$$

$$\underline{0} = \underline{g}(\underline{x}, \underline{a}, \underline{u}, t) \tag{5.6}$$

5.1.1 Echtzeitfähige Verfahren zur numerischen Simulation

Nachfolgend werden die Grundlagen für die numerische Echtzeitsimulation von Modellen in linearer und nichtlinearer Zustandsraumdarstellung skizziert.

Lineare numerische Simulation

Die lineare Simulation basiert auf der numerischen oder symbolischen Lösung der Zustandsgleichung (Gl. 5.3, Gl. 5.4) des zugehörigen linearen Systems. Die Lösung der Zustandsgleichung im

Zeitbereich hat allgemein folgende Form [Ludyk 1990]:

$$\underline{x}(t) = e^{\underline{A} \cdot t} \cdot \underline{x}_0 + \int_0^t e^{\underline{A} \cdot (t-\tau)} \cdot \underline{B} \cdot \underline{u}(\tau) d\tau \tag{5.7}$$

Dabei existieren effiziente numerische Verfahren, um die sogenannte Fundamentalmatrix des Systems zu bestimmen:

$$\underline{\Phi}(t) = e^{\underline{A} \cdot t} \tag{5.8}$$

Nähert man die Lösung $\underline{x}(t)$ für diskrete Zeitpunkte kT (k=0,1,2,3...) mit konstanten Werten $\underline{u}(k)$, $\underline{y}(k)$ und $\underline{x}(k)$ während des Abtastintervalls T für die Verarbeitung im Digitalrechner, so lässt sich schreiben:

$$\underline{x}(k+1) = e^{\underline{A} \cdot T} \cdot \underline{x}(k) + \int_0^T e^{\underline{A} \cdot (T-\tau)} d\tau \cdot \underline{B} \cdot \underline{u}(k) \tag{5.9}$$

Dies bildet die Basis zur digitalen, linearen, numerischen Simulation mit Hilfe der Fundamentalmatrix [Ludyk 1990] und kann folgendermaßen umgeformt werden:

$$\underline{x}(k+1) = \underline{\Phi}(T) \cdot \underline{x}(k) + (\underline{\Phi}(T) - \underline{I}) \cdot \underline{A}^{-1} \cdot \underline{B} \cdot \underline{u}(k) \tag{5.10}$$

Dieses rekursive, zeitinvariante Differenzengleichungssystem kann sehr effizient und mit konstanter Laufzeit digital gelöst werden und ist daher sehr gut für eine Echtzeitverarbeitung geeignet.

Nichtlineare numerische Simulation

Bei hoher Komplexität von Modellen in nichtlinearer Zustandsraumdarstellung ist eine analytische Lösung der Differentialgleichungen nicht möglich. Hier bietet sich eine numerische Lösung des in Gl. 5.1 beschriebenen Anfangswertproblems auf Basis der Integration der Gleichung an:

$$\underline{x}(t_1) = \underline{x}_0 + \int_{t_0}^{t_1} \underline{f}(\underline{x}(t), \underline{u}(t), \underline{p}) dt \tag{5.11}$$

Nähert man die Lösung $\underline{x}(t)$ für diskrete Zeitpunkte kT (k=0,1,2,3...) mit konstanten Werten $\underline{u}(k)$, $\underline{y}(k)$ und $\underline{x}(k)$ im Abtastintervall $T=t_{k+1} - t_k$, so lässt sich schreiben:

$$\underline{x}(k+1) = \underline{x}(k) + \int_{t_k}^{t_{k+1}} \underline{f}(\underline{x}(k), \underline{u}(k), \underline{p}) dt \tag{5.12}$$

Zur Lösung dieser Gleichung existieren diverse Verfahren zur numerischen Integration

([Schwarz 1993]).

Die einfachste Möglichkeit ist das Euler Einschrittverfahren [Bronstein, Semendjajew 1989]. Hierbei wird der Integralausdruck aus Gl. 5.12 als Rechteckfläche der Höhe $\underline{f}(\underline{x}(k), \underline{u}(k), \underline{p})$ und der Breite h (mit $h=t_{k+1} - t_k$) interpretiert. Man erhält eine nichtlineare Differenzengleichung erster Ordnung (Anfangswerte $\underline{x}(0)$ und $\underline{u}(0)$ sind bekannt):

$$\underline{x}(k+1) = \underline{x}(k) + h \cdot \underline{f}(\underline{x}(k), \underline{u}(k), \underline{p}) \tag{5.13}$$

Das Euler-Verfahren hat prinzipbedingt einen grossen Verfahrensfehler, so dass mit sehr kleinen Schrittweiten h gerechnet werden muss. Daraus resultieren zusätzlich grosse akkumulierte Rundungsfehler.

Bessere numerische Genauigkeit liefern Einschrittverfahren höherer Ordnung, wie das Heun-Verfahren (2 Auswertungen von $\underline{f}(\underline{x},\underline{u},\underline{p})$ im Intervall $t_{k+1} - t_k$) und das Runge-Kutta-Verfahren 4. Ordnung (4 Auswertungen von $\underline{f}(\underline{x},\underline{u},\underline{p})$ im Intervall $t_{k+1} - t_k$) [Föllinger 1994]. Diese Verfahren erlauben aufgrund der Zwischenauswertungen eine größere Integrationsschrittweite mit besserer numerischer Genauigkeit, verursachen aber auch erhöhten Rechenaufwand (für verteilte Simulation auch erhöhten Kommunikationsaufwand) als die Verfahren 1. Ordnung.

Die genannten Integrationsverfahren sind im Bereich der Echtzeitsimulation häufig angewendet. Sie erlauben eine effiziente und deterministische Verabeitung mit in jedem Rechenzyklus gleichem Verarbeitungsaufwand. Von der numerischen Genauigkeit her sind sie allerdings ausgeklügelten Verfahren mit variabler Schrittweite und Unstetigkeitsdetektion ([Oberschelp 1998], [Stumpe 1996]) deutlich unterlegen, so dass ihre Anwendung für die Echtzeitsimulation immer nur ein Kompromiss zwischen Genauigkeit und Echtzeitfähigkeit sein kann.

5.1.2 Datenflussgraph zur Nichtlinearen Simulation

Die Betrachtung des Datenflusses eines Systems auf Verarbeitungsebene ist ein wichtiges Hilfsmittel um das System zu analysieren und das Potential für Parallelisierung und Verteilbarkeit zu bestimmen. Der Berechnungsgraph zur Systemrepräsentation auf der Verarbeitungsebene des OMM (vgl. Gl. 5.1 bis Gl. 5.4) kann als Datenflussgraph (DFG) aufgefasst werden. Nach [Teich 1997] sind Datenflussgraphen gerichtete Graphen in denen Knoten Aktivitäten und Kanten gerichteten Datenfluss repräsentieren. In einem Datenfluss werden die Aktivitäten (Berechnungen) allein durch die Verfügbarkeit der Daten gesteuert.

Entscheidender Punkt ist die Auswahl einer geeigneten Granularität zur Defintion der Knoten von Datenflussgraphen. Im Hinblick auf eine verteilte Simulation besteht ein Knoten des DFG für die NL-Simulation aus Gl. 5.1 und Gl. 5.2, wobei Gl. 5.2 noch einmal aufgeteilt wird ([Deppe, Homburg 1998], [Stolpe 2004]):

1) ***N**icht-**D**urchgriffs-Knoten des DFG* **(**ND-Knoten**)**: Berechnung aller Ausgänge $\underline{y}_{ND}$, die sich aufgrund der aktuellen Zeit, der inneren Zustände oder deren Anfangsbedingungen und der Parameter berechnen lassen. Dieser Knoten hat die Eigenschaft, dass er immer sofort ausgeführt werden kann, da keine Eingänge abgefragt werden müssen:

$$\underline{y}_{ND}(t) = \underline{g}_{ND}(\underline{x}(t), \underline{p}) \tag{5.14}$$

2) ***D**urchgriffs-Knoten des DFG* (D-Knoten): Berechnung aller restlichen Ausgänge $\underline{y}_D$, die direkt von Eingängen $\underline{u}_D$ abhängen. Durch die direkte Verknüpfung von Ein- und Ausgängen bilden die Durchgriffs-Schritte sequentielle Abhängigkeiten zwischen Knoten:

$$\underline{y}_D(t) = \underline{g}_D(\underline{x}(t), \underline{u}_D(t), \underline{p}) \tag{5.15}$$

3) *Zustands-(**S**tate-)Knoten des DFG* (S-Knoten): Berechnung der neuen inneren Zustände des DGL-Systems. Die Berechnung kann auch von Eingängen $\underline{u}_S$ abhängen, Ausgänge werden nicht berechnet. Die Änderungen der Zustände wirken sich beim nächsten Durchlaufen des ND-Schritts aus. Für die NL-Simulation besteht der Zustands-Schritt in der Berechnung der Zustandsableitungen und der Berechnung der neuen Zustände mit Hilfe von numerischer Integration:

$$\underline{x}(t_1) = \underline{x}_0 + \int_{t_0}^{t_1} \underline{f}(\underline{x}(t), \underline{u}_S(t), \underline{p})dt \tag{5.16}$$

Die Anwendung der oben beschriebenen Regeln zur Definition von Knoten und Kanten für den Datenflussgraphen der NL-Simulation führt unmittelbar auf folgende graphische Darstellung, die eine Mischung aus Blockdiagramm und Graphendarstellung ist:

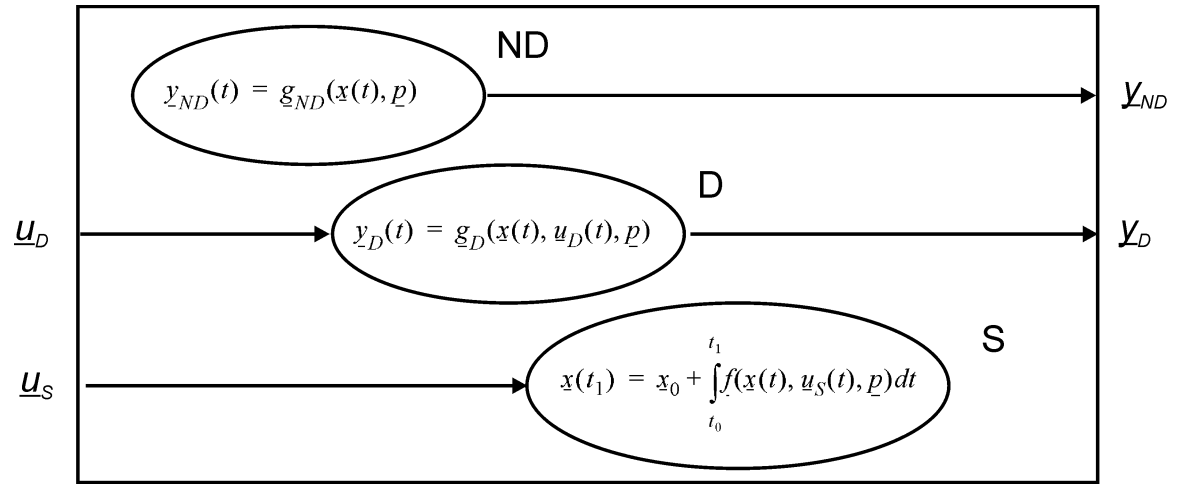

Abbildung 5-1: Datenflussgraph für die NL-Simulation

5.1.3 Verteilte Nichtlineare Simulation

Zur durchgängigen Erhaltung der modular-hierarchischen Systemstruktur (VMS, AMS, MFM) des OMM bis auf die Verarbeitungsebene muss die Simulation auf modularen Datenflussgraphen basieren. Dabei hat jeder einzelne DFG die Struktur und die Schnittstelle gemäß Abbildung 5-1. Jeder Teil-DFG kann dann in einem Rechnernetzwerk platziert werden und kommuniziert seine Ein-/Ausgänge über Nachrichtenkopplung.

Die nachfolgende Abbildung zeigt links modular-hierarchisch organisierte Systemelemente

(z. B. MFMs) mit deren Ein-/Ausgangsbeziehungen. Auf der rechten Seite sind die zugehörigen gekoppelten Datenflüsse nebeneinander dargestellt, so dass die vorhandene Parallelität bei der NL-Simulation erkennbar wird:

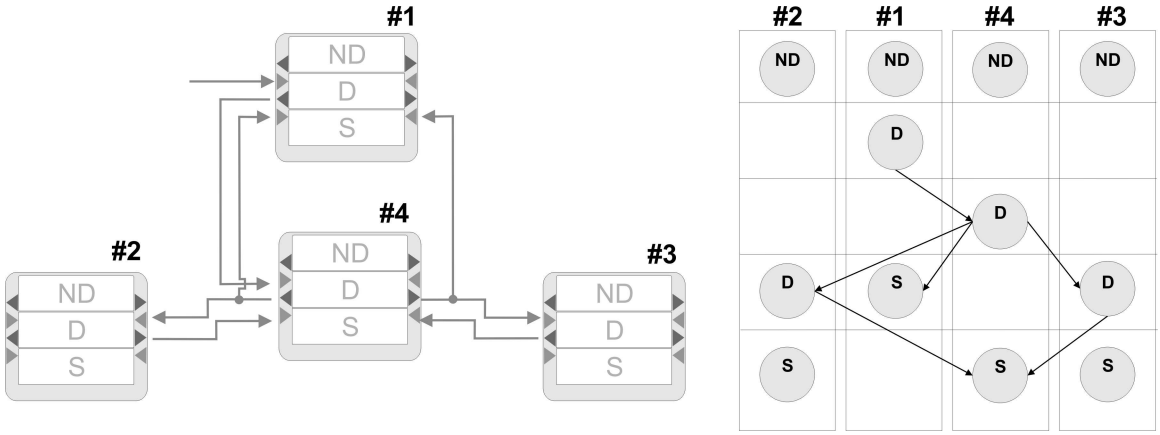

Abbildung 5-2: Beispiel für eine modulare NL-Simulation mit gekoppelten DFGs

Diese Art der modularen Simulation auf Basis gekoppelter DFGs bietet viele Vorteile:

1. Die physikalische Systemstruktur kann erhalten bleiben und dient als Vorgabe zur Verteilung.
2. Simulierte Module sind für die HIL-Simulation einzeln gegen reale Module austauschbar.
3. Test und Inbetriebnahme können schrittweise erfolgen.
4. Mit verteilter Hardware lässt sich ggf. die Parallelität im DFG in Rechenzeitgewinn umsetzen.
5. Teile können für sicherheitskritische Anwendungen dupliziert werden (Redundanz).

Randbedingungen und Einschränkungen

Grundvoraussetzung für die verklemmungsfreie Ausführbarkeit der modularen Simulation ist die Kreisfreiheit des gekoppelten DFGs. Ist dies nicht der Fall, so muss eine andere Aufteilung des Gesamtsystems in Teil-DFGs vorgenommen werden.

Die gewählte Schnittstelle (Ausgänge $\underline{y}_D$, $\underline{y}_{ND}$ und Eingänge $\underline{u}_D$, $\underline{u}_S$) für jeden Teil-DFG dient zur Blackbox-Kopplung der Module. D. h. es werden keine weiteren Daten aus Zwischenberechnungen in den ND-, D-, S-Knoten benötigt, so dass der mathematische Inhalt der Knoten nach aussen nicht bekannt ist. Trotzdem reicht eine Schnittstellen-Kompatibilität zur Kopplung nicht aus, um das gleiche Ergebnis wie bei einer zentralen Lösung des ursprünglichen Gesamtsystems zu erzielen. Es sind folgende Randbedingungen zwingend einzuhalten:

- gleiche Simulationsschrittweite
- gleiches numerisches Integrationsverfahren
- Datenaustausch in den Haupt- und Nebenschritten des Integrationsverfahrens [Eppinger 1994]
- gleiche numerische Zahlengenauigkeit

Werden diese Randbedingungen verletzt, so entstehen zusätzliche numerische Fehler, die das Ergebnis der Simulation völlig unbrauchbar machen können. Will man zur Rechenzeitersparnis dennoch verschiedene Simulationsschrittweiten nutzen, so ist eine Fehlerschätzung notwendig

[Rükgauer 1996].

5.2 Laufzeitplattform zur modularen Echtzeit-Simulation

Für die modulare Echtzeit-Simulation wird IPANEMA (vgl. Kapitel 2.4) als Laufzeitplattform verwendet, da IPANEMA nach einem offenen einfach zu erweiternden Baukastenprinzip aufgebaut ist. D. h. es liegen modulare SW-Bausteine als Quelltext in ANSI-C vor, die weitest möglich Hardware-unabhängig weiterverwendet werden können. Der Vollständigkeit halber seien auch zwei weitverbreitete industrielle Lösungen zur modularen Echtzeit-Simulation im HIL-Umfeld erwähnt, die allerdings nicht vollständig offen und erweiterbar sind:

Kiffmeier [Kiffmeier 1995] realisiert modulare Echtzeit-Implementierungen von regelungstechnischen Anwendungen durch automatische Code-Generierung auf der Basis von Blockdiagrammen. Dabei wird in einem SIMULINK-Blockdiagramm [MathWorks 1992] nicht nur die eigentliche Modelldynamik, sondern auch die Struktur des Netzwerkes inklusive der Kommunikationsverbindungen zwischen den Prozessoren festgelegt. Dieser grafische Ansatz wird bei dem von der Firma dSPACE angebotenen Real-Time-Interface zu SIMULINK für Multiprozessorsysteme (RTI-MP) verfolgt [dSPACE 1995].

Die Firma ETAS bietet mit ihrer Entwicklungsungebung ASCET-SD [ETAS 1996] ein ähnliches Konzept an. Auf der Ebene von Blockdiagrammen und Statecharts können Systeme modelliert werden. Die Codegeneriung für die Zielplattformen INTEL-Pentium (Offline Simulation), Transputer und Power-PC (Echtzeitsimulation) erfolgt automatisch. Multiprozessorsysteme werden prinzipiell unterstützt. Die Zuordnung von Berechnungsmodulen zu Prozessoren erfolgt wie bei RTI-MP von dSPACE manuell.

5.2.1 Verwendete Laufzeitplattform IPANEMA

Aufgrund neuer Anforderungen durch leistungsfähigere Echtzeit-Hardware und durch die Notwendigkeit zur Umsetzung von verschiedenen Taktraten (Multirate) sind komplexe Erweiterungen der ursprünglichen IPANEMA Variante vorgenommen worden.

In den ersten IPANEMA-Anwendungen auf Transputer-Systemen wurde eine sehr feingranulare Struktur der Anwendung angestrebt, da die einzelnen Rechenknoten des verteilten Systems nur sehr wenig Rechenleistung und Hauptspeicher zur Verfügung stellen konnten [Honekamp 1998]. Jedes IPANEMA-Objekt wurde auf genau einem Prozessor abgebildet. Für leistungsstarke Echtzeit-Hardware (z. B. MPC555, MPC750, etc.) ist es problemlos möglich, mehrere IPANEMA-Objekte pro Prozessor zu verarbeiten. Es muss daher möglich sein, mehrere IPANEMA-Objekte pro Prozessor/Prozess in Form von Tasks zu implementieren. Die datengetriebene Ausführung von verteilten Applikationen durch blockierendes Empfangen ist bei der Verwendung von Tasks nicht mehr möglich. Hier wird eine zeitgesteuerte Ausführung auf der Basis eines Master-Timers vorgeschlagen.

Für das Konzept der Calculator-Auswertung in drei Schritten (ND-, D-, S-Schritt) sind weitere Änderungen des internen Kontrollflusses der Calculatoren notwendig. Dies geht einher mit der Realisierung verschiedener Taktraten auf der Basis von präemptivem Multitasking für die Calculatoren. Bisher waren nur Singlerate-Anwendungen möglich.

5.2.2 Abbildung der IPANEMA-Objekte auf Multitasking

Um die IPANEMA-Objekte auf ein Echtzeitsystem abzubilden, gibt es grundsätzlich die Möglichkeit, Single- oder Multitasking zu verwenden. Dabei wird von folgenden Begriffen ausgegangen [Ungerer 1993]:

1. **Programmebene**:
   Mehrere untereinander kommunizierende Prozesse auf einem oder mehreren Prozessoren laufen (quasi) gleichzeitig ab und benötigen keinen Datenaustausch untereinander.

2. **Prozessebene**:
   Auf dieser Ebene werden innerhalb eines Programms konkurrierende Tasks (Prozess) realisiert. Jeder Task (Prozess) wird sequentiell abgearbeitet und besitzt einen eigenen Daten- und Programmbereich (Multitasking).

3. **Blockebene**:
   Leichtgewichtige Prozesse (auch Threads genannt) ohne eigenen Daten- und Programmbereich kommunizieren auf der Basis von gemeinsamen Daten.

4. **Anweisungsebene** (oder Befehlsebene):
   Elementare Anweisungen (in der Sprache nicht weiter zerlegbare Datenoperationen) werden nebenläufig ausgeführt.

5. **Suboperationsebene**:
   Eine elementare Anweisung wird durch den Compiler oder in der Maschine in Suboperationen aufgebrochen, die parallel ausgeführt werden. Beispiel: Vektoroperationen.

Für Echtzeitsysteme unterscheidet man sinnvollerweise zusätzlich zwischen verschiedenen Task-Typen:

- **Echtzeit-Task**:
  Echtzeit-Tasks sind Tasks mit harten Zeitschranken, die an periodische Echtzeit-Timer gekoppelt sind.

- **Hintergrund-Task**:
  Hintergrund-Tasks sind Tasks mit weichen Zeitschranken, die an asynchrone Anwenderzugriffe gekoppelt sind.

Für die Einbeziehung von Multitasking zur Umsetzung wird die folgende Erweiterung vorgenommen: Es gibt genau einen sogenannten IPANEMA-Master, der mindestens den Moderator und ein Calculator-Assistant- oder Adaptor-Assistant-Paar enthält. Die einzelnen IPANEMA-Objekte werden im Master als Echtzeit- und Hintergrund-Tasks implementiert. Der Master läuft auf genau einem Rechenknoten ab. Für verteilte Anwendungen wird pro externem Rechenknoten ein sogenannter Servant implementiert. Ein Servant enthält keinen Moderator, aber mindestens

ein Calculator-Assistant- oder Adaptor-Assistant-Paar:

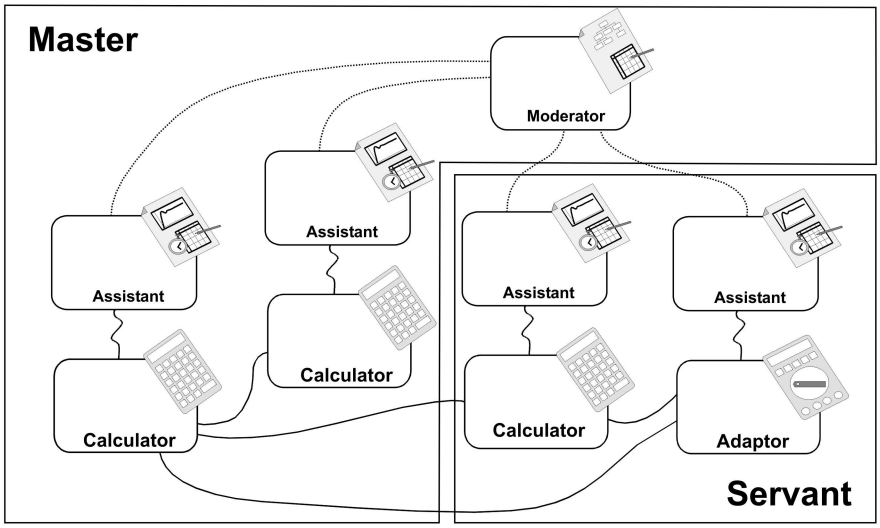

Abbildung 5-3: Master und Servant in IPANEMA

Die nachfolgende Abbildung verdeutlicht die Kombinationsmöglichkeiten für Programme, Tasks und Calculatoren. Sie zeigt drei CPUs mit je einem Programm und drei Timer-Tasks, an die vier Calculatoren gekoppelt sind. Die Timer sind lokal auf der Master-CPU angeordnet und lösen lokale Interrupts und Inter-Prozessor-Interrupts (IPI) auf den Slave-CPUs aus. Schreiben und Lesen von Daten (nicht abgebildet) erfolgen vor bzw. nach den Berechnungen im Calculator:

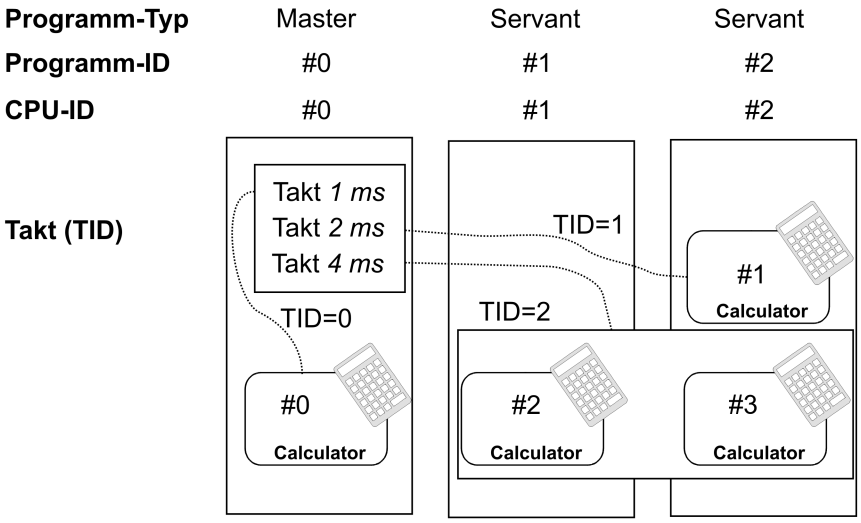

Abbildung 5-4: Organisation der Programme, Calculatoren und Taktraten

Der Anwender ordnet jedem Calculator eine Taktrate, eine Programm-ID und die zugehörige CPU zu. Damit steht die Anwendungstopologie der Software fest. Es lässt sich jede beliebige (sinnvolle) Topologie konfigurieren. Die einfachste Anwendung besteht aus einem Programm, einer Task und einem Calculator. Komplexe Anwendungen bestehen aus mehreren Programmen mit jeweils mehreren eigenen Tasks. Die Organisation für alle Calculatoren, Tasks und Programme erfolgt nach bestimmten Regeln:

- Ein Calculator ist genau einer CPU zugeordnet.
- Ein Adaptor wird genauso wie ein Calculator behandelt. Bei Adaptoren ist die jitterfreie Ein-/Ausgabe der Mess- und Stellgrößen zu beachten; die Messgrößen werden innerhalb des ND-Teils gelesen (Datenquelle), die Stellgrößen im S-Teil ausgegeben.
- Tasks können CPU-übergreifend organisiert sein.
- Die Zykluszeit einer Tasks (Task-Periodendauer) muss immer ein ganzzahliges Vielfaches der kleinsten vorkommenden Zykluszeit betragen.
- Die Pufferung von Daten im Assistant erfolgt entsprechend der lokalen Abtastrate des zugehörigen Calculators. Für die langsameren Abtastungen werden die letzten Werte mehrfach eingetragen (Data-Hold), so dass die Ringpuffer aller Assistants von der Datengröße her synchronisiert sind.

Für die CPUs, Programme, Calculatoren und Taktraten werden Identifier (IDs) nach den folgenden Regeln vergeben:

- Die Takt-Prioritäten werden nach Taktrate vergeben, der schnellste Takt hat die ID null.
- Der Master besitzt immer die Programm-ID null.
- Der schnellste Calculator (Adaptor) hat die Calculator-ID null und gehört damit zu Takt-ID null.
- Der langsamste Calculator (Adaptor) hat die höchste Calculator-ID.
- Calculatoren und Adaptoren werden nicht getrennt numeriert.

5.2.3 Scheduling

Der Scheduler hat die Aufgabe, den Ausführungszeitpunkt einer Tasks zu bestimmen. IPANEMA nutzt einen zentralen Scheduler, der im Master untergebracht ist. Für Echtzeitanwendungen ist dies eine einfache und robuste Lösung, die ein besser überschaubares Zeitverhalten auch für verteilte Anwendungen ermöglicht. Das Grundproblem der präzisen Timer-Synchronisation (globale Zeit) in verteilten Echtzeitanwendungen wird somit entschärft.

Die IPANEMA-Schicht zur Task-Verwaltung wird vom Betriebssystem-Scheduler aufgerufen. IPANEMA unterstützt dabei das Konzept des präemptiven Multitasking, d. h. aktive Tasks können durch neue Tasks unterbrochen (verdrängt) werden. Voraussetzung ist, dass der Betriebssystem-Scheduler diese Funktionalität bietet. Für Offline-Simulationen ohne Echtzeitbedingungen wird kein präemptives Multitasking verwendet. Die Tasks werden rein sequentiell abgearbeitet, so dass eine sehr hohe Effizienz bei der Verarbeitung aufgrund wegfallender Zeiten für den Taskwechsel und das Scheduling etc. gewährleistet ist.

Die nachfolgende Abbildung verdeutlicht den Vorgang von der Auslösung eines Timer-Interrupts durch die RT-Hardware bis zur Aktivierung eines Calculator-Blocks (ND, D, S). IPANEMA konfiguriert für jede Taktrate einen Timer des RTOS (Real-Time Operating System) und registriert eine zugehörige Interrupt-Service-Routine (ISR) als Einsprungfunktion beim RTOS-Scheduler. Wird eine ISR aufgerufen, so kann IPANEMA auf der Basis einer vorab bestimmten Task-Matrix die zugehörige Calculator-Funktion aufrufen, gerade aktive Berechnungen werden unterbrochen

und später fortgeführt. Die Task-Verwaltung von IPANEMA enthält dabei selbst keinen hardwareabhängigen Code:

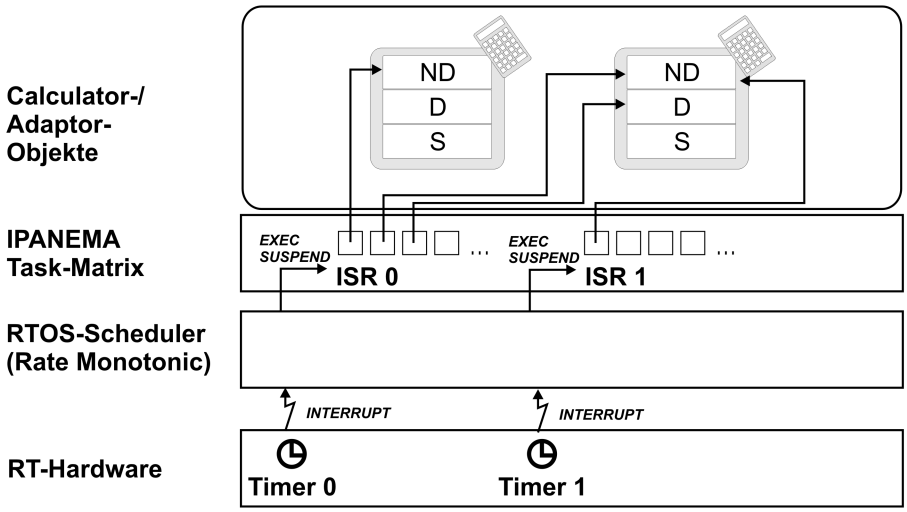

Abbildung 5-5: Task-Verwaltung von IPANEMA

Für die Echtzeitverarbeitung eignet sich als Scheduling-Verfahren das Rate-Monotonic-Scheduling (RMS) besonders. Es beruht auf dem Prinzip, dass eine Task um so wichtiger ist, je höher ihre Taktrate ist [Buttazzo 1997]. Damit verdrängt eine schnellere Task immer eine langsamere. Dieses Verfahren wird z. B. für das DS1005-System durch den dSPACE-Echtzeit-Kern [Otterbach, Leinfellner 1999] bereitgestellt. Es gilt zu beachten, dass die Anzahl der Tasks möglichst niedrig gewählt wird. Dadurch werden die Zeiten für das Scheduling und den Task-Wechsel minimiert. So wird die Prozessorauslastung U_P verbessert:

$$U_P(n) = \sum_{i=1}^{n_T} \frac{T_{ET,\,i}}{T_{s,\,i}} \tag{5.17}$$

TABELLE 3. Formelzeichen zu Gl. 5.17

Prozessorauslastung	Anzahl von Tasks	Ausführungszeit[a]	Task-Periodendauer
U_P	n_T	$T_{ET,i}$	$T_{s,i}$

a. Eigene Ausführungszeit ohne Unterbrechungen durch andere Tasks (Task-Execution-Time)

Die theoretische Obergrenze der Prozessorauslastung für das Rate-Monotonic-Scheduling $U_{P,max}$ lautet [Liu, Layland 1973]:

$$U_{P,\,max}(n_T) = n_T \cdot (\sqrt[n_T]{2} - 1) \tag{5.18}$$

Die Prozessorauslastung für zwei Tasks kann damit einen Wert von *82,8 %* und für drei Tasks einen von *78,0 %* nicht übersteigen. Für $U_{P,max}(\infty)$ ergibt sich ein Wert von *69,3 %*.

5.2.4 IPANEMA-Task-Matrix

In der IPANEMA-Task-Matrix werden die Informationen über die Auswertereihenfolge der Cal-

culator-Blöcke (ND, D, S) zeilenweise und deren Zuordnungen zu Tasks spaltenweise gespeichert. Die Anzahl der Spalten ist größer als die Anzahl der Tasks, da zu einem Zeitpunkt mehrere Tasks gleichzeitig aktiviert werden können. Vor dem Start einer IPANEMA-Echtzeitapplikation wird die Task-Matrix nach folgendem Verfahren im Moderator automatisch erzeugt und geprüft:

1. Durch eine topologische Sortierung (vgl. Kap. 2.2) des Datenflussgraphen (drei Knoten pro Calculator) wird die sequentielle Reihenfolge für eine vollständige Auswertung bestimmt (sortierte Liste als Ergebnis). Dies ist nur für einen kreisfreien DFG möglich. Die vollständige Auswertung wird für den Zeitpunkt *t = 0* und für alle Zeitpunkte, in denen alle Tasks aktiv sind, benötigt. Diese vollständige Auswertung entspricht dem Singlerate-Anwendungsfall.

2. Für jede Kombination aktiver Tasks wird eine Submatrix folgendermaßen ermittelt:
   2a) Das Ergebnis aus 1) wird in die erste Spalte der Submatrizen kopiert, und alle Einträge, die nicht zum zugehörigen Zeitpunkt aktiviert werden sollen, werden gelöscht.
   2b) Die Spalten werden auf die Spalte der jeweils langsamsten Tasks, für die Einträge existieren, verschoben.

Das Verfahren bedeutet eine Zeitersparnis zur Laufzeit, da die Prüfung auf Ausführung nicht notwendig ist. Die Multirate-Ausführung erfolgt vorgeplant. Zur Verdeutlichung wird exemplarisch die Task-Matrix für die nachfolgende Multirate-Anwendung ermittelt:

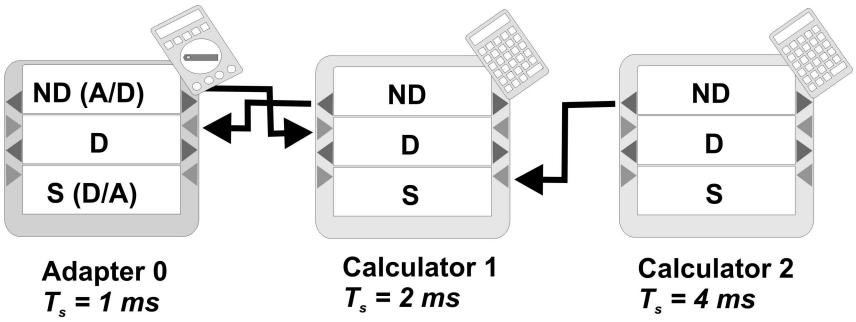

Abbildung 5-6: Beispiel für eine Multirate-Anwendung

Abbildung 5-7 zeigt die zugehörige Task-Matrix für das obige Beispiel. Sie besteht aus neun Zeilen und zwölf Spalten, die auf vier Submatrizen verteilt sind.

Für die frei gewählten Abtastraten werden drei zugehörige Timer konfiguriert. Für den Zeitpunkt *t = 0* werden alle Timer einen Interrupt in der Reihenfolge 0, 1, 2 auslösen. Bei Aktivierung von Task 2 erfolgt die sequentielle Berechnung aller Blöcke. Durch diese Zuordnung kann die Berechnung aufgrund des RMS-Verfahrens nach einer Millisekunde durch Task 0, nach zwei Millisekunden durch Task 1 und nach drei Millisekunden erneut durch Task 0 unterbrochen werden. Die Echtzeitbedingungen sind dann eingehalten, wenn kein Calculator-/Adaptor-Block eine erneute Berechnung beginnt, bevor eine vorangegangene beendet ist. Diese Bedingungen werden zur Laufzeit überprüft. Die Sende-/Empfangs-Vorgänge finden am Anfang/Ende der Calculator-/Adaptor-Blöcke statt. Durch die vorsortierte (kreisfreie) Ausführung ist die korrekte Sende-/

Empfangsreihenfolge sichergestellt:

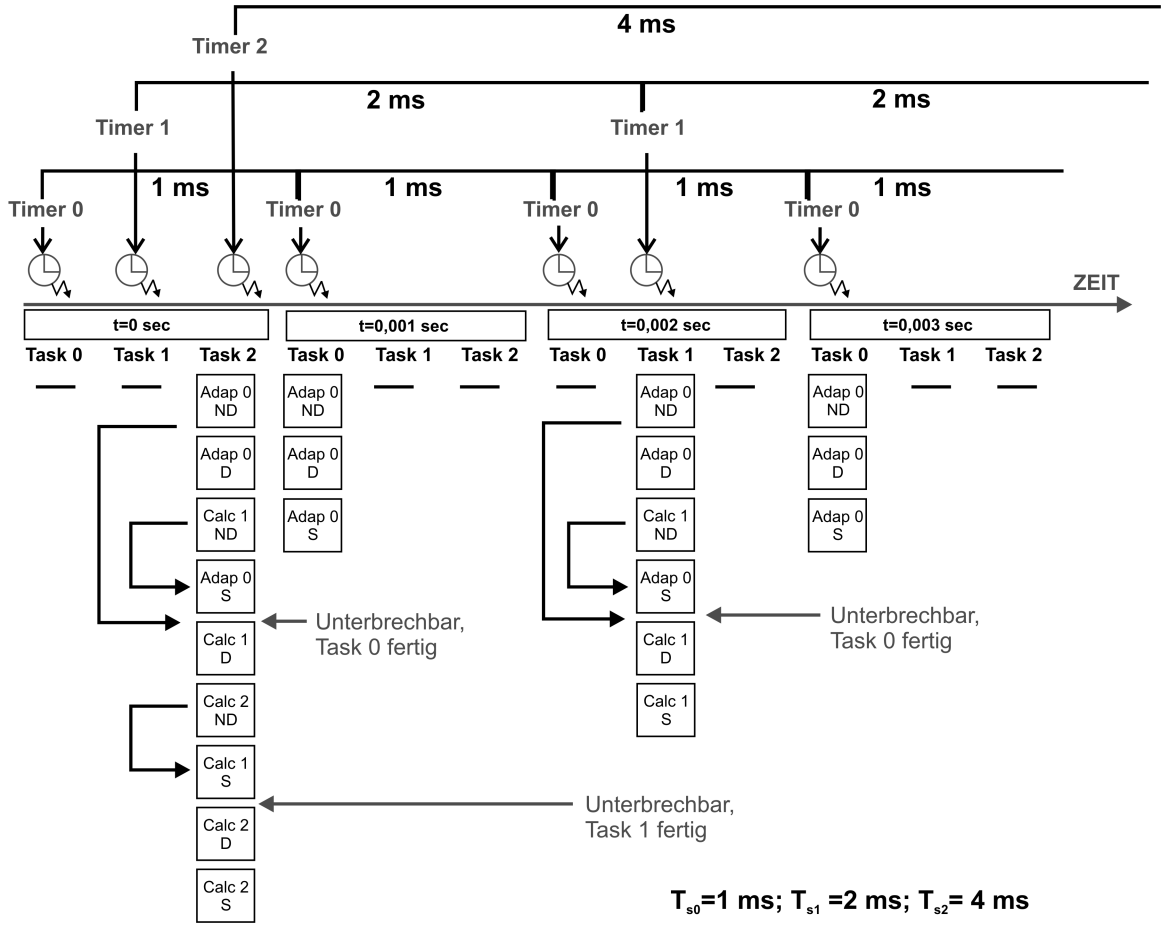

Abbildung 5-7: IPANEMA-Task-Matrix

5.3 Parallelverarbeitung auf AMS-/MFM-Ebene

Die Streckenmodelle mechatronischer Systeme (Systemmodelle) besitzen aufgrund ihrer modular-hierarchischen Struktur eine natürliche inhärente Parallelität. Im Folgenden soll das Potential der Parallelisierung von Systemmodellen zur schnellen Simulation erörtert werden.

5.3.1 Parallelisierung von Systemmodellen zur schnellen Simulation

Das Systemmodell als Teil des erweiterten Streckenmodells beschreibt die mechanischen, die elektrischen und die hydraulischen Bestandteile der Aktorik, der Sensorik und der Tragstrukturen eines mechatronischen Systems auf den AMS-/MFM-Ebenen. Mit Hilfe des OMM lassen sich die Beschreibungsebenen Topologie, Verhalten und Verarbeitung erzeugen und verwalten. Die topologische Struktur eines Systemmodells orientiert sich an der Aggregat-Struktur des mechatronischen Systems und ist daher konsequent modular-hierarchisch aufgebaut (siehe Kap. 3).

Es soll im Folgenden aufgezeigt werden, welche Erfahrungen mit dem Laufzeitverhalten bei der nichtlinearen Simulation von parallelisierten Systemmodellen auf der Basis von Datenflussgraphen auf Blockebene gemacht wurden.

Für eine Parallelverarbeitung mit dem Ziel, die Simulationszeit gegenüber einem sequentiellen

Ablauf zu minimieren (d. h. einen Speedup zu erzielen), ist das Verhältnis zwischen der sequentiellen und der parallelen Rechenzeit inkl. dem Zeitverlust durch die Kommunikation von entscheidender Bedeutung. Zur Beurteilung des Potentials für eine Rechenzeitersparnis müssen die sequentielle Rechenzeit, der längste sequentielle Berechnungspfad (Critical Path) im Datenflussgraph und die Zeitverluste durch Kommunikation entlang des kritischen Pfades geschätzt werden.

Die sequentielle Rechenzeit wird dabei für einen Simulationsschritt betrachtet. Ein sequentieller Simulationsschritt bedeutet die Berechnung der ND-, D- und S-Knoten durch eine Task. Die Rechenzeit für einen Simulationsschritt einer modular verteilten Simulation (vgl. Abbildung 5-2) ergibt sich aus der einmaligen Auswertung aller verteilten ND-, D- und S-Knoten inkl. Kommunikationszeiten. In beiden Fällen ist im S-Knoten die Rechenzeit für einen Einschritt DGL-Löser (z. B. Euler) enthalten, die aber vernachlässigbar gering ist (Bei DGL-System-Ordnung N: N Multiplikationen und N Additionen, vgl. Gl. 5.13).

Die maximale Simulationsgeschwindigkeit eines parallelisierten Systemmodells hängt demnach von der Struktur des Datenflussgraphen und der Rechen- und Kommunikationsleistung der verwendeten Hardware ab.

5.3.2 Analyse von Systemmodellen und Lastverteilung

Zur Analyse der Systemmodelle im Hinblick auf parallele Strukturen auf der Verarbeitungsebene werden Datenflussgraphen verwendet. Knoten in diesem gerichteten Graph sind die ND-, D- und S-Bestandteile der Basissysteme (Koppelsysteme sind auf der Verarbeitungsebene nicht mehr vorhanden). Kanten werden zwischen den zusammengehörigen ND-, D- und S-Knoten aufgrund der inneren Datenabhängigkeiten (z. B. Systemzustände) und zwischen Knoten mit Ein-/Ausgangsabhängigkeiten eingefügt. Die Knoten werden mit den Rechenzeiten (für eine spezifische Hardware) und die Kanten mit den Kommunikationszeiten (ebenfalls spezifisch) gewichtet. Die nachfolgende Abbildung zeigt exemplarisch einen Datenflussgraphen für das Modell einer aktiv

gefederten Achse eines Fahrzeugs (ohne Gewichtungen):

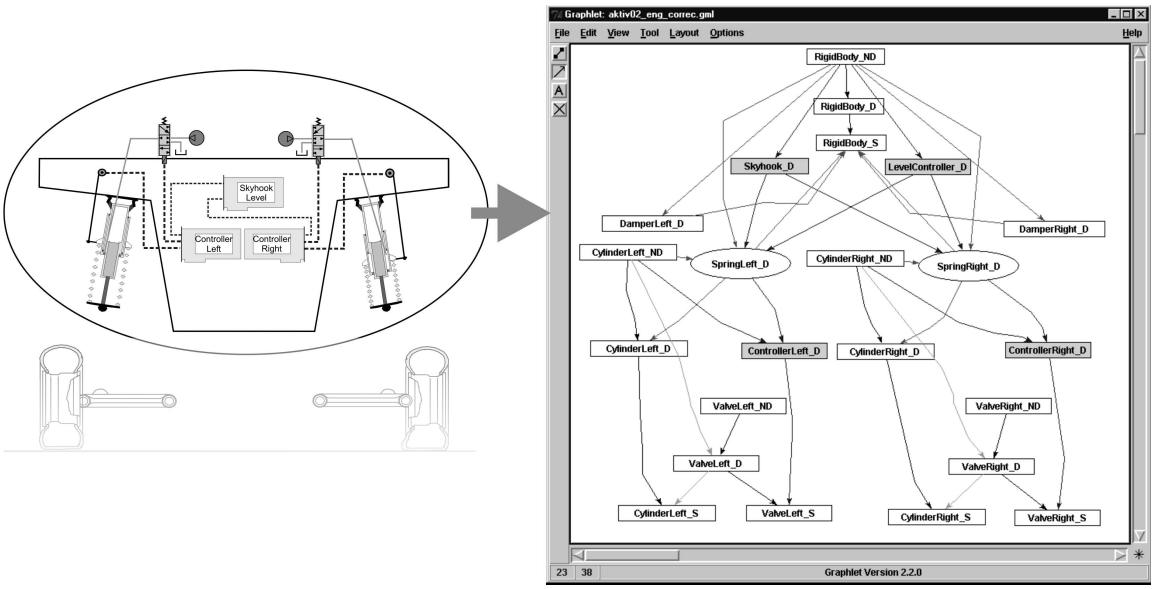

Abbildung 5-8: Beispiel für einen Datenflussgraph (aus [Deppe, Homburg 1998])

Vor einer Analyse des DFG ist es sinnvoll, zunächst die sequentielle Laufzeit für ein Modell zu bestimmen. Nachfolgend sind dann eine Vorauswahl der parallelen Hardware und die sinnvolle Aufteilung (Lastverteilung) des DFG möglich. Im Folgenden wird als Beispiel das Modell eines Waldnutzfahrzeuges auf sein Laufzeitverhalten hin untersucht.

Sequentielle Laufzeit für das Modell eines Waldnutzfahrzeugs

Das Modell beschreibt das passive Fahrwerk eines dreiachsigen Waldnutzfahrzeuges mit zwei gelenkig verbundenen Chassisteilen. Der vordere Aufbau besitzt dabei zwei, der hintere vier Räder. Es ist lediglich die Funktion "Federung" modelliert, die Funktionen "Lenkung" und "Antrieb" werden nicht berücksichtigt:

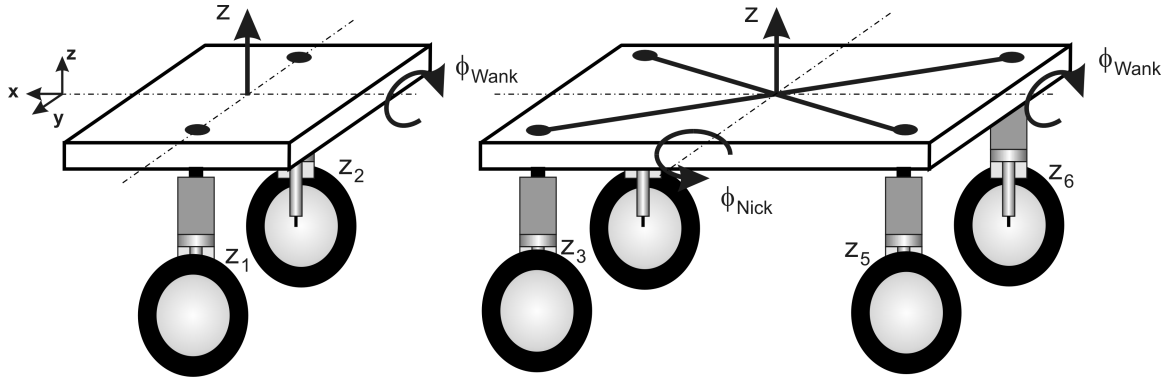

Abbildung 5-9: Waldnutzfahrzeug: Starrkörpermodell

Die sechs Räder und zwei Aufbauteile sind durch acht kraftgekoppelte Starrkörper modelliert. Die Räder sind durch je ein Gelenk, das nur eine vertikale Translation zuläßt, an den Aufbau gekoppelt. Die beiden Aufbauteile sind durch ein Gelenk verbunden. Hier ist nur eine Rotation um die vertikale Gelenkachse möglich. Ein weiteres Gelenk dient zur Anbindung des vorderen Aufbaus an ein Umgebungsmodell (Terrain- bzw. Straßenmodell). Es ergeben sich insgesamt

dreizehn Freiheitsgrade aus den sechs Freiheitsgraden der Räder, aus einem Freiheitsgrad des Aufbaugelenks und aus sechs Freiheitsgraden der Ankopplung des vorderen Aufbaus an die Umgebung.

Das Modell wurde in CAMeL-View ([iXtronics 2001/1], [iXtronics 2001/2]) erstellt. Es ergibt sich ein eng gekoppeltes ODE-System mit der Ordnung *104* (d. h. es existieren *104* skalare Systemzustandsvariablen). Insgesamt sind *20.911* mathematische Funktionen für einen Auswerteschritt der Modellgleichungen auf Verarbeitungsebene des OMM notwendig:

TABELLE 4. Funktionsaufschlüsselung für das Waldnutzfahrzeug-Modell

Funktion	**Anzahl im Modell**	**Funktion**	**Anzahl im Modell**
Multiplikation	9.345	Sinus	81
Addition	5.384	IF-Abfrage (ohne ELSE)	67
Zuweisung	3.699	IF-Abfrage	51
Subtraktion	939	Arcus Sinus	48
Vorzeichenwechsel	842	Betrag	32
Division	233	weitere Funktionen	85
Cosinus	105		
		Gesamt:	**20.911**

Es entsteht kein reiner Datenfluss: Es existieren *51* IF-Abfragen und nochmal *67* IF-Abfragen ohne ELSE-Zweig. Die in Relation zu den anderen Funktionen geringe Anzahl der IF-Abfragen und die Tatsache, daß die TRUE- und FALSE-Zweige jeweils wenige Anweisungen enthalten, bewirken daher nur eine geringe Schwankungsbreite der Ausführungszeit. Die IF-Abfragen entstehen z. B. durch notwendige Fallunterscheidungen zur numerischen Berechnung von Koordinatentransformationen etc. in den Bewegungsgleichungen des Systems.

Stellt man die Durchgriffe (Datenabhängigkeiten innerhalb eines Zeitschrittes) des Modells auf Blockebene (siehe [Stolpe et al. 1997], S. 19 ff.) dar, so erhält man einen gerichteten azyklischen Graphen mit *180* Knoten (Basissystemen) und *165* Kanten. Bei zusätzlicher Aufteilung der Basissysteme in ihre ND-, D- und S-Teile ergeben sich *549* Knoten und mehr als *525* Kanten.

TABELLE 5. Laufzeitmessung für das Waldnutzfahrzeug-Modell

Betriebssystem	**CPU**	**Compiler**	**Programmgröße**	**Rechenschritte**	**Mittlere Realzeit pro Schritt**[a]
Windows 2000	Athlon MP 1800+	Watcom 10.6[b]	869 KB	10.000	58 µs

a. Nur Modellgleichungen ohne DGL-Löser, aus Gesamtlaufzeit durch Anzahl Rechenschritte bestimmt

b. Höchste Optimierungsstufe eingeschaltet

Aus der Funktionsaufschlüsselung und der Leistungs-Angabe einer CPU lässt sich natürlich nicht ohne weiteres auf die Ausführungszeit der Evaluierung der Modellgleichungen schließen, da komplexere Funktionen in Software abgebildet sind und Caching- und Pipelining-Effekte berücksichtigt werden müssen [Puschner 2002]. Auch haben Compiler, Mainboard, Speicher, Betriebssystem etc. Einfluss auf die Ausführungszeit. Bei datenflussorientierten Anwendungen

ist eine Messung eine effiziente Methode zur Laufzeitbestimmung (vgl. Tabelle 5).

Aufgrund des geringen Laufzeitwertes von $58\,\mu s$ (vgl. Tabelle 5) zur einmaligen sequentiellen Auswertung des DFG und der hohen Dichte des DFG und dem damit verbundenen Kommunikationsaufwand lässt sich auf einem PC keine effiziente parallele Simulation des Modells erwarten.

Lastverteilung

Die Lastverteilung von Datenflussgraphen, d. h. die Partitionierung und die Zuordnung zu Prozessoren, wurde ursprünglich auf der Ebene von skalaren Gleichungen betrachtet und als heuristisches Optimierungsproblem behandelt ([Naumann, Homburg 1996], [Deppe 1995]). Wegen der großen Anzahl von Auswertungen für das heuristische Verfahren (z. B. Simulated Annealing) und die sehr komplexen Datenflussgraphen ergaben sich sehr schnell lange Optimierungszeiten im Bereich von Stunden.

Im Rahmen einer Diplomarbeit [Hees 1999] wurden verschiedene algorithmische Methoden zu Clustering (Zusammenfassen von Knoten), Mapping (Zuordnung von Prozessoren zu Clustern) und Kommunikationszusammenfassung für Systemmodelle untersucht. Der Datenflussgraph des Systemmodells wurde auf Basissystemebene durch Aufteilung der mathematischen Blöcke in ND-, D- und S-Anteile erstellt.

Die Ergebnisse der Untersuchungen basieren auf den Daten von Transputer-Hardware (T800). Diese Technologie ist inzwischen veraltet, bot aber eine hervorragende Plattform für verteilte Anwendungen. Dies basierte vor allem auf dem günstigen Verhältnis zwischen Rechen- und Kommunikationsleistung der Transputer. Dieses Verhältnis bestimmt aber gerade die hier interessierende relative Performance-Verbesserung einer verteilten Anwendung im Vergleich zu einer sequentiellen Lösung.

Die Kommunikationszeiten wurden mit Hilfe des sog. *LogP-Modells* [Culler et al. 1993] modelliert, das Latenz- und Overheadzeiten berücksichtigt. Für die untersuchten Fahrzeugmodelle ergaben sich sog. dichte Graphen mit größerer Kanten- als Knotenanzahl. Die Ergebnisse lauten, tabellarisch zusammengefasst:

TABELLE 6. Ergebnisse für die untersuchten Systemmodelle

Modellname	Ordnung	Anzahl Starrkörper	DFG-Größe	Seq. Zykluszeit T800, 20 MHz	Theoretischer max. Speedup
Aktive Achse[a]	17	1	39 Knoten 57 Kanten	1,34 ms	1,59[b] (drei T800)
Zweispur[c]	32	5	153 Knoten 212 Kanten	51,3 ms	2,81 (vier T800)
EduTruck[d]	173	13	891 Knoten 1.204 Kanten	91,4 ms	3,66 (vier T800)

a. vgl. Abbildung 5-8

b. Dieser Wert ergibt sich durch Zeitmessung (gemessene parallele Rechenzeit = 0,84 ms)

c. handcodiertes 3D-Fahrzeugmodell mit Aufbaumasse und 4 Radmassen ohne Antriebsstrang, siehe [Stolpe et al. 1997]

d. steht für Educational-Truck. Das Modell wird in [Hahn 1999] beschrieben

5.3.3 Zusammenfassung

Die untersuchten Systemmodelle besitzen feingranulare, dichte Datenflussgraphen mit inhärenter Parallelität. Ein Speedup für die Simulation auf Transputer-Hardware war bereits recht begrenzt (*<4*). Für moderne Plattformen lassen sich aufgrund eines ungünstigeren Verhältnisses zwischen Rechen- und Kommunikationsgeschwindigkeit eher noch geringere Speedups erwarten. Will man die topologische Aggregat-Struktur miteinbeziehen, verschärft sich die Situation weiter, da zusätzlich einschränkende Randbedingungen erwachsen. Die rasante Prozessorentwicklung läuft dieser feingranularen Art von Anwendungen gewissermaßen davon, da die Taktfrequenzen auf lokal begrenzten, immer kleineren Prozessorkernen sehr viel schneller steigen als die mögliche Kommunikationsgeschwindigkeit zum Systembus bzw. Netzwerk.

Dieses Ergebnis scheint zunächst gegen eine Parallelverarbeitung zu sprechen. Hier gilt es zu bedenken, dass nur der Aspekt der Laufzeiteffizienz im Kontext einer schnellen Simulation von Streckenmodellen betrachtet wurde. Diese Modelle bestehen aus den systembeschreibenden Gleichungen zu Mechanik, Hydraulik und Elektrik der Strecke, die offensichtlich sehr feingranular verwoben sind (dichte Datenflussgraphen, d. h. mehr Kanten als Knoten). Im Fall der Modelle "Aktive Achse" und "EduTruck" sind zwar noch die Controller auf AMS- und MFM-Ebene enthalten, sie machen aber nur einen sehr geringen Teil der Gleichungen aus.

Was spricht dennoch für eine Parallelverarbeitung auf AMS-/MFM-Ebene?

1. Die schrittweise Entwicklung komplexer mechatronischer Systeme funktioniert nur mit Hilfe der HILS und erfordert die Verwendung von modularer Echtzeitsimulation ([Gambuzza 2002], [Gambuzza et al. 2003]) zur Nachbildung der Aggregat-Struktur des mechatronischen Systems [Stolpe 2004].
2. Zum *Rapid Prototyping* der Informationsverarbeitung mechatronischer Systeme ([Stolpe et al. 2000], [Deppe at al. 2001/1], [Deppe at al. 2003/3], [Deppe, Zanella 2002]) während der HILS und für den ersten Gesamtsystem-Prototypen ist eine Verteilung von Reglern/OCMs auf verteilte Steuergeräte erforderlich.
3. Für sicherheitskritische Systeme ist die Vorhaltung von Redundanzen unverzichtbar.

Zusammenfassend lässt sich somit feststellen, daß eine Parallelverarbeitung auf AMS-/MFM-Ebene unabdingbar ist, aber bei Schnitten durch Systemmodelle schnell zu einem Laufzeitanstieg führen kann.

6 Konzept zur verteilten Online-Mehrziel-Optimierung

6.1 Einleitung

Die Anwendungen der Online-Optimierung lassen sich nach [DFG-Antrag zum SFB 614 2001] grob in zwei Klassen einteilen, die hier "modellbasierte Optimierung" und "Nachoptimierung am realen System" genannt werden (vgl. auch Abbildung 6-2 und Abbildung 6-3):

Der Begriff "modellbasierte Optimierung" wird hier so verstanden, dass neue optimierte Systemparameter anhand eines Optimierungslaufs auf der Basis des erweiterten Streckenmodells (vgl. Kap. 3.3) ermittelt und erst anschließend in geeigneter Weise (Übertragungs- und Überblendfunktionen) auf die reale Strecke bzw. den realen Controller übertragen werden. Die Vorteile der modellbasierten Optimierung sind:

- Keine Gefahr bei instabilem System-Verhalten während der Optimierung
- Optimierungen schneller als Echtzeit sind möglich (Vorhersagen)
- Anregungsart und -energie können frei gewählt werden

Die direkte Kopplung eines Optimierungslaufs mit einer realen Strecke ist nur für HILS-Anwendungen geeignet. Eine Nachoptimierung am realen System benötigt eine Rückfallebene für den Fall von instabilen Reglern und eine verstärkte Systemüberwachung. Bedingt durch den Optimierungs-Algorithmus, kann es zur Auswahl von instabilen Parameter-Sätzen kommen. Für die Nachoptimierung am HILS-Prüfstand dürfen Instabilitäten nur über wenige Abtastschritte auftreten, bevor auf eine stabile Regelung zurückgeschaltet wird.

6.2 Modularität

Für die Implementierung von Online-Optimierung wurde am MLaP das sogenannte Operator-Controller-Modul (OCM**)** konzipiert und weiterentwickelt [DFG-Antrag zum SFB 614 2001]. Das OCM lässt sich prinzipiell in zwei Bereiche einteilen: den "Operator" und den "Controller". Der Operator beinhaltet die diskreten Elemente der Informationsverarbeitung, wie Notfall-Routinen, Regler-Überwachung und die Optimierung. Der Controller umfasst die kontinuierlichen, regelnden Teile der Informationsverarbeitung. Während der Controller im Wesentlichen auf der "quasi-kontinuierlichen" digitalen Regelungstheorie basiert, fußt der Operator auf diskreter Logik. Implementierungstechniken umfassen z. B. Zustandsmaschinen und prozessbasierte Systeme wie z. B. Work-Flow-Diagramme.

Die Funktionen "Überwachung" und "Übertragung" formen mittels spezieller Routinen kontinuierliche Signale in diskrete Signale um und umgekehrt. Die Überwachung von Zustandsvariablen des zeitkontinuierlichen Reglers und der physikalischen Strecke ist eine weitere wichtige Aufgabe, welche die Stabilität des Gesamtsystems bei Parameterwechseln in den Reglern sichert.

Zur Realisierung von hierarchischen und verteilten Optimierungsanwendungen auf Basis des OCM ist ein entsprechendes Modularisierungskonzept erforderlich. Hier wird eine Einteilung in drei Arten von Modulen vorgeschlagen, mit der auch komplexe OCMs verteilt und hierarchisch umsetzbar sind (siehe auch Abbildung 6-1):

1. **Optimierer**:
   Implementiert den eigentlichen Optimierungsalgorithmus und die Leitwarte zur Steuerung des gesamten Ablaufs. In jeder Anwendung existiert nur genau ein Optimierer.

2. **Evaluator** (Bewerter):
   Ist mit dem Optimierer verbunden und hat zwei Aufgaben: die Weitergabe von Parameteränderungen vom Optimierer zum Simulator und die Berechnung von Zielgrößen. Pro Anwendung existieren beliebig viele Evaluatoren.

3. **Simulator**:
   Simuliert die ODE-basierten Modelle der Anregung, des Controllers und der Strecke (System) numerisch und implementiert Parameter-Übertragung, System-Überwachung und Controller-Rückfallebene.

Durch diese Einteilung in die **O**ptimierer-**E**valuator-**S**imulator Struktur (**OES**) ergibt sich auch der OES-Datenflussgraph, der als Basis für die verteilte Realisierung von OCMs verwendet wird:

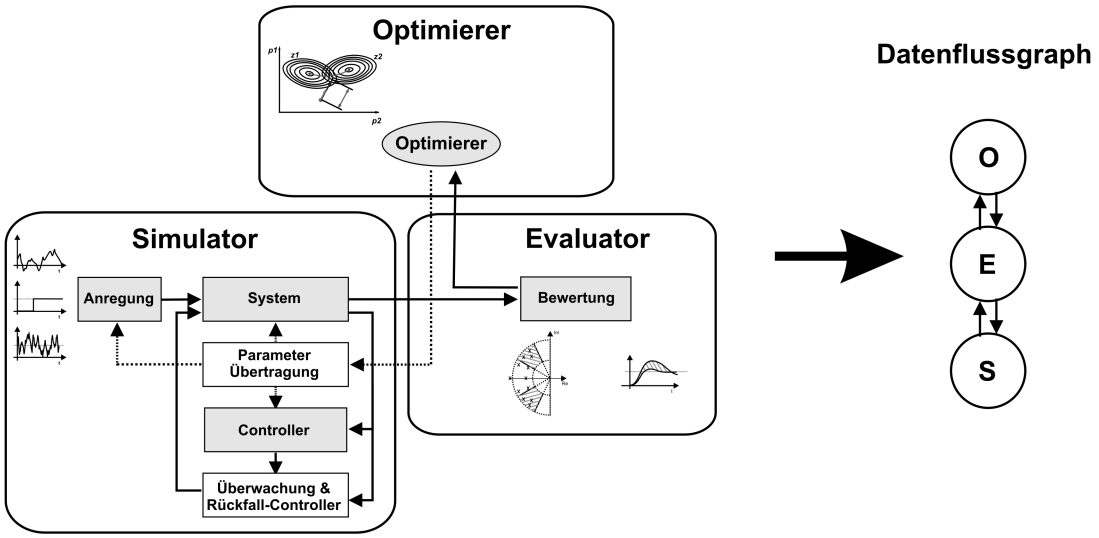

Abbildung 6-1: Modularisierung und Datenfluss für ein OCM

Die Ankopplung eines realen, online zu optimierenden Systems mit Hilfe der Konzepte "modellbasierte Optimierung" und "Nachoptimierung am realen System" ist kompatibel zum OES-Modularisierungskonzept. Dies ist möglich, da sich Struktur und Schnittstelle des Simulator-Moduls an der Umsetzung eines realen Systems orientieren. Für die "modellbasierte Optimierung" wird nach dem erfolgreichen Optimierungslauf der OES-Module ein neuer Parametervek-

tor auf das reale System aufgeschaltet:

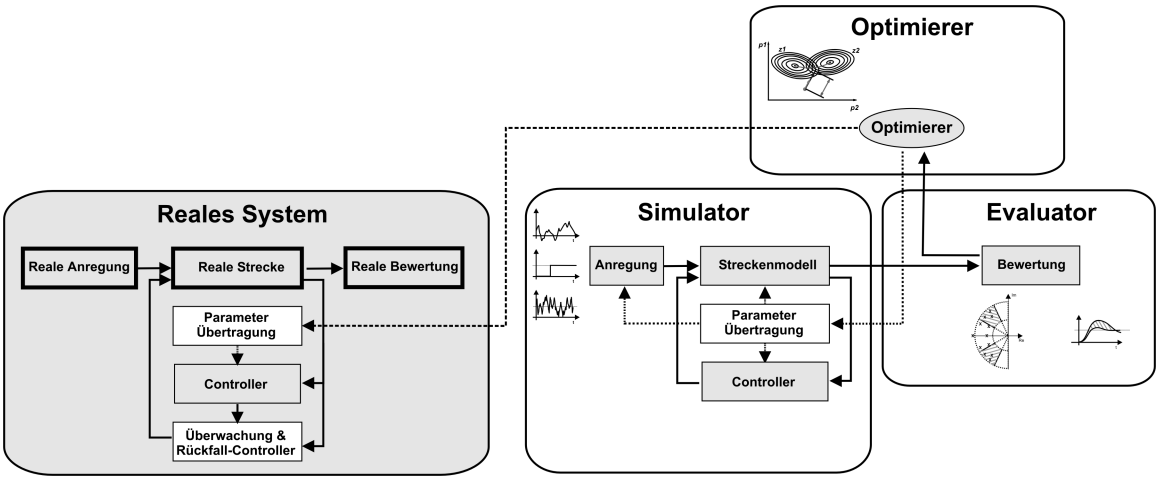

Abbildung 6-2: Modellbasierte Optimierung mit Optimierer, Evaluator und Simulator

Für die "Nachoptimierung am realen System" entfällt das Simulator-Modul und wird durch das reale System ersetzt:

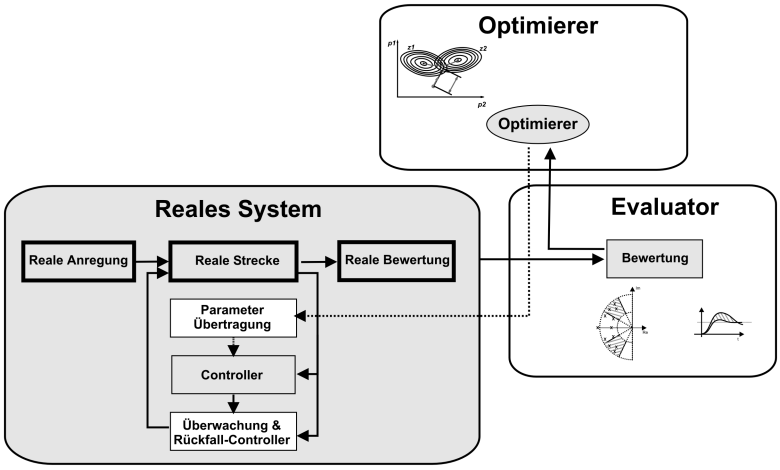

Abbildung 6-3: Online-Nachoptimierung am realen System mit Optimierer und Evaluator

6.3 Hierarchie

Das MLaP-Konzept zur modular-hierarchischen Strukturierung mechatronischer Systeme (vgl. Kap. 3.1) zeichnet sich dadurch aus, dass innerhalb eines Systems eine klare ebenenweise Hierarchisierung der OCMs existiert. Dabei hat jedes OCM die Fähigkeit zur modellbasierten Mehrziel-Optimierung. Dazu wird ein mathematisches Modell der eigenen Ebene und der unterlagerten Ebenen (in geeigneter Modellierungstiefe) verwendet: Ein OCM auf MFM-Ebene beschreibt das eigene MFM mit großer Modellierungstiefe, ein OCM auf AMS-Ebene beschreibt dasselbe MFM mit geringerer Tiefe.

Modelliert wird sinnvollerweise das dynamische Verhalten der geregelten physikalischen Aggregate der unterlagerten Ebenen. Eine Modellierung der gesamten Ebenen inklusive der Vorgänge in den Operator-Teilen der unterlagerten OCMs ist nicht zielführend bzw. zu komplex.

Durch die enthaltenen erweiterten Systemmodelle zur modellbasierten Optimierung ist in einem mechatronischen System quasi Wissen über die eigenen physikalischen Zusammenhänge und Funktionsanforderungen abgelegt. Es ist dadurch möglich, modellbasierte Vorhersagen zu Reaktionen auf bestimmte Systemveränderungen (z. B. Parameteränderungen) zu treffen. Dieses Prinzip wird im Rahmen des Sonderforschungsbereichs 376 „Massive Parallelität - Algorithmen, Entwurfsmethoden, Anwendungen“ als "Imagination", also als Vorstellung des Systems von sich selbst, bezeichnet.

Abbildung 6-4 zeigt die ebenenweise Hierarchie der OCMs am Beispiel "Selbstorganisierendes Kreuzungsmanagement" des SFB 376 (vgl. Kap. 7.4):

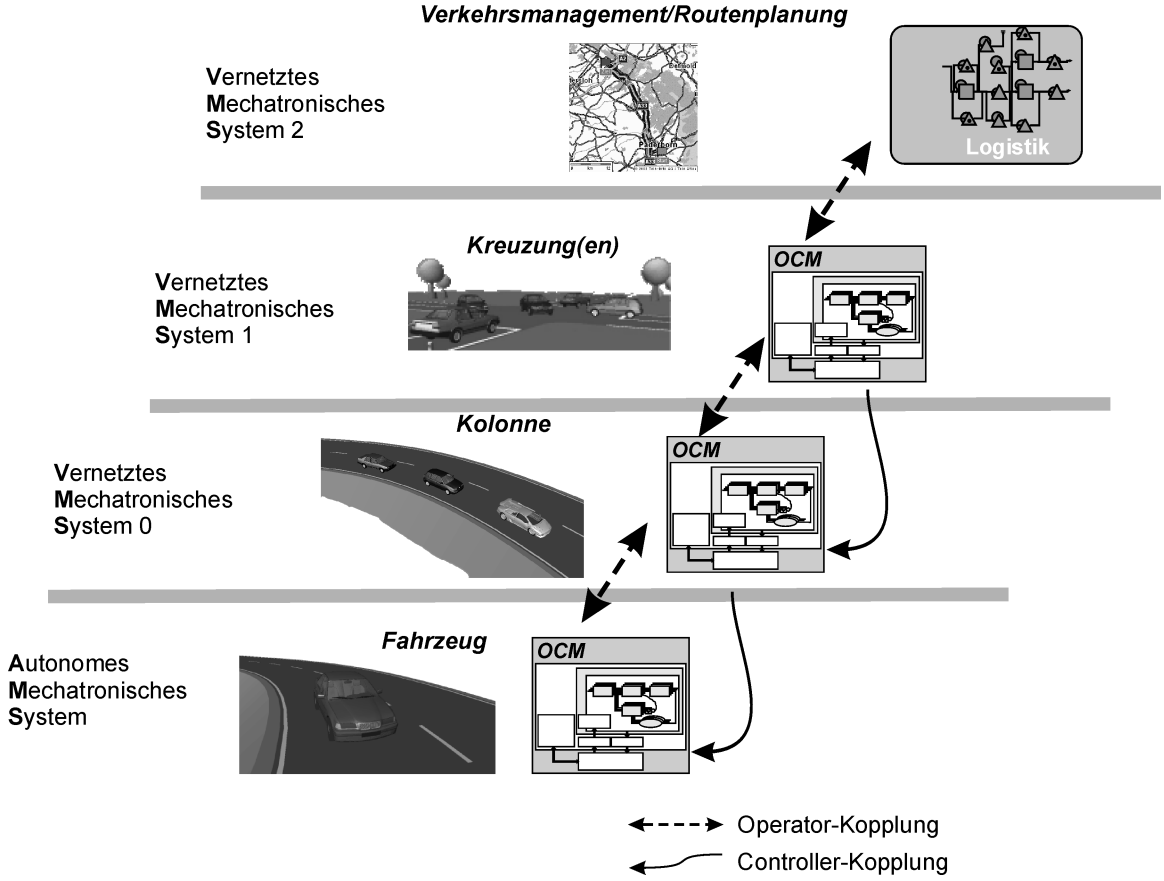

Abbildung 6-4: Ebenen AMS, VMS0, VMS1 und VMS2

Die VMS-Ebene ist in sich weiter hierarchisiert, so dass zwischen Fahrzeugkolonne (VMS0), Kreuzungsmanagement (VMS1) und Verkehrsmanagement (VMS2) unterschieden wird. Die VMS2-Ebene ist eine Betrachtung der Logistik (Verkehrsmanagement/Routenplanung) mit speziellen Modellen und Verfahren ausserhalb der Mechatronik.

6.3.1 Besonderheiten auf VMS1-Ebene

Zur Implementierung von Funktionen auf VMS1-Ebene, wie etwa ein selbstorganisierendes Kreuzungsmanagement, ist die Verwendung von Inter-Fahrzeug Kommunikation (z. B. durch ein Funknetzwerk) erforderlich, da sich die beteiligten Fahrzeuge ausserhalb des Bereiches von KFZ-Umfeld-Sensorik befinden. Dies bedeutet, dass die OCMs auf VMS1-Ebene aller Fahr-

zeuge gekoppelt sind und gemeinsam eine verteilte Online-Mehrziel-Optimierung durchführen müssen. Innerhalb der Hierarchieebene VMS1 ist also die Kooperation der beteiligten OCMs erforderlich.

Die nachfolgende Abbildung zeigt zwei denkbare Varianten zur Realisierung dieser Kooperation von OCMs verschiedener Fahrzeuge auf Basis von OES-Datenflussgraphen:

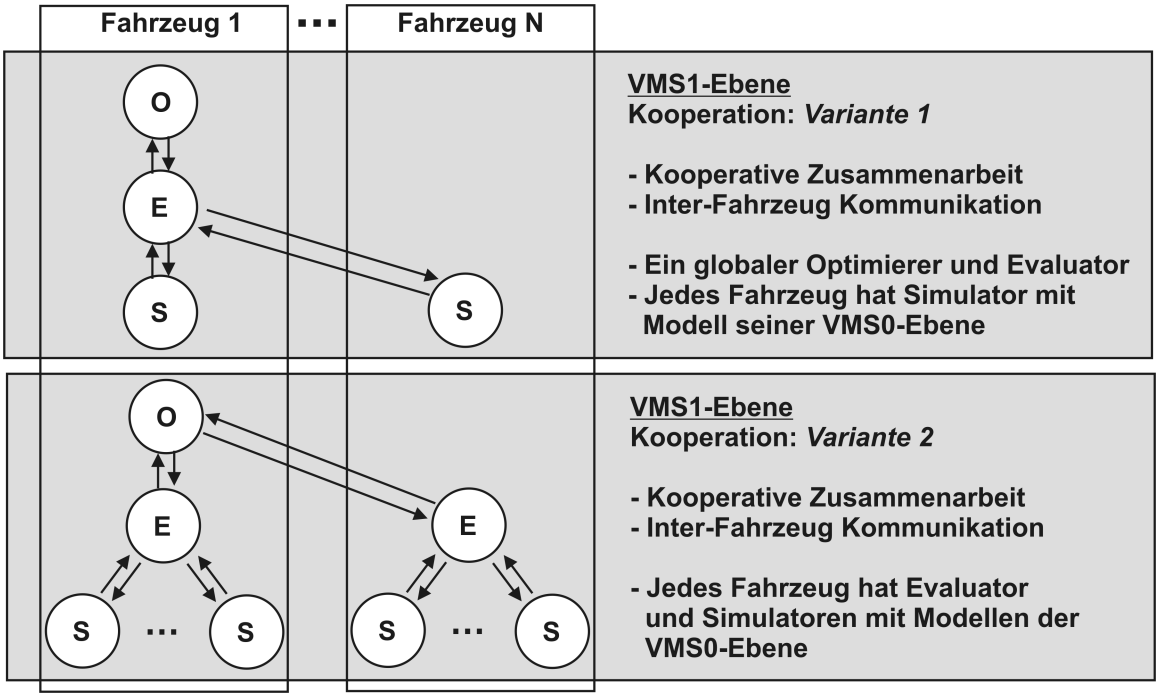

Abbildung 6-5: Varianten von verteilten OES-Strukturen auf VMS1-Ebene

6.4 Parallelität

Ein wichtiges Hilfsmittel zur Umsetzung in eine ausreichend schnelle Online-Optimierung ist die Parallelverarbeitung. Es lassen sich folgende Ebenen mit hohem Parallelisierungspotential identifizieren:

1. **Optimierungsebene**:
   Parallele Auswertung von unterschiedlich parametrierten Kopien der Systembewertung (Zielgrößenauswertung) durch mehrere Evaluatoren

2. **VMS1-Ebene (vgl. Abbildung 6-5, Variante 1)**:
   Strukturbedingt vorhandene Parallelität durch die Vernetzung (einer zeitlich veränderlichen Anzahl) kooperierender OCMs

Auch die Kombination beider Ebenen ist möglich, so dass eine OES-Struktur mit mehreren Evaluatoren entsteht, die dann jeweils mehrere Simulatoren verwenden (vgl. Abbildung 6-5, Variante 2).

6.4.1 Optimierungsebene

Auf der Optimierungsebene wird davon ausgegangen, dass immer nur ein Optimierer während

eines Optimierungslaufs aktiv ist. Die Verkopplung von mehreren gleichzeitig aktiven Optimierern führt zu großen Problemen und ist als zukünftiger Forschungsgegenstand anzusehen. Für diese Anwendungsklasse ist das vorliegende OES-Konzept prinzipiell auch geeignet, löst aber natürlich nicht deren mathematische und algorithmische Probleme. Eine Möglichkeit, ein Optimierungsproblem in Teiloptimierungen zu zerlegen und diese wiederum so aufeinander abzustimmen, dass ihre Lösung auch die Gesamtlösung der ursprünglichen Aufgabe darstellt, ist z. B. durch theoretische Methoden der Optimierung dynamischer Systeme gegeben, die unter dem Begriff "Dynamic Decomposition Principle" in der Literatur zu finden sind ([Wismer 1971], [Lasdon 1970]).

Die verteilte Verarbeitung auf der Optimierungsebene beruht auf der nebenläufigen Auswertung von Zielfunktionen. Deren Auswertung ist erfahrungsgemäß der bei weitem rechenintensivste Teil eines Optimierungslaufs. Das Optimierungsverfahren an sich, d. h. der Algorithmus zur Minimumsuche, soll nicht verteilt werden. Nach dem Fork-Join-Prinzip [Towsley et al. 1990] erzeugt der Optimierungsalgorithmus unabhängige Parametervektoren, startet dann nebenläufige Bewertungsprozesse (Fork) und wartet, bis die Ergebnisse aller Prozesse vorliegen (Join):

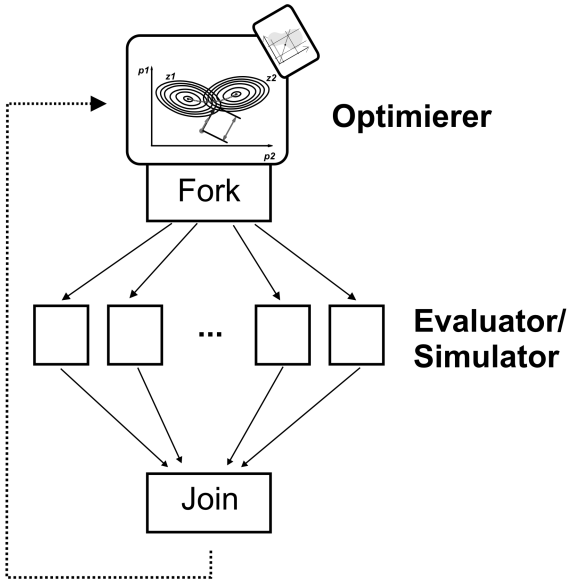

Abbildung 6-6: Paralleler Ablauf für die Mehrziel-Optimierung

6.4.2 VMS1-Ebene

Auf der Ebene der Vernetzten Mechatronischen Systeme (VMS1) führen die kooperierenden OCMs der Einzelsysteme die Online-Optimierung gemeinsam durch. Da hier keine verteilten Optimierungsalgorithmen betrachtet werden, existiert nur eine zentrale Instanz eines Optimierers und des Evaluators.

Für die zentrale Optimierer-Instanz muss ein Einzelsystem ausgewählt werden (Leader-Election). Die Evaluatoren und Simulatoren dagegen sind auf alle OCMs verteilt. Eine besondere Eigenschaft von VMS1 in speziellen Anwendungen kann die zeitlich veränderliche Anzahl der beteiligten Einzelsysteme sein. Für die Optimierung auf VMS1-Ebene ist eine dynamische Verwaltung der Simulatoren vorzusehen (da z. B. zeitlich veränderliche Anzahl von Fahrzeugen im

Kreuzungsbereich).

6.5 Echtzeitverarbeitung

Nachfolgend werden die Maßnahmen zur Sicherstellung der Online-Fähigkeit eines Optimierungslaufs beschrieben. Die Echtzeitfähigkeit der Informationsverarbeitung in mechatronischen Systemen wird hier folgendermaßen definiert:

> *Eine Informationsverarbeitung ist echtzeitfähig, wenn das Maximum und die Schwankungsbreite ihrer Reaktionszeit garantiert unterhalb der für ihre physikalische Umgebung als notwendig spezifizierten Schranken bleiben.*

Als Reaktionszeit (Deadline) wird dabei die Zeit zwischen dem Einlesen von Messgrößen und dem Aufschalten von Stellgrößen oder Parameteränderungen verstanden. Ein Überschreiten der Reaktionszeit ist also unter allen Umständen zu vermeiden. Eine Maximierung der Unterschreitung der Reaktionszeit bringt auf der anderen Seite aber auch keine Vorteile mit sich. Die Reaktionszeit wird insbesondere bei einer digitalen Realisierung immer einer gewissen Schwankungsbreite (Jitter) unterliegen. Je nach Anwendungsfall darf eine bestimmte Schwankungsbreite nicht überschritten werden. Die maximale Schwankungsbreite und die obere Zeitschranke für die Reaktionszeit sind in der Entwurfsphase der Informationsverarbeitung festzulegen. Nachfolgend wird eine Klassifizierung von Reaktionszeiten (Deadlines) vorgestellt, die zur Einteilung der OES-Funktionen in Echtzeitklassen verwendet wird.

6.5.1 Klassen von Echtzeit-Tasks

Bei der Definition geeigneter Systemarchitekturen für Realzeitsysteme entsteht ein Konflikt zwischen garantierbaren, deterministischen Ausführungszeiten und der nutzbaren Rechenleistung. Bei einer ungünstigen Kombination von rechenintensiven Tasks und solchen mit kurzer Deadline kann beispielsweise bei modernen Prozessoren mit Pipelines und Caches nur noch ein Bruchteil der Rechenleistung genutzt werden. Die größte Rolle spielen hierbei die Kriterien "maximale Reaktionszeit" und "rechnerische Komplexität". Bei Tasks mit sehr kurzer Deadline ist daher oft eine Realisierung mit Hilfe von rekonfigurierbarer Hardware sinnvoller als eine Software-Lösung [Bednara et al. 2003]. In [Färber et al. 1997] wird ein Modell zur Einteilung verschiedener Tasks eines Realzeitsystems in Klassen vorgeschlagen:

- **Klasse 0**:
  Tasks mit einer Deadline kürzer als *1 µs* führen Funktionen sehr geringer Komplexität aus.
- **Klasse 1**:
  Interrupthandler oder primäre Antworttasks mit sehr kurzen Antwortzeiten (einige µs) und geringer Codegröße (weniger als *10.000* dynamische Instruktionen)
- **Klasse 2**:
  Die Standard-Realzeittasks sind durch Deadlines ab *1 ms* und Codegrößen im *5* bis *50-kByte*-Bereich charakterisiert.

- **Klasse 3**:
  Sehr rechenintensive Tasks, z. B. Optimierungsaufgaben mit mehreren Millionen Instruktionen, meist mit Antwortzeiten von *20 ms* und mehr.
- **Klasse 4**:
  Spezialisierte Aufgaben, z. B. Bildverarbeitung, die am besten von Spezialprozessoren ausgeführt werden. Üblicherweise eine begrenzte Zahl wohldefinierter Algorithmen mit typischen Antwortzeiten zwischen *20 ms* und *400 ms*.

6.5.2 Echtzeitfähigkeit für die Mehrziel-Optimierung

Zunächst wird der Begriff Online-Optimierung, wie er für diese Arbeit zugrunde gelegt wird definiert. Diese Definition schließt ganz bewusst Aussagen über die Konvergenzgeschwindigkeit von Optimierungsläufen aus, da es extrem schwierig ist, Konvergenzaussagen für komplexe Mehrziel-Probleme zu treffen. Es wird mit der nachfolgenden eigenen Definition nur eine konstante Anzahl von Optimierungsschritten pro Zeiteinheit gefordert:

> *Für eine Online-Optimierung muss sichergestellt sein, dass in einem bestimmten Zeitintervall eine Mindestanzahl von Optimierungsschritten erfolgt. Das zeitliche Verhalten des Optimierungsverfahrens und der Zielgrößenberechnung muss dazu echtzeitfähig sein. Das Finden einer lokalen oder globalen Pareto-optimalen Lösung des Optimierungsproblems muss dabei nicht garantiert sein.*

Für eine Echtzeit-Informationsverarbeitung kann eine grobe empirische Zuordnung von Taskklassen zu OES-Funktionen folgendermaßen lauten: Simulationen sind typischerweise im Bereich Klasse 2, Bewertungen (Evaluatoren) im Bereich Klasse 2/3 und Optimierer im Bereich Klasse 2 angesiedelt (frü MOPO vgl. Kap. 6.5.3). Damit ergibt sich für einen gesamten OES-Ablauf der Bereich Klasse 3/4 (vgl. Abbildung 6-7).

Um diesen Anforderungen gerecht zu werden, ist eine effiziente Implementierung in Form von kompilierbarem Code (z. B. ANSI-C Code) eine wichtige Voraussetzung. Für die Auswahl eines geeigneten Optimierungsverfahrens sollte man sich auf lokale Verfahren (zur Berechnung einer

Lösung) beschränken:

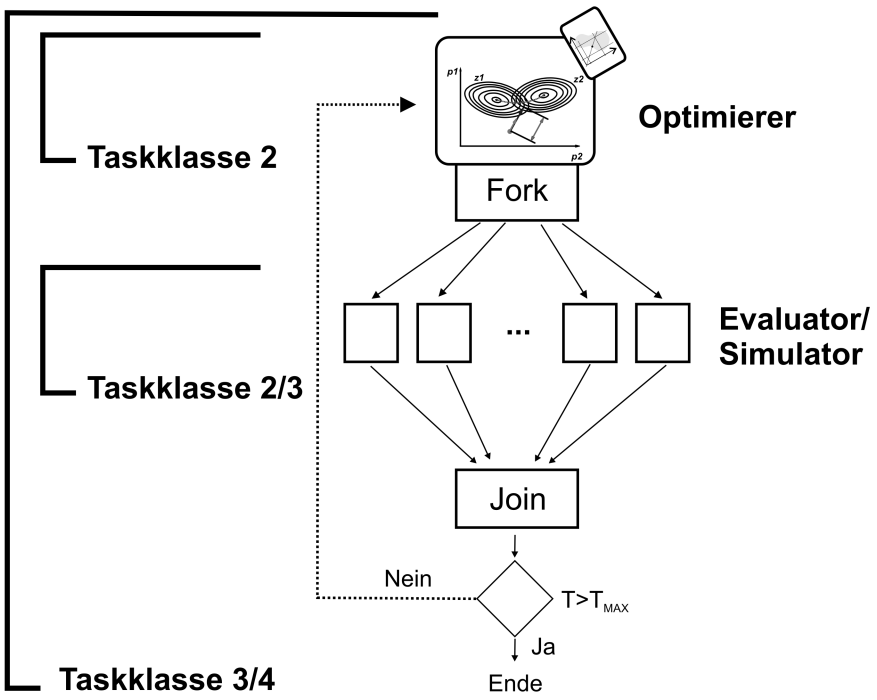

Abbildung 6-7: Taskklassen für die Online-Optimierung

6.5.3 Laufzeitverhalten des MOPO-Gradientenverfahrens

Für die Echtzeitfähigkeit des Optimierungslaufs ist die Dauer des längsten Berechnungspfades (WCET) von entscheidender Bedeutung. Aufgrund der Komplexität der Algorithmen wird vorgeschlagen, das Laufzeitverhalten der Optimierungsalgorithmen über eine Messung auf der jeweiligen Zielhardware zu bestimmen. Hier gibt es zwei Möglichkeiten, um die Ausführungszeit für den längsten Berechnungspfad zu messen:

1. Messung des Laufzeitverhaltens für ein bekanntes Optimierungsproblem und ggf. Hochrechnung auf den längsten Berechnungspfad, falls dieser nicht oder selten durchlaufen wird.
2. Implementierung eines Messlaufes in der Software, der garantiert den längsten Pfad durchläuft, dabei aber keine sinnvolle Lösung eines Optimierungsproblems liefert.

Im Folgenden wird exemplarisch eine Messung auf einer LINUX-Workstation beschrieben. Beim hier vorgestellten Gradientenverfahren ergibt sich der längste Pfad aus zwei dominierenden Anteilen. Dies sind die Bestimmung der Suchrichtung $\underline{d}_k$ (lokales Problem) im Parameterraum und die Ermittlung der optimalen Schrittweite λ_k für das Fortschreiten in die gefundene Richtung (globales Problem). Pro Optimierungsschritt wird entweder die Suchrichtungsbestimmung oder die Bestimmung der Schrittweite notwendig. Es werden daher beide Algorithmen getrennt betrachtet.

Lokales Problem

Die Bestimmung der Suchrichtung -$\underline{d}_k$ wird als quadratisches Optimierungsproblem formuliert [Münch 2001]. Dieses Problem wird in der beschriebenen Implementierung mit Hilfe der *Active-*

Set-Methode [Gill, Murray 1978] gelöst. Bei n_z Zielgrößen müssen im Zuge dieses Verfahrens lineare Gleichungssysteme mit variablen Ordnungen zwischen zwei und n_z gelöst werden. Eine Messung auf einer LINUX-Workstation (siehe Tabelle 7) ergab das Zeitverhalten für das Lösen von linearen Gleichungssystemen. Die Kurven "Minimale Zeit für ein Gleichungssystem" und "Maximale Zeit für ein Gleichungssystem" in Abbildung 6-8 zeigen das Zeitverhalten, aufgetragen über die Ordnung des Gleichungssystems. Man bleibt bis zur Ordnung *10* unterhalb von *0,5* Millisekunden (*0,416 ms*).

TABELLE 7. Randbedingungen für die durchgeführte Messung

Kategorie	**Randbedingungen**
Hardware	PentiumII-400, 384 MB RAM
Betriebssystem	SuSE-Linux 7.0, glibc-2.1.3, SuSE-Kernel 2.2.16 (Pentium optimiert)
Compiler	gcc-2.95.2, keine Optimierungen

Für die Active-Set-Methode sind die Laufzeiten im Diagramm unter "minimale gemessene Gesamtzeit", "mittlere gemessene Gesamtzeit" und "maximale gemessene Gesamtzeit" ablesbar. Für die Ordnung *10* ergibt sich eine maximale Laufzeit von *2,12* Millisekunden. Die starke Streuung begründet sich in der variablen Ordnung der bei der Active-Set-Methode zu lösenden Gleichungssysteme, die problemabhängig variiert. Daher wird für die Abschätzung einer Obergrenze die Annahme getroffen, dass alle zu lösenden Gleichungssysteme immer die maximal mögliche Ordnung n_z besitzen. Dazu wird die maximale Zeit zur Lösung eines Gleichungssystems mit n_z multipliziert. Den entsprechenden Kurvenverlauf findet man als "geschätzte maximale Gesamtzeit", der sich durch ein Polynom 4. Ordnung approximieren läßt. Man erkennt eine pessimistische Abschätzung, die immer über den gemessenen maximalen Werten liegen wird. Bei $n_z=10$ schätzt man *4,16* Millisekunden im Vergleich zu maximal gemessenen *2,12* Millisekunden:

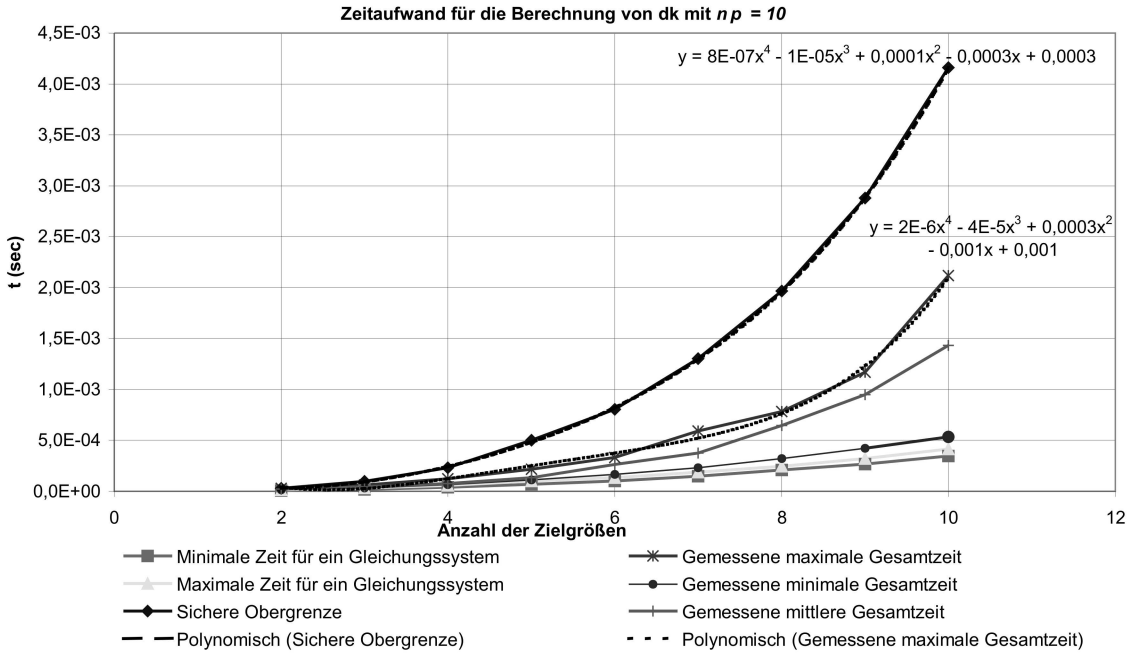

Abbildung 6-8: Zeitaufwand für die Suchrichtungsbestimmung

Globales Problem

Bei bekannter Suchrichtung muß das sog. globale, nichtlineare Problem gelöst werden. Hier soll die optimale Schrittweite für das Fortschreiten in die Richtung $\underline{d}_k$ ermittelt werden, so dass sich eine möglichst große Verbesserung aller Zielgrößen ergibt. Dazu wird eine iterative eindimensionale Minimumsuche (Liniensuche) verwendet. Es wird solange iteriert, bis sich eine Zielgröße verschlechtert. Tritt bereits beim ersten Schritt der Minimumsuche eine Verschlechterung einer Zielgröße auf, so wird die Iteration von dort aus rückwärts weitergeführt. Beim Vorwärtsschreiten wird die Schrittweite in jedem Schritt verdoppelt, beim Rückwärtsschreiten in jedem Schritt halbiert. Mit Hilfe der letzten zulässigen Zielfunktionswerte wird abschließend für jede Zielgröße eine quadratische Interpolation durchgeführt. Als Ergebnis erhält man eine Schrittweite λ_i für jede einzelne Zielgröße, die anhand der Lage des Minimums der interpolierten Zielfunktion ausgewählt wird. Als globale Schrittweite wird der kleinste λ_i -Wert ausgewählt.

Abbildung 6-9 verdeutlicht den Zusammenhang zwischen der Dauer der Interpolationen und der Anzahl der Zielgrößen. Der Zusammenhang ist, wie erwartet, quasi-linear. Die Rechenzeiten sind um zwei Größenordnungen geringer als die Zeiten zur Suchrichtungsbestimmung:

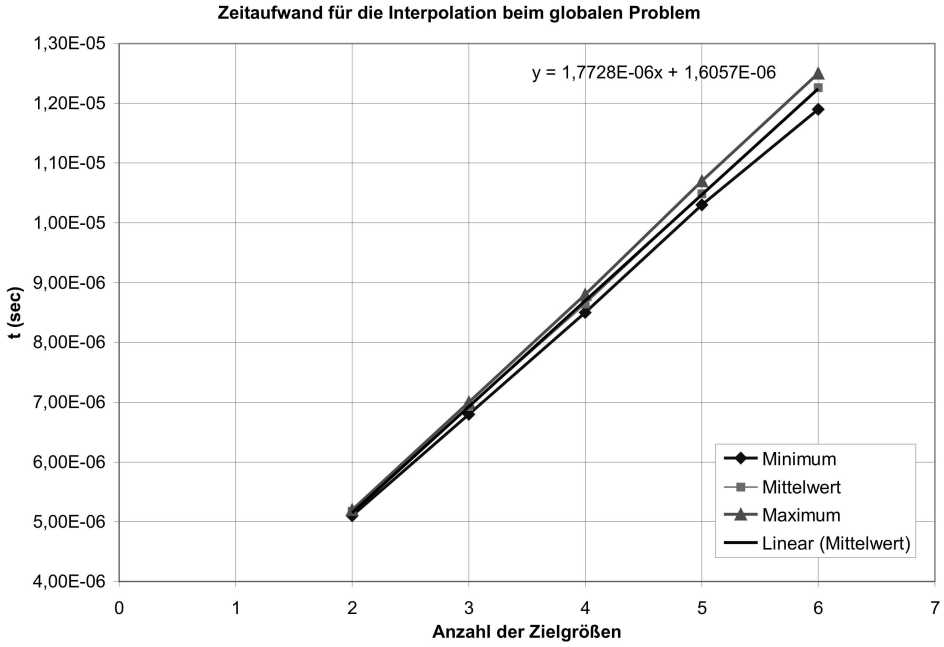

Abbildung 6-9: Zeitaufwand für die Interpolation beim globalen Problem

Ergebnis

Es zeigt sich, dass die Bestimmung der Suchrichtung $\underline{d}_k$ den größten Rechenaufwand verursacht und damit den längsten Berechnungspfad bestimmt (vgl. Tabelle 8). Für die beschriebene Rechnerkonfiguration konnte exemplarisch eine Obergrenze ermittelt werden, welche die maximale Rechenzeit durch ein Polynom 4. Ordnung in Abhängigkeit der Zielgrößenanzahl beschreibt. Für andere Rechnersysteme bzw. Echtzeitsysteme ist eine ähnliche Charakteristik zu erwarten, die jedoch individuell durch Messung der Zeiten ermittelt werden muss.

Die maximale Berechnungsdauer ist im Wesentlichen von der Anzahl der Zielgrößen abhängig.

Die Anzahl n_p der Parameter hat weniger Bedeutung (hier wird eine Parameteranzahl von *10* verwendet). Allerdings gilt es zu berücksichtigen, dass sich durch die implizite Berücksichtigung von Parameterschranken die Zahl der Zielfunktionen dynamisch ändern kann. Während weniger Optimierungsschritte müssen evtl. Parameter durch künstliche Zielfunktionen beschränkt werden, d. h. für eine sichere Abschätzung bzw. Messung muss von einer maximalen Zielgrößenanzahl $n_z + n_p$ ausgegangen werden.

TABELLE 8. Maximal zu erwartende Rechenzeiten für die beschriebene Rechnerkonfiguration

Anzahl Zielgrößen	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
T_{Max} **für** $\underline{d}_k$ **[ms]**	**0,028**	**0,096**	**0,232**	**0,500**	**0,804**	**1,302**	**1,986**	**2,880**	**4,160**
T_{Max} für λ_k [ms][a]	0,005	0,007	0,0088	0,011	0,013	*0,014*	*0,016*	*0,018*	*0,019*

a. Die Tabellenwerte für 7, 8, 9 und 10 Zielgrößen sind linear extrapoliert

6.5.4 Prinzip der Vorausschau

Setzt man die Echtzeitbedingungen für einen Optimierungslauf fest, so ist dies von der spezifischen Anwendungssituation abhängig. Ist man aufgrund ausreichender Rechenleistung dazu in der Lage, die Bewertung (Evaluation) schneller als in Echtzeit auszuführen, dann ist eine modellbasierte zeitliche Vorhersage der unmittelbaren Zukunft möglich.

Bildet man den Quotienten aus der im Modell fortschreitenden Zeit ("Modellzeit") und der tatsächlich verstreichenden physikalischen Rechenzeit ("Realzeit") pro Evaluierung, so ergibt sich der hier eingeführte *Echtzeitfaktor* τ_{RT}. Wenn dieser größer als 1 ist, so sind zeitliche Vorhersagen für die Bewertung möglich:

$$\tau_{RT} = \frac{T_{Modell}}{T_{Realzeit}} \tag{6.1}$$

Wählt man die vorgegebene Maximalanzahl $n_{opt,max}$ von Optimierungsschritten kleiner als den Echtzeitfaktor, so ist eine Vorab-Optimierung von Situationen der unmittelbaren Zukunft möglich. Die Zeit für den gesamten Optimierungslauf ist dann kleiner als das zugrundeliegende Modellzeitintervall (vgl. Tabelle 9). Diese Technik der Vorab-Optimierung wird z. B. für das Kreuzungsmanagement verwendet, um in ausreichender Entfernung vor der Kreuzung bereits deren Überquerung zu optimieren:

$$T_{Optimierung} = n_{opt,\,max} \cdot T_{Realzeit} = n_{opt,\,max} \cdot \frac{T_{Modell}}{\tau_{RT}} \tag{6.2}$$

TABELLE 9. Beispiel-Parameter für eine Vorab-Optimierung

Modellzeit [s] T_{Modell}	**Echzeitfaktor** τ_{RT}	**Vorgegebene Anzahl von Optimierungs-schritten**	**Benötigte Optimierungszeit [s]**
100	20.000	$n_{opt,max}$=100	0,5
100	20.000	$n_{opt,max}$=200	1,0

6.5.5 Verschiedene Taktraten

Bildet man die OES-Funktionen im Rechner ab, so müssen die verschiedenen Taktraten des Optimierungslaufs berücksichtigt werden. Da die Systembewertung sich üblicherweise auf ein vorgegebenes Modellzeitintervall bezieht, werden Simulator und Evaluator zwangsläufig häufiger ausgewertet als der Optimierer. Simulator und Evaluator müssen nicht mit identischer Taktrate ausgewertet werden. Es entsteht ein *Multirate*-System mit bis zu drei Taktraten (Optimierungs-, Evaluator-, Simulator-Taktrate).

Dieses *Multirate*-System lässt sich nicht vergleichen mit Multirate-Simulationen von Systemmodellen, die numerisch sehr schwierig zu handhaben sind, da im Regelfall Rückkopplungen über Taktgrenzen hinweg bestehen [Rükgauer 1996]. Die Rückkopplung zwischen Evaluator und Optimierer und zwischen Evaluator und Simulator ist ablaufgeprägt, d. h. Daten werden zu anderen Teilen erst dann zurückgegeben, wenn der eigene Durchlauf (eine ganze Simulation oder eine ganze Bewertung) vollständig ist. Hier gibt es also keine Notwendigkeit zur Interpolation oder Extrapolation von Daten zwischen OES-Funktionen.

Für eine echtzeitfähige Single-Prozessor-Anwendung sollen die OES-Funktionen auf präemptive Tasks abgebildet werden. Die präemptiven Tasks ermöglichen eine ungestörte Verarbeitung des reinen Reglercodes (Controller) als hochpriorisierte Task, die mit möglichst geringer und konstanter Totzeit ausgeführt werden muss. Besonders im Hinblick auf die hohen Rechenzeitanforderungen durch Bewertungs- und Optimierungsalgorithmen sind diese besonderen Vorkehrungen unbedingt zu treffen.

6.6 Parallelisierung des MOPO-Verfahrens

Die Idee der Parallelverarbeitung in MOPO (Multi-Objective Parameter Optimization) beruht auf der Nutzung von unterschiedlich parametrierten, parallel rechnenden Kopien der Systembewertung. Dieses Konzept [Deppe et al. 2003/1] ist sehr erfolgversprechend, da die Erfahrung zeigt, dass in praktischen Anwendungen der Mechatronik die Rechenlast der eigentlichen Optimierungsalgorithmen gegenüber den Zielfunktionsauswertungen vernachlässigbar klein ist. Entscheidend für eine effiziente Parallelverarbeitung ist daher die Fähigkeit eines Optimierungsverfahrens, möglichst viele Systembewertungs-Instanzen gleichzeitig nutzen zu können. Nachfolgend werden die dazu implementierten Möglichkeiten für das MOPO-Gradienten- und -Quasi-Newton-Verfahren vorgestellt.

6.6.1 Performance Kriterien

Nach [Lootsma 1985] sind die wichtigsten "Performance"-Kriterien für die Mehrziel-Optimierung auf Parallelrechnern **Robustheit** (d. h. Eignung der Optimierungsmethode das spezifische Problem hinreichend genau zu lösen), **Kapazität** (d. h. Maximale Problemgröße, die gelöst werden kann) und **Effizienz**. Dabei ist für das letztere Kriterium völlig offen, wie es bestimmt werden kann. Folgende Vergleiche zur Bestimmung von Effizienz wären z. B. denkbar:

- Vergleich eines parallelen Algorithmus und des entsprechenden sequentiellen auf einem Parallelrechner
- Vergleich verschiedener paralleler Algorithmen auf einer identischen parallelen Hardware
- Vergleich verschiedener Parallelrechner-Architekturen für einen identischen parallelen Algorithmus

Hier wird für die Mehrziel-Optimierung das MOPO-Verfahren vorgeschlagen, da es im Laufe der Jahre speziell an die Problemstellungen der Mechatronik angepasst wurde und daher Robustheit bieten kann. Die Kapazität des Gradienten- und Quasi-Newton-Verfahrens ist für kleine (bis 30 Zielgrößen/Parameter) und mittlere (bis 100 Zielgrößen/Parameter) Probleme geeignet [Lootsma 1985].

6.6.2 MOPO-Gradientenverfahren

Das Gradientenverfahren besteht aus dem lokalen und dem globalen Schritt (vgl. Kap. 4.8). Beide Schritte bieten unterschiedlich gute Möglichkeiten zur parallelen Implementierung.

Lokales Problem

Für den lokalen Schritt werden die Zielgrößen zur Ermittlung der Empfindlichkeitsmatrix $\underline{S}$ benötigt (vgl. Kap. 4.8.1). Diese hat die Dimension $n_p \; x \; n_z$ (mit n_p Parameteranzahl und n_z als Zielgrößenanzahl). Zur Berechnung der Empfindlichkeitsmatrix werden ausgehend vom aktuellen Parametervektor $\underline{p}$ alle skalaren Elemente p_i einzeln um Δp ausgelenkt. Die resultierende Parametermatrix $\underline{P}$ hat folgende Form:

$$\underline{P} = \Delta p \cdot \underline{I} + \begin{bmatrix} \underline{p} \\ \dots \\ \underline{p} \end{bmatrix} = \begin{bmatrix} (\Delta p + p_1) & \dots & p_{n_z} \\ \dots & \dots & \dots \\ p_1 & \dots & (\Delta p + p_{n_z}) \end{bmatrix} \tag{6.3}$$

Verwendet man eine Differentiation 1. Ordnung zur Gradientenberechnung, so müssen pro Zeile der Empfindlichkeitsmatrix $\underline{S}$ alle Zielgrößendifferenzen zwischen Parametervektor $\underline{p}$ und zugehörigem Zeilenvektor der Parametermatrix $\underline{P}$ bestimmt werden:

$$\underline{S_1} = \begin{bmatrix} \frac{z_1\left(\left[(\Delta p + p_1)\ \ldots\ p_{n_z}\right]\right) - z_1(\underline{p})}{\Delta p} & \ldots & \frac{z_{n_z}\left(\left[(\Delta p + p_1)\ \ldots\ p_{n_z}\right]\right) - z_{n_z}(\underline{p})}{\Delta p} \\ \ldots & \ldots & \ldots \\ \frac{z_1\left(\left[p_1\ \ldots\ (\Delta p + p_{n_z})\right]\right) - z_1(\underline{p})}{\Delta p} & \ldots & \frac{z_{n_z}\left(\left[p_1\ \ldots\ (\Delta p + p_{n_z})\right]\right) - z_{n_z}(\underline{p})}{\Delta p} \end{bmatrix} \tag{6.4}$$

Die Berechnung der Gradienten in der Empfindlichkeitsmatrix erfolgt im Optimierer, dazu muss der Evaluator alle Zielfunktionen der Matrix $\underline{Z}$ berechnen und an den Optimierer zurücksenden:

$$\underline{Z_1} = \begin{bmatrix} z_1\left(\left[(\Delta p + p_1)\ \ldots\ p_{n_z}\right]\right) & \ldots & z_{n_z}\left(\left[(\Delta p + p_1)\ \ldots\ p_{n_z}\right]\right) \\ \ldots & \ldots & \ldots \\ z_1\left(\left[p_1\ \ldots\ (\Delta p + p_{n_z})\right]\right) & \ldots & z_{n_z}\left(\left[p_1\ \ldots\ (\Delta p + p_{n_z})\right]\right) \end{bmatrix} \tag{6.5}$$

Differentiation 2. Ordnung

Verwendet man eine Differentiation 2. Ordnung zur Gradientenberechnung, so werden für die partiellen Ableitungen der Zielfunktionen ein vorangegangener Parameter-Stützpunkt p_k und zwei neue Parameter-Stützpunkte (p_k+Δp und p_k-Δp) benötigt. Damit verdoppelt sich die Zeilenanzahl von $\underline{Z}$, d. h. der Evaluator muss doppelt so viele Zielgrößen berechnen:

$$\underline{Z_2} = \begin{bmatrix} z_1(p_1 + \Delta p) & z_2(p_1 + \Delta p) & \ldots & z_{n_z}(p_1 + \Delta p) \\ z_1(p_1 - \Delta p) & z_2(p_1 - \Delta p) & \ldots & z_{n_z}(p_1 - \Delta p) \\ \ldots & \ldots & \ldots & \ldots \\ z_1(p_{n_p} + \Delta p) & z_2(p_{n_p} + \Delta p) & \ldots & z_{n_z}(p_{n_p} + \Delta p) \\ z_1(p_{n_p} - \Delta p) & z_2(p_{n_p} - \Delta p) & \ldots & z_{n_z}(p_{n_p} - \Delta p) \end{bmatrix} \tag{6.6}$$

Die MOPO-Algorithmen sind so realisiert, dass sie unabhängige Parametervektoren erzeugen, die zu einer parallelen Zielfunktionsauswertung genutzt werden. Die nachfolgende Abbildung zeigt ein gedachtes Szenario mit 4 Zielfunktionen und 4 Parametern und Differentation 1. Ordnung. Die 16 skalaren Elemente der Matrix $\underline{Z}$ werden 8 Evaluatoren zugeordnet, die dann in einem Prozessornetzwerk verteilt werden können. Die Zuordnung zu Evaluatoren richtet sich nach praktischen Gesichtspunkten der jeweiligen Anwendung: Etwa falls Zielfunktionen von verschiedenen Arten von (Evaluator-)Software-Werkzeugen berechnet werden müssen, da sie spezifisch Zeit- oder Frequenzbereich oder der linearen- oder der nichtlineare-Analyse zugeordnet sind. Speziell wenn ein Evaluator die Ergebnisse eines Simulators für die Berechnung mehrerer Zielfunktionen einer Zeile von $\underline{Z}$ verwenden kann ist es sinnvoll mehrerer Zielfunktionen in

einer Evaluator-Instanz zu berechnen:

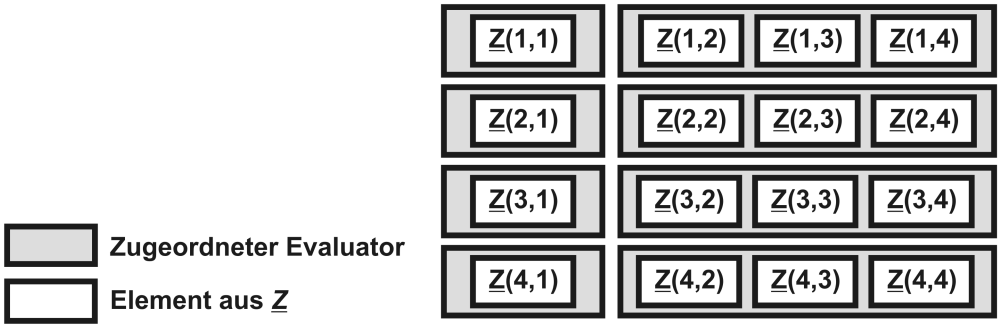

Abbildung 6-10: Beispiel für die parallele Auswertung einer Matrix $\underline{Z}$

Sind alle Elemente von $\underline{Z}$ sinnvoll unabhängig auswertbar, so ergibt sich die theoretisch maximal verwendbare Prozessoranzahl N_P für eine Parallelverarbeitung des lokalen Problems zu (n_D ist die Ordnung des verwendeten numerischen Differentationsverfahrens):

$$N_P = n_D \cdot n_p \cdot n_z \tag{6.7}$$

Globales Problem

Bei bekannter Suchrichtung muss noch das globale Problem gelöst werden. Hier soll die optimale Schrittweite λ_k für das Fortschreiten in die im lokalen Schritt gefundene Richtung $\underline{d}_k$ ermittelt werden, so dass sich eine möglichst große Verbesserung der Zielgrößen ergibt. Dazu wird eine iterative eindimensionale Minimumsuche pro Zielfunktion verwendet: $\underline{p}_{k+1} = \underline{p}_k + \lambda_k \cdot \underline{d}_k$

Die Strategie für die Parallelisierung dieses iterativen Suchvorgangs ist die spekulative Vorabauswertung mehrerer Stützstellen. Dazu wird in einem Raster mit Schrittweitenverdopplung ein Parametervektor bestimmt, für den die Zielgrößen parallel ermittelt werden können. Abbildung 6-11 zeigt hierfür ein Beispiel: In zwei parallelen Schritten wird die Zielfunktion für je drei Stützstellen gleichzeitig ausgewertet. Das Minimum der Zielfunktion wird zwischen λ_4 und λ_5 lokalisiert, die spekulative Berechnung von λ_6 war eigentlich unnötig.

Die Umsetzung ist aufgrund einer Vielzahl von zu beachtenden Sonderfällen mit erheblichem

Programmieraufwand verbunden:

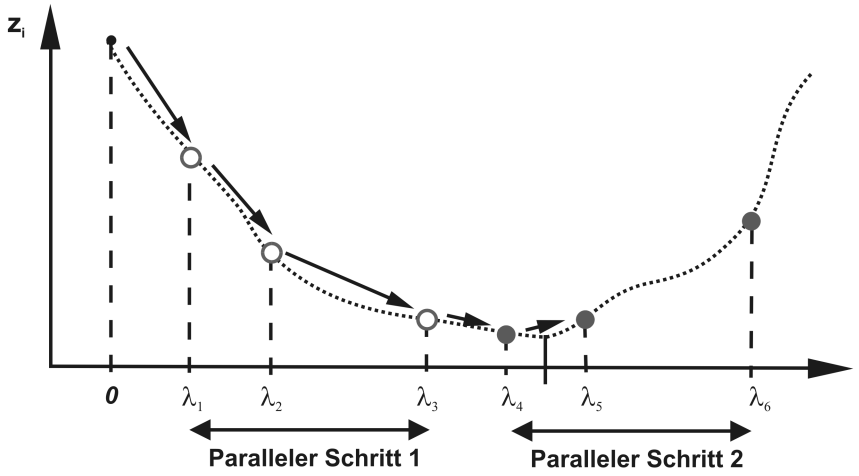

Abbildung 6-11: Parallele Auswertung bei der eindimensionalen Minimumsuche

Zusammenfassung

Durch diese Strategie zur Parallelisierung des globalen Schritts eignet sich das Gradientenverfahren sehr gut für die Parallelverarbeitung. Nachfolgend ist der Programmablauf für das Gradientenverfahren inkl. paralleler Zielgrößenauswertung dargestellt:

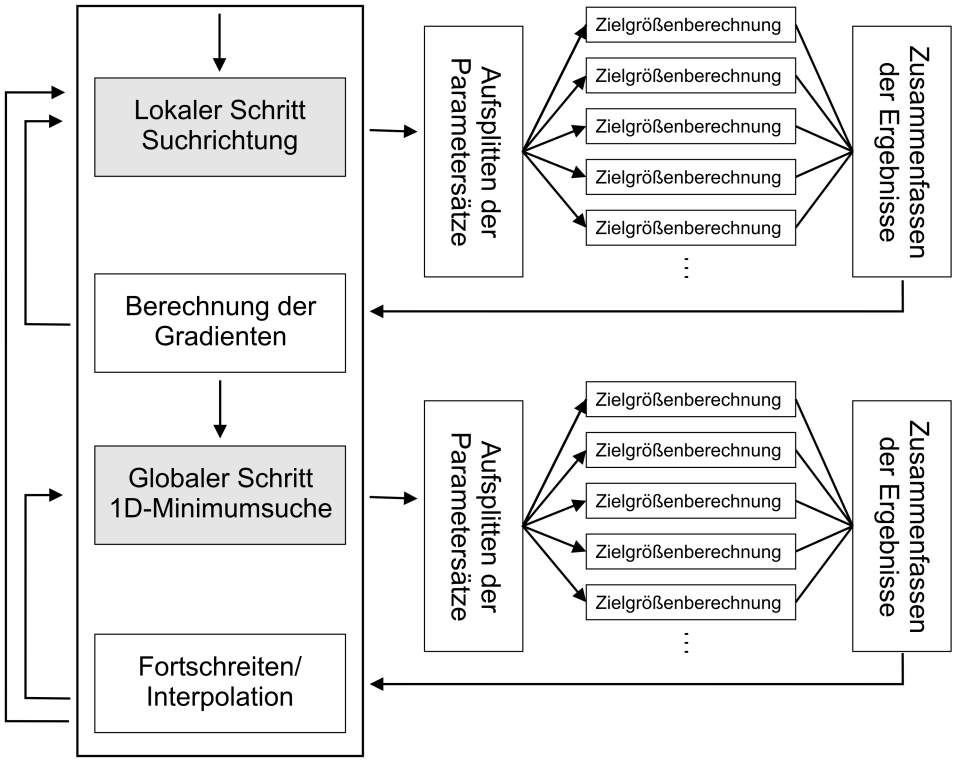

Abbildung 6-12: Flussdiagramm für das Gradientenverfahren

6.6.3 MOPO-Quasi-Newton-Verfahren

Beim Quasi-Newton Verfahren wird die Auswertung der Zielfunktionen zur Gradientenberech-

nung benötigt. Die weitere Auswertung von Zielfunktionen stützt sich anschließend rein auf die gefundenen analytischen Ersatz-Zielfunktionen 2. Ordnung. Das Quasi-Newton-Verfahren hat also für die Gradientenberechnung das gleiche Parallelisierungspotential in der Zielfunktionsauswertung wie der lokale Schritt des Gradientenverfahrens. Dagegen steht ein insgesamt höherer Rechenaufwand für die sequentiell implementierten internen Berechnungen durch Schätzung der Hesse-Matrix und Lösung des Ersatzproblems. Diese sequentiellen Berechnungen haben allerdings nur eine geringe Auswirkung auf die erzielbare Leistungssteigerung durch die parallele Zielgrößenberechnung:

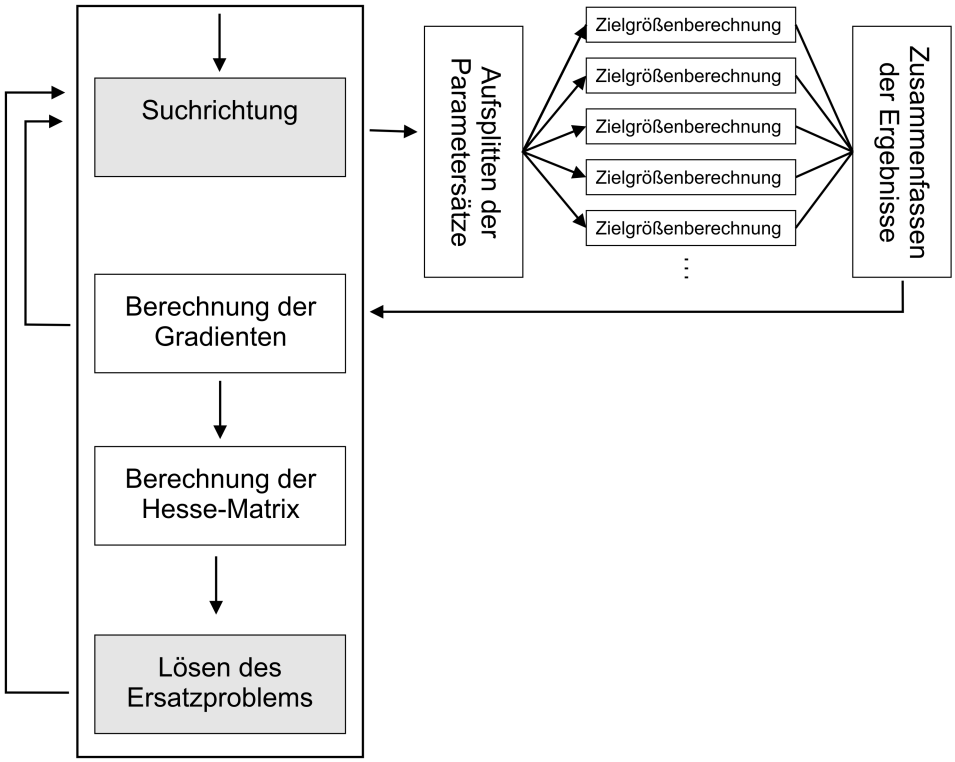

Abbildung 6-13: Flussdiagramm für das Quasi-Newton-Verfahren

7 Anwendungsbeispiele

Nachfolgend werden drei Anwendungsbeispiele mit jeweils unterschiedlichen Schwerpunkten in Bezug auf den Anwendungsfall für die Online-Mehrziel-Optimierung vorgestellt:

Die Regleroptimierung auf AMS-Ebene für ein aktiv gefedertes Fahrzeug (als Einspurmodell mit Knickfreiheitsgrad modelliert) entspricht einem "Standard-Anwendungsfall" zur Reglersynthese während des Systementwurfs, daher wird er hier aufgenommen. Eine verteilte Verarbeitung ist dabei zur Zeitersparnis bei der Behandlung sehr komplexer Systeme von großer Bedeutung. Aspekte wie Echtzeit und Multitasking sind nicht relevant.

Für das Feder-/Neigemodul der NBP-Shuttles (Railcabs) steht ein HILS-Prüfstand zur Verfügung, der sehr gut für die Überprüfung der Ergebnisse im Bereich der Online-Fähigkeit der Mehrziel-Optimierung geeignet ist. Hier wird auf einem multitaskingfähigen Echtzeitsystem eine Regler-Nachoptimierung auf MFM-Ebene unter Echtzeitbedingungen implementiert.

Die komplexeste Anwendung ist das selbstorganisierende Kreuzungsmanagement. Prinzipbedingt unterliegt es harten Echtzeitbedingungen. Zum jetzigen Zeitpunkt ist allerdings nur eine Offline-Variante implementiert, wobei konzeptionell eine spätere Echtzeitverarbeitung stets mitbedacht wird. Das Kreuzungsmanagement zeichnet sich durch eine hochgradig verteilte Verarbeitung mit gleichzeitiger dynamischer Anzahl von beteiligten Fahrzeugen aus. Das Anwendungsbeispiel Kreuzungsmanagement zeigt die grundlegenden Verfahren zur Kollisionsvermeidung und eine exemplarische, modellbasierte Mehrziel-Optimierung auf VMS-Ebene auf.

7.1 Kriterien zur Auswahl der Anwendungsbeispiele

Insgesamt wird nach den folgenden vier Kriterien unterschieden, um die Beispiele zu charakterisieren:

1. **Hierarchieebene**:
   Auf welcher Hierarchieebene ist das Optimierungsexperiment angesiedelt?
2. **Echtzeit**:
   Unterliegt die Anwendung inkl. der Mehrziel-Optimierung harten Echtzeitbedingungen?
3. **Multitasking**:
   Muss ein präemptives Multitasking verwendet werden, um mehrere Taktraten abzubilden?
4. **Verteilt**:
   Ist eine verteilte Verarbeitung sinnvoll bzw. inhärent durch die Anwendung vorgezeichnet?

Tabelle 10 gibt den entsprechenden Überblick über die Eigenschaften der Anwendungsbeispiele:

TABELLE 10. Eigenschaften der Anwendungsbeispiele

Optimierungs-Anwendung	Hierarchie-ebene	Echtzeit	Multi-tasking	Verteilt	Gewählte Rechnerplattform
Aktiv gefedertes Einspurmodell	**AMS**			**X**	Workstation-Cluster (Linux/Windows)
Bahntechnik Feder-/Neige-Modul	**MFM**	**X**	**X**	**(X)**[a]	Power PC 750 (dSPACE RTK)
Selbstorganisierendes Kreuzungsmanage-ment	**VMS**	**(X)**[b]	**(X)**	**X**	Workstation-Cluster (Linux/Windows)

a. War aufgrund der Leistungsfähigkeit der gewählten Rechnerplattform nicht notwendig

b. Fernziel für eine spätere reale Umsetzung

7.2 Regleroptimierung der aktiven Federung an einem Einspurmodell

Für die Anwendung der Mehrziel-Optimierung in den frühen Phasen der Entwicklung eines mechatronischen Systems beginnt man sinnvollerweise mit linearen Modellen und greift auf PC-basierte Hard- und Software zurück. Der Aspekt der Online-Fähigkeit steht damit zunächst nicht im Vordergrund. Allerdings kann die Einbeziehung der Parallelverarbeitung für die Optimierung sehr komplexer Systemmodelle von Anfang an nützlich oder sogar notwendig sein. Am Beispiel einer Regleroptimierung für das aktive Fahrwerk eines Einspurmodells auf AMS-Ebene (vgl. Abbildung 7-2) wird das typische Vorgehen für eine Mehrziel-Optimierung in frühen Entwicklungsphasen erläutert. Insbesondere das Instrumentarium zur Systembewertung mit Hilfe der Theorie linearer Systeme (Varianzen, Pollagen etc.) spielt dabei eine große Rolle.

7.2.1 Fahrzeugmodell

Zugrundegelegt wird ein aktiv gefedertes Einspurmodell mit hydraulischen ABC-Federbeinen (Active Body Control [Hestermeyer et al. 2001]) auf MFM-Ebene. Der Fahrzeugaufbau des physikalischen Ersatzmodells besteht aus zwei Starrkörpern, die sich gegeneinander verdrehen lassen (Knickfreiheitsgrad mit Drehsteifigkeit). Mit diesem Freiheitsgrad lässt sich in erster Nähe-

rung die Karosseriedurchbiegung eines Fahrzeugs modellieren:

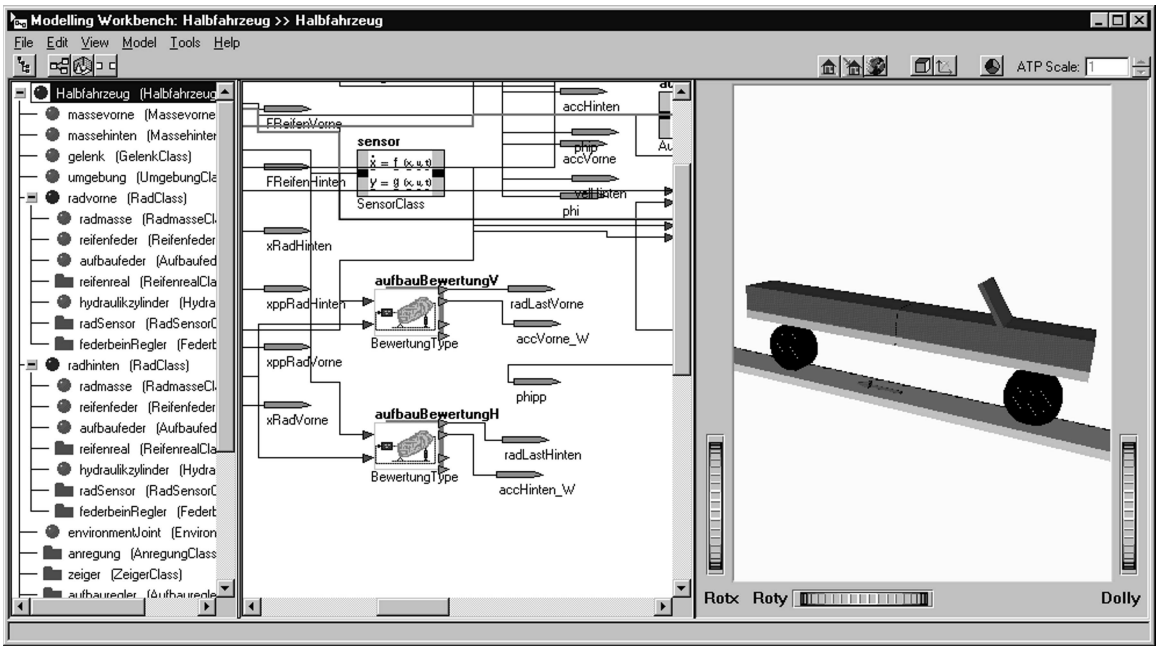

Abbildung 7-1: CAMeL-View-Modell des aktiv gefederten Einspurmodells

Die Räder sind auch Starrkörper modelliert. Die Radmassen sind über die Reifensteifigkeit (lineares Feder-/Dämpfergesetz) mit der Straße verbunden. Die Hydraulik der ABC-Federbeine wird vereinfacht als idealer Kraftsteller modelliert. Es ergeben sich somit fünf mechanische Freiheitsgrade für die Vertikaldynamik des Modells. Insgesamt ergibt sich ein System 10. Ordnung. Dieses Fahrzeugmodell wird mit Hilfe von CAMeL-View (vgl. Abbildung 7-1) auf der Basis von Minimalkoordinaten [Hahn 1999] modelliert:

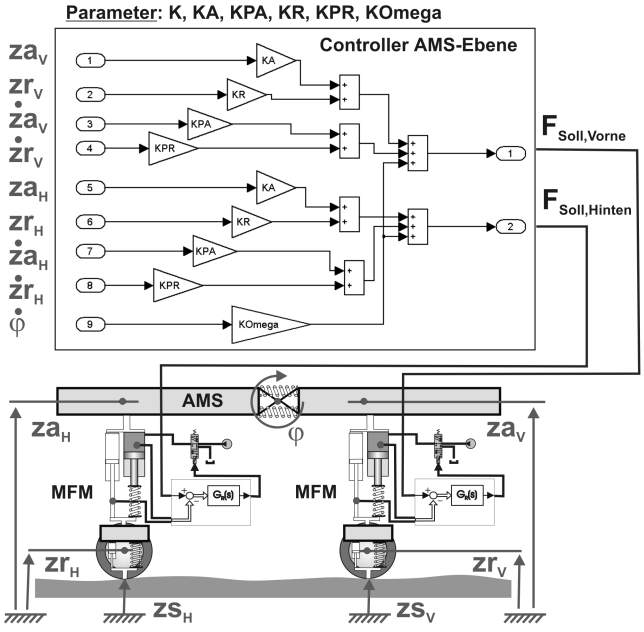

Abbildung 7-2: Controller auf AMS-Ebene und physikalisches Ersatzmodell

Die Vertikaldynamik des Modells ist überwiegend linear modelliert allerdings ergibt sich durch die Kinematik des Knickens eine Nichtlinearität, die jedoch für kleine Auslenkungen als quasi

linear angesehen werden kann. Der zu optimierende Regler ist ebenfalls linear.

Das Fahrzeugmodell ist so modelliert, dass auch die Bewegung in Quer- und Längsrichtung möglich ist (Antrieb und Lenken). Dadurch enthält die Verhaltensebene des Modells (Zustandsraumdarstellung in ODSL, [Hahn 1999]) eine Vielzahl von Nichtlinearitäten zur Koordinatentransformation vom Fahrzeug-Koordinatensystem in das inertiale Koordinatensystem (Quer- und Längsrichtung).

Die Aufgabenstellung für die Regelung auf AMS-Ebene besteht darin, die optimalen Sollwerte (Vertikalkräfte) für die ABC-Federbeine auf MFM-Ebene vorzugeben. Als Messgrößen stehen die Radlagen (zr_H, zr_V), die Aufbaulagen (za_H, za_V), der Knickwinkel φ und alle zugehörigen Geschwindigkeiten zur Verfügung (vgl. Abbildung 7-2).

7.2.2 Softwarearchitektur

Bevor das eigentliche Optimierungsexperiment im nächsten Abschnitt beschrieben wird, soll auf die verwendete Software und deren Architektur eingegangen werden. Ein wichtiger Bestandteil ist das Werkzeug *Scilab*, das sehr gut für die Implementierung eines Evaluators zur Offline-Systembewertung geeignet ist.

Scilab

Scilab, entwickelt seit 1989 durch INRIA (Institut National de Recherche en Informatique et Automatique), ist eine offene Software-Umgebung für numerische Berechnungen im Bereich wissenschaftlicher Anwendungen. Die Software ist seit 1994 im Internet frei verfügbar (*Scilab Homepage: http://www-rocq.inria.fr/scilab*) und wird einschließlich Quellcode ausgeliefert. Seit 2003 existiert auch ein Scilab-Konsortium (*Homepage: http://www.scilab.org*).

Scilab enthält eine Vielzahl von hochwertigen mathematischen Funktionen aus folgenden Bereichen:

- Lösungen linearer Gleichungssysteme (auch dünn besetzter)
- Berechnung von Eigenwerten und Eigenvektoren
- Singulärwertzerlegung und Pseudo-Inverse
- schnelle Fourier-Transformation
- mehrere Methoden zur Lösung von (auch steifen) Differentialgleichungen
- mehrere Optimierungsverfahren
- Lösung nichtlinearer Gleichungssysteme
- mehrere Methoden der linearen Algebra für optimale Steuerungen

Scilab bietet auch die Möglichkeit, eigene Erweiterungen in Form von C- oder FORTRAN 77-Funktionen hinzuzufügen. Das Werkzeug verfügt über einen Interpreter und eine eigene Programmiersprache, die Matrizen als integralen Bestandteil hat. Aus eigenen C- oder FORTRAN 77-Hauptprogrammen heraus kann man Scilab auch als Bibliothek einbinden. Dazu wird die Software als "Engine" ohne eigene Oberfläche zum eigenen Programm hinzugelinkt.

Ab Version 2.7 verwendet Scilab die FORTRAN 77 LAPACK-Programmbibliotheken (*http://*

www.netlib.org/lapack/). Sie enthalten Unterprogramme zur numerischen Lösung der häufigsten Aufgaben aus den Gebieten lineare Algebra, lineare Gleichungssysteme, Methode der kleinsten Quadrate und Eigenwerte von Matrizen. LAPACK stützt sich wiederum auf die BLAS Level 3 (Basic Linear Algebra Subprograms, *http://www.netlib.org/blas/*) Programmbibliotheken, die jeweils spezifisch pro Rechnerplattform implementiert und optimiert sind.

Scilab läuft auf den meisten Unix-Systemen einschließlich Linux und auf Windows 9X/NT/ 2000/XP und vereint insgesamt folgende Vorteile in sich:

- Moderne, leistungsfähige Numerik, die schnelle und genaue Ergebnisse liefert
- Offenheit
- Freie Verfügbarkeit
- Plattformunabhängigkeit
- Interpretierte oder kompilierte Verarbeitung nach Bedarf
- Einfach Integrierbarkeit in eigene Anwendungen

Optimierer-Evaluator-Simulator-Struktur

Zur Umsetzung der Anwendung wird die einfachste OES-Struktur in Form eines Optimierers, eines Evaluators und eines Simulators verwendet. Der Optimierer wird durch MOPO bereitgestellt und über eine TCP/IP-Schnittstelle mit dem Evaluator verbunden. Dieser besteht aus einer Scilab-Engine (extern gesteuerte Scilab-Anwendung ohne Anwenderoberfläche). Der Evaluator ist wiederum über eine TCP/IP-Schnittstelle (Sci-Server genannt) mit dem Simulator verbunden. Der Simulator wird durch die Simulationsplattform IPANEMA implementiert.

Die Offline-IPANEMA-Anwendung mit je einem Moderator, Assistant und Calculator kann automatisch aus CAMeL-View generiert werden. Der Calculator enthält das Fahrzeugmodell in Form von kompiliertem C-Code. Für den Evaluator stellt IPANEMA die Dienste Simulation und

numerische Linearisierung zur Verfügung:

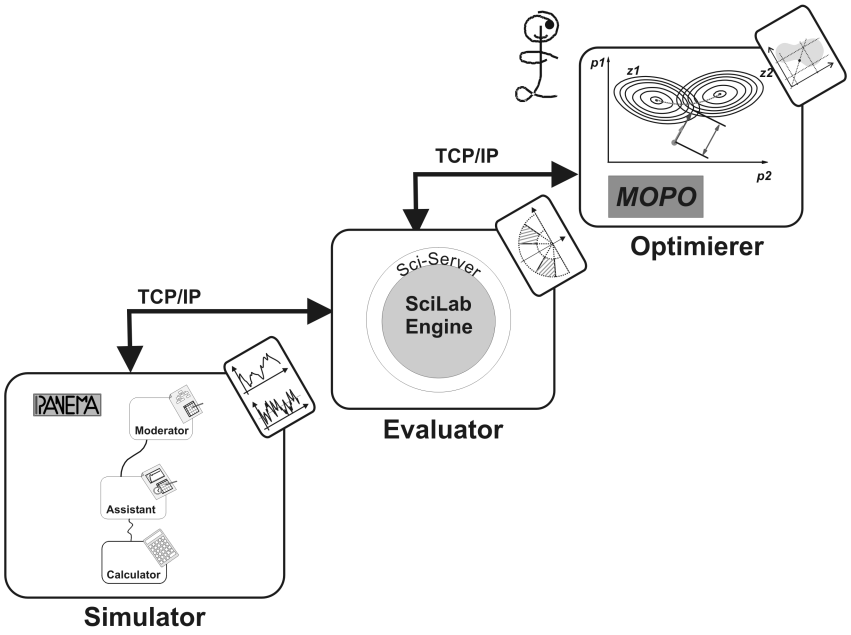

Abbildung 7-3: OES-Struktur für die Optimierung des Einspurmodells

7.2.3 Optimierungsexperiment

Die Aufgabenstellung für die Optimierung besteht darin, die Reglerparameter der AMS-Ebene zur Berechnung der Sollvertikalkräfte für die ABC-Federbeine auf MFM-Ebene anzupassen. Ausgangspunkt ist das passive System, d. h. der Kraftsteller ist starr und die Reglerparameter (vgl. Tabelle 12) sind Null. Zielsetzung für die Optimierung ist es, die Aufbaubeschleunigung weiter zu vermindern, die dynamische Radlastschwankung weiter zu minimieren und gleichzeitig das Knicken erheblich besser zu dämpfen. Dabei ist das passive System bereits gut abgestimmt, hat allerdings eine geringe Knickdämpfung. Diese Forderungen werden mit Hilfe von sechs Zielgrößen (vgl. Tabelle 11) mathematisch formuliert. Als Modelleingänge dienen stochastische Anregungen aus dem Straßenprofil (zs_H, zs_V).

Die Zielgrößen werden auf der Basis des numerisch linearisierten Systemmodells ermittelt. Wird ein Reglerparameter durch den Optimierer modifiziert, wird zunächst das nichtlineare System ansimuliert (vier Sekunden Modellzeit), damit sich die neue statische Ruhelage einstellen kann. In diesem Betriebspunkt wird dann die numerische Linearisierung durch IPANEMA vorgenommen. Die ermittelten ABCD-Matrizen werden an Scilab gesendet. Dort finden alle weiteren Berechnungen zur Systembewertung statt (vgl. Kap. 10). Für dieses Experiment werden Varianzen/Streuungen und Pollagen zur Bewertung herangezogen:

TABELLE 11. Erläuterung der Zielgrößen des Optimierungsexperiments

Name	Beschreibung
rms_zapp	Summe der Streuungen der Vertikalbeschleunigung der beiden Aufbaumassen[a]
rms_rlast	Summe der Streuungen der dynamischen Radlastschwankungen beider Räder
rms_phipp	Streuung der Knickwinkelbeschleunigung

Name	Beschreibung
dist_r	Lage der Radeigenwerte im Kreisringsektor (10,55 Hz bis 15,82 Hz und 47 ° bis 65 °)
dist_a	Lage der Eigenwerte für die Aufbautranslation im Kreisringsektor (1,08 Hz bis 1,63 Hz und 52 ° bis 72 °)
dist_rot	Lage der Eigenwerte für die Biege-(Knick-)Schwingung im Kreisringsektor (3,41 Hz bis 5,12 Hz und 5 ° bis 68 °)

a. Das Modell ist so parametriert, dass symmetrische Verhältnisse bzgl. Vorder- und Hinterwagen herrschen

Über die nachfolgend beschriebenen Parameter lassen sich die oben genannten Zielgrößen beeinflussen:

TABELLE 12. Erläuterung der Parameter des Optimierungsexperiments

Name	Beschreibung
KA	Parameter für die Aufbau-Federkonstante
KPA	Parameter für die Aufbau-Dämpferkonstante
KR	Parameter für die Rad-Federkonstante
KPR	Parameter für die Rad-Dämpferkonstante
KOMEGA	Rückführverstärkung für die Winkelgeschwindigkeit im Knickgelenk

Abbildung 7-4 zeigt einen Teil der MOPO-Anwenderoberfläche. Hier ist die Konfiguration der Parameter und Zielgrößen ersichtlich. Diese Konfiguration kann interaktiv während des Optimierungslaufs modifiziert werden, um das Experiment zu steuern.

Zur Laufzeit lassen sich zusätzliche Plots der Parameter- und Zielgrößenverläufe anzeigen. Dargestellt ist nur der sog. "Radar-Plot" (vgl. Kap. 4.7.3), der die Erfüllungsgrade aller Zielgrößen in einem Kreisdiagramm darstellt. Die Zielgrößen sind dort aus Platzgründen über Indizes referenziert, die der Reihenfolge in der Tabelle der Zielgrößen-Konfigurations-Tabelle entsprechen. Ein Erfüllungsgrad von *100* entspricht dem oberen Limit, ein Erfüllungsgrad von *0* dem unteren

Limit:

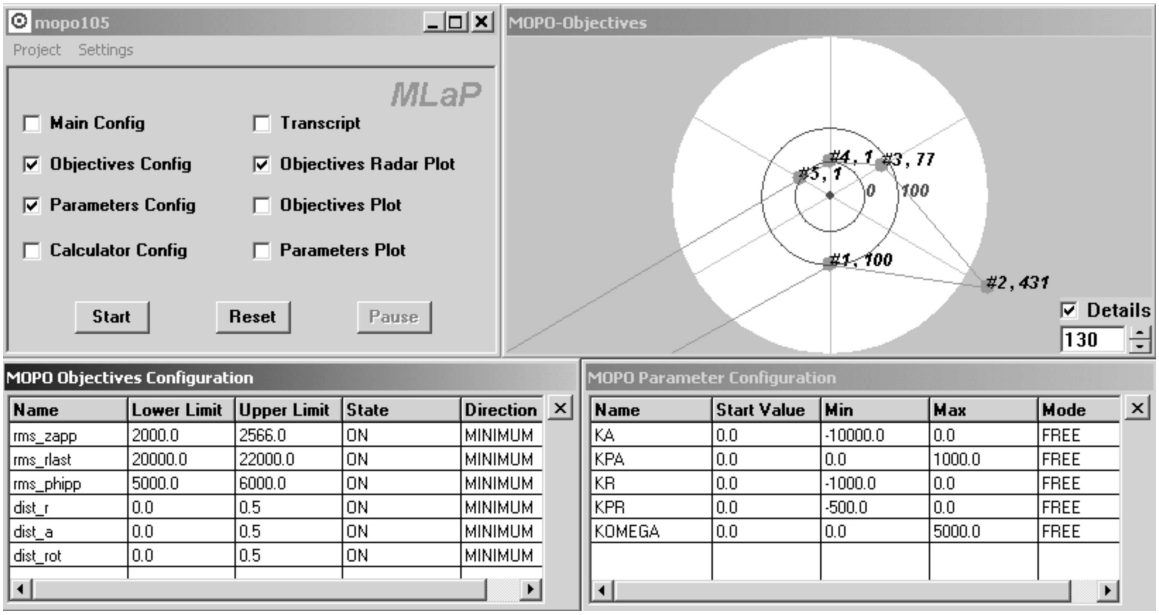

Abbildung 7-4: MOPO-Anwenderoberfläche: "Radar-Plot", Parameter- und Zielgrößen-Konfiguration

Unter dem Punkt "Main Config" des Hauptfensters lässt sich die Konfigurationsmaske für die Auswahl des Optimierungsverfahrens und dessen Grundeinstellungen öffnen. Für diese Anwendung wird das Quasi-Newton-Verfahren verwendet. In Tabelle 13 werden die Bedeutungen der einzelnen Konfigurationsparameter näher erläutert:

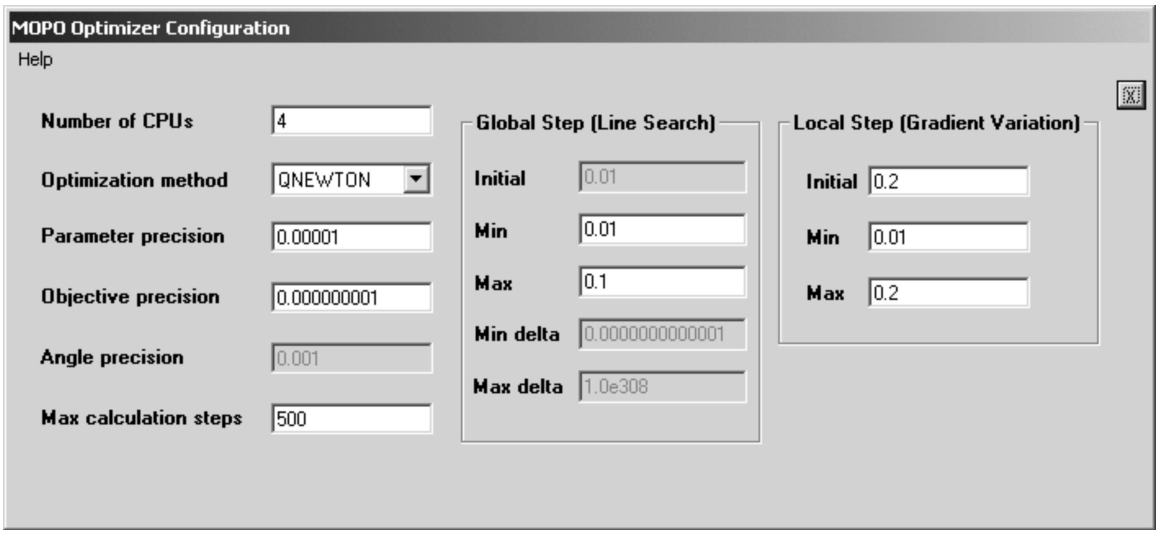

Abbildung 7-5: Eingabe der Optimierer-Konfiguration in der MOPO-Anwenderoberfläche

TABELLE 13. Bedeutung der MOPO-Konfigurationsparameter für das Quasi-Newton Verfahren

MOPO-Parameter	Bedeutung
Number of CPUs	Begrenzt die Anzahl der Parametervektoren, die pro Optimierungsschritt gleichzeitig bewertet werden sollen.
Parameter Precision	Liegt die Änderung der skalierten Parameter-Werte unter diese Schranke wird die Optimierung beendet.

MOPO-Parameter	Bedeutung
Objective Precision	Liegt die Änderung der skalierten Zielgrößen-Werte unter diese Schranke wird die Optimierung beendet.
Max. Calculation Steps	Nach dieser Anzahl von Schritten wird die Optimierung beendet.
Globales Problem: Min	Kleinste Schrittweite für die Liniensuche. Wird dieser Wert erreicht erfolgt ein neuer lokaler Schritt.
Globales Problem: Max	Obergrenze für die Schrittweite der Liniensuche.
Lokales Problem: Initial	Startwert für die erste Parameter-Auslenkung zur Gradientenberechnung.
Lokales Problem: Min	Kleinste Parameter-Auslenkung zur Gradientenberechnung.
Lokales Problem: Max	Größte Parameter-Auslenkung zur Gradientenberechnung.

7.2.4 Optimierungsergebnis

Die untenstehende Abbildung zeigt die Verläufe der optimierten Parameter, aufgetragen über die Optimierungsschritte. Dabei zeigen die gestrichelten Linien zusätzlich die Parametergrenzen:

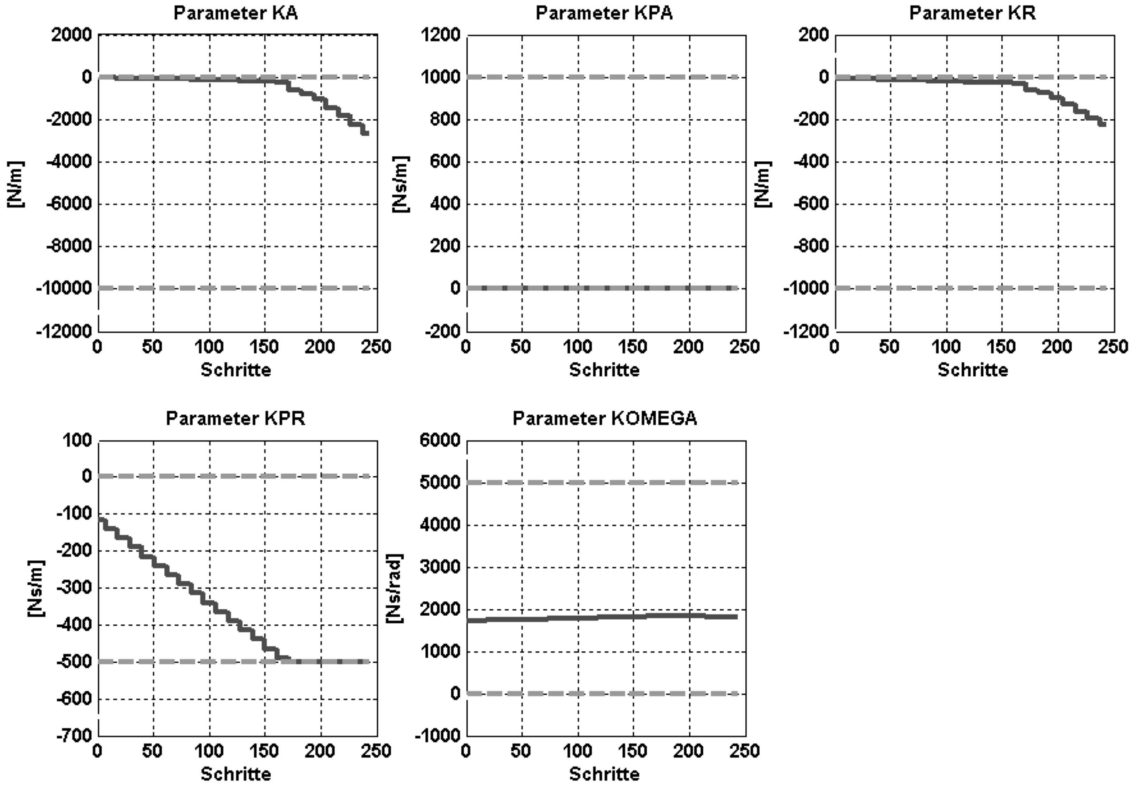

Abbildung 7-6: Parameterverläufe während der Optimierung

Die Polverläufe im Optimierungsverlauf zeigen, dass sich im wesentlichen die Pole des Rades geändert haben. Sie bleiben dabei überwiegend im Bereich der Vorgaben des Kreisringsektors. Die Pole von Translation und Knicken des Aufbaus haben sich nicht signifikant verändert, wobei für das Knicken die Vorgaben des Kreisringsektors nie erfüllt werden. Abbildung 7-7 zeigt die zugehörigen Polverläufe in der oberen linken, komplexen Halbebene (positive Imaginärteile,

negative Realteile) als Scilab-Plot:

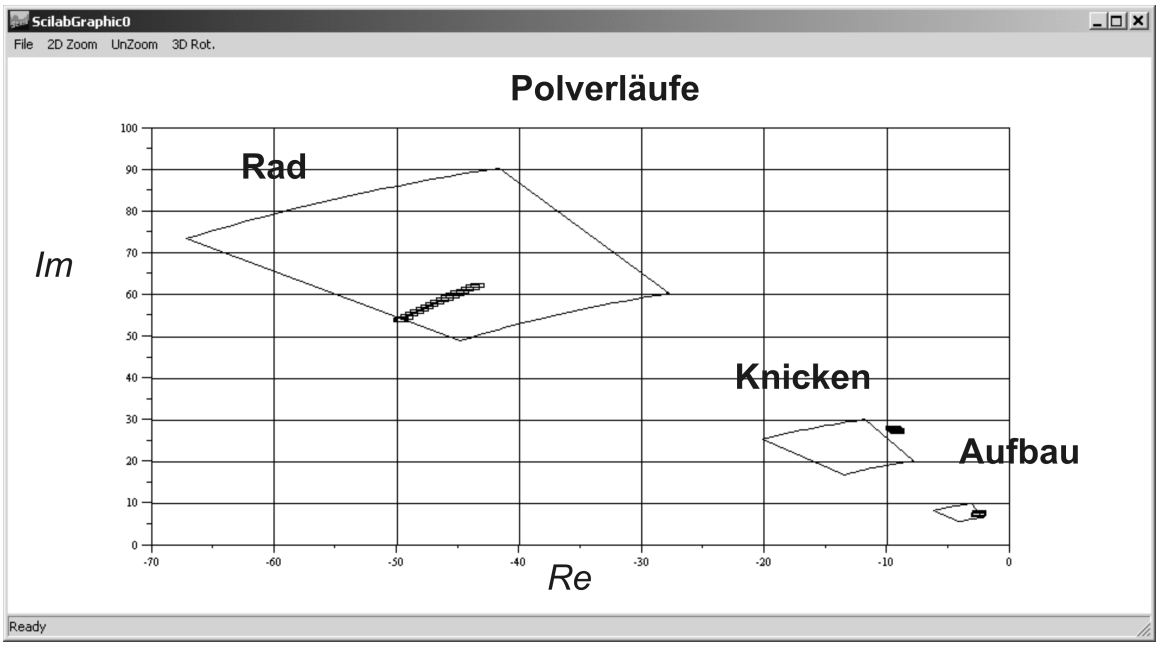

Abbildung 7-7: Polverläufe in der komplexen Ebene (nur positive Imaginärteile)

Betrachtet man die Zielgrößenverläufe in Abbildung 7-8, so zeigt sich, dass die Streuung für die Radlast vermindert wird, während die Streuungen für Aufbaubeschleunigung und Knick-Winkelbeschleunigung vergrößert werden. Insgesamt können alle drei Streuungen nicht unterhalb der oberen Limits (gestrichelte Linien in Abbildung 7-8) gebracht werden. Die unteren Limits für die Pollagen von Aufbau und Rad werden nahezu eingehalten:

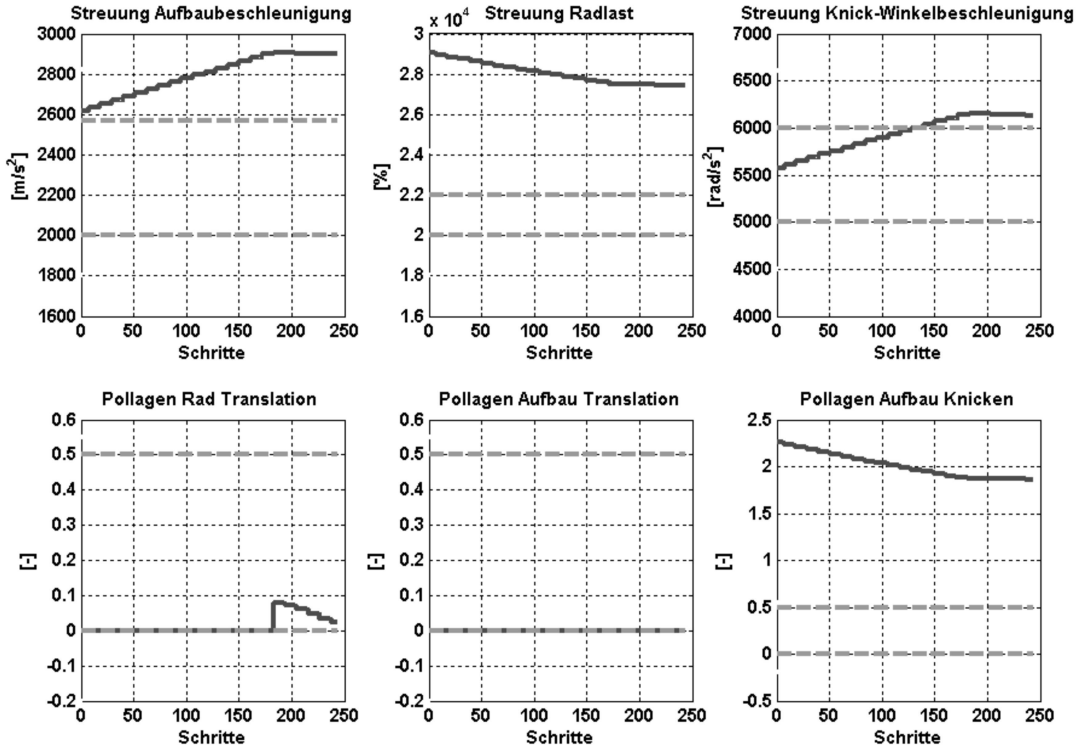

Abbildung 7-8: Zielgrößenverläufe während der Optimierung

Vergleichende Simulation

Durch die Wahl der Reglerparameter und die entsprechende Kraftüberlagerung des Kraftstellers können gezielt die Feder-/Dämpfereigenschaften des aktiv gefederten Modells beeinflusst werden. Durch Addition der passiven Feder-/Dämpferparameter mit den optimierten Reglerparametern können die Parameter des aktiven Systems beschrieben werden:

TABELLE 14. Ausgewählte Parameter des passiven & aktiven Systems

Name	Beschreibung	Parameterwerte Passiv (Reglerparameter Null)	Parameterwerte Aktiv
ma	Einzelne Aufbaumasse	430 kg	430 kg
mr	Einzelne Radmasse	30 kg	30 kg
ca+KA	Federbein Steifigkeit	20000 N/m	17349 N/m
da+KPA	Federbein Dämpfung	2000 Ns/m	2000 Ns/m
cr+KR	Reifen Steifigkeit	200000 N/m	199782 N/m
dr+KPR	Reifen Dämpfung	500 Ns/m	0 Ns/m
crot	Knicksteifigkeit	50000 N/rad	50000 N/rad
drot+KOMEGA	Knickdämpfung	50 Ns/rad	1865 Ns/rad

Zum Vergleich des dynamischen Verhaltens des passiven und aktiven Systems wird das Einschwingen und eine nachfolgende Sprungantwort (*5 cm* Strassenanregung xs_v, vorne bei *3.5 s*, hinten bei *3.8 s*) für vorderen Aufbauhub, Knickwinkel, vordere Radlast und vordere Aufbaubeschleunigung dargestellt. Im Falle des Einschwingens zeigen sich im aktiven Fall Verbesserungen (kleinere Amplituden) für alle dargestellten Größen. Bei der Sprungantwort fällt insbeson-

dere die stark verbesserte Dämpfung des Knickwinkels auf:

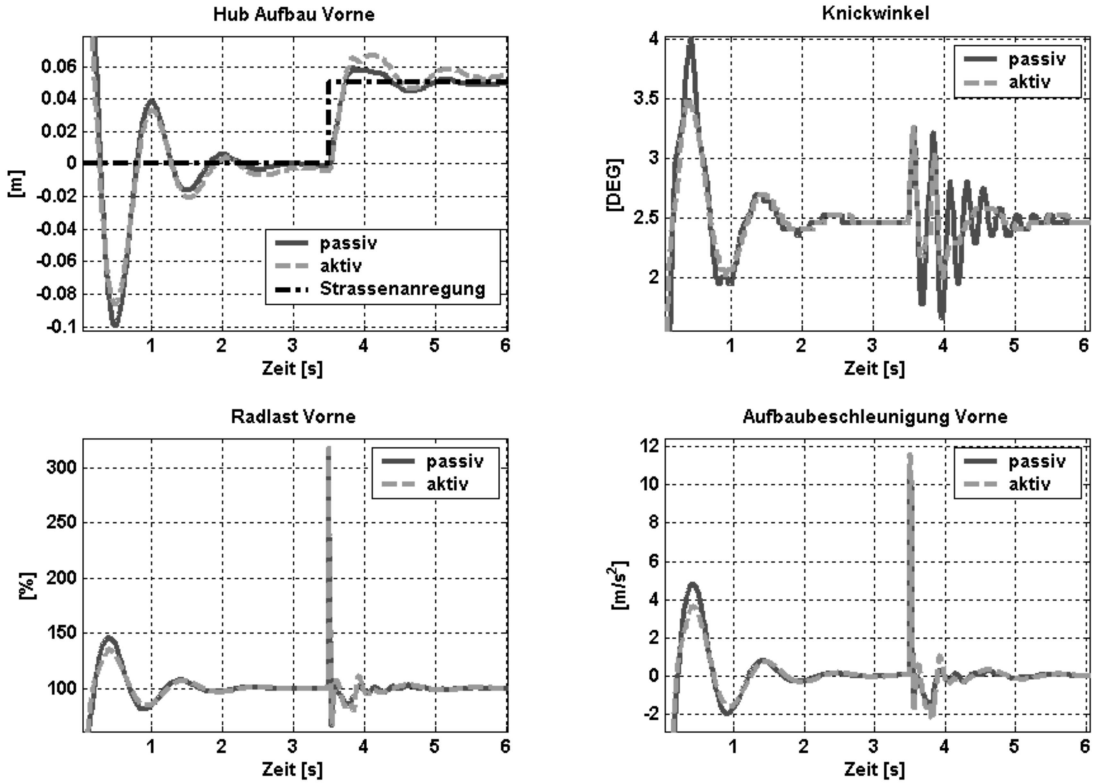

Abbildung 7-9: Vergleich des dynamischen Verhaltens vor und nach der Optimierung

7.3 Online-Nachoptimierung von Reglerparametern am HIL-Prüfstand

Im Projekt „Neue Bahntechnik Paderborn“ wird derzeit ein modulares Bahnkonzept entwickelt, das moderne Fahrwerkstechnologie mit den Vorteilen des Transrapid und der Nutzung der bestehenden Bahntrassen vereint. Am Beispiel des aktiv geregelten Feder-/Neige-Moduls als Teil des Federungskonzepts für den Wagenkasten der Bahn-Shuttles werden Echtzeit-Optimierung und -Überwachung vorgestellt. Die Anwendung erfolgt an einem vorhandenen HILS-Prüfstand, der zum Test des aktiven Fahrwerks und der Neigetechnik dient:

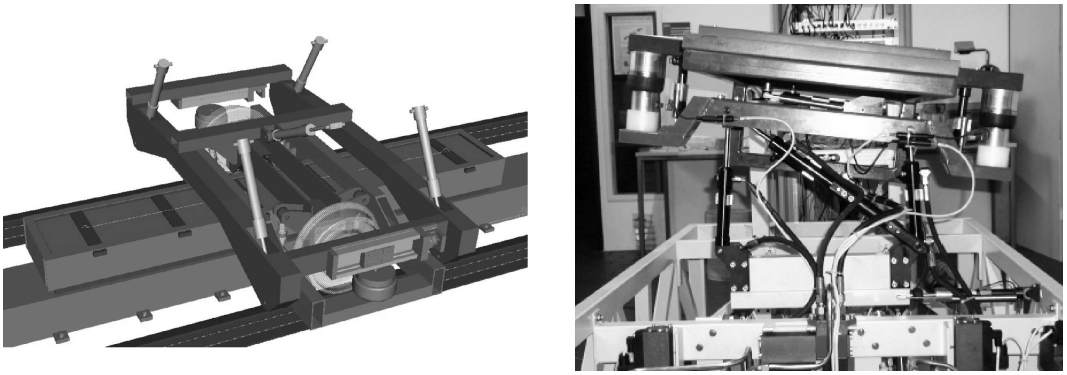

Abbildung 7-10: Feder-/Neigemodul als CAD-Modell und auf dem HILS-Prüfstand

Abbildung 7-10 zeigt den Aufbau des Feder-/Neigemoduls als CAD-Modell und einen ersten Prototypen im Maßstab *1:2,5* auf dem HILS-Prüfstand. Der Fahrzeugaufbau (hier Stahlplatten)

ist ausschließlich über Luftfedern mit dem Fahrwerksträger verbunden. Die Aktorik des Feder-/Neigemoduls besteht aus drei Hydraulikzylindern, die in der Bildmitte zu erkennen sind. Durch die Luftfedern ist eine optimale schwingungstechnische Entkopplung zwischen Aufbau und Rädern gegeben. Allerdings erzeugen die Luftfedern prinzipbedingt eine geringe Dämpfung der Aufbaubewegung. Den Aufbau optimal zu dämpfen ist daher die Aufgabe der Hydraulikaktoren, die dazu über ein geeignetes Regelgesetz (auf AMS-Ebene) angesteuert werden.

Vor dem Bau des Prüfstands erfolgten umfangreiche Arbeiten im Bereich Modellbildung und Reglerentwurf ([Liu-Henke et al. 2000/1], [Liu-Henke et al. 2000/2], [Henke et al. 2000]). Anhand des Prüfstands kann die Übereinstimmung zwischen Modell und Realität überprüft werden. Wendet man eine Online-Nachoptimierung auf das reale System an, die der vorher verwendeten modellbasierten Optimierung entspricht, so kann anhand der Unterschiede der Optimierungsergebnisse auf die Modell-/Reglergüte geschlossen werden. Durch Reglernachoptimierung am realen System können auch die durch Herstellungstoleranzen hervorgerufenen Unterschiede von Komponenten (Aktoren, Sensoren etc.), die aufwendig zu modellieren sind, berücksichtigt werden. Insgesamt ist aufgrund von Parameter-Unsicherheiten, Nichtlinearitäten, Effekten aus der Messtechnik und der digitalen Reglerrealisierung etc. davon auszugehen, dass es generell kaum möglich ist, ein völlig exaktes Modell eines mechatronischen Systems zu erstellen.

Die Online-Optimierung an sich ist keine neue Technik, komplette Fahrwerke von Fahrzeugen auf Hydro-Pulser-Prüfständen zu optimieren ist beispielsweise eine eingeführte Methode [Rutz, Winkler 1994]. In der vorliegenden Anwendung wird jedoch noch einen Schritt weitergegangen: Motiviert durch die Idee der Optimierung im Betrieb über die gesamte Lebensdauer eines mechatronischen Systems, werden die Systembewertung und der Optimierungsalgorithmus (MOPO-Gradientenverfahren) mit in die harte Echtzeitschleife des Regelsystems des Feder-/Neigemoduls integriert ([Deppe, Oberschelp 2000], [Deppe at al. 2001/2]).

7.3.1 Operator-Controller-Modul auf MFM-Ebene

Im Zuge der Online-Mehrziel-Optimierung sollen die Reglerparameter auf MFM-Ebene für die Hydraulikventile und die Differentialzylinder modifiziert werden. Dies soll allerdings nicht für alle Komponenten gleichzeitig geschehen, sondern für jede Aktor-Ventil-Kombination einzeln. Dabei wird die Anwendung auf die Optimierung der Parameter für die vertikal angeordneten Aktoren I und III (vgl. Abbildung 7-11) beschränkt. Pro Aktor-Ventil-Gruppe gilt es drei Parameter zu optimieren. Dies sind die beiden Parameter des unterlagerten PD-Reglers für die Ventilschieberposition und die Verstärkung des Lagereglers für den Differentialzylinder.

Für diese Anwendung ist ein Operator-Controller-Modul (OCM) umzusetzen, das durch Umschalten für beide Aktoren sequentiell genutzt werden kann. Aus der Echtzeit-Implementierung von Evaluator und Optimierer erwachsen zwei wichtige Forderungen an die Realisierung dieses OCMs:

1. Da der Optimierer die Reglerparameter zur Laufzeit aktiv verändert, ist es aus Sicherheitsgründen notwendig, eine modellbasierte Online-Überwachung im Zeit- und im Frequenzbereich zu implementieren. Sie muss in der Lage sein, ein Fehlverhalten (Instabilität, große Regelabweichung etc.) zu erkennen und durch Einschalten einer Rückfallregelung das System innerhalb weniger Millisekunden wieder zu stabilisieren.
2. Aufgrund der unterschiedlichen Taktraten für Bewertung und Optimierung und zur unbedingten Sicherstellung der Echtzeitfähigkeit des OCMs ist eine Multirate-Realisierung auf der Basis von präemptivem Multitasking erforderlich. Die Regler-/Überwachungsroutinen werden dabei der Task mit der höchsten Priorität zugeordnet, die von keiner anderen Task unterbrochen werden kann.

Abbildung 7-11 zeigt die OE(S)-Struktur der Controller-Nachoptimierung für das reale System "Feder-Neige Prüfstand". Ein eigenes Simulator Modul ist nicht erforderlich, da direkt an das reale System angekoppelt wird. Ausgänge des Evaluators an das reale System sind die aktuellen Reglerparameter sowie Daten aus dem Zeit- und Frequenzbereich, die zur Systemüberwachung herangezogen werden. In diesem Fall werden die Lageabweichung des Aktors und die Abweichung des Autoleistungsdichtespektrums des Ventilschieberwegs zwischen der Messung und einem Referenzmodell zur Überwachung verwendet.

Die Aufgabe der "Parameter-Übertragung" besteht darin, die jeweiligen neuen Reglerparameter auf den Controller zu übertragen. Diese Übertragung ist in diesem Fall nicht aufwändig, da keine internen Reglerzustände neu initialisiert werden müssen (P-PD-Kaskade). Die Reglerverstärkungen können "hart" umgeschaltet werden, sobald der Rückfall-Controller aktiv ist. Die Umschaltung zwischen Controller und Rückfall-Controller erfolgt durch Überblenden der Stellgröße.

Die "Überwachung" prüft, ob die Zeit- und Frequenzdaten aus dem Vergleich von Messung und Referenzmodell innerhalb vorgegebener Grenzen (Limits) liegen. Werden Grenzwerte überschritten schaltet die "Überwachung" auf einen Rückfall-Controller um. Dessen Reglerparameter

sind zur Laufzeit konstant und wurden vorab ausgelegt:

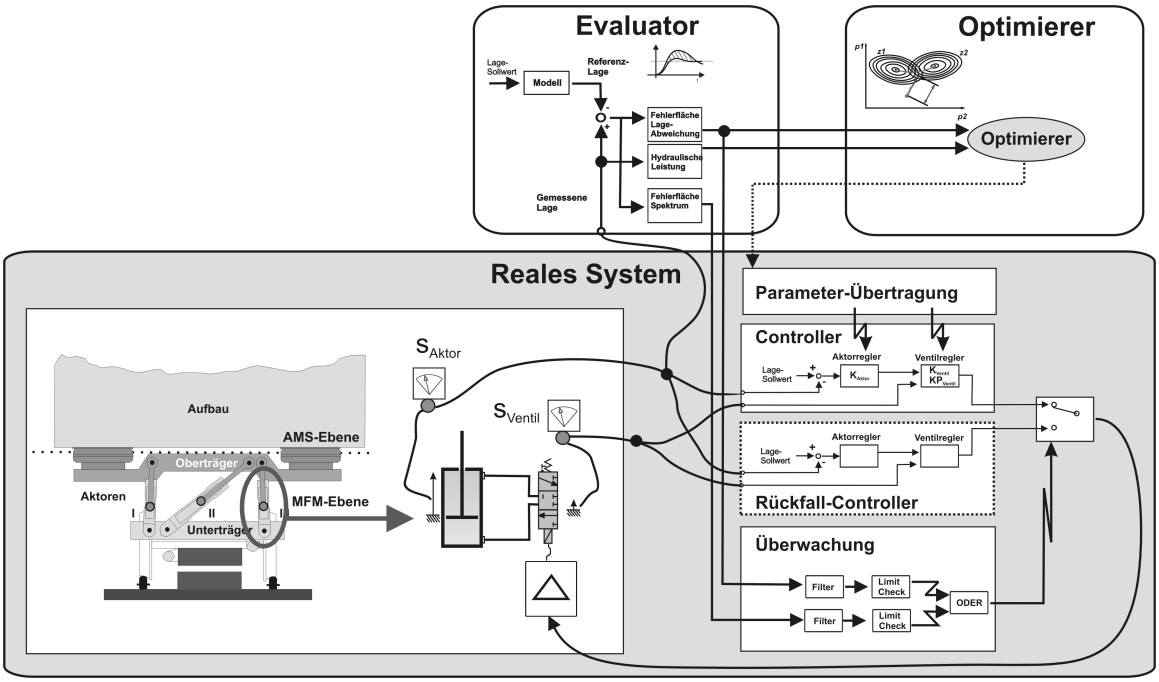

Abbildung 7-11: OES-Struktur der Nachoptimierung am realen System "Feder-Neige Prüfstand"

7.3.2 Implementierung

Zur echtzeitfähigen Implementierung wird zunächst eine Zuordnung der verschiedenen OES-Komponenten zu IPANEMA-Objekten vorgenommen. Zur Anbindung an die Mess- und Stellsignale des HILS-Prüfstands besitzt der Simulator neben einem IPANEMA-Calculator auch einen IPANEMA-Adaptor. Softwaretechnisch existieren somit drei Calculatoren und ein Adaptor, wobei die OES-Funktionen der gedanklichen Strukturierung der Anwendung dienen.

Für die Anwendung werden zwei Echtzeit-Tasks konfiguriert. Der Optimierer-Calculator wird dabei einer niederpriorisierten Task mit einer Zykluszeit von zwei Millisekunden zugeordnet. Die Calculatoren und der Adaptor für den Simulator und den Evaluator laufen in einer hochpriorisierten Task mit einer Zykluszeit von einer Millisekunde und können damit nie von der Optimierungs-Task unterbrochen werden.

Als Rechen-Hardware wird eine PPC750-Prozessorkarte mit entsprechenden I/O-Karten (Fa. dSPACE) verwendet. IPANEMA setzt dabei auf den mitgelieferten multitaskingfähigen Real-Time-Kernel (RTK) von dSPACE auf. Der RTK stellt im Wesentlichen Dienste zum Zugriff auf die I/O-Karten und zur Organisation des Multitasking zur Verfügung. Als Scheduling-Verfah-

ren wird das enthaltene Rate-Monotonic-Scheduling (RMS) genutzt (vgl. Kap. 5.2):

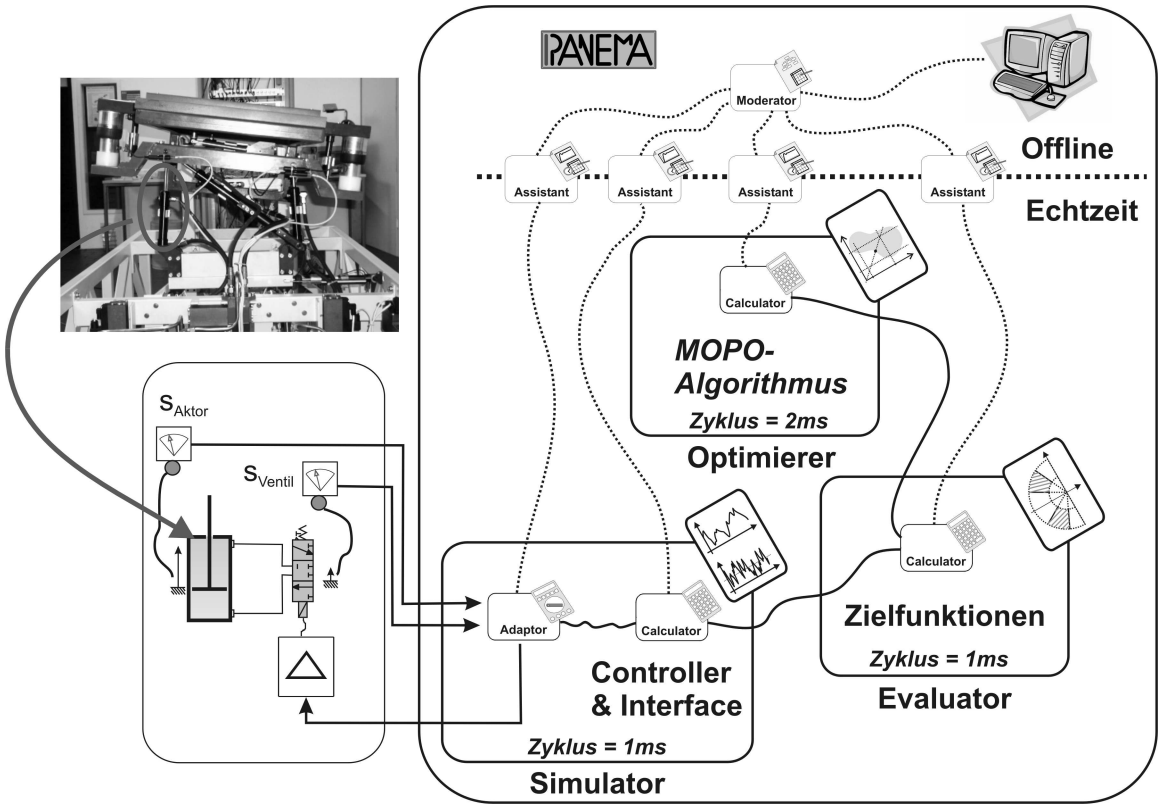

Abbildung 7-12: Mit IPANEMA realisierte echtzeitfähige OES-Struktur

7.3.3 Parameter und Zielgrößen

Die zu optimierenden Parameter sind die Verstärkungen für die Lagerückführung K_{Ventil}, die Geschwindigkeitsrückführung KP_{Ventil} des Ventilschiebers und die Verstärkung des Lagereglers K_{Aktor} für den Differentialzylinder. Bewertet werden Stellgenauigkeit und hydraulische Leistung des Aktors. Es ergibt sich insgesamt eine Optimierung mit zwei Zielgrößen und drei Parametern.

Zur Bewertung der Aktor-Positionsgenauigkeit wird das Systemverhalten mit einem ideal gedämpften System 2. Ordnung verglichen (Dämpfungsmaß *0,7* und Eigenfrequenz *20 Hz*). Dieses findet auch Verwendung für der Überwachung des Systems. Die quadratische Fehlerfläche ISE (Integral Squared Error) zwischen der Aktorposition und der Position des Referenzsystems wird als Maß für die Güte der Regelung verwendet.

Die hydraulische Leistung P_H eines Aktors wird als direkte Zielgröße verwendet. Zur Schätzung der hydraulischen Leistung dient eine Druck- und Volumenstrom-Betrachtung für den Differentialzylinder. Vereinfachend wird eine mittlere Kolbenfläche für beide Kammern angenommen, da hier weniger der absolute Wert der hydraulischen Leistung als vielmehr ihre relative Änderung wichtig ist. Der dynamische Druck in der belasteten Zylinder-Kammer wird über die Relativgeschwindigkeit und die Relativbeschleunigung der Kolbenstange, die aus der Messung der Aktorauslenkung durch Differentiationen gewonnen werden können, bestimmt. Die Kompressibilität des Hydrauliköls wird vernachlässigt:

TABELLE 15. Erläuterungen zur Schätzung der hydraulischen Leistung

Name	Formelzeichen	Einheit	Wert/Berechnung	Typ
Bewegte Masse	m	[kg]	52,5[a]	Parameter
Kolbenfläche	A_K	[m^2]	1,2566e-3	Parameter
Viskose Reibung	d	[Ns/m]	100[b]	Parameter
Statischer Druck	p_{stat}	[N/m^2]	41e5[c]	Parameter
Aktor Auslenkung	s	[m]	aus Messung	Eingang
Aktor Relativ-geschwindigkeit	$\dot{s}$	[m/s]	s einmal differenzieren	Eingang
Aktor Relativ-beschleunigung	$\ddot{s}$	[m/s^2]	s zweimal differenzieren	Eingang
Volumenstrom	q_H	[m^3/s]	$q_H = A_K \cdot \dot{s}$	Ausgang
Dynamischer Druck	p_{dyn}	[N/m^2]	$p_{dyn} = p_{stat} + \frac{m \cdot \ddot{s} + d \cdot \dot{s}}{A_K}$	Ausgang
Hydraulische Leistung	P_H	[W]	$P_H = q_H \cdot p_{dyn}$	Ausgang

a. Hälfte der Aufbaumasse (ca. 105 kg)

b. Geschätzter Wert

c. Entspricht dem Hydraulikdruck im Aktor bei 52,5 kg Gewichtsbelastung

7.3.4 Optimierungsexperiment

Für die Optimierung wird das echtzeitfähige MOPO-Gradientenverfahren genutzt. Bei der Konfiguration der Schrittweiten für die Paramtersuche gilt es zu beachten, dass die minimale Auslenkung eines Parameters eine deterministische Veränderung der Zielgrößen bewirken muss, die größer als ihr Rauschpegel ist. Dies ist ein signifikanter Unterschied zu Offline-Optimierungen, bei denen keine Beachtung von Messrauschen o. ä. notwendig ist. Die optimale Schrittweiten-Konfiguration muss ggf. mit Hilfe mehrerer Testläufe bestimmt werden. Um eine möglichst gute Mittelung der Zielgrößen zu erreichen, wird ein Parametervektor über zwei Sekunden (entspricht *2.000* Simulationsschritten) konstant gehalten und bewertet, bevor eine neuer Parametervektor aufgeschaltet wird. Jeweils *200* Messwerte werden für eine Zielfunktionsauswertung herangezo-

gen:

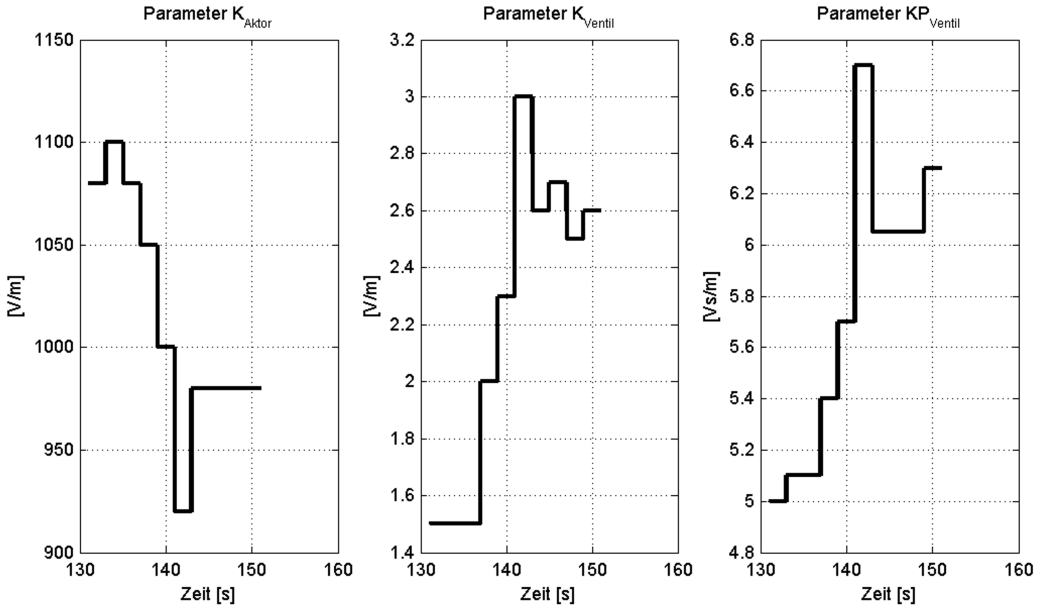

Abbildung 7-13: Parameteränderung von Aktor I während der Optimierung

Der HILS-Prüfstand bietet die Möglichkeit, über Hydraulikaktoren eine vertikale Gleisanregung auf den Unterträger (vgl. Abbildung 7-11) zu erzeugen. Für das Experiment hat sich eine Rechteckanregung mit einer Frequenz von *2 Hz* und einer Amplitude von *1 mm* als ausreichend herausgestellt, wobei diese auch im Bereich von realistischen Gleisanregungen liegt.

Die nachfolgende Abbildung zeigt die Änderung der Zielgrößen für Aktor I. Hier ist zu sehen, dass sowohl Lageabweichung als auch Leistung gleichzeitig innerhalb von *20* Sekunden minimiert werden können:

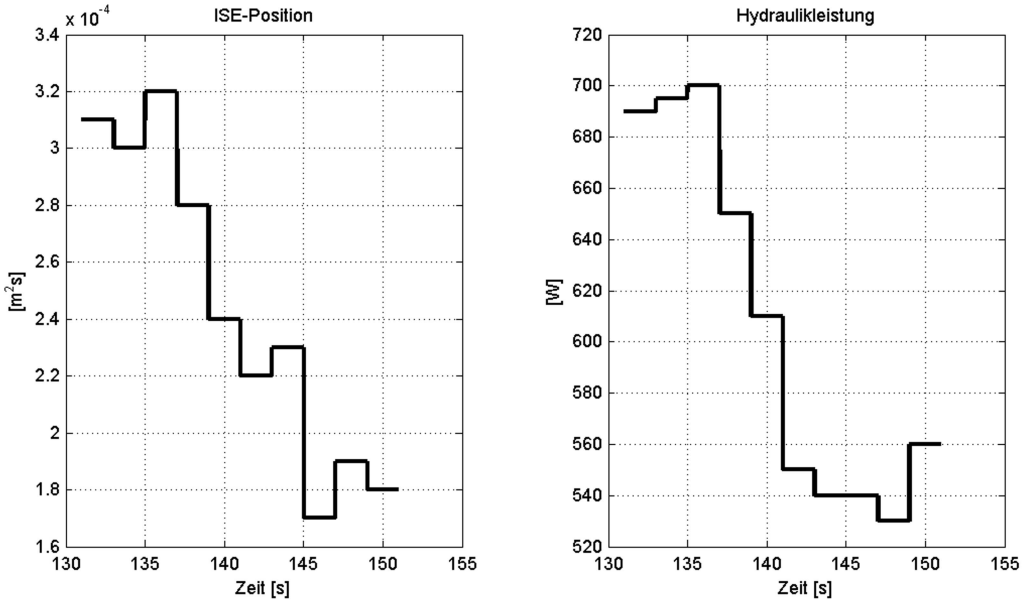

Abbildung 7-14: Zielgrößenänderung von Aktor I während der Optimierung

7.3.5 Zusammenfassung

Die Einbindung des Optimierers in die Echtzeitschleife hat sich als sehr effizient und robust herausgestellt. Mit Hilfe der Überwachungsfunktionen ist ein zuverlässiger Betrieb möglich. Die unterschiedlichen Optimierungsergebnisse für die baugleichen und symmetrisch angeordneten Aktoren I und III zeigen, dass beide Komponenten nicht als identisch angesehen werden können. Sämtliche akkumulierten Abweichungen der verwendeten Regelkreisglieder (Sensoren, Aktoren, Messtechnik etc.) werden somit durch die Online-Optimierung aufgedeckt. Diese Ergebnisse sind für eine Modellverbesserung und somit für eine weitere, modellbasierte Optimierung sehr wertvoll:

TABELLE 16. Start- und Endparameter für Aktor I und Aktor III

Parameter	**K_{Aktor} [V/m]**	**K_{Ventil} [V/m]**	**KP_{Ventil} [Vs/m]**
Startwerte	1.000,0	1,5	5,0
Ergebnis Aktor I	980,0	2,6	6,3
Ergebnis Aktor III	784,0	2,2	6,2

„Dass intelligente Verkehrstechnologie allein die Probleme des wachsenden Verkehrsaufkommens bewältigen würde, hat nie jemand behauptet. Aber umgekehrt wird ein Schuh daraus. Zu glauben, man könne auf intelligente Verkehrslenkung verzichten, obwohl einem die Probleme über den Kopf wachsen, das ist eine Illusion.“

Der Münchner Oberbürgermeister Christian Ude zur Vertragsunterzeichnung MOBINET (*www.mobinet.de*)

7.4 Selbstorganisierendes Kreuzungsmanagement

Betrachtet man die kritische Zunahme der Verkehrsdichte, so ist die massive Vernetzung von Fahrzeugen zur Vermeidung des Verkehrsinfarkts unumgänglich. Am Beispiel einer Straßenkreuzung wird aufgezeigt, wie durch die Vernetzung von autonomen Einzelfahrzeugen ein selbstorganisierendes Kreuzungsmanagement zur kollisionsfreien Überquerung einer Straßenkreuzung entsteht. Ziel ist es, durch die Kooperation der Einzelfahrzeuge eine online-optimierte Überquerung ohne Stillstand der Fahrzeuge umzusetzen.

Die Ergebnisse wurden im Rahmen des Sonderforschungsbereichs 376 „Massive Parallelität - Algorithmen, Entwurfsmethoden, Anwendungen“ erarbeitet.

Im Laufe der Zeit hat das Kreuzungsmanagement verschiedene Entwicklungsstufen durchlaufen. Anfänglich wurden potentielle Kollisionspunkte mit Hilfe eines kombinierten Semaphoren-/ Token-Verfahrens ([Naumann, Rasche 1997], [Rasche et al. 1997], [Rasche 2004]) verwaltet (vgl. Abbildung 7-15). Später wurde die zeitliche Überlappung des Aufenthalts in kritischen Zonen der Kreuzung mit Hilfe des MOPO-Gradientenverfahrens im Zuge einer Online-Optimierung überschneidungsfrei eingestellt ([Lückel et al. 1999], [Rasche 2004]):

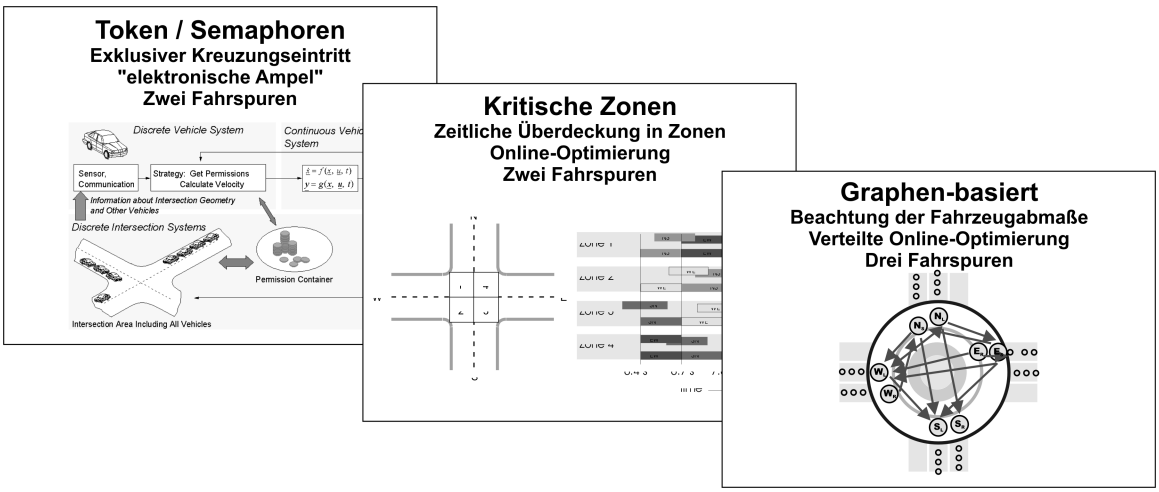

Abbildung 7-15: Entwicklungsstufen des Kreuzungsmanagements

Das nachfolgend vorgestellte neue Kreuzungsmanagement ist in wesentlichen Punkten erweitert worden, um eine größere Realitätsnähe zu erzielen.

7.4.1 Neues Kreuzungsmanagement

Die Erweiterungen des Kreuzungsmanagements umfassen weite Bereiche der zugehörigen Modelle, Simulations- und Optimierungsansätze [Neuendorf, Deppe 2003]. Eine echtzeitfähige Umsetzung war dabei nicht im Fokus; als Hardware-Plattform sind daher vernetzte Workstations (NOW) vorgesehen.

Die wichtigsten Veränderungen gegenüber den bisherigen Verfahren lassen sich folgendermaßen zusammenfassen:

1. Erweiterung auf zwei Einfahrtsspuren (Linksabbieger- und Rechts-/Geradeausspur)
2. Ebenenweise Online-Optimierung mit neuem Quasi-Newton-Verfahren
3. Einbeziehung der Fahrzeugabmaße und variabler Abbiegetrajektorien
4. Dynamische Fahrzeuganzahl (An-/Abmelden eines Fahrzeuges entspricht der Lebensdauer eines zugehörigen Simulators)
5. Variable Modellierungstiefe auf AMS-Ebene
6. Graphenbasierte Vorfahrtsbestimmung
7. Parallelverarbeitung auf Basis der OES-Struktur (zunächst offline)

Das Kreuzungsmanagement kann zur Leistungssteigerung des Verfahrens in eine zweistufige Optimierung aufgeteilt werden. Es findet eine lokale Optimierung der Quer- und der Längsdynamik auf Fahrzeugebene (AMS) statt. Auf Kreuzungsebene (VMS1) wird das dynamische Netz der Fahrzeugabhängigkeiten gebildet („virtuelle Kolonnen“) und optimiert. Die Abstands-/ Kolonnenregler (VMS0) setzen die Vorgaben der Kreuzungsebene um:

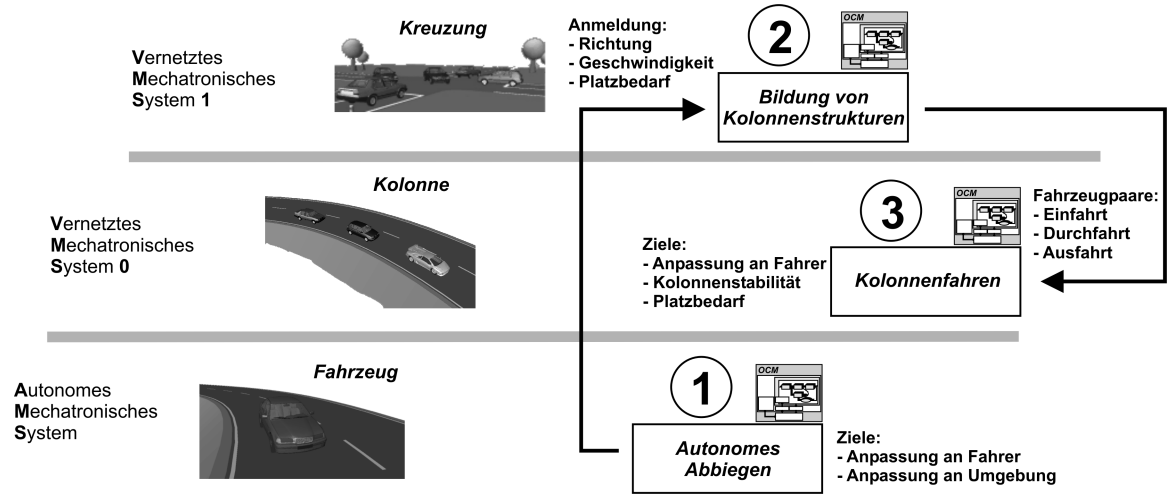

Abbildung 7-16: Vorgehensweise für die mehrstufige Optimierung

Die Aufgaben der einzelnen Ebenen sind:

- Fahrzeugintern (AMS-Ebene):
  Anpassung der Abbiegetrajektorie und des Geschwindigkeitsprofils an dynamisch veränderliche Randbedingungen: Fahrerwünsche (sportlich/komfortorientiert → maximale Beschleunigungen), Straßenverhältnisse, Abbiege-Korridor (Platzbedarf) u. ä. Diese Optimierungsstufe ist als Ersatz oder Assistent für den menschlichen Fahrer interpretierbar.

- Kreuzung (VMS1-Ebene):
  Bestimmung der Vorfahrt und Optimierung der virtuellen Kolonnen. Ergebnis sind die Geschwindigkeitsprofile als Führungsgrößen für jedes beteiligte Fahrzeug (initiale modellhafte Lösung für das geregelte Kolonnenfahren) und die Fahrzeugpaarungen für das Kolonnenfahren. Berücksichtigt werden die Vorgaben aus der Logistik-/Routenplanungsebene (VMS2), die Optimierungsergebnisse aus der AMS-Ebene (minimale Abweichung hiervon), der Durchsatz, der Energieverbrauch u. ä. Zielgrößen. Auf dieser planenden Ebene ist hohes Optimierungspotential vorhanden.

- Kolonne (VMS0-Ebene):
  Der Abstands-/Kolonnenregler führt die Vorgaben der VMS1-Ebene aus. Bei Abweichung von den errechneten Geschwindigkeitsprofilen werden die Profile aller relevanten nachfolgenden Fahrzeuge entsprechend korrigiert, um Kollisionen zu vermeiden. Da es sich um virtuelle Kolonnen handelt, d. h. um Fahrzeugkombinationen aus unterschiedlichen Richtungen der Kreuzung, muss mit entsprechender Technik (z. B. Funk) eine Kommunikation der Fahrzeuge untereinander sichergestellt sein.

7.4.2 Einteilung der Kreuzung in Zonen

Die Fahrt durch die Kreuzung wird in folgende Phasen bzw. Zonen eingeteilt:

- In der *Kontaktphase/-zone* wird dem Fahrer die Kontrolle entzogen und das Fahrzeug auf eine vordefinierte konstante Geschwindigkeit gebracht. In dieser Zeit kann der Fahrer auch seinen Richtungswunsch angeben. Das Fahrzeug wird in das Kommunikationsnetzwerk der Kreuzung eingebunden. Es werden Informationen über die Kreuzungsgeometrie und die anderen Verkehrsteilnehmer, mit denen es zu einer Kollision kommen könnte, ausgetauscht.

- In der *Strategiephase/-zone* wird schnell eine sichere suboptimale Lösung gesucht. Die Fahrzeuge werden in den Kollisionsgraphen eingebunden, und ihnen wird ein Geschwindigkeitsprofil zugewiesen, mit dem sie unter allen Umständen die Kreuzung sicher überqueren können.

- Diese in der Strategiephase ermittelte Lösung dient als Startwert für die Optimierung in der *Optimierungsphase/-zone*. In dieser Phase wird ein Modell der nachfolgenden Handlungsphase genutzt, mit dem vorausschauend die Kollisionsfreiheit überprüft und die optimalen Fahrzeugabstände und -geschwindigkeiten ermittelt werden können. Dazu wird das eigene MOPO-Verfahren zur kontinuierlichen Mehrziel-Parameter-Optimierung verwendet. Die Optimierung muss bei Eintritt in die Handlungsphase/-zone beendet sein.

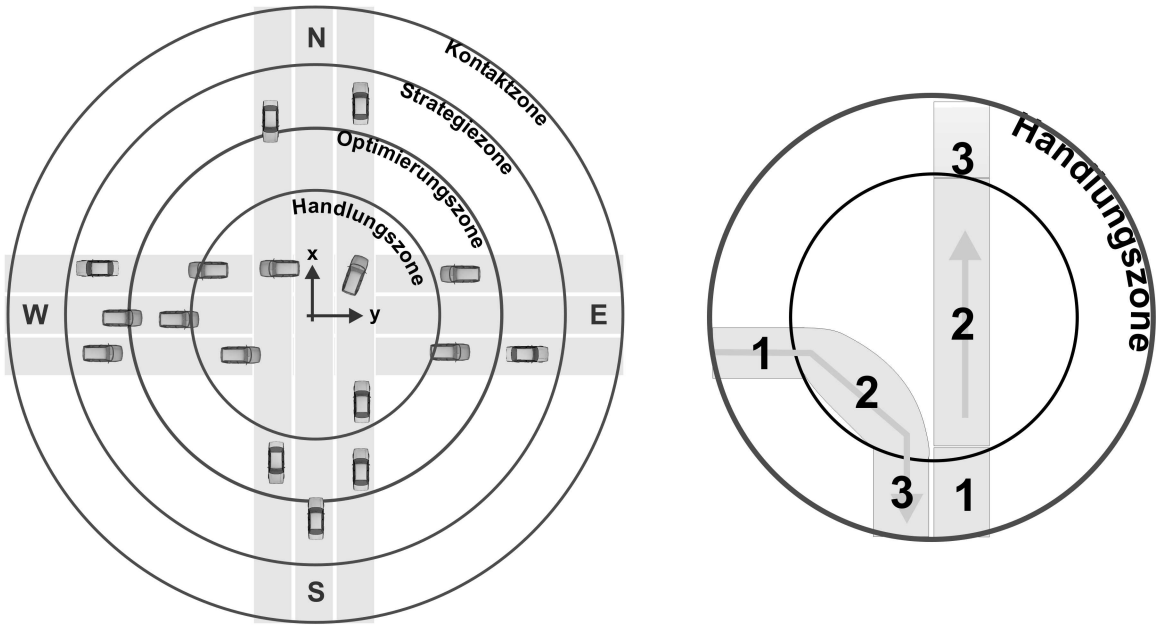

Abbildung 7-17: Zoneneinteilung der Kreuzung

- Die *Handlungsphase/-zone* ist diejenige Zone, die in der Strategie- und der Optimierungsphase betrachtet wird. Die Handlungszone ihrerseits ist unterteilt in
  - die Verzögerungszone (1): Zone, in der die in der Strategie- und der Optimierungszone ermittelten Geschwindigkeitsprofile umgesetzt werden,
  - die Durchfahrtszone (2): Zentraler Kreuzungsbereich, der die potentiellen Kollisionsorte enthält; hier findet auch das Abbiegen der Fahrzeuge statt,
  - die Beschleunigungszone (3): Ausfahren und ggf. Beschleunigen der Fahrzeuge auf ein höheres Geschwindigkeitsniveau.

Optimiert wird das Verhalten der Fahrzeuge in der Handlungszone. Daher wird auch nur diese in den Optimierungsexperimenten berücksichtigt.

7.4.3 Algorithmus zur Vorfahrtsbestimmung

Bei der Handlungszone der Kreuzung steht die Vorfahrtsbestimmung zu ihrer kollisionsfreien Überquerung im Mittelpunkt. Zum Einen wird die Handlungszone für die Vorhersage-Simulationen in den Strategie- und den Optimierungszonen simuliert, zum Anderen findet in der realen Handlungszone die vorgeplante Überquerung durch die Fahrzeuge statt.

Durch spezielle Abstandsvorgaben und Auswahl der „logischen“ Kolonnenvorgänger wird das Problem der kollisionsfreien Kreuzungsüberquerung auf eine Abstands-/Kolonnenregelung zurückgeführt. Eine vereinfachte Modellierung besteht in der Vorgabe von statisch vorab berechneten Geschwindigkeitsprofilen zur Einhaltung des Abstands. Hierbei erfolgt der Übergang zu einer „echten“ Abstands-/Kolonnenregelung dann, wenn das Geschwindigkeitsprofil des Nachfolgenden permanent durch das des Vorherfahrenden verändert wird.

Das Modell der Handlungszone soll dazu dienen, eine wenig rechenintensive, aber gute globale

Überquerung als Startpunkt für eine nachfolgende Optimierung zu bestimmen. Zur Bestimmung der globalen Überquerung werden folgende vereinfachende Annahmen für das Modell der Handlungszone getroffen:

1. Eine echte Abstands-/Kolonnenregelung ist aufgrund der idealisierten Annahme von fehlenden Störgrößen nicht notwendig. Stattdessen wird die Fahrzeuglängsbewegung durch Geschwindigkeits-Zeit- und Geschwindigkeits-Weg-Profile gesteuert. Dabei werden sinnvolle Begrenzungen für die Beschleunigung bei der Berechnung der Geschwindigkeits-Profile mitberücksichtigt.

2. Die Fahrzeugkontur wird zur effizienten Kollisions-/Abstandsprüfung durch zwei Kreise beschrieben (reine 2D-Betrachtung).

3. Jedes Fahrzeug kennt die Kreuzungsgeometrie.

4. Jedes Fahrzeug hat eine vorgeplante Bahn und nur der Geschwindigkeitsverlauf kann variiert werden.

5. Fahrzeuge können aus Ihren Bahnvorgaben gegenseitig ihre potentiellen Kollisionsorte in der Kreuzung berechnen (Schnittpunkte von bikubischen 2D-Spline-Kurven).

6. Die Fahrzeuge können in einem Funknetzwerk kommunizieren.

Zur Modellierung der Fahrzeugabhängigkeiten bei der Überquerung wird im neuen Kreuzungsmanagement ein gerichteter Graph verwendet. Die Fahrzeuge sind die Knoten des Graphen, wobei jedes Fahrzeug mit seinem Nachfolger in der Einfahrtsspur und allen potentiellen Kollisionspartnern (potentiell heisst: es existiert ein Bahnschnittpunkt im Kreuzungsbereich) über gerichtete Kanten verbunden wird. Es entsteht jeweils ein azyklischer gerichteter Graph (vgl. Kap. 2.2) mit einer Ausrichtung von der Kreuzungsmitte nach außen. Dieser Graph wird Kollisionsgraph genannt und kann maximal vier Komponenten enthalten (so ergeben vier Rechtsabbieger einen Graphen mit vier Knoten ohne Kanten, da die Fahrzeuge völlig unabhängig voneinander die Kreuzung überqueren können). Eine von einem Fahrzeug (Knoten) ausgehende Kante bedeutet Vorgänger gegenüber dem Ziel-Fahrzeug der Kante zu sein, d. h. in einer Fahrzeugkolonne zeigen die Kanten des Kollisionsgraphen entgegen der Fahrtrichtung.

Im gezeigten Beispiel (vgl. Abbildung 7-18) existiert nur eine Graphen-Komponente. Dies ist für eine hohe Verkehrsdichte allerdings auch der wahrscheinlichste Fall. Ziel ist es nun, eine Reihenfolge zu finden, in der die Zuteilung der Durchfahrt erfolgen kann. Der Lösungsalgorithmus arbeitet nach dem First-In-First-Get-Prinzip, d. h. je eher ein Fahrzeug eingebunden wird, desto weniger Einschränkungen aufgrund anderer Fahrzeuge (in Form von auf das Fahrzeug gerichtete Kanten des Kollisionsgraphen) sind zu erwarten.

Der Kollisionsgraph ist kein statisches Gebilde, sondern ständigen Veränderungen unterworfen (dynamisches Netz), da ständig neue Fahrzeuge hinzukommen und auch andere den Kreuzungsbereich verlassen. Für das Beispiel werden die Prioritäten willkürlich gemäß den Himmelsrichtungen in der Reihenfolge {North, East, West, South} gewählt. Als Randbedingung gilt: „Innere

haben Vorrang vor Äußeren“. Nach dieser Regel ergeben sich die Kanten des Kollisionsgraphen:

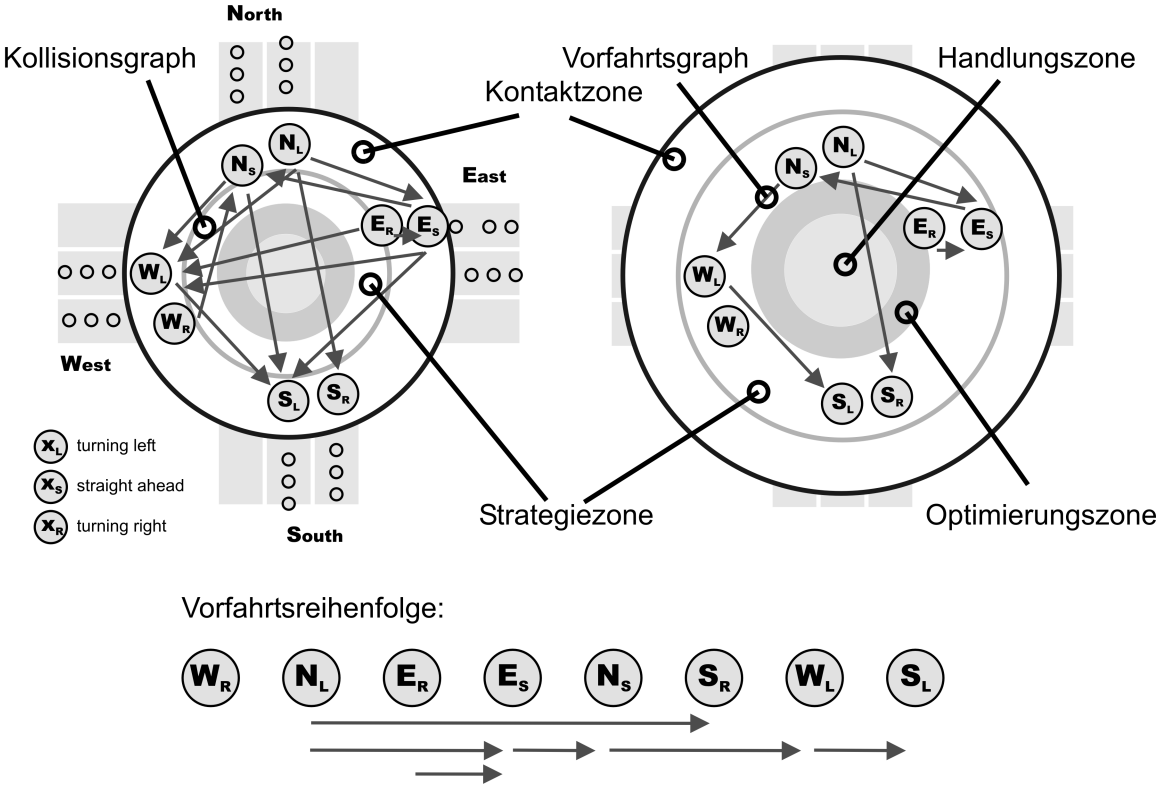

Abbildung 7-18: Kollisionsgraph und Vorfahrtsgraph

Für die Überquerung werden nun Fahrzeugkolonnen gebildet. Dabei wählt jedes Fahrzeug den Vorgänger aus, für den es am meisten verzögern muss. Somit ist gewährleistet, dass allen potentiellen Kollisionspartnern ausgewichen wird. Nach dieser Regel bildet sich eine neue Graph-Struktur (Vorfahrtsgraph) für die Vorfahrt aus dem Kollisionsgraphen heraus. Die Reihenfolge der Vorfahrt kann aus den möglichen topologischen Sortierungen (vgl. Kap. 2.2) des Vorfahrtsgraphen ausgewählt werden. Letztlich wird eine eindeutige Reihenfolge erzeugt, indem frühe Fahrzeuge mit geringerem Abstand zur Kreuzung bevorzugt werden (vgl. Kap. 7.4.8).

Der Kollisionsgraph wird in der Kontaktzone aufgebaut, der Vorfahrtsgraph in der Strategiezone bestimmt. Liegt die Vorfahrtsreihenfolge der Fahrzeuge fest, so werden für jedes Fahrzeug zwei Fälle unterschieden:

- Freies Fahrzeug: Es braucht keine Rücksicht auf andere Fahrzeuge genommen zu werden; die Geschwindigkeit kann vor der Absenkung auf die Durchfahrtsgeschwindigkeit zur Rückstau-Minimierung sogar kurzzeitig erhöht werden (Beispiel W_R, $N_L$und E_R).
- Abhängiges Fahrzeug: Es gibt genau einen Vorausfahrenden, anhand von dessen Durchfahrtsdaten (Bahn- und Geschwindigkeits-Trajektorie) die notwendige eigene relative Verschiebung berechnet wird. Dabei müssen potentielle Kollisionen in der gesamten Handlungszone (Verzögerungs-/Durchfahrts-/Beschleunigungszone) vermieden werden.

Die notwendige Verschiebung zum Vorausfahrenden wird durch den Vergleich der Weg-Zeit-Profile ermittelt. Dazu wird zunächst der potentielle erste Kollisionspunkt als Spline-Schnittpunkt der XY-Trajektorien der Fahrschläuche beider Fahrzeuge bestimmt. Anhand der beiden Weg-

Zeit-Profile, der unterschiedlichen Bahnlängen zum potentiellen Kollisionspunkt und eines vorgebbaren Sicherheitsabstands ergibt sich dann eine positive oder negative Verschiebung zum Vorausfahrenden.

Für das Kreuzungsmanagement erfolgt die Darstellung der Geschwindigkeitsverläufe mit Hilfe von kubischen 1D-Splines ([Späth 1973], [Schwarz 1993]). Diese Splines werden über Stützstellenwertepaare sowie die Anfangssteigung und die Endsteigung der Spline-Kurve konfiguriert.Zu einem Eingangswert werden der interpolierte Ausgangswert und dessen erste und zweite Ableitung ausgegeben.

Mit Hilfe der 2D-Splines lassen sich beliebige 2D-Trajektorien abbilden, die keine Funktionen sein müssen. Die 2D-Splines werden zur Definition der Abbiegespuren im Kreuzungsmanagement verwendet. Die Parametrisierung der Splines erfolgt über die Bogenlänge, wie auch in der Robotersteuerung üblich [Olomski 1989]. Die Definition erfolgt über Stützpunkte in der xy-Ebene (xi,yi) und einen Anfangs- und Endbahnwinkel. In Abhängigkeit der Bahnlänge werden die interpolierten Punkte, der Bahnwinkel und die Bahnkrümmung ausgegeben.

Zur Berechnung des zugehörigen Geschwindigkeitsverlaufs werden dreiecksförmige Beschleunigungsprofile vorgegeben, um den Ruck klein und abschnittsweise konstant zu halten. Das Abbremsen von der Einfahrts- auf die Durchfahrtsgeschwindigkeit wird in gleich große Zeitintervalle mit jeweils konstantem Ruck eingeteilt. Das Beschleunigen auf die Ausfahrtsgeschwindigkeit erfolgt nach dem gleichen Prinzip. Das nachfolgende Diagramm zeigt ein Beispiel für die Abstimmung eines Linksabbiegers aus Süden (Follower-SL) auf einen Rechtsabbieger aus Norden (Leader-NR), die gleichzeitig und mit gleicher Geschwindigkeit in die Handlungszone eintreten. Die Fahrzeuge können in der Durchfahrtszone kollidieren und bilden in der Ausfahrtszone eine Kolonne. Daher muss der Nachfolgende in der Einfahrtszone einen Sicherheitsabstand (hier

z. B. *20 m*) aufbauen, der in Durchfahrt und Ausfahrt bestehen bleibt:

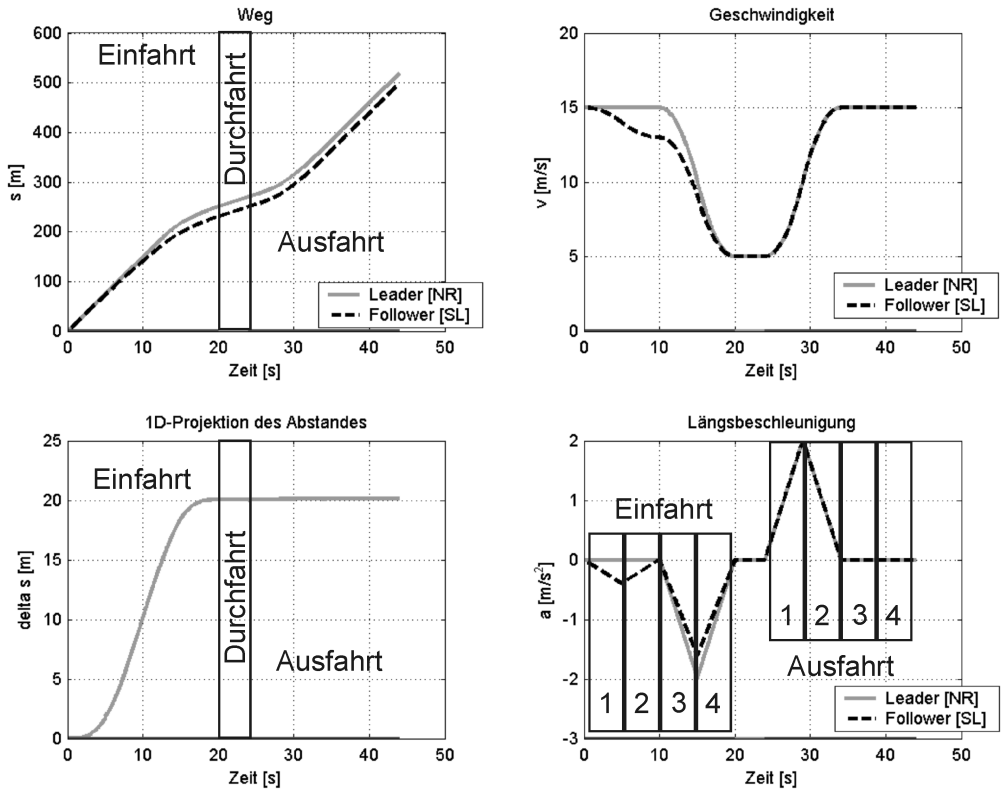

Abbildung 7-19: Verschiebung zweier Fahrzeuge zur Kollisionsvermeidung

Zur Berechnung werden zunächst vier gleich große Zeitintervalle gebildet, in denen jeweils konstanter Ruck herrschen soll. Diese Vorgaben werden gewählt, da die menschliche Wahrnehmung der Beschleunigungsänderung (durch die Fahrzeuginsassen) auf lineares Verhalten beschränkt ist. Für das erste Zeitintervall bedeutet dies einen zeitlichen Wegverlauf $s_1(t)$ gemäß:

$$s_1(t) = v_1 \cdot t + \frac{1}{6} \cdot \dddot{s}_{1,\,Max} \cdot t^3 \tag{7.1}$$

mit Anfangsgeschwindigkeit v_1, Zeit t und max. Ruck $\dddot{s}_{1,\,Max}$. Die zugehörigen Geschwindigkeits- und Beschleunigungsverläufe lauten:

$$\dot{s}_1(t) = v_1 + \frac{1}{2} \cdot \dddot{s}_{1,\,Max} \cdot t^2 \tag{7.2}$$

$$\ddot{s}_1(t) = \dddot{s}_{1,\,Max} \cdot t \tag{7.3}$$

Die restlichen drei Zeitintervalle werden entsprechend ähnlich berechnet. Gibt man Anfangsgeschwindigkeit v_1, Endgeschwindigkeit v_2, Zeit T und Weglänge L insgesamt über alle vier Zeitintervalle vor, so ergibt sich der maximale Ruck für die Abschnitte 1 und 2 zu:

$$\ddot{s}_{1,\,2,\,Max}(L,\,T,\,v_1,\,v_2) \;=\; -8 \cdot \frac{v_2 \cdot T + 3 \cdot v_1 \cdot T - 4 \cdot L}{T^3} \tag{7.4}$$

Für die Abschnitte 3 und 4 lautet er:

$$\ddot{s}_{3,\,4,\,Max}(L,\,T,\,v_1,\,v_2) \;=\; 8 \cdot \frac{3 \cdot v_2 \cdot T + v_1 \cdot T - 4 \cdot L}{T^3} \tag{7.5}$$

Soll ein Fahrzeug lediglich von v_1 auf v_2 verzögert werden, so wird die Normalzeit T_N zugrundegelegt. Die Zeitvorgabe T_N für konstante Geschwindigkeit in den Abschnitten 1 und 2 und Abbremsen von v_1 auf v_2 in den Abschnitten 3 und 4 bei gegebener Weglänge L_N berechnet sich mit $\ddot{s}_{1,\,Max} = 0$ nach:

$$T_N \;=\; \frac{4 \cdot L_N}{v_2 + 3 \cdot v_1} \tag{7.6}$$

Eine Umstellung nach L_N ergibt:

$$L_N \;=\; \frac{1}{4} \cdot T_N \cdot (v_2 + 3 \cdot v_1) \tag{7.7}$$

Will man ein Fahrzeug zur Kollisionsvermeidung um den Weg Δs relativ verschieben, so setzt man für Gl. 7.4 und Gl. 7.5:

$$T \;=\; T_N \tag{7.8}$$

$$L \;=\; L_N + \Delta s \;=\; \frac{1}{4} \cdot T_N \cdot (v_2 + 3 \cdot v_1) + \Delta s \tag{7.9}$$

Ein positiver Wert Δs bewirkt so, dass ein Fahrzeug im gegebenen Zeitintervall eine höhere Durchschnittsgeschwindigkeit als die „Norm-Durchschnittsgeschwindigkeit“, die durch die Normalzeit T_N und die Weglänge L_N definiert ist, haben wird.

Das nachfolgende Diagramm zeigt ein Beispiel für zwei im Abstand von *2 s* hintereinander fahrenden Rechtsabbiegern (aus Norden). Für den Vorgänger (Leader-NR) ist ein negativer Wert Δs von *-60 m* angenommen. Um einen Abstand zum Vorausfahrenden von z. B. *23 m* zu halten, wird hier ein negativer Wert Δs von *-70 m* für den Nachfolgenden (Follower-NR) gewählt. Zu beachten ist, dass der Nachfolgende den Beschleunigungsvorgang aufgrund des Kolonnenfahrens mit einer niedrigeren Startgeschwindigkeit v_1 (*ca. 14,5 m/s*) beginnt als der Vorausfahrende (*15,0 m/s*). Für die gegebene Weglänge $L_N = 250\ m$ und *v2 = 5 m/s* ändert sich daher auch der Wert der

Normalzeit T_N für den Nachfolgenden von *20,0 s* auf ca. *20,587 s*:

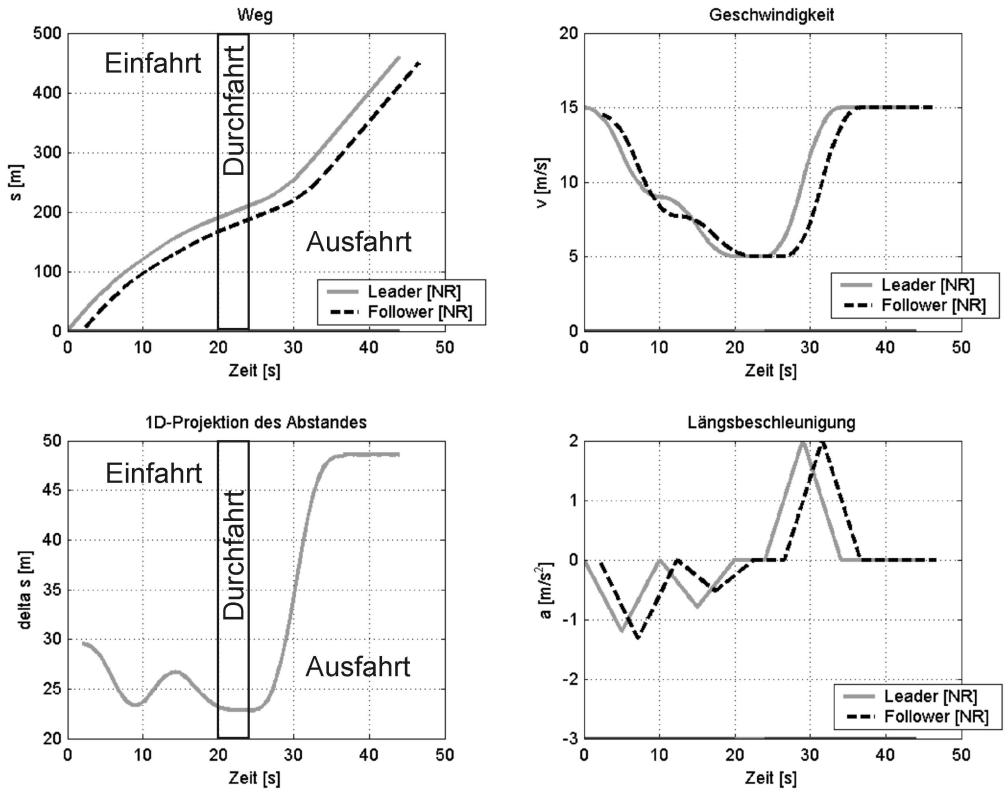

Abbildung 7-20: Fahrzeugkolonne in derselben Einfahrtsspur

Für die Berechnung der maximalen Längsbeschleunigung wird der maximale Ruck in Abhängigkeit von Δs benötigt. Einsetzen von Gl. 7.8 und Gl. 7.9 in Gl. 7.4 liefert:

$$\dddot{s}_{1,\,2,\,Max}(\Delta s,\, T_N) \;=\; \frac{32 \cdot \Delta s}{T_N{}^3} \tag{7.10}$$

Einsetzen von Gl. 7.8 und Gl. 7.9 in Gl. 7.5 liefert:

$$\dddot{s}_{3,\,4,\,Max}(\Delta s,\, T_N,\, v_1,\, v_2) \;=\; \frac{16 \cdot T_N \cdot (v_2 - v_1) - 32 \cdot \Delta s}{T_N{}^3} \tag{7.11}$$

Betrachtet man den Beschleunigungsverlauf innerhalb jedes der 4 äquidistanten Zeitintervalle, so tritt die maximale Längsbeschleunigung aufgrund des intervallweise konstanten Rucks am Ende des 1. und des 3. Intervalls nach der Zeit *0,25* T_N auf:

$$\left|\ddot{s}_{Max}(\Delta s,\, T_N,\, v_1,\, v_2)\right| \;=\; max\left\langle \left|\ddot{s}_{1,\,2,\,Max}(\Delta s,\, T_N)\right|,\, \left|\ddot{s}_{3,\,4,\,Max}(\Delta s,\, T_N,\, v_1,\, v_2)\right| \right\rangle \cdot \frac{T_N}{4} \tag{7.12}$$

7.4.4 Fahrzeugmodelle und Umgebungsmodelle

Die Bahnen, auf denen sich Fahrzeuge bewegen können, sind fest vorgegeben. Für die notwendige geometrische Kollisionsprüfung werden den Fahrzeugen jeweils bei Einfahrt in den Kreuzungsbereich alle Bahnen bekannt gemacht. Die Bahnen sind dabei bezüglich eines Koordinatensystems in der Kreuzungsmitte definiert. Um den Rechenaufwand für die Kollisionsprüfung während der Optimierung auf VMS1-Ebene gering zu halten, werden die Fahrzeuggeometrien durch Kreise approximiert (Abbildung 7-21). Diese Ersatzgeometrien werden den anderen Verkehrsteilnehmern bei Einfahrt in den Kreuzungsbereich ebenfalls mitgeteilt. Die modelltechnische Realisierung der Bahndefinition für die Fahrzeuge erfolgt mit Hilfe von bikubischen Splines:

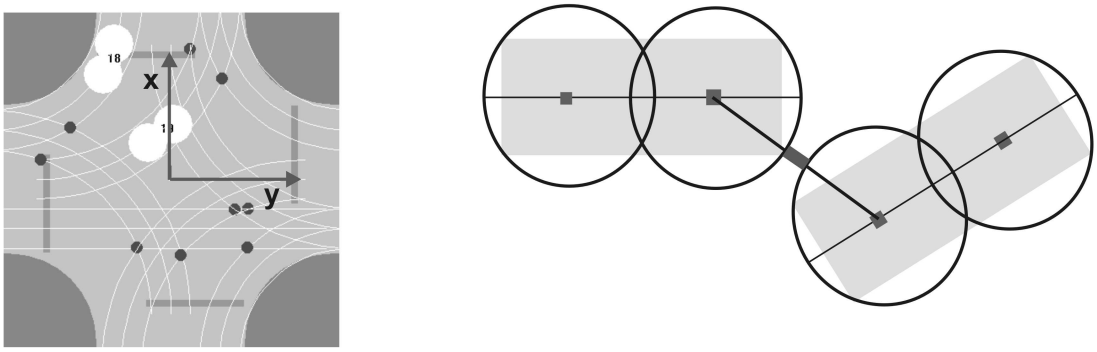

Abbildung 7-21: Effiziente Bestimmung des minimalen Fahrzeugabstands durch Kreisgeometrien

Beim Problem des Kreuzungsmanagements ist zu beachten, dass im Rahmen der Optimierung sehr viele Vorabsimulationen der Handlungszone notwendig sind (s. u.) und dass die Simulation der Fahrzeuge in der Kreuzung der entscheidende Faktor für die Simulationskosten ist. Daher ist es sinnvoll, die Modellierungstiefe der verwendeten Fahrzeugmodelle an die jeweilige Aufgabe anzupassen. Für das Kreuzungsmanagement sind daher insgesamt drei Fahrzeugmodelle unterschiedlicher Komplexität entstanden. Alle Fahrzeugmodelle werden dem Verhalten eines BMW 325i, Baujahr 1988, angepasst, dessen Parameter bekannt waren. Bei den drei dynamischen Modellen handelt es sich um:

- Lineares Einspurmodell: Dieses Modell entspricht weitgehend den aus der Literatur bekannten linearen Einspurmodellen. In das Modell integriert sind lineare Regler für die Fahrzeuggeschwindigkeit $\dot{x}$ und die Orientierung des Geschwindigkeitsvektors (Summe von Gierwinkel ψ und Schwimmwinkel β).
  Kenngrößen des Modells: *3* Freiheitsgrade und Systemordnung *7*.

- Nichtlineares Zweispurmodell: Es besteht aus einer Aufbaumasse und vier Radmassen. Dazwischen wirken nichtlineare Feder/Dämpfer-Gesetze. Die Vorderräder sind lenkbar, wobei die Lenkkinematik realer Fahrzeuge berücksichtigt wird. Das Fahrzeugmodell wird durch Reifenmodelle (erweitertes EZ2Use) und einen Antriebsstrang in Form elektrischer Antriebsmaschinen vervollständigt. Bei der Modellbildung wird auf eine geeignete Strukturierung und

Modularisierung des Modells (s. o.) geachtet. Dadurch werden die Identifikation und die Verifikation der Systemparameter vereinfacht.
Kenngrößen des Modells: *12* Freiheitsgrade und Systemordnung *83*.

- Nichtlineares Einspurmodell: Dieses Modell enthält alle Komponenten des nichtlinearen Zweispurmodells (s. u.). Jedoch wird die mechanische Struktur vereinfacht. Dieser Schritt führt zu einer ca. dreimal schnelleren Simulation auf einem Einzelprozessorsystem, verglichen mit dem Zweispurmodell.
  Kenngrößen des Modells: *8* Freiheitsgrade und Systemordnung *63*.

Eine vollständige Simulationsumgebung für das Kreuzungsmanagement erfordert weitere Komponenten. Dazu zählen beispielsweise die Fahrdynamikregler (Geschwindigkeits-, Spurfolge- und Abstands-/Kolonnenregler), Fahrermodelle für die Bereiche außerhalb der Handlungszone (inkl. Fahrertypklassifizierung für die AMS-Optimierung), die realistische Definition einer Kreuzungsgeometrie und deren Umsetzung in Form von Bahndefinitionen und 3D-Ansichten sowie die Verknüpfung von Fahrzeugposition und -lage mit den Bahndaten.

7.4.5 Lineares Einspurmodell für die Vorausschau

Das lineare Einspurmodell stellt die niedrigste Modellierungstiefe von Fahrzeugen im Kreuzungsmanagement dar. Es wird auf der Basis des klassischen Einspurmodells [Rickert, Schunck 1940] an die eigenen Erfordernisse angepasst. Das Modell ist deshalb sehr wichtig, da es gleichzeitig physikalisch anschaulich und schnell simulierbar ist. Gerade für Simulationen zur Vorausschau auf VMS-Ebene, die schneller als Echtzeit sein müssen, ist das Modell sehr gut geeignet. Durch die Annahme einer linearen Schräglaufsteifigkeit der Reifen wird das reale Fahrverhalten nur bis zu einer Querbeschleunigung von ca. $4\ m/s^2$ hinreichend genau approximiert [Mitschke 1990]. Im Kreuzungsmanagement wird diese Querbeschleunigungsgrenze aber nicht erreicht. Zur Linearisierung des Modells sind weitere Einschränkungen zu beachten. Zum Einen können nur kleine Lenkwinkel δ betrachtet werden, so dass gilt: $sin(\delta) = 0$, $cos(\delta) = 1$ und $tan(\delta) = 0$. Zum Anderen existiert in den Bewegungsgleichungen eine einseitige Verkopplung zwischen der Fahrzeuglängsgeschwindigkeit $\dot{x}$ und dem Schwimmwinkel β (s. Gl. 7.18). Da jedoch für das Kreuzungsmanagement in der Verzögerungszone nur die Längsdynamik und in der Durchfahrtszone (Abbiegen mit $v = const$) nur die Querdynamik eine Rolle spielt, kann die Fahrzeuggeschwindigkeit im Querdynamikteil des Modells ohne Probleme als Parameter eingesetzt werden. Der Parameter v beschreibt damit einen Betriebspunkt zur Betrachtung der Querdynamik.

Bild 7-22 verdeutlicht die Größen zur Beschreibung des Einspurmodells. Dargestellt sind der Momentanpol der Geschwindigkeit, die Antriebskraft F_a, die Reifenquerkräfte F_{qh} und F_{qv}, die

Geschwindigkeit *v*, der Lenkwinkel δ, der Schwimmwinkel β und der Gierwinkel ψ:

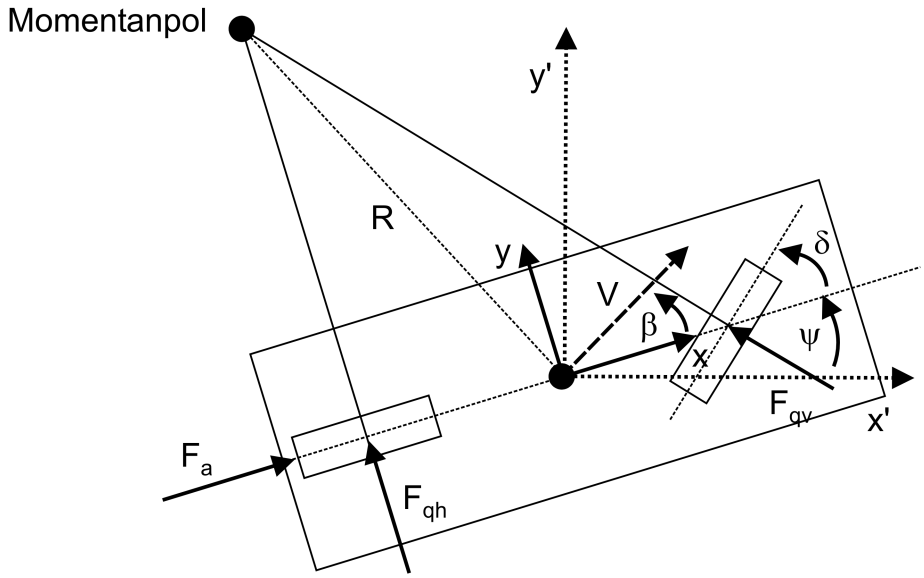

Abbildung 7-22: Einspurmodell

Zur Integration des Modells in das Kreuzungsmanagement müssen entsprechende Regelsysteme für das Fahrzeug verwendet werden. Zum Einen muss der Geschwindigkeitsbetrag in Fahrzeuglängsrichtung eingeregelt werden, zum anderen muss der Richtung der Geschwindigkeit im ortsfesten 2D-Kreuzungskoordinatensystem (*x',y'*) gefolgt werden. Dazu wird der Winkel des Geschwindigkeitsvektors um die Hochachse als Führungsgröße verwendet (Kurswinkel). Für das lineare Einspurmodell ergibt sich der Kurswinkel zu:

$$\gamma = \beta + \psi \tag{7.13}$$

Als Stellgröße dient der Lenkwinkel δ, für den folgendes Regelgesetz aufgestellt wird:

$$\delta = KL \cdot \left(\gamma_{soll} - \frac{KLB}{KL} \cdot \beta - \psi\right) + \frac{KL}{TLI} \cdot \int_0^t (\gamma_{soll} - \beta - \psi) d\tau \tag{7.14}$$

Dieser PI-Regler wird für eine Geschwindigkeit $v_0 = 5{,}0$ *m/s* ausgelegt und wird automatisch über den Parameter $KLB = KL*(v/v_0)$ an den jeweiligen Geschwindigkeitsbetriebspunkt (Parameter *v*) angepasst. D. h. die Reglereigenschaften im Bezug auf den Schwimmwinkel bleiben auch bei Änderung des Parameters *v* durch Kompensation in der ersten Zeile der $\underline{A}$- (s. Gl. 7.18) und $\underline{B}$-Matrix (s. Gl. 7.17) gleich. Über den Integralanteil des PI-Reglers kann der sog. Geschwindigkeitsstellfehler bei rampenförmigem Führungsgrößenverlauf zu Null gemacht werden.

Um den Geschwindigkeitsbetrag zu regeln, wird als Führungsgröße die Antriebskraft F_a am Hinterrad verwendet. Dafür wird folgendes PI-Regelgesetz formuliert:

$$F_a = KV \cdot (v_{soll} - \dot{x}) + \frac{KV}{TVI} \cdot \int_0^t (v_{soll} - \dot{x}) d\tau \tag{7.15}$$

Die Führungsgrößen der Regler werden zu Eingängen in das geregelte lineare Einspurmodell:

TABELLE 17. Eingänge des geregelten linearen Einspurmodells

Name	Einheit	Beschreibung
γ_{Soll}	[rad]	Geschwindigkeitssollwinkel um die Hochachse
v_{Soll}	[m/s]	Geschwindigkeitssollbetrag des Fahrzeugs

Zusätzlich zu fünf Zustandsgrößen zur Beschreibung der Strecke kommen zwei weitere Zustände aus den Integralanteilen der beiden PI-Regler hinzu:

TABELLE 18. Zustände des geregelten linearen Einspurmodells

Name	Einheit	Beschreibung
β	[rad]	Schwimmwinkel
ψ	[rad]	Gierwinkel
$\dot{\psi}$	[rad/s]	Gierrate
x	[m]	Weg
$\dot{x}$	[m/s]	Geschwindigkeit
ε_v	[m]	Integral über den Geschwindigkeitsfehler: $\dot{\varepsilon}_v = v_{soll} - \dot{x}$
ε_ψ	[rad s]	Integral über den Winkelfehler: $\dot{\varepsilon}_\psi = \gamma_{soll} - \beta - \psi$

Das Fahrzeugmodell wird dem Verhalten eines BMW 325i, Baujahr *1988*, angepasst, dessen Parameter bekannt waren. Die Reglerparameter werden in Scilab mit Hilfe des Frequenzkennlinienverfahrens ermittelt und anhand von Eigenwertanalysen und linearer Simulation überprüft.

Die Bandbreiten des geregelten Systems liegen für den Betriebspunkt $v_0 = 5{,}0$ *m/s* oberhalb von *1 Hz*.

TABELLE 19. Parameter des geregelten linearen Einspurmodells

Name	Wert	Einheit	Beschreibung
v	variabel	[m/s]	Geschwindigkeit im zu wählenden Betriebspunkt
v_0	5,00	[m/s]	Geschwindigkeit für die Reglerauslegung
m	1.296,00	[kg]	Fahrzeugersatzmasse (alle Trägheiten)
I_z	1.750,00	[kg m^2]	Fahrzeugträgheitsmoment um die Hochachse
l_v	1,25	[m]	Abstand der Vorderachse vom Schwerpunkt
l_h	1,32	[m]	Abstand der Hinterachse vom Schwerpunkt
c_v	70.000,00	[N/rad]	Gesamtfederkonstante für den vorderen Schräglauf
c_h	70.000,00	[N/rad]	Gesamtfederkonstante für den hinteren Schräglauf
ρ	1,20	[kg/m^3]	Luftdichte
A_w	2,00	[m^2]	Stirnfläche des Fahrzeugs
c_w	0,40	[-]	Luftwiderstandsbeiwert

Name	Wert	Einheit	Beschreibung
KV	8.242,00	[kg/s]	Reglerverstärkung für den Geschwindigkeitsregler
TVI	0,18	[s]	Zeitkonstante für Geschwindigkeits-PI-Regler
KL	6,03	[rad]	Reglerverstärkung für den Lenkregler
KLB	KL*(v/v_0)	[rad]	Reglerverstärkung für den Lenkregler mit v-Kompensation (Reglerkoeffizienten abhängig vom Betriebspunkt *v*)
TLI	0,36	[s]	Zeitkonstante für Gierwinkel-PI-Regler

Für die Herleitung der Bewegungsgleichungen für die Querdynamik des Einspurmodells sei auf die Literatur verwiesen [Mitschke 1990]. Im Längsdynamikteil ist neben der Massenträgheit des Fahrzeugs der Luftwiderstand F_L in linearisierter Form berücksichtigt:

$$F_{L,\,Lin}(\dot{x}) \;=\; \left[\frac{1}{2}\cdot \rho \cdot c_w \cdot A_w \cdot v\right]\cdot \dot{x} \tag{7.16}$$

Die Bewegungs- und Reglergleichungen des gesamten Modells in Zustandsraumdarstellung lauten somit:

$$\begin{bmatrix} \dot{\beta} \\ \dot{\psi} \\ \ddot{\psi} \\ x \\ \dot{x} \\ \varepsilon_v \\ \varepsilon_\psi \end{bmatrix} = \underline{A}\cdot \begin{bmatrix} \beta \\ \psi \\ \dot{\psi} \\ x \\ \dot{x} \\ \varepsilon_v \\ \varepsilon_\psi \end{bmatrix} + \begin{bmatrix} \frac{KLB\cdot c_v}{m\cdot v} & 0 \\ 0 & 0 \\ \frac{KL\cdot c_v\cdot l_v}{I_z} & 0 \\ 0 & 0 \\ 0 & \frac{KV}{m} \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \gamma_{Soll} \\ v_{Soll} \end{bmatrix} \tag{7.17}$$

$$\underline{A} = \begin{bmatrix} \frac{-c_h-(KLB+1)\cdot c_v}{m\cdot v} & \frac{-KLB\cdot c_v}{m\cdot v} & \frac{-m\cdot v - c_v\cdot \frac{l_v}{v} + c_h\cdot \frac{l_h}{v}}{m\cdot v} & 0 & 0 & 0 & \frac{KLB\cdot c_v}{TLI\cdot m\cdot v} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{-(KL+1)\cdot c_v\cdot l_v + c_h\cdot l_h}{I_z} & \frac{-KL\cdot c_v\cdot l_v}{I_z} & \frac{-c_v\cdot \frac{l_v^{\,2}}{v} - c_h\cdot \frac{l_h^{\,2}}{v}}{I_z} & 0 & 0 & 0 & \frac{KL\cdot c_v\cdot l_v}{TLI\cdot I_z} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{-KV-\frac{\rho\cdot c_w\cdot A_w\cdot v}{2}}{m} & \frac{KV}{TVI\cdot m} & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.18}$$

Da für die Optimierung im Kreuzungsmanagement auf VMS-Ebene die Durchfahrtsgeschwindigkeit (Abbiegegeschwindigkeit) ein variabler Parameter ist, werden die Zustandsgleichungen zur Simulation nicht diskretisiert, sondern in symbolischer Form als C-Code formuliert und mit Hilfe eines Heun-Verfahrens online gelöst. Aufgrund der analysierten Eigenfrequenzen ist dabei eine Integrationsschrittweite von $h = 0{,}01\ s$ ausreichend. Die gemessenen Rechenzeiten pro Simulationsschritt gemäß Tabelle 20 haben gezeigt, dass eine verteilte Simulation des linearen

Einspurmodells ineffizient wäre. Daher sind die Simulationen des linearen Einspurmodells direkt in Form von C-Code in einen Evaluator integrierbar. Mit einem Athlon-1800 MP-System kann man das Verhalten des linearen Einspurmodells für eine gedankliche Vorausschau von z. B. *100 s* innerhalb von *100 s/25.000 = 4 ms* voraussimulieren:

TABELLE 20. Rechenzeiten für einen Simulationsschritt des geregelten linearen Einspurmodells

Rechenzeit T	Faktor h/T	Hardware	Numerische Lösung
1,30 [µs]	7.692	Intel PIII 500	Heun-Löser, h = 0,01 s, 64 bit double
0,40 [µs]	25.000	Athlon 1800 MP	Heun-Löser, h = 0,01 s, 64 bit double

7.4.6 Simulator für die VMS0-Ebene

Um die Einzelfahrzeuge simulieren zu können, wird auf die bewährte Simulationsplattform IPANEMA zurückgegriffen. Pro Fahrzeug existiert eine vollständige IPANEMA-Applikation. Die Verwendung von IPANEMA bietet folgende Vorteile:

1. Echtzeitfähigkeit (wird hier nicht ausgenutzt, ist aber eine wichtige Option)

2. Verteilbarkeit durch definierte netzwerkfähige Objektkommunikation

3. Plattformunabhängigkeit

4. Adaptor-Konzept zur Kommunikation mit der Umgebung

5. Anbindung an das Modellbildungswerkzeug CAMeL-View

Innerhalb einer IPANEMA-Applikation kann bei Bedarf mit Hilfe der Calculatoren/Adaptoren auf den MFG-/MFM-Ebenen weiter modularisiert werden. Mit Hilfe der Adaptor-Objekte findet die Vernetzung der Fahrzeuge statt (Modellkopplung). Dazu besitzt der Adaptor ein auf TCP/IP [Stevens 1992] basierendes Kommunikationsmodul zum Senden und Empfangen von Daten:

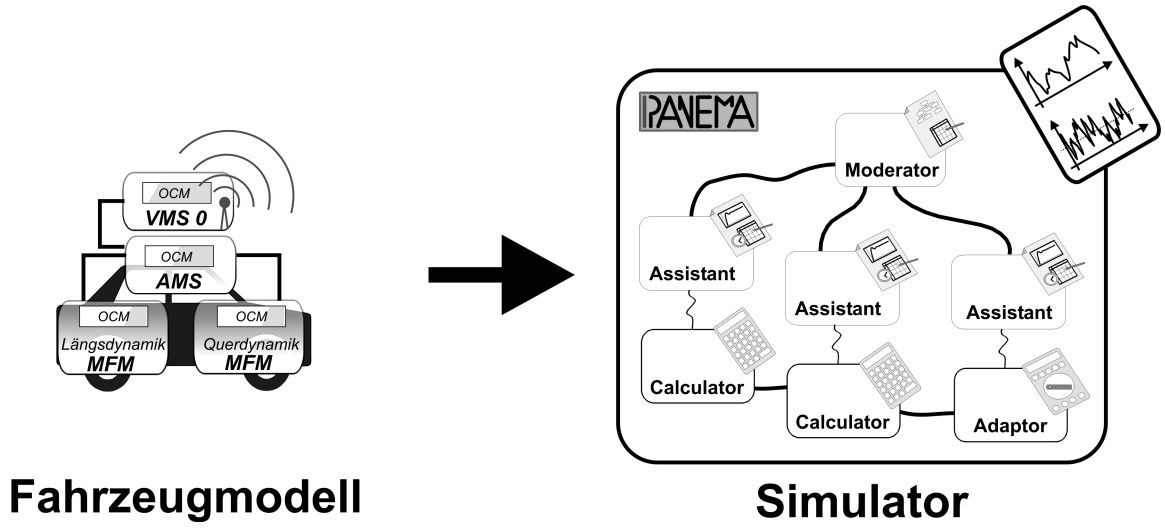

Abbildung 7-23: IPANEMA als Simulator für die VMS0-, AMS- und MFM-Ebenen

7.4.7 Evaluator für die Handlungszone

Zur Bewertung der Vorgänge in der Handlungszone wird ein Evaluator benötigt, der die folgenden Aufgaben erfüllt:

1. Umsetzung des beschriebenen graphenbasierten Verfahrens zur Vorfahrtsbestimmung
2. Anbindung von geeigneten Simulatoren (IPANEMA) zur Darstellung der VMS0-Ebene (dynamische Fahrzeuganzahl)
3. Realisierung von Zielfunktionen zu den Vorgängen in der Handlungszone (Durchsatz, max. Quer-/Längsbeschleunigung, min. Fahrzeugabstand, Energieverbrauch etc.)
4. Bereitstellung einer Schnittstelle zum Optimierer

Ein solcher Evaluator für die Handlungszone lässt sich nicht mit Hilfe von ODSS modellieren; auch das Werkzeug Scilab bietet hier keine Unterstützung an. Daher wird der Evaluator in Form eines handcodierten C-Programms umgesetzt. Da der Evaluator ursprünglich als reiner Simulator für die Handlungszone entstanden ist und erst später erweitert wurde, heißt er iSIM (Intersection Simulator):

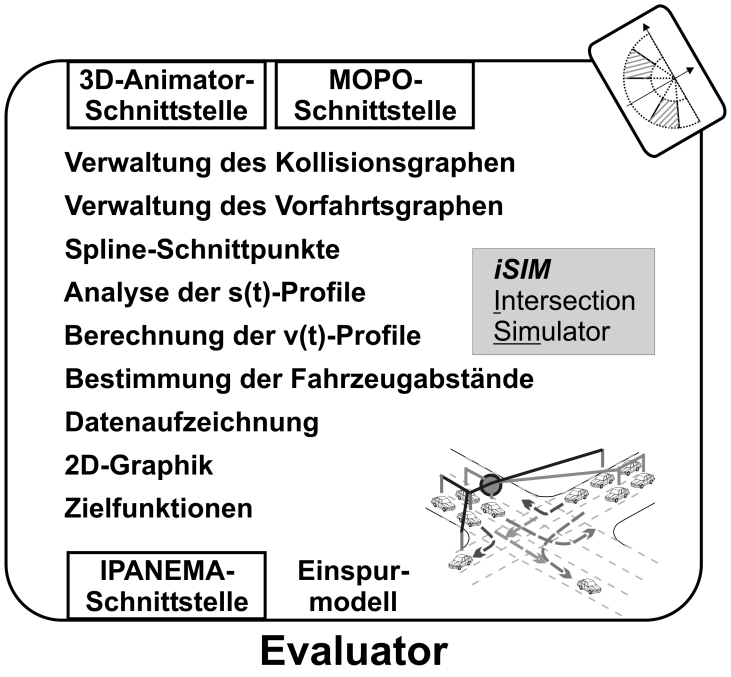

Abbildung 7-24: Evaluator iSIM

Alternativ zur Anbindung von Einzelfahrzeugen über IPANEMA kann auch das beschriebene lineare Einspurmodell zur Simulation verwendet werden. Das lineare Einspurmodell wird dazu direkt in iSIM integriert, da eine Verteilung zu einer zu feingranularen Parallelität führen würde.

Optional können die 3D-Koordinaten (x-, y-, z-Position und Kardanwinkel) aller Fahrzeugkarosserien und Räder über eine TCP/IP-Schnittstelle zyklisch zu einem 3D-Animator gesendet werden. Die Schnittstelle zum Optimierer (MOPO) erfolgt ebenfalls über ein TCP/IP-Protokoll. Dieses Protokoll ist identisch mit dem MOPO-Scilab-Protokoll (siehe Anhang).

iSIM-Simulationsergebnisse

Der iSIM-Kern ist plattformunabhängig; nur die 2D-Visualisierung benötigt eine Windows-Plattform. Unter Linux werden für eine maximale Anzahl von gleichzeitig *25* Fahrzeugen (lineares Einspurmodell) *ca. 7 MB* RAM benötigt. Die Simulation der Handlungszone ist bei Verwendung einer AMD-Athlon 1800-MP-CPU ca. 6-fach schneller als Echtzeit:

TABELLE 21. Gemessene Simulationszeiten der Handlungszone (lin. Einspurmodell auf AMS-Ebene)

Anzahl der CPUs	**Modellzeit [s]**	**Realzeit [s]**	**Modellzeit/Realzeit**
1	100	16	6,25

Die nachfolgende Tabelle zeigt Randbedingungen und Ergebnisse für eine Kreuzungssimulation. Dabei treten immer maximal 8 Fahrzeuge am Rand der Handlungszone (im *10*-Sekunden-Abstand) in den Kollisionsgraphen und den Vorfahrtsgraphen nach einem festen Ablaufschema ein. Auf diese Weise können Simulationen der Handlungszone mit verschiedenen Fahrzeugmodellen verglichen werden.

TABELLE 22. Randbedingungen und Ergebnisse der Simulation

Mittlere Kurvenradien	13,0 m links; 9,0 m rechts
Sicherheitsabstand	3 Fahrzeuglängen
Maximale Verschiebung der Fahrzeuge	[+150 m ; -15 m]
Durchfahrtlänge durch Verzögerungszone	230 m
Zeitlücke (Time Headway [Mayr 2001]) bei der Einfahrt (d. h. bei 8 Einfahrtsspuren ergibt sich ein maximal möglicher Eingangsdurchsatz von 48 Fzg./Min. = 8 Fzg./10 s)	10 s (= 150 m bei 15,0 m/s)
Einfahrts-/Ausfahrtsgeschwindigkeit	15,0 m/s
Durchfahrtsgeschwindigkeit	5,0 m/s
Simulationsdauer (Modellzeit)	100,0 s
Schrittweite der Kreuzungs-Simulation	0,01 s
Kommunikationsschrittweite	0,01 s
Schrittweite Fahrzeugmodelle	0,001 s
Simulator für Fahrzeugmodelle	IPANEMA mit Runge-Kutta4-Löser
Ausgabeschrittweite (2D-Animation)	1,0 s
Fahrzeugmodelle BMW 325i (1988): Lineares Einspurmodell Einspurmodell aus CAMeL-View Zweispurmodell aus CAMeL-View	 Ordnung = 7; DOF = 3 Ordnung = 63; DOF = 8 Ordnung = 83; DOF = 12
Lastverteilung	Gleiche Fahrzeuganzahl pro Prozessor
Simulationsergebnisse	
Maximale Fahrzeuganzahl gleichzeitig simuliert Summe simulierter Fahrzeuge	27 60
Betrag maximale Quer- und Längsbeschleunigung	ca. 3 m/s^2
Durchschnittlicher Durchsatz am Ausgang bei Simulationsende (Simulationsbeginn mit leerer Kreuzung, 25 Fahrzeuge bleiben am Ende der Simulation in der Kreuzung) Durchschnittlicher Durchsatz am Eingang bei Simulationsende	 21 Fahrzeuge/Min.((60-25) Fzg./100 s) 36 Fahrzeuge/Min. (60 Fzg./100 s)

Abbildung 7-25 zeigt die Daten der ersten *16* Fahrzeuge für eine Simulation der Handlungszone

auf der Basis des linearen Einspurmodells. Die Zeitdaten zeigen, dass bei den angegebenen Randbedingungen die Werte der Längs- und der Querbeschleunigungen in physikalisch realistischen Bereichen liegen. Die Abbiegevorgänge finden auf Kreisbahnen statt, was sich anhand der rechteckförmigen Querbeschleunigungsverläufe erkennen lässt. Intern werden jedoch kubische 2D-Splines zur Bahndefinition verwendet, so dass Bahnen sehr flexibel definiert werden können. Die Rechtsabbieger unterliegen wegen der geringen Kurvenradien den höchsten Querbeschleunigungen beim Abbiegen (bei vergleichbaren Geschwindigkeiten). Die drei ersten Fahrzeuge erhöhen ihre Geschwindigkeit vor der Kreuzung kurzfristig, um den Nachfolgenden Platz zu machen; ansonsten vermindern alle Fahrzeuge ihre Geschwindigkeit je nach berechneter Verschiebung zum Kollisionspartner mehr oder weniger schnell auf die Durchfahrtsgeschwindigkeit. Deutlich sind die dreiecksförmigen Beschleunigungsprofile zu erkennen:

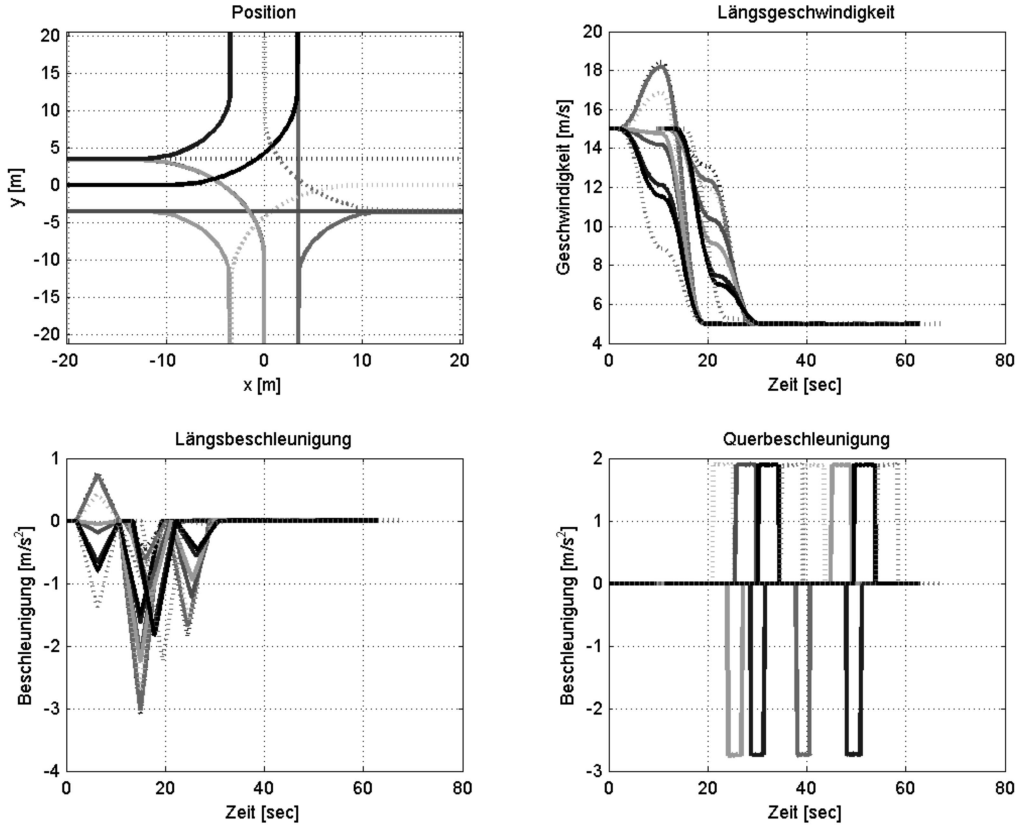

Abbildung 7-25: Simulationsdaten für 16 Fahrzeuge

7.4.8 Fahrzeugprioritäten bei der Vorfahrtsreihenfolge

Die Strukturen des Kollisionsgraphen und des daraus entstehenden Vorfahrtsgraphen hängen von der Reihenfolge der Knoten- bzw. Fahrzeug-Einbindung ab. Für jede Reihenfolge des Einbindens ist zwar eine kollisionsfreie Überquerung der Kreuzung garantiert, jedoch ist die Größe des Rückstaus in den Einfahrtsspuren der Kreuzung vom längsten Pfad im Vorfahrtsgraphen abhängig.

Falls die Reihenfolge des Einbindens in den Kollisionsgraphen völlig frei wählbar wäre, würden

bei *N* Fahrzeugen theoretisch *N!* Möglichkeiten der Vorfahrtsvergabe entstehen, wobei Möglichkeiten durch unabhängige Fahrzeuge und Symmetrie ausscheiden. Für acht Fahrzeuge (S_L, N_L, W_R, E_S, S_S, W_L, N_R, E_L) und eine leere Kreuzung wird exemplarisch eine Eingrößen-Optimierung zur Rückstau-Minimierung mit Hilfe von Simulated Annealing [Van Laarhoven 1987] durchgeführt. Dazu wird eine entsprechende Optimierungsfunktion temporär in iSIM integriert. Eine Verwendung der eigenen MOPO-Optimierungsalgorithmen war nicht möglich, da die Zielfunktion diskontinuierlichen Charakter hat. Die Randbedingungen für die Simulationen entsprechen den bereits beschriebenen, nur dass die Fahrzeuganzahl auf acht Fahrzeuge mit spezieller Eintrittsreihenfolge begrenzt wurde.

Die Optimierung kann in *30* Schritten die maximale Verschiebung Δs (s. Gl. 7.9) zwischen den Fahrzeugen von *54.2 m* auf *20.9 m* minimieren (positive Verschiebung bedeutet verringerte Durchschnittsgeschwindigkeit). Bild 7-26 zeigt die zugehörige Abkühlkurve (Temperatur des Simulated Annealing in Grad Kelvin), den Kostenverlauf (maximale Verschiebung Δs in Meter) und den resultierenden besten Vorfahrtsgraphen. Für die maximale Verschiebung wird für jeden Optimierungsschritt der beste und der aktuelle Wert verglichen. Schon nach 8 Schritten wird die beste Reihenfolge gefunden (und gespeichert). Alle nachfolgenden zufällig gewählten Reihenfolgen erzeugte eine größere maximale Verschiebung. Eine Erhöhung der Durchschnittsgeschwindigkeit (negative maximale Verschiebung) konnte für die gegebene Konfiguration nicht gefunden werden.

Es zeigt sich zunächst ein recht großes Optimierungspotential. Für realistische Simulationsszenarien mit einer vollen Kreuzung (>30 Fahrzeuge), in die am Rand auf den Einfahrtsspuren Fahrzeuge einfahren, zeigt sich jedoch nur ein sehr geringes Optimierungspotential. Es ist daher am sinnvollsten die Fahrzeuge nach Reihenfolge des Eintretens/Anmeldens in den Vorfahrtsgraphen einzubinden (First-In-First-Get-Strategie). Dies wurde durch umfangreiche Simulationsläufe mit kontinuierlich einfahrenden Fahrzeugen geprüft (Simulationskonfiguration wie Kap. 7.4.7) und hat zwei Gründe:

1. Will man eine optimale Vorfahrtsreihenfolge für neu in eine Kreuzung einfahrende Fahrzeuge bei hoher Verkehrsdichte ermitteln, so schränkt ein schon bestehender Rückstau in den Einfahrtsspuren das Optimierungspotential drastisch ein, falls er durch die Limitierung der Abbiegegeschwindigkeit (Querbeschleunigungsgrenze) dominiert wird.

2. Die Anzahl der "neuen" Fahrzeuge mit noch unbestimmter Vorfahrtsreihenfolge ist klein im Vergleich mit den vorhandenen Fahrzeugen im Inneren der Kreuzung mit schon festgelegter Vorfahrtsreihenfolge.

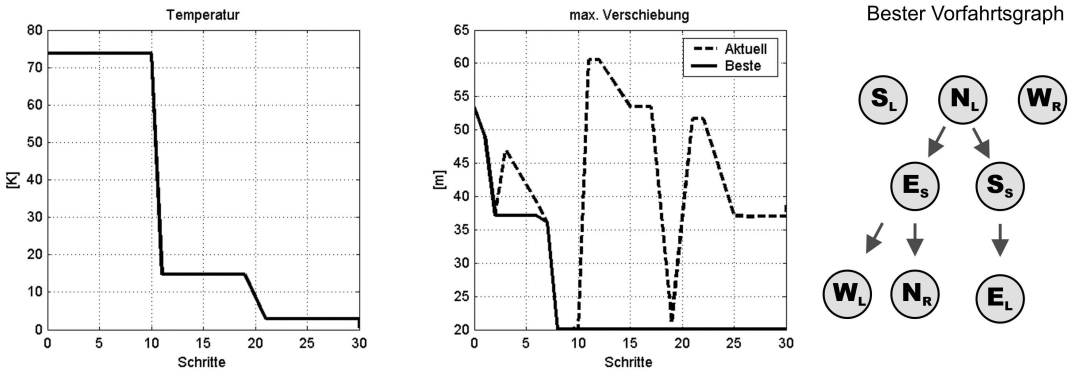

Abbildung 7-26: Minimierung des Rückstaus für 8 Fahrzeuge

7.4.9 Parallelverarbeitung für die Bewertung der Handlungszone

Will man detailliertere Bewertungen zur Längs-, Quer- und Vertikaldynamik der Einzelfahrzeuge bei der Überquerung der Kreuzung vornehmen, so ist die Verwendung der komplexeren nichtlinearen Fahrzeugmodelle notwendig. Diese Modelle benötigen ein Vielfaches des Rechenaufwandes für die Simulation im Vergleich zum linearen Einspurmodell. Da die Evaluierung der Handlungszone während eines Optimierungslaufs sehr häufig vorgenommen wird, ist es sinnvoll, eine parallele Simulation der Einzelfahrzeuge zu realisieren. Im Folgenden werden die Laufzeiten für die nichtlinearen Modelle auf einem Workstation-Cluster mit zehn CPUs dargestellt.

Der Workstation-Cluster besteht aus fünf Dualprozessor-Systemen mit AMD-Athlon 1800-MP-CPUs. Die Vernetzung erfolgt über einen Gigabit-Switch. Ein System (zwei CPUs, *2 GB* RAM) läuft unter Windows 2000 und dient als Leitwarte (2D-/3D-Animation etc.). Die vier Rechnerknoten (8 CPUs, je *1 GB* RAM) laufen unter Suse-Linux 8.0. Der Evaluator-Prozess iSIM läuft auf dem Windows-Knoten und startet von dort aus die Simulatoren auf den Linux-Knoten über RSH-Kommandos (Remote Shell); die Interprozess-Kommunikation erfolgt durch TCP/IP-Sokkets.

Die Lastverteilung folgt einem sehr einfachen Schema, da die Lasten sowie die Rechnerknoten homogen sind. Zudem stehen die Rechnerknoten exklusiv für diese Anwendung zur Verfügung, so dass keine externen Lasten zu erwarten sind. Zur Lastverteilung sorgt iSIM daher lediglich dafür, dass die Anzahl der Simulatoren auf jedem Rechnerknoten möglichst gleich ist. Beim Start eines neuen Simulators wird dazu aus einer Liste der Rechnerknoten der Knoten mit der momentan geringsten Simulator-Anzahl ausgewählt. Bei gleichen Simulator-Anzahlen auf mehreren oder allen Rechnerknoten erfolgt eine Zufallsentscheidung.

Die Anwendungstopologie wird durch einen hierarchischen Evaluator-Simulator-Graph beschrieben. Dieser Graph hat eine dynamische Struktur, da die Anzahl der Simulatoren der aktuellen Fahrzeuganzahl in der Handlungszone entspricht und deshalb zeitlich veränderlich ist. Für die oben beschriebene Simulation (Tabelle 22) treten minimal *8* und maximal *27* Fahrzeuge

gleichzeitig auf:

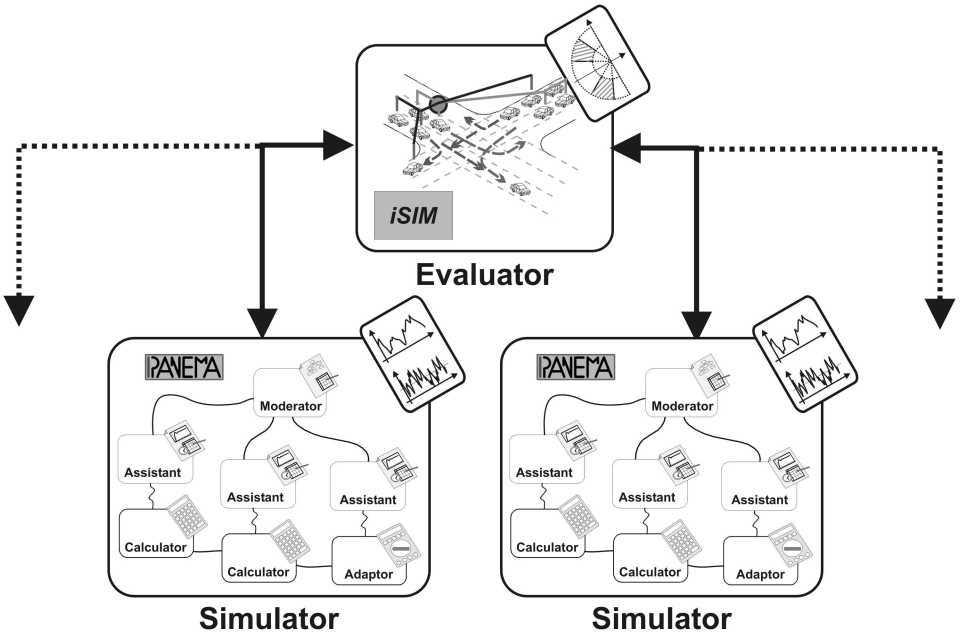

Abbildung 7-27: Dynamischer Evaluator-Simulator-Graph

Die Bewertung der Handlungszone wird für das nichtlineare Einspur- und Zweispurmodell in verschiedenen CPU-Konfigurationen der Linux-Rechenknoten durchgeführt. Zur Referenzsimulation für die nachfolgende Ermittlung eines Speedup-Faktors wird zunächst ein Linux-Rechenknoten in Single-CPU-Konfiguration zusammen mit dem Leitwartenrechner verwendet. Anschließend erfolgen die Simulationen der beiden Fahrzeugmodelle für 2, 4, 5 und 8 CPUs. Das nachfolgende Diagramm zeigt die zugehörigen Rechenzeiten und den Speedup für die jeweilige CPU-Anzahl:

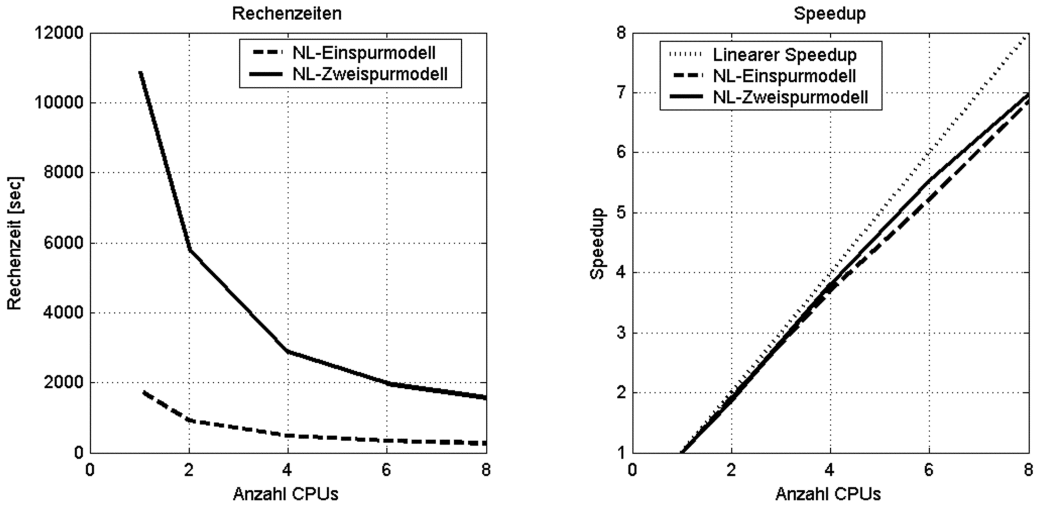

Abbildung 7-28: Laufzeitergebnisse für die parallele Simulation von Fahrzeugen

Die Ergebnisse zeigen die gute Skalierbarkeit der Anwendung. Trotz verbleibender sequentieller Abläufe im Evaluator (Bahnberechnung, Kollisionsprüfung, Vorfahrtsbestimmung etc.) können die Laufzeiten durch eine Parallelverarbeitung drastisch gesenkt werden. Auch die Zeitverluste durch Kommunikation und das dynamische Starten und Beenden der IPANEMA-Anwendungen

haben geringen Einfluss auf die parallele Performance.

Nach dem Gesetz von Amdahl [Amdahl 1967] ergibt sich für den gemessenen Speedup von *6,8* bei 8 Prozessoren der parallele Anteil am Gesamtcode zu *0,97*. Extrapoliert man mit diesen Werten auf eine Anwendung mit 27 Prozessoren (ein Prozessor pro Fahrzeug), so ließe sich theoretisch ein Speedup von *15,1* erzielen. Angesichts dieser Schätzung wäre die Verwendung von noch mehr Rechenknoten sehr sinnvoll.

7.4.10 Modellbasierte Online-Optimierung auf VMS1-Ebene

In den vorangegangenen Abschnitten wurden die Verfahren zur effizienten Simulation und Bewertung der Vorgänge in der Handlungszone beschrieben. Auf dieser Grundlage bauen die Optimierungsexperimente auf VMS1-Ebene auf. Die Optimierung basiert auf dem MOPO-Verfahren. Die nachfolgende Abbildung zeigt die Optimierer-Evaluator-Simulator-Topologie für das Kreuzungsmanagement. Die Parallelität auf VMS1-Ebene (dynamische Simulatoranzahl) und diejenige auf Optimierungsebene (mehrere Evaluator-Instanzen) sind bereits mitdargestellt. Ein Fahrzeug übernimmt die Optimierung, da von diesem immer nur eine Instanz zu einem Zeitpunkt aktiv ist. Ein Verteilungsvorschlag für die Evaluator-Simulator-Funktionen ist ebenfalls skizziert. Die parallele OES-Struktur zur modellbasierten Online-Optimierung findet sich innerhalb der gekoppelten Operator-Controller-Module auf VMS1-Ebene der Einzelfahrzeuge. Die weiteren OCMs auf VMS0-, AMS- und MFM-Ebene sind ebenfalls angedeutet:

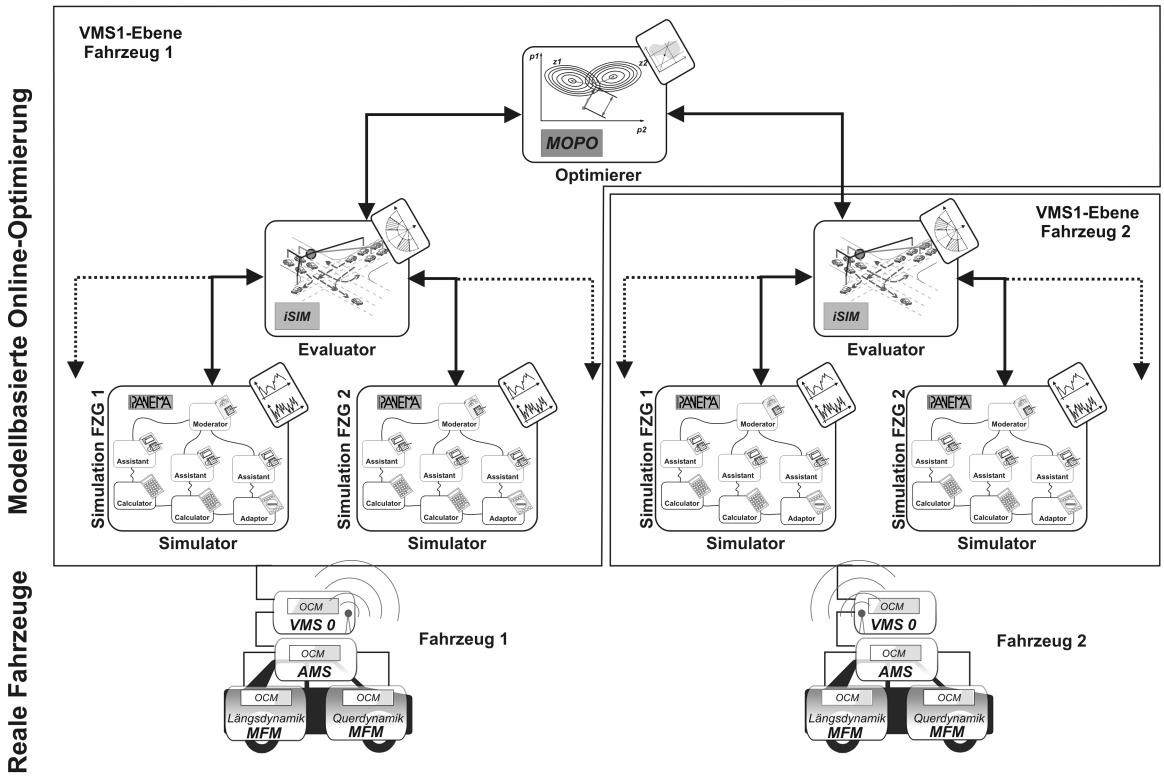

Abbildung 7-29: OES-Struktur mit Parallelität auf Optimierungs- und VMS1-Ebene

Ein sehr wichtiges kontinuierliches Mehrziel-Problem in der Optimierungszone auf VMS1-Ebene besteht im Auffinden der besten Parameter für die zeitlichen Abstände (Time Headway [Mayr 2001]) und die Geschwindigkeiten für die Verzögerungs- und die Durchfahrtszone: v_{Ein-}

$_{fahrt}$, $v_{Durchfahrt}$, $th_{Einfahrt}$, $th_{Durchfahrt}$. Durch diese Größen kann der Durchsatz in der Durchfahrtszone optimiert und gleichzeitig der dazu passende optimale Eingangsdurchsatz in die Verzögerungszone gefunden werden. Für die Mehrziel-Optimierung dieses Problems werden sechs Zielfunktionen verwendet:

TABELLE 23. Ziele für die Mehrziel-Optimierung auf VMS1-Ebene

Ziel	Beschreibung
Maximierung der minimalen Geschwindigkeit	Die minimale Geschwindigkeit in der Verzögerungszone sollte nicht unter der Abbiegegeschwindigkeit ($v_{Durchfahrt}$) liegen
Maximierung des Gesamtdurchsatzes	Die Fahrzeugfrequenz auf allen vier Ausfahrtsspuren der Handlungszone zusammen (Eingangsdurchsatz bezieht sich auf acht Spuren)
Minimierung der Energie	Kinetische Energieänderung aller Fahrzeuge zum Bremsen und zum Beschleunigen
Minimierung der maximalen Verschiebung Δs	Größe der Verschiebung zur Kollisionsvermeidung; geringe Verschiebung bedeutet wenig Rückstau
Minimierung der maximalen Querbeschleunigung	Tritt beim Rechtsabbiegen auf
Minimierung der maximalen Längsbeschleunigung	Tritt bei Verschiebung der Fahrzeuge in der Verzögerungszone auf

Die Minimierung der maximalen Fahrzeugverschiebung Δs als Maß für den Rückstau ist eine sehr wichtige Größe für das Kreuzungsmanagement. Anhand dieser Größe kann beobachtet werden, ob eine stillstandsfreie Überquerung der Kreuzung möglich ist. Ist der Eingangsdurchsatz höher als der Durchfahrtsdurchsatz, so wird Δs im Laufe der Evaluierungszeit monoton steigen, ansonsten stellt sich ein oberer Grenzwert ein. Sobald die minimale Geschwindigkeit in der Verzögerungszone einen konfigurierbaren Grenzwert (Voreinstellung: *1 %* von $v_{Einfahrt}$) unterschritten hat, wird die Evaluierung der Handlungszone beendet. Als Zielfunktionswerte werden die um *100 %* verschlechterten Werte des vorherigen Optimierungsschrittes zurückgegeben.

Gemäß der Grundidee des Kreuzungsmanagements, dass die beteiligten Kraftfahrzeuge selbst die erforderliche Rechenleistung in Form eines Prozessornetzwerkes bereitstellen, ist die Optimierung als verteilte Anwendung umgesetzt. Verwendet wird das beschriebene Quasi-Newton-Verfahren. Es können bei den vier beschriebenen Optimierungsparametern die vier Zeilen der Empfindlichkeitsmatrix $\underline{S}$ (vgl. Kap. 6.6) parallel durch vier Evaluator-Instanzen berechnet werden. Es ergibt sich eine Topologie für die verteilte Anwendung mit bis zu *177* $(1 + 4 + 4 \cdot 43)$ Prozessen für die oben beschriebenen Simulationsrandbedingungen (für iSim): Ein Optimierer (MOPO), vier Evaluatoren (iSIM) und bis zu *43* Simulatoren (IPANEMA) pro Evaluator. Die hohe Anzahl von Fahrzeugen (vormals *27*) ist durch die Vergrößerung der Durchfahrtslänge durch die Verzögerungszone von *230 m* auf *350 m* bedingt (bewirkt geringere Längsbeschleunigungen).

7.4.11 Optimierungsergebnisse

Die Optimierung wird sowohl mit dem nichtlinearen Einspur- als auch mit dem Zweispurmodell durchgeführt, um die Ergebnisse abzusichern und das Laufzeitverhalten zu vergleichen. Mit Hilfe des Quasi-Newton-Verfahrens ergeben sich identische Optimierungsergebnisse nach *137*

Schritten für beide Modellierungstiefen. Die nachfolgenden Diagramme zeigen den zugehörigen Optimierungsverlauf für die Zielgrößen und die Parameter:

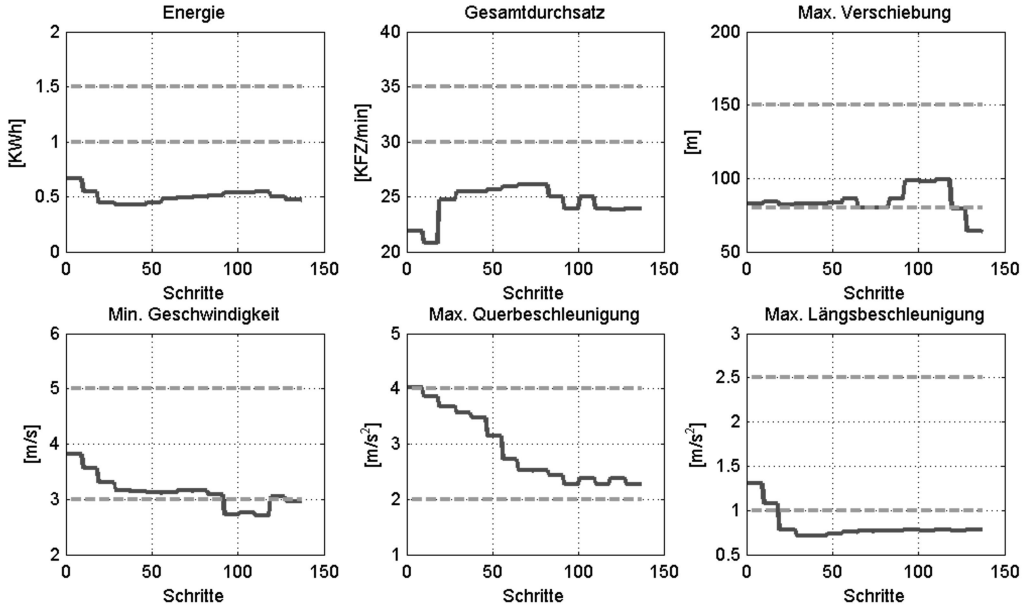

Abbildung 7-30: Zielgrößen-Verläufe während der Optimierung

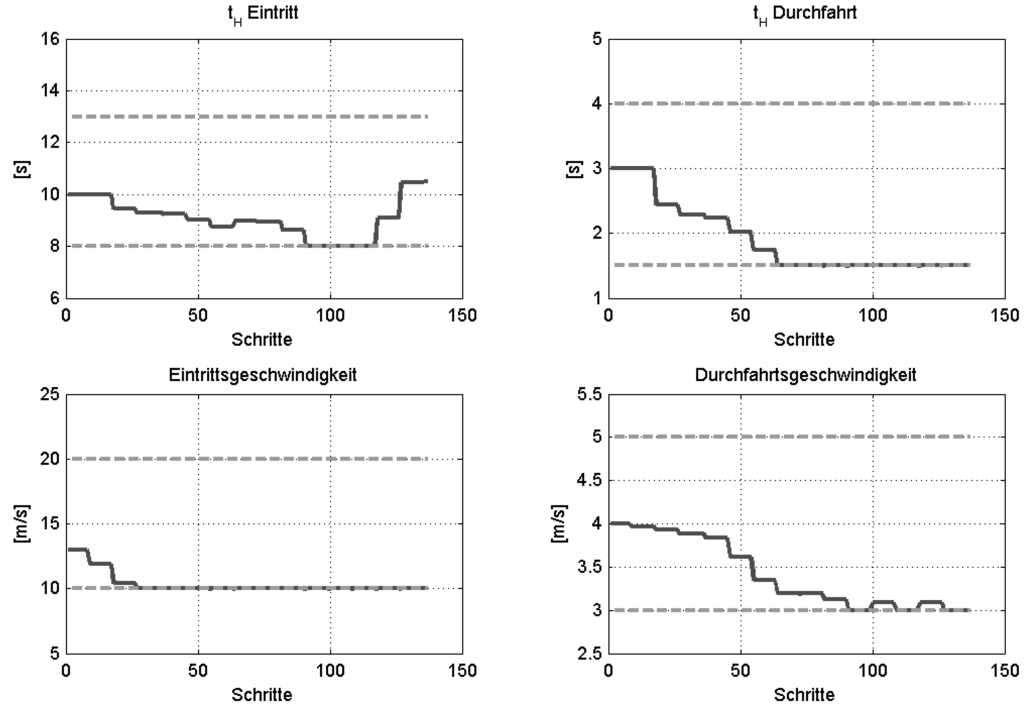

Abbildung 7-31: Parameter-Verläufe während der Optimierung

Relevante Verbesserungen können für die maximale Querbeschleunigung, die maximale Längsbeschleunigung und die maximale Fahrzeugverschiebung erreicht werden. Das Optimierungsergebnis ist speziell auf die jeweilige Vorfahrts-/Abbiegesituation zugeschnitten; neue Situationen

erfordern eine erneute Optimierung. Insbesondere der Kolonnenabstand in der Durchfahrtszone und die Abbiege-/Durchfahrtsgeschwindigkeit haben sehr großen Einfluss auf alle Zielgrößen. Im Optimierungsverlauf erreichen daher beide Parameter ihre unteren Grenzen.

Für die Optimierung wird der bereits beschriebene Workstation-Cluster verwendet. Zwei Prozessoren werden für den MOPO-Prozess und für Visualisierungen genutzt, auf die restlichen acht Prozessoren werden die iSIM- und die IPANEMA-Prozesse gleichmäßig verteilt. Die Laufzeiten und Speedups ergeben sich wie folgt:

TABELLE 24. Rechenzeiten und Speedups für die parallele Optimierung

Modell	Rechenzeit 1 CPU [min]	Rechenzeit 8 CPUs [min]	Speedup
NL-Einspur	6.099	890	6,8
NL-Zweispur	(> 35.000)[a]	5.356	(6,8)[b]

a. Hochgerechnet auf Basis des angenommenen Speedups

b. Annahme des Speedups des NL-Einspurmodells zur Schätzung der sequentiellen Rechenzeit

Der Speedup für die Verwendung von 8 CPUs entspricht demjenigen, der für die reine Simulation (ein Bewertungslauf) beobachtet wird. Dies ist nicht überraschend, da die Bewertung der Handlungszone den dominierenden Rechenanteil bildet. Die Übereinstimmung im Speedup zeigt dabei die Effizienz der Implementierung auf. Trotz der hohen Prozessanzahl und den daraus resultierenden Startup- und Kommunikationszeiten können die 8 CPUs sehr gut ausgenutzt werden. Die hohe Anzahl der verteilten Prozesse lässt vermuten, dass auch wesentlich größere Rechencluster (>100 CPUs) effizient genutzt werden können.

7.4.12 Zusammenfassung

Anhand der hierarchischen Struktur, der Zoneneinteilung, der Fahrzeug- und Umgebungsmodelle und der Algorithmen zur Kollisionsvermeidung/Vorfahrtsbestimmung des Kreuzungsmanagements wurde zunächst die Bewertung der Handlungszone vorgestellt. Die Ergebnisse unter Verwendung der komplexen Modelle haben gezeigt, dass die Verwendung des linearen Einspurmodells bereits Ergebnisse mit guter Genauigkeit liefert. Weitergehende Aussagen zu Quer-, Längs- und Vertikaldynamik der Fahrzeuge während der Überquerung sind jedoch nur mit den komplexeren Modellen zu erzielen. Für verschiedene Modellierungstiefen wurden zusätzlich die Laufzeitergebnisse für eine parallele Simulation der Fahrzeuge verglichen.

Für die Bestimmung einer Vorfahrtsreihenfolge hat sich gezeigt, dass für realistische Szenarien, bei denen einzelne Fahrzeuge zeitverschoben in eine bereits "volle" Kreuzung einfahren, die First-In-First-Get-Strategie gut geeignet ist.

Für die verteilte Optimierung auf VMS1-Ebene wurde die zugehörige Optimierer-Evaluator-Simulator-Struktur vorgestellt. Exemplarisch wurde ein Optimierungsexperiment zur Abstimmung der Fahrzeugabstände und -geschwindigkeiten für die Handlungszone beschrieben.

Für das Kreuzungsmanagement hat sich gezeigt, dass nur bei ausreichend großen Zonen und geeigneten Fahrzeugabständen überhaupt genügend Flexibilität für die Optimierung des Verkehrsflusses vorhanden ist. Der Verkehrsablauf enthält eine Neigung zur Selbsthemmung, wenn

die Zahl der in die Kreuzung einfahrenden Fahrzeuge größer ist als die Zahl der Fahrzeuge, die die Kreuzung verlassen [Cremer 1979]. Dies ist ein entscheidendes Charakteristikum, das den Verkehrsfluss von anderen Verteilprozessen mit beschränkter Transportkapazität unterscheidet: Wenn einmal ein gewisser Grenzwert der Verkehrsdichte überschritten wird, kommt es zum Zusammenbruch des Prozesses, wobei die sich dann einstellende Verkehrsstärke weit unter der maximal möglichen liegen wird.

8 Zusammenfassung und Ausblick

Entstanden ist die Arbeit im Rahmen des DFG-Sonderforschungsbereichs 376 "Massive Parallelität - Algorithmen, Methoden, Anwendungen" im Teilprojekt C1. Hier wird mit dem selbstorganisierenden Kreuzungsmanagement ein hochkomplexes Anwendungsbeispiel bearbeitet, das die Bereiche Online-(Echtzeit-) und Mehrziel-Optimierung mit dem Kontext massiver Parallelität und dem autonomen mechatronischen Fahren verbindet.

In der vorliegenden Arbeit wurde ein Konzept für die verteilte und echtzeitfähige Mehrziel-Optimierung in mechatronischen Systemen vorgestellt. Mit Hilfe der neuen Definition von Optimierer-, Evaluator- (Bewerter-) und Simulator-Funktionen (OES-Struktur genannt) lassen sich hierarchisch organisierte Anwendungen der Mehrziel-Optimierung realisieren, die mit der modular-hierarchischen Struktur mechatronischer Systeme kompatibel sind. Das Konzept umfasst die Definition des notwendigen Echtzeitverhaltens und von zwei Ebenen für die Parallelverarbeitung. Aufgrund der Erfahrungen mit der verteilten Echtzeitsimulation von eng verwobenen Systemmodellen basiert die Verteilung auf der grob granularen OES-Struktur, von der gezeigt werden konnte (Beispiel Kreuzungsmanagement), dass eine effiziente Parallelverarbeitung daraus resultiert.

Zur Umsetzung der in der Arbeit beschriebenen Anwendungsbeispiele, die zur Überprüfung des Konzepts für die verteilte und echtzeitfähige Mehrziel-Optimierung dienen, mussten die MOPO-Gradienten- und Quasi-Newton-Verfahren erweitert werden: Für das Gradientenverfahren ist eine Echtzeit-Variante realisiert worden. Für Gradienten- und Quasi-Newton-Verfahren ist nun die vollständig parallele Auswertung der Zielfunktionen möglich.

Ein Anwendungsbeispiel ist die echtzeitfähige Implementierung einer optimierenden Regelung am HILS-Prüfstand für ein aktives Feder-Neige-Fahrwerksmodul für ein Bahnshuttle (Neue Bahntechnik Paderborn). Es ist dort gelungen, eine modellgestützte Überwachung und Bewertung im Zeit- und im Frequenzbereich zusammen mit dem MOPO-Gradientenverfahren zur Mehrziel-Optimierung in die Echtzeitschleife der Prüfstandsinformationsverarbeitung zu integrieren. Dafür wurde die verteilte Simulationsplattform IPANEMA um Multitasking-Verarbeitung erweitert.

Am Anwendungsbeispiel Kreuzungsmanagement wurde die Verfeinerung und Weiterentwicklung des vorherigen Standes in Bezug auf Modelle, Verfahren zur Vorfahrtsbestimmung und verteilte Mehrziel-Optimierung aufgezeigt. Wichtige Veränderungen gegenüber dem bisherigen Verfahren sind die Erweiterung auf zwei Einfahrtsspuren (Linksabbieger und Rechts-/ Geradeausspur), eine graphenbasierte Vorfahrtsbestimmung und die Nutzung von Parallelverarbeitung. Für die Optimierung wurde eine parallele Offline-Verarbeitung auf vernetzten Workstations mit Hilfe der OES-Struktur (Optimierer-Evaluator-Simulator) umgesetzt. Das Besondere hierbei ist die dynamische Struktur der Anwendung, die sich in einer zeitlich veränderlichen Anzahl von Simulatoren äußert. Die verwendeten 8 Prozessoren konnten sehr gut ausgenutzt werden (Speedup *6,8*) und auch wesentlich größere Rechencluster (*>100* Prozessoren) könnten effizient genutzt werden.

Im Bereich der Echtzeitverarbeitung ist es sinnvoll, zukünftig auch eine echtzeitfähige Variante des vorgestellten Quasi-Newton-Verfahrens zu entwickeln. Dies kann in Anlehnung an das Vorgehen zur Realisierung des echtzeitfähigen Gradientenverfahrens erfolgen. Als wesentlich aufwendiger einzuschätzen ist jedoch die zukünftige Umsetzung einer Mehrziel-Optimierung (z. B. im Rahmen der Neuen Bahntechnik Paderborn), die gleichzeitig massiv-parallel und echtzeitfähig ist.

Für die Implementierung von Optimierungsverfahren, die aus mehreren **gleichzeitig** aktiven Optimierern bestehen, ist weitere Forschungsarbeit notwendig. Dies betrifft die Bereiche der hierarchischen Verknüpfung von Optimierungsexperimenten auf den AMS-/MFM-Ebenen und der kooperativ organisierten Optimierung auf der VMS-Ebene. Die gleichzeitige Aktivierung von mehreren lose verknüpften Optimierungsexperimenten führt dann zukünftig zu Anwendungen, die noch hochgradiger verteilt und parallel sein werden, als es bisher schon der Fall ist.

9 Anhang A: Grundlagen zur Systembewertung

9.1 Wichtige Analyseverfahren für lineare Systeme

Ausgangspunkt sind die zeitinvarianten linearen Systeme (LZI) in Zustandsraumdarstellung [Ludyk 1990] mit den Eingängen $\underline{u}(t)$, den Ausgängen $\underline{y}(t)$ und den Zuständen $\underline{x}(t)$:

$$\dot{\underline{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{B} \cdot \underline{u}(t) \tag{9.1}$$

$$\underline{y}(t) = \underline{C} \cdot \underline{x}(t) + \underline{D} \cdot \underline{u}(t) \tag{9.2}$$

Bestimmung der Eigenwerte

Jeder quadratischen Matrix $\underline{A}$ ist ein charakteristisches Polynom $P(s)$ zugeordnet:

$$P(s) = det(s \cdot \underline{I} - \underline{A}) \tag{9.3}$$

Die Nullstellen $\lambda_1, \ldots, \lambda_n$ dieses Polynoms bezeichnet man als *Eigenwerte* von $\underline{A}$. Sie liefern wichtige Aussagen zu Stabilität, Dämpfung und Zeitkonstanten des Schwingungsverhaltens des linearen Systems.

Eigenwertbereiche

Erfahrungsgemäß ist es besser, Eigenwerte nicht in feste Lagen zu zwingen, sondern ihnen lediglich gewisse Bereiche der komplexen Ebene vorzuschreiben. So wird die Gefahr verringert, die Strecke zu einem ihrer Natur stark widersprechenden Verhalten zu zwingen, was sich durch hohe Stellbeträge, starke Anfangspendelungen und dergleichen rächen wird [Föllinger 1994].

Ein konjugiert komplexes Eigenwertpaar λ, λ' sollte in einem symmetrischen Paar von Kreisringsektoren liegen, wodurch Dämpfung und Abklingzeit der zugehörigen Schwingung innerhalb vorgegebener Schranken liegen:

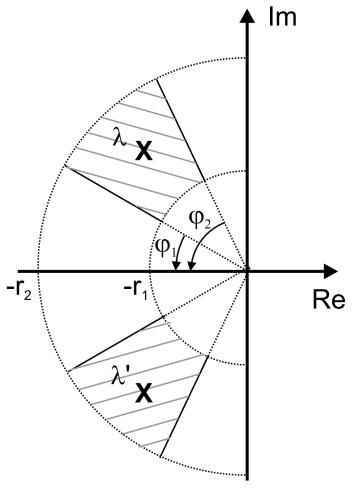

Abbildung 9-1: Eigenwertbereiche

Frequenzgangsrechnung

Für stabile eingeschwungene Systeme (abgeklungenes Eigenverhalten) lässt sich das Störverhalten $\underline{x}_p(t)$ mit Hilfe der Frequenzgangsmatrix beschreiben:

$$\underline{x}_p(t) = \underline{F}_x(j\omega) \cdot \underline{u}(t) = \underline{F}_x(j\omega) \cdot \underline{u}_f \cdot e^{j\omega t} \tag{9.4}$$

Aus der partikulären Lösung der Zustandsgleichung ergibt sich die Frequenzgangsmatrix (siehe auch [Kwakernaak, Sivan 1972]) zu:

$$\underline{F}_x(j\omega) = (j\omega \cdot \underline{I} - \underline{A})^{-1} \cdot \underline{B} \tag{9.5}$$

Damit lässt sich der Zusammenhang zwischen $\underline{u}(t)$ und $\underline{y}(t)$ darstellen:

$$\underline{F}_y(j\omega) = \underline{C} \cdot \underline{F}_x(j\omega) + \underline{D} \tag{9.6}$$

Kovarianzanalyse

Die Varianzen der Ausgangsgrößen eines linearen Systems (Diagonalelemente der Kovarianzmatrix) sind ein Maß für die Leistungsübertragung eines Systems. Es kann z. B. ermittelt werden, wie stark ein Ausgang oder Zustand auf einen Eingangspegel reagiert. Mit Hilfe der Streuungen sind Aussagen über die maximal auftretenden Amplituden möglich. Die Bestimmung der Kovarianzmatrix kann im Zeit- oder im Frequenzbereich erfolgen. Bei einer Betrachtung im Modalraum können sogar mittlere Leistungen modalen Eigenwerten zugeordnet werden [Lückel, Kasper 1981]. Hier wird von einem allgemeinen LZI-System ausgegangen, dessen Eingänge durch einen stochastischen Vektorprozess $\underline{w}(t)$ definiert sind:

$$\dot{\underline{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{B} \cdot \underline{w}(t) \tag{9.7}$$

$$\underline{x}(t_0) = \underline{x}_0 \tag{9.8}$$

Dabei besteht $\underline{w}(t)$ aus skalaren stochastischen Prozessen, die mittelwertfrei sind:

$$\underline{w}(t) = \left[w_1(t) \; w_2(t) \; \ldots \; w_n(t) \right]^T \tag{9.9}$$

$$E\{\underline{w}(t)\} = \underline{0} \tag{9.10}$$

Die Kovarianzmatrix wird aus dem Erwartungswert des Produkts der beiden gegeneinander zeitverschobenen stochastischen Vektorprozesse $\underline{w}(t_1)$ und $\underline{w}(t_2)$ mit positiv definiter Intensität $\underline{V}$ gebildet [Kwakernaak, Sivan 1972]:

$$\underline{R}_u(t_1, t_2) = E\{\underline{w}(t_1) \cdot \underline{w}(t_2)^T\} = \underline{V} \cdot \delta(t_1 - t_2) \tag{9.11}$$

Die Varianzmatrix $\underline{Q}$ ergibt sich aus der Kovarianzmatrix für die Zeitverschiebung 0 zwischen $\underline{w}(t_1)$ und $\underline{w}(t_2)$:

$$\underline{Q}_u(t) = \underline{R}_u(t, t) \tag{9.12}$$

Die Varianzen der Systemzustände im Zeitbereich lauten definitionsgemäß:

$$\underline{Q}_x(t) = E\{\underline{x}(t) \cdot \underline{x}(t)^T\} \tag{9.13}$$

Nach Gl. 5.7 und Gl. 9.13 ergibt sich folgende Darstellung:

$$\underline{Q}_x(t) = e^{\underline{A} \cdot t} \cdot \underline{Q}_{x0} \cdot e^{\underline{A}^T \cdot t} + \int_0^t e^{\underline{A} \cdot (t-\tau)} \cdot \underline{B} \cdot \underline{V} \cdot \underline{B}^T \cdot e^{\underline{A}^T \cdot t} d\tau \tag{9.14}$$

Wenn $\underline{A}$ asymptotisch stabil ist, ergibt sich folgender Grenzwert für die Varianzmatrix für beliebige $\underline{Q}_{x0}$:

$$\lim_{t \to \infty} \underline{Q}_x(t) = \underline{Q}_x = \int_0^\infty e^{\underline{A} \cdot t} \cdot \underline{B} \cdot \underline{V} \cdot \underline{B}^T \cdot e^{\underline{A}^T \cdot t} d\tau \tag{9.15}$$

Die Varianzen der Ausgänge können über die Ausgangsgleichung der Zustandsraumdarstellung mit Hilfe der $\underline{C}$-Matrix sofort abgeleitet werden zu:

$$\underline{Q}_y = \int_0^\infty \underline{C} \cdot e^{\underline{A} \cdot t} \cdot \underline{B} \cdot \underline{V} \cdot \underline{C}^T \cdot \underline{B}^T \cdot e^{\underline{A}^T \cdot t} d\tau \tag{9.16}$$

Die Darstellung im Frequenzbereich wird mit Hilfe der Frequenzgangsmatrizen formuliert:

$$\underline{Q}_x = \frac{1}{\pi} \cdot \int_0^\infty \underline{F}_x(j\omega) \cdot \underline{V} \cdot \underline{F}_x{}^T(-j\omega) d\omega \tag{9.17}$$

$$\underline{Q}_y = \frac{1}{\pi} \cdot \int_0^\infty \underline{F}_y(j\omega) \cdot \underline{V} \cdot \underline{F}_y{}^T(-j\omega) d\omega \tag{9.18}$$

Die Lösung dieser Integralgleichungen führt auf eine Gleichung in der sogenannten *Lyapunov*-Form, für die in der Literatur zahlreiche Lösungsverfahren beschrieben werden [Kwakernaak, Sivan 1972] :

$$\underline{0} = \underline{A} \cdot \underline{Q} + \underline{Q} \cdot \underline{A}^T + \underline{B} \cdot \underline{V} \cdot \underline{B}^T \tag{9.19}$$

9.2 Wichtige Analyseverfahren für nichtlineare Systeme

Fourier-Transformation

Die Fourier-Transformation (FT) ist ein wichtiges Werkzeug zur Berechnung des Spektrums eines Signals im Frequenzbereich. Die Formel zur kontinuierlichen Fourier-Transformation ([Johnson 1991], [Babovsky et al. 1987]) eines Signals *u(t)* lautet:

$$U(j\omega) = \int_{-\infty}^{\infty} u(t) \cdot e^{-j\omega t} d\tau \tag{9.20}$$

Um die Diskrete Fourier-Transformation (DFT) zu berechnen, wird das Integral durch die Summe von *N* Rechtecken der Höhe *u(nT)* angenähert. Dabei entspricht *T* der Abtastzeit, *F = 1/T* der Abtastfrequenz und *N* der Anzahl der Messwerte:

$$U_d(\omega_k) = \sum_{n=0}^{N-1} u(nT) \cdot e^{-j\omega_k nT} \tag{9.21}$$

Die Spektren aus der Fourier-Transformation kennzeichnen, vereinfacht ausgedrückt, den Schwingungsanteil aller harmonischen Schwingungen im gesamten Frequenzbereich $-\infty < \omega < \infty$. Möchte man die Leistung eines Signals pro jeweiliger Frequenz ermitteln, so verwendet man das Autoleistungsdichtespektrum S_{UU}:

$$S_{UU}(\omega) = \lim_{T \to \infty} \frac{U(j\omega) \cdot U'(j\omega)}{T} = \lim_{T \to \infty} \frac{|U(j\omega)|^2}{T} = \int_{-\infty}^{\infty} R_{uu}(\tau) \cdot e^{-j\omega\tau} d\tau \tag{9.22}$$

Das Autoleistungsdichtespektrum ist ein Maß dafür, wie sich der quadratische Mittelwert über die Frequenzen verteilt. Für mittelwertfreie Signale mit $E\{u\} = 0$ entspricht der quadratische Mittelwert auch der Varianz:

$$q_u^2(u, \omega_1, \omega_2) = \int_{\omega_1}^{\omega_2} S_{UU}(\omega) \cdot d\omega \tag{9.23}$$

Frequenzgang

Für nichtlineare Systeme kann kein Frequenzgang im Sinne der linearen Systemtheorie berechnet werden. Mit Hilfe der Fourier-Analyse kann jedoch der Frequenzgang in einem stabilen Betriebspunkt messtechnisch bestimmt werden. Hier gilt es zu beachten, dass die resultierenden Frequenzkennlinien mit dem Betriebspunkt und der Anregungsamplitude variieren. Für die Messung des Frequenzgangs zwischen dem Eingang *u(t)* und dem Ausgang *y(t)* eines Systems wird zunächst das Kreuzleistungsdichtespektrum S_{YU} benötigt:

$$S_{YU}(\omega) = \lim_{T \to \infty} \frac{Y(j\omega) \cdot U'(j\omega)}{T} = \int_{-\infty}^{\infty} R_{uy}(\tau) \cdot e^{-j\omega\tau} d\tau \tag{9.24}$$

Der Frequenzgang eines Systems lässt sich mit Hilfe von Autoleistungsdichte- und Kreuzleistungsdichtespektrum folgendermaßen ermitteln:

$$F(j\omega) = \frac{S_{YU}(\omega)}{S_{UU}(\omega)} \tag{9.25}$$

9.3 Berechnung von Fehlerflächen

Die Berechnung von Fehlerflächen kann durch Integralfunktionen formuliert werden. Dabei wird der Fehler ε_∞ zwischen Soll- und Istkurven im Zeit- oder im Frequenzbereich integriert und gewichtet. Tabelle 25 listet einige Integralfunktionen im Zeitbereich auf. Integralfunktionen für Fehlerflächen im Frequenzbereich lassen sich analog konstruieren:

TABELLE 25. Integralfunktionen zur Berechnung von Fehlerflächen im Zeitbereich

Abkürzung	Bezeichnung	Integralfunktion
IAE	Betragslineare Fehlerfläche (*Integral Absolute Error*)	$\int_0^\infty \lvert\varepsilon(t)\rvert d\tau$
ISE	Quadratische Fehlerfläche (*Integral Squared Error*)	$\int_0^\infty \varepsilon^2(t) d\tau$
ITAE	Zeitgewichtete betragslineare Fehlerfläche (*Integral Time-Weighted Absolute Error*)	$\int_0^\infty t \cdot \lvert\varepsilon(t)\rvert d\tau$
ITSE	Zeitgewichtete quadratische Fehlerfläche (*Integral Time-Weighted Squared Error*)	$\int_0^\infty t \cdot \varepsilon^2(t) d\tau$

10 Anhang B: Verwendung von Scilab

Im Folgenden werden wichtige Scilab-Funktionalitäten skizziert, die bei einer Verwendung von Scilab als Evaluator für lineare Systeme zum Tragen kommen. Die dargestellten Scilab-Skripte entstammen den Anwendungsbeispielen aus Kap. 7. Zusätzlich wird die Anbindung von Scilab bzw. des Scilab-Servers an MOPO und IPANEMA beschrieben.

10.1 Bewertung von Eigenwertlagen

Ein konjugiert komplexes Eigenwertpaar sollte in einem symmetrischen Paar von Kreisringsek-

toren liegen, wodurch Dämpfung und Abklingzeit der zugehörigen Schwingung innerhalb vorgegebener Schranken liegen. Mit Hilfe der Funktion *calcpoly()* werden zunächst geeignete Polygone über die Vorgabe von Start-/Endwinkeln (zur imaginären Achse) und Start-/Endradien (Abstand zum Ursprung der komplexen Halbebene) definiert. Die Funktion *poleMaxDistance()* errechnet den maximalen Abstand eines Eigenwertes zu einem gegebenen Polygon. Liegt der Eigenwert innerhalb des Polygons oder auf seinem Rand, wird als Abstandswert 0 zurückgegeben:

```
// Beispiel für die Bewertung von Eigenwerten in Scilab
// Autor: M.Deppe

// eigene Funktionen definieren
exec("calcpoly.sci");
exec("poleMaxDistance.sci");
...
// Polygone erzeugen
// Parameter:
// Anfangs-,Endradius,Anfangs-,Endwinkel der Kreisringsektoren und
// Anzahl der Stützstellen
polygon_a = calcpoly( 3*2*pi,   6*2*pi, 30, 70, 20);
polygon_r = calcpoly( 8*2*pi,  15*2*pi, 30, 70, 20);
polygon_k = calcpoly( 1*2*pi,   5*2*pi, 30, 70, 20);
...
// Eigenwerte der A-Matrix berechnen
s=spec(A);

// Auswahl bestimmter Eigenwerte
posspecmat = [];
n = size(s);
j = 1;
for i=1:n(1)
  if imag(s(i)) > 0.0 then
    posspecmat(j,1) = real(s(i));
    posspecmat(j,2) = imag(s(i));
    j = j+1;
  end
end

// Abstand der Eigenwerte vom vorgegebenen Polygon berechnen
distance = [];
// <eigene externe C-Routine: poleMaxDistance>
distance(1) = poleMaxDistance((posspecmat(1,:)), polygon_r);
distance(3) = poleMaxDistance((posspecmat(3,:)), polygon_k);
distance(5) = poleMaxDistance((posspecmat(5,:)), polygon_a);
...
return;
```

Listing 10-1: Bewertung von Eigenwerten in Scilab

Mit Hilfe der Scilab-Funktionen *link()* und *fort()* wird die in C-Code implementierte Funktion *poleMaxDistance()* folgendermaßen auf die gleichnamige Scilab-Funktion gemappt:

```
// Einbinden der externen C-Funktion poleMaxDistance
x = link('polemaxdistance.dll", 'poleMaxDistance', 'c');
```

```
function maxDist = poleMaxDistance(cur, ref)
  maxDist = -1.0;         // Rückgabewert,         index 1, double*
  curLen = size(cur, 1);  // Anzahl Eigenwerte,    index 2, integer*
  curX   = cur(:,1);      // Realteile,            index 3, double*
  curY   = cur(:,2);      // Imaginärteile,        index 4, double*
  refLen = size(ref, 1);  // Anzahl Polygonpunkte, index 5, integer*
  refX   = ref(:,1);      // Realteile,            index 6, double*
  refY   = ref(:,2);      // Imaginärteile,        index 7, double*
  maxDist = fort('poleMaxDistance',maxDist,1,'d', curLen, 2, 'i',...
                 curX, 3, 'd', curY, 4, 'd', ...
                 refLen, 5, 'i', refX, 6, 'd', refY, 7, 'd', ...
                 'out', size(0), 1, 'd'); // Rückgabewert ist index 1
endfunction
```

Listing 10-2: Definition der Funktion *poleMaxDistance()*

Nachfolgend ist der zugehörige C-Code der Funktion *poleMaxDistance()* gelistet. Als Besonderheit ist zu nennen, dass Scilab Funktionen mit dem Rückgabetyp *void* und mit den Adresszeigern der Ein- und Ausgangsgrößen als Funktions-Parametern erwartet:

```
/*  Datei: <poly.c> */
/*  Autor: E. Münch */
#include <math.h>

void poleMaxDistance(double *MaxDist, int *curLen, double *curX, double *curY,
                     int *refLen, double *refX, double *refY) {
  int i;
  int Result=0;
  double d;

  *MaxDist = 0.0;
  for(i = 0; i < *curLen; i++) {
    Result = PtInPoly(curX[i], curY[i], *refLen, refX, refY);
    if(!Result) {
      double dmin = HUGE_VAL;
      int j0, j1;
      for(j0 = 0; j0 < *refLen; j0++) {
        j1 = (j0+1) % *refLen;
       d = distance(curX[i], curY[i], refX[j0], refY[j0], refX[j1], refY[j1]);
        sciprint("%s: distance from Pole %d to
                 line %d = %g\r\n", "poleMaxDistance", i, j0, d);
        if(d < dmin) dmin = d;
      }
      if(dmin > *MaxDist) *MaxDist = dmin;
    }
  }
  return;
}

int PtInPoly(double pX, double pY, int refLen, const double *refX,
             const double *refY) {
  int i, j, Count = 0;
  for(i = 0; i < refLen; i++) {
    double dr, xr;
    j = ( i+1 ) % refLen;
    if( (refY[i] > pY ? 1 : 0) == (refY[j] > pY ? 1 : 0) ) continue;
    if(refX[i] <= pX && refX[j] <= pX) continue;
```

```
    if(refX[i] >  pX && refX[j] >  pX) { Count++; continue; }
    dr = (refY[j] - refY[i]) / (refX[j] - refX[i]);
    if(pX < refX[i] + (pY - refY[i]) / dr) Count++;
  }
  return Count & 1;
}

double distance(double pX, double pY, double refX0, double refY0,
                double refX1, double refY1) {
  double t, dx, dy, dpx, dpy, dq;
  dx  = refX1 - refX0;
  dy  = refY1 - refY0;
  dpx = pX - refX0;
  dpy = pY - refY0;
  dq = dx*dx + dy*dy;
  if(dq == 0.0)
    t = -1.0;
  else
    t = ( dpx*dx + dpy*dy ) / dq;
  if(t < 0.0) {
    double x, y;
    x = pX - refX0;
    y = pY - refY0;
    return sqrt(x*x + y*y);
  }
  else if(t > 1.0) {
    double x, y;
    x = pX - refX1;
    y = pY - refY1;
    return sqrt(x*x + y*y);
  }
  return fabs( (dpx*dy - dpy*dx) / sqrt(dq) );
}
```

Listing 10-3: Externe C-Routine `poleMaxDistance`

10.2 Berechnung von Varianzen

Wie in Kap. 9 gezeigt, lässt sich die Berechnung von Varianzen für die Zustände/Ausgänge linearer Systeme auf das Lösen einer Lyapunovgleichung zurückführen. Hierfür ist in Scilab die Funktion *lyap()* vorgesehen. Nachfolgendes Skript verdeutlicht den prinzipiellen Ablauf zur Berechnung von Varianzen in Scilab:

```
// Beispiel für die Berechnung von Varianzen/Streuungen in Scilab
// Autor: M.Deppe
...
// A,B,C,D zu einem kontinuierlichen linearen System umformen
S1=syslin('c',A,B,C,D);

// Kovarianzmatrix der Zustände für die Eingänge 1 und 2
// über die Lyapunovgleichung
V=eye(2,2);
Q=lyap(S1.A', -S1.B*V*S1.B', 'c');

// RMS-Werte (Streuungen) der Zustände
rmsx=sqrt(diag(Q));
```

```
// RMS-Werte (Streuungen) der Ausgänge
// S1.C*Q*S1.C' liefert nicht den Durchgriffs-Einfluß!
rmsy = sqrt(abs(diag(S1.C*Q*S1.C')));

return;
```

Listing 10-4: Scilab-Beispiel zur Berechnung von Varianzen bzw. Streuungen von linearen Systemen

10.3 Lineare Simulation (Einspurmodell)

Die lineare Simulation ist ein wichtiges Hilfsmittel zur Bewertung linearer Systeme. Die Beschreibung des linearen Systems in Zustandsraumdarstellung kann dabei rein numerisch oder auch symbolisch vorliegen. Das untenstehende Skript zeigt die lineare Simulation in Scilab auf der Basis des in Kap. 7.4.5 beschriebenen linearen Einspurmodells. Zusätzlich ist die MLaP-Funktion *lsim()* dokumentiert.

```
// Lineares Einspurmodell mit Geschwindigkeits- und "Lenkregler"
// Symbolische Bewegungsgleichungen und lineare Simulation
// Autor: M.Deppe

exec("deffunc.sci"); // Definition der eigenen Funktion lsim() und damp()
// Alle Werte in SI-Einheiten!
vlin =     5.00;      // Geschwindigkeit im Betriebspunkt
m    =  1296.00;      // Fahrzeugmasse
Iz   =  1750.00;      // Trägheitsmoment um die Hochachse
lv   =     1.25;      // Abstand Schwerpunkt Vorderachse
lh   =     1.32;      // Abstand Schwerpunkt Hinterachse
cv   = 70000.00;      // Gesamtfederkonstante vorderer Schräglauf
ch   = 70000.00;      // Gesamtfederkonstante hinterer Schräglauf
Aw   =     2.00;      // Stirnfläche des Fahrzeugs
cw   =     0.40;      // Luftwiderstandsbeiwert
rho  =     1.20;      // Luftdichte
KV   =  8242.00;      // P-Verstärkung V-Regler
TVI  =     0.18;      // Zeitkonstante PI-Regler
KL   =     6.03;      // Verstärkung Lenkregler
KLB  = KL*(vlin/5.0); // Verstärkung Lenkregler mit v-Kompensation
TLI  =     0.36;      // Zeitkonstante PI-Regler
// Koeffizienten A-Matrix (NUR STRECKE)
  a11 = -1.0*(cv+ch)/(m*vlin);
  a13 = -1.0/(m*vlin)*(m*vlin+cv*lv/vlin-ch*lh/vlin);
  a23 =  1.0;
  a31 = -1.0/Iz*(cv*lv-ch*lh);
  a33 = -1.0/Iz*(cv*lv*lv/vlin+ch*lh*lh/vlin);
  a45 =  1.0;
  a55 =  (-KV-0.5*rho*cw*Aw*vlin) / m;
// Koeffizienten B-Matrix (NUR STRECKE)
  b11 = cv/(m*vlin);
  b31 = cv*lv/Iz;
  b52 = 1.0/m;
// Systemmatrizen des geregelten Systems
A =  [ a11-b11*KLB -b11*KLB a13  0   0     0          b11*KLB/TLI;...
       0            0       1    0   0     0          0;          ...
       a31-b31*KL  -b31*KL  a33  0   0     0          b31*KL/TLI; ...
       0            0       0    0   1     0          0;          ...
```

```
       0            0       0    0   a55   b52*KV/TVI  0;          ...
       0            0       0    0  -1     0           0;          ...
      -1           -1       0    0   0     0           0;          ...
  ];
B =  [  b11*KLB 0;       ...
        0       0;       ...
        b31*KL  0;       ...
        0       0;       ...
        0       b52*KV;  ...
        0       1;       ...
        1       0;       ...
  ];
C = diag([1 1 1 1 1 1 1]);
D = zeros(7,2);
Sg = syslin('c',A,B,C,D);
Sg.X0=[ 0.0 0.0 0.0 0.0 0.0 0.0 0.0]';
d = damp(Sg.A); // Eigenwerte mit Freq und Dämpfung
len = 1500;
// Zeitvektor
t=linspace(0,15,len);
// Sprunganregung
s_upsi=10*1.57*[zeros(1,250) ones(1,500) +1.0*ones(1,500) ones(1,250)];
s_uv=5.0*[zeros(1,250) ones(1,500) ones(1,750)];
// Lineare Simulation mit der eigenen Funktion lsim()
s_data=lsim(Sg,[s_upsi; s_uv;],0.01);
return;
```

Listing 10-5: Lineare Simulation des linearen Einspurmodells als Scilab-Skript

Für die lineare Simulation wird die eigene Scilab-Funktion *lsim()* verwendet:

```
// Simulation fuer ein zeitkontinuierliches lineares System
// Autor: U. Dierkes
function [y] = lsim(S,u,dt)
  n=size(S.A);
  // Fundamentalmatrix berechnen
  Phi=expm(S.A*dt); // phi(T)=e^AT
  // Theta = (phi(T)-I)*inv(A)
  F=eye(S.A)*dt;
  E=zeros(S.A);
  k=2;
  while norm(E+F-E,1)>0
    E=E+F;
    F=S.A*F*dt/k;
    k=k+1;
  end
  Theta=E;
  // Simulation
  x=S.x0;
  n=size(u);
  m=size(S.C);
  y=zeros(m(1),n(2));
  for i = 1 : n(2)
    // Zustaende
    x=Phi*x + Theta*S.B*u(:,i);
    // Ausgaenge
    y(:,i)=S.C*x+S.D*u(:,i);
```

```
  end
endfunction
```

Listing 10-6: Funktion *lsim()* zur linearen Simulation

11 Literaturverzeichnis

[Allgower, Georg 1990] Allgower, E.; Georg, K. (1990). *Numerical Continuation Methods.* Springer-Verlag, Berlin, Heidelberg, New York.

[Amdahl 1967] Amdahl, G. M. (1967). *Validity of single-processor approach to achieving large-scale computing capability.* Proceedings of AFIPS Conference, Reston, VA, pp. 483-485.

[Babovsky et al. 1987] Babovsky, H.; Beth, T.; Neunzert, H.; Schulz-Reese, M. (1987). *Mathematische Methoden in der Systemtheorie: Fourieranalysis.* Teubner-Verlag, Stuttgart.

[Bednara et al. 2003] Bednara, M.; Danne, K.; Deppe, M.; Oberschelp, O.; Slomka, F.; Teich, J. (2003). *Design and Implementation of Digital Linear Control Systems on Reconfigurable Hardware*. EURASIP Journal on Applied Signal Processing, Volume 2003, No. 6, pp. 594-602.

[Buttazzo 1997] Buttazzo, G. C. (1997). *Hard Real-Time Computing Systems.* Kluwer Academic Publishers.

[Bronstein, Semendjajew 1989] Bronstein, I. N.; Semendjajew, K. A. (1989). *Taschenbuch der Mathematik.* Verlag Harri Deutsch, Thun, Frankfurt/Main.

[Broyden 1967] Broyden, C. G. (1967). *Quasi-Newton Methods and their Applications to Function Minimization.* Mathematics of Computation 21, pp. 368-381.

[Castiglioni et al. 1992] Castiglioni, G.; Jäker, K.-P.; Lückel, J.; Rutz, R. (1992). *Active Vehicle Suspension with an Active Vibration Absorber.* Proceedings of the International Symposium on Advanced Vehicle Control AVEC '92, Society of Automotive Engineers of Japan, Inc., Yokohama, pp. 148-153.

[Coello, Lechunga 2002] Coello, C. A. C.; Lechunga, M. S. (2002). *MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization.* Proceedings of the IEEE World Congress on Computational Intelligence, Hawaii.

[Cremer 1979] Cremer, M. (1979). *Der Verkehrsfluß auf Schnellstraßen.* Springer-Verlag, Berlin, Heidelberg, New York.

[Culler et al. 1993] Culler, D.; Karp, D.; Patterson, D.; Sahay, A.; Schauser, K. E.; Santos, E.; Subramanian, R.; von Eicken, T. (1993). *Logp: Towards a realistic model of parallel computation.* 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.

[Das, Dennis 1996/1] Das, I.; Dennis, J.E. (1996). *Normal-Boundary Intersection: An Alternate Approach for Generating Pareto-optimal Points in Multicriteria Optimization Problems.* ICASE-NASA Tech. Report 96-62. SIAM J. on Optimization.

[Das, Dennis 1996/2] Das, I.; Dennis, J.E. (1996). *A Closer Look at Drawbacks of Minimizing Weighted Sums of Objectives for Pareto Set Generation in Multicriteria Optimization Problems.* Dept. of CAAM Tech. Report 96-36, Rice University, Houston, TX.

[Deppe 1995] Deppe, M. (1995). *Programmierung des Lastverteilungstools LODIT (Load Distribution Tool).* Studienarbeit im Fachbereich Maschinentechnik, Universität Paderborn.

[Deppe 1997] Deppe, M. (1997). *Parallele Simulation in heterogenen Rechnernetzen - Erweite-*

rung und Test der verteilten Simulationsplattform IPANEMA. Diplomarbeit im Fachbereich Maschinentechnik, Universität Paderborn.

[Deppe, Homburg 1998] Deppe, M.; Homburg, C. (1998). *Rapid Prototyping of Distributed Mechatronic Applications.* DIPES ´98, Schloss Eringerfeld.

[Deppe, Oberschelp 2000] Deppe, M.; Oberschelp, O. (2000). *Real-time Support for Online Controller Supervision and Optimization.* International IFIP Workshop on Distributed and Parallel Embedded Systems, Paderborn University, Schloß Eringerfeld.

[Deppe, Rasche 2000/1] Deppe, M.; Rasche, R. (2000). *Sonderforschungsbereich 376: Massive Parallelität - Algorithmen, Entwurfsmethoden, Anwendungen - Arbeits- und Ergebnisbericht Juli 1998 - Dezember 2000*. Teilprojekt C1, Universität Paderborn.

[Deppe, Rasche 2000/2] Deppe, M.; Rasche, R. (2000). *Sonderforschungsbereich 376: Massive Parallelität - Algorithmen, Entwurfsmethoden, Anwendungen - Finanzierungsantrag Januar 2001 - Dezember 2003*. Teilprojekt C1, Universität Paderborn.

[Deppe et al. 2001/1] Deppe, M.; Robrecht, M.; Zanella, M.; Hardt, W. (2001). *Rapid Prototyping of Real-Time Control Laws for Complex Mechatronic Systems.* 12th IEEE International Workshop on Rapid System Prototyping, Monterey, CA.

[Deppe et al. 2001/2] Deppe, M.; Oberschelp, O.; Münch, E. (2001). *Echtzeit-Parameter-Optimierung und Überwachung in mechatronischen Systemen.* In: 5. Magdeburger Maschinenbautage, Otto-von-Guericke-Universität, Entwicklungsmethoden und Entwicklungsprozese im Maschinenbau, Logos Verlag, Berlin.

[Deppe, Zanella 2002] Deppe, M.; Zanella, M. (2002). *Design and Realization of Distributed Real-Time Controllers for Mechatronic Systems.* World Computer Congress, Stream 7, DIPES, Montreal, Quebec, Canada; Kluwer Academic Publishers, Norwell, MA.

[Deppe et al. 2003/1] Deppe, M.; Neuendorf, N.; Scharfeld, F. (2003). *Sonderforschungsbereich 376: Massive Parallelität - Algorithmen, Entwurfsmethoden, Anwendungen - Arbeits- und Ergebnisbericht Januar 2000 - Juli 2003*. Teilprojekt C1, Universität Paderborn.

[Deppe et al. 2003/2] Deppe, M.; Neuendorf, N.; Scharfeld, F. (2003). *Sonderforschungsbereich 376: Massive Parallelität - Algorithmen, Entwurfsmethoden, Anwendungen - Finanzierungsantrag Januar 2003 - Dezember 2005*. Teilprojekt C1, Universität Paderborn.

[Deppe et al. 2003/3] Deppe, M.; Robrecht, M.; Zanella, M.; Hardt, W. (2003). *Rapid Prototyping of Real-Time Control Laws for Complex Mechatronic Systems: A Case Study.* The Journal of Systems and Software, No. 70/3, pp. 263-274.

[DFG-Antrag zum SFB 614 2001] DFG-Antrag zum SFB 614 (SFB-Antragsnummer 1799) (2001). *"Selbstoptimierende Systeme des Maschinenbaus"*, Universität Paderborn.

[Dorigo, Gambardella 1997] Dorigo, M.; Gambardella, L. M. (1997). *Ant Colonies for the Traveling Salesman Problem.* J. Biosystems, Vol. 43.

[Dorißen, Höver 1996] Dorißen, H. T.; Höver, N. (1996). *Autonome Intelligente Geschwindigkeitsregelung (AICC) - Ein Beitrag zur Steigerung des Komforts und der aktiven Fahrsicherheit.*

Automobiltechnische Zeitschrift (ATZ) 98, 7/8, pp. 396-405.

[dSPACE 1995] dSPACE GmbH. (1995). *Real-Time Interface to SIMULINK, User's Guide*. dSPACE GmbH, Paderborn.

[Entenmann 1976] Entenmann, W. (1976). *Optimierungsverfahren*. Hüthig-Verlag, Heidelberg.

[Eppinger 1994] Eppinger, A. (1994). *Rechnerintegrierte Systemtechnik - objektorientiertes Konzept und Realisierung mit Parallelrechnern*. Dissertation am Fachbereich Maschinentechnik, Universität Paderborn.

[ETAS 1996] ETAS GmbH&Co.KG. (1996). *ASCETS-SD: User Guide Version 1.0*. ETAS GmbH&Co.KG, Schwieberdingen.

[Färber et al. 1997] Färber, G.; Fischer, F.; Kolloch, T.; Muth, A. (1997). *Improving Processor Utilization with a Task Classification Model based Application Specific Hard Real-Time Architecture.* Proceedings of the 6th International Workshop on Real-Time Computing Systems and Applications (RT-CSA ’97), Academia Sinica, Taipei, Taiwan.

[Fletcher 1980] Fletcher, R. (1980). *Practical Methods of Optimization.* Volume 1: Unconstrained Optimization. Wiley, New York.

[Föllinger 1994] Föllinger, O. (1994). *Regelungstechnik: Einführung in die Methoden und ihre Anwendung*. 8. überarb. Aufl., Hüthig, Heidelberg.

[Fonseca, Fleming 1995] Fonseca, C.; Fleming, P. (1995). *An overview of evolutionary algorithms in multiobjective optimization.* Evolutionary Computation, 3 (1), pp. 1-16.

[Gambuzza 2002] Gambuzza, A. (2002). *Konzept zur verteilten modularen Simulation mechatronischer Systeme.* Diplomarbeit im Fachbereich Mathematik/Informatik, Universität Paderborn.

[Gambuzza et al. 2003] Gambuzza, A.; Deppe, M.; Oberschelp, O. (2003). *Verteilte modulare Simulation mechatronischer Systeme.* 5. Mechatroniktagung des VDI, Fulda.

[Gill, Murray 1978] Gill, P. E.; Murray, W. (1978). *Numerically Stable Methods for Quadratic Programming*. Mathematical Programming, 14, pp. 349–372.

[Göpfert, Nehse 1990] Göpfert, A.; Nehse, R. (1990). *Vektoroptimierung.* BSB Teubner Verlagsgesellschaft, Leipzig.

[Hahn et al. 1997] Hahn, M.; Lückel, J.; Wittler, G. (1997). *Eine Entwurfsmethodik für mechatronische Systeme*. Magdeburger Maschinenbau-Tage: Entwicklungsmethoden u. Entwicklungsprozesse im Maschinenbau, Magdeburg.

[Hahn 1999] Hahn, M. (1999). *OMD - Ein Objektmodell für den Mechatronikentwurf. Anwendung in der objektorientierten Modellbildung mechatronischer Systeme unter Verwendung von Mehrkörpersystemformalismen.* Dissertation am Fachbereich Maschinentechnik, Universität Paderborn, VDI-Verlag, Düsseldorf.

[Haimes 1973] Haimes, Y. (1973). *Integrated system identification and optimization.* Control and Dynamic Systems: Advances in Theory and Applications 10, pp. 435-518.

[Hanselmann 1993] Hanselmann, H. (1993). *Hardware-in-the-Loop Simulation as Standard*

Approach for Development, Customization, and Production Test of ECU's. Seventh International Pacific Conference and Exposition on Automotive Engineering, Phoenix, AZ.

[Hees 1999] Hees, K. (1999). *Kommunikationseffiziente Lastverteilungsverfahren zur Simulation mechatronischer Systeme*. Diplomarbeit im Fachbereich Mathematik/Informatik, Universität Paderborn.

[Henke et al. 2000] Henke, M.; Liu-Henke, X.; Lückel, J.; Grotstollen, H.; Jäker, K.-P. (2000). *Design of a Railway Carriage, driven by a linear Motor with Active Suspension/Tilt Module.* 9th IFAC Symposium on Control in Transportation Systems, Braunschweig.

[Hestermeyer et al. 2001] Hestermeyer, T.; Becker, M.; Neuendorf, N. (2001). *Nichtlineare ABC-Regelungen mit Operator-Controller-Struktur, abgestimmt auf Führung und Störung der Straße.* Haus der Technik: Driveability, Essen.

[Hillermeier 2001] Hillermeier, C. (2001). *Nonlinear multiobjective optimization: a generalized homotopy approach.* Birkhäuser-Verlag, Basel, Boston, Berlin.

[Honekamp et al. 1997] Honekamp, U.; Stolpe, R.; Naumann, R.; Lückel, J. (1997). *Structuring Approach for Complex Mechatronic Systems*. ISATA'97, Florence.

[Honekamp 1998] Honekamp, U. (1998). *IPANEMA - Verteilte Echtzeit-Informationsverarbeitung in mechatronischen Systemen*. Dissertation am Fachbereich Maschinentechnik, Universität Paderborn, VDI-Verlag, Düsseldorf.

[iXtronics 2001/1] iXtronics GmbH. (2001). *CAMeL-View Reference Guide,* Paderborn.

[iXtronics 2001/2] iXtronics GmbH. (2001). *CAMeL-View User Guide,* Paderborn.

[Jäker 1991] Jäker, K.-P. (1991). *Entwicklung realisierbarer hierarchischer Kompensatorstrukturen für lineare Mehrgrößensysteme mittels CAD.* Dissertation am Fachbereich Maschinentechnik, Universität Paderborn, VDI-Verlag, Düsseldorf.

[Johnson 1991] Johnson, J. R. (1991). *Digitale Signalverarbeitung.* Hanser, München, Wien; Prentice-Hall, London.

[Kasper 1985] Kasper, R. (1985). *Entwicklung und Erprobung eines instrumentellen Verfahrens zum Entwurf von Mehrgrößenregelungen*. Dissertation am Fachbereich Maschinentechnik, Universität Paderborn, VDI-Verlag, Düsseldorf.

[Kasper et al. 1990] Kasper, R.; Lückel, J.; Jäker, K.-P.; Schröer, J. (1990). *CACE tool for multi-input, multi-output systems using a new vector optimization method.* International Journal of Control, Vol. 51, No. 5, pp. 963-993.

[Kiffmeier 1995] Kiffmeier, U. (1995). *Multi-DSP-Power auf Knopfdruck.* Elektronik, Ausg. 12, Franzis Verlag, München.

[Koch 2005] Koch, T. (2005). *Integration von Konstruktion und mechatronischer Komposition während des Entwurfs mechatronischer Systeme am Beispiel eines integrierten Radmoduls.* Dissertation am Fachbereich Maschinentechnik, Universität Paderborn, VDI-Verlag, Düsseldorf.

[Kreisselmeier, Steinhauser 1979] Kreisselmeier, R.; Steinhauser, R. (1979). *Control Design by Optimizing a Vector Performance Index*. IFAC Symposium on CADCS, Zürich, pp. 113-117.

[Kuhn, Tucker 1951] Kuhn, H.; Tucker, A. (1951). *Nonlinear programming.* Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, (ed. J. Neymann), University of California Press, Berkeley, CA, pp. 481-492.

[Kwakernaak, Sivan 1972] Kwakernaak, H.; Sivan, R. (1972). *Linear Optimal Control Systems.* Wiley-Interscience, New York, London, Sydney.

[Lasdon 1970] Lasdon, L. S. (1970). *Optimization Theory for Large Systems.* McMillan, New York.

[Liu, Layland 1973] Liu, C. L.; Layland, J. W. (1973). *Scheduling algorithm for multiprogramming in a hard real time environment.* Journal of ACM, vol. 20, pp. 46-61.

[Liu-Henke et al. 2000/1] Liu-Henke, X.; Lückel, J.; Jäker, K.-P. (2000). *Development of an Active Suspension/Tilt System for a Mechatronic Railway Carriage.* 1st IFAC-Conference on Mechatronics Systems (Mechatronics 2000), Darmstadt.

[Liu-Henke et al. 2000/2] Liu-Henke, X.; Lückel, J.; Jäker, K.-P. (2000). *Ganzheitlicher mechatronischer Entwurf eines aktiven Feder-/Neigemoduls.* VDI-Tagung: Mechatronik - Mechanisch/ Elektrische Antriebstechnik, Wiesloch.

[Ludyk 1990] Ludyk, G. (1990). *CAE von Dynamischen Systemen*. Springer-Verlag, Berlin, Heidelberg, New York.

[Lückel, Kasper 1981] Lückel, J.; Kasper, R. (1981). *Strukturkriterien für die Steuer-, Stör- und Beobachtbarkeit linearer, zeitinvarianter, dynamischer Systeme.* Regelungstechnik, 29. Jahrgang, Heft 10.

[Lückel et al. 1985] Lückel, J.; Kasper, R.; Jäker, K.-P. (1985). *Interactive Optimization of Controller and Plant Parameters in the Case of Multiple Design Objectives.* Hansen, N. E.; Larsen, P. M. (Hrsg.): Preprints of the 3rd IFAC/IFIP International Symposium CADCE´85: Computer Aided Design in Control and Engineering Systems. Advanced Tools for Modern Technology; Lyngby, Danmark.

[Lückel 1992] Lückel, J. (1992). *The Concept of Mechatronic Function Modules applied to Compound Active Suspension Systems.* Symposium: Research Issues in Automotive Integrated Chassis Control Systems, International Association for Vehicle System Dynamics, Herbertov.

[Lückel et al. 1999] Lückel, J.; Naumann, R.; Rasche, R. (1999). *Systematic Design of Crosslinked Mechatronic Systems, Exemplified by a Decentralized Intersection Management.* European Control Conference 1999, Karlsruhe.

[Lückel et al. 2000] Lückel, J.; Schmitz, J.; Koch, T. (2000). *Systematische Entwicklung mechatronischer Produkte am Beispiel eines hybrid angetriebenen Verteilerfahrzeugs.* VDI-Tagung: Mechatronik - Mechanisch/Elektrische Antriebstechnik, Wiesloch.

[Lückel 2000] Lückel, J. (2000). *Systemkonzept Neue Bahntechnik Paderborn.* 4. Internationales HNI-Symposium, Paderborn.

[Lückel et al. 2001] Lückel, J.; Hestermeyer, T.; Liu-Henke, X. (2001). *Generalization of the Cascade Principle in View of a Structured Form of Mechatronic Systems.* 2001 IEEE/ASME

International Conference on Advanced Intelligent Mechatronics (AIM 2001), Villa Olmo, Como.

[Lückel et al. 2002/1] Lückel, J.; Ettingshausen, C.; Hestermeyer, T.; Schlautmann, P. (2002). *Neue Bahntechnik Paderborn - Eine Anwendung der verallgemeinerten Kaskade*. Innovative Antriebssysteme, Erstes Internationales Symposium für Mechatronik (ISOM´02), Chemnitz.

[Lückel et al. 2002/2] Lückel, J.; Biber, H.; Koch, T.; Schlautmann, P. (2002). *Das Wechselspiel zwischen Konstruktion und Auslegung der Dynamik während des Entwurfs mechatronischer Systeme.* Festschrift zum 90. Geburtstag von Herrn Prof. Dr. rer. nat. Dr.-Ing. E. h. Kurt Magnus, München, pp. 195-209.

[Lootsma 1985] Lootsma, F.A. (1985). *Comparative performance evaluation, experimental design, and generation of test problems in nonlinear Optimization.* In: K. Schittkowski, ed., Computational Mathematical Programming, Springer-Verlag, Berlin, pp. 249-260.

[Marglin 1967] Marglin, S. (1967). *Public Investment Criteria.* MIT Press, Cambridge, MA.

[MathWorks 1992] MathWorks, Inc. (1992). *SIMULINK Dynamic System Simulation Software.* MathWorks, Inc., Natick, Mass.

[Mayr 2001] Mayr, R. (2001). *Regelungsstrategien für die automatische Fahrzeugführung: Längs- und Querregelung, Spurwechsel- und Überholmanöver.* Springer-Verlag, Berlin, Heidelberg, New York.

[Mitschke 1990] Mitschke, M. (1990). *Dynamik der Kraftfahrzeuge.* Band C: Fahrverhalten. 2. Aufl., Springer-Verlag, Berlin, Heidelberg, New York.

[Münch 2001] Münch, E. (2001). *Fortentwicklung und Realisierung eines Verfahrens zur gleichzeitigen Optimierung mehrerer Zielgrößen.* Studienarbeit im Fachbereich Maschinentechnik, Universität Paderborn.

[Münch 2003] Münch, E. (2003). *Mehrgrößenoptimierung - Algorithmusentwicklung und Anwendung an der Spurführung der NBP (Neue Bahntechnik Paderborn).* Diplomarbeit im Fachbereich Maschinentechnik, MLaP, Universität Paderborn.

[Nägler, Walter 1987] Nägler, G.; Walter, H. (1987). *Graphen, Algorithmen, Programme*. Springer-Verlag, Wien, New York.

[Naumann, Homburg 1996] Naumann, R.; Homburg, C. (1996). *LoDiT - Automatisches Partitionieren von mechatronischen Systemen für die verteilte Simulation.* Proceedings of the ASIM'96, Dresden.

[Naumann, Rasche 1997] Naumann, R.; Rasche, R. (1997). *Intersection Collision Avoidance by Means of Decentralized Security and Communication Management of Autonomous Vehicles.* Proceedings of the 30th ISATA Conference on ATT/ITS, Florence.

[Naumann 2000] Naumann, R. (2000). *Modellierung und Verarbeitung vernetzter intelligenter mechatronischer Systeme.* Dissertation am Fachbereich Maschinentechnik, Universität Paderborn, VDI-Verlag, Düsseldorf.

[Neuendorf, Deppe 2003] Neuendorf, N.; Deppe, M. (2003). *Vernetzte mechatronische Systeme am Beispiel eines dezentralen Kreuzungsmanagements für Kfz.* 5. Mechatroniktagung des VDI,

Fulda.

[Oberschelp 1998] Oberschelp, O. (1998). *Multirate-Integrationsverfahren zur Lösung von Differentialgleichungen bei der Simulation mechatronischer Systeme.* Studienarbeit im Fachbereich Maschinentechnik, Universität Paderborn.

[Olomski 1989] Olomski, J. (1989). *Bahnplanung und Bahnführung von Industrierobotern.* Fortschritte der Robotik, Band 4, Vieweg-Verlag, Braunschweig, Wiesbaden.

[Otterbach, Leinfellner 1999] Otterbach, R.; Leinfellner, R. (1999). *Virtuelles Ausprobieren - Vom Entwurf zur Simulation in Echtzeit - Stand der Technik bei Rapid Prototyping.* Zeitschrift Elektronik, Nr. 8.

[Papageorgiou 1996] Papageorgiou, M. (1996). *Optimierung: Statische, dynamische, stochastische Verfahren für die Anwendung.* Oldenbourg Verlag, München, Wien.

[Pareto 1971] Pareto, V. (1971). *Manual of Political Economy (English translation of 'Manuale di Economica Politica').* MacMillan Company, New York.

[Puschner 2002] Puschner, P. (2002). *Transforming Execution-Time Boundable Code into Temporally Predictable Code.* World Computer Congress, Stream 7, DIPES, Montreal, Quebec, Canada; Kluwer Academic Publishers, Norwell, MA.

[Rakowska et al. 1991] Rakowska, J.; Haftka, R,; Watson, L. (1991). *Tracing the efficient curve for multiobjective control-structure optimization.* Computing Systems in Engineering, 2 (6), pp. 461-471.

[Rasche et al. 1997] Rasche, R.; Naumann, R.; Tacken, J.; Tahedl, C. (1997). *Validation and Simulation of a Decentralized Intersection Collision Avoidance Algorithm.* Proceedings of the IEEE Conference on Intelligent Transportation Systems (ITSC'97), Boston, MA.

[Rasche 2004] Rasche. R. (2004). *Kreuzungsmanagement - informationstechnische Vernetzung autonomer Fahrzeuge als Beispiel für Selbstoptimierung im Maschinenbau.* Dissertation am Fachbereich Maschinentechnik, Universität Paderborn.

[Rickert, Schunck 1940] Rickert, P.; Schunck, T. (1940). *Zur Fahrmechanik des gummibereiften Kraftfahrzeugs.* Ingenieur-Archiv 11, pp. 210-224.

[Rükgauer 1996] Rükgauer, A. (1996). *Modulare Simulation mechatronischer Systeme mit Anwendung in der Fahrzeugdynamik.* Dissertation am Institut B für Mechanik, Universität Stuttgart.

[Rutz, Winkler 1994] Rutz, R.; Winkler, M. (1994). *Mechatronic Suspension Design using Online Optimization.* International Symposium on Advanced Transportation Applications (ISATA), Aachen.

[Schwarz 1993] Schwarz, H. R. (1993). *Numerische Mathematik.* 3. Aufl., Teubner-Verlag, Stuttgart.

[Schweitzer 1989] Schweitzer, G. (1989). *Mechatronik - Aufgaben und Lösungen*. VDI-Tagung "Kontrollierte Bewegungen im Maschinen- und Fahrzeugbau", Bad Homburg.

[Späth 1973] Späth, H. (1973). *Spline-Algorithmen zur Konstruktion glatter Kurven und Flä-*

chen. Oldenbourg Verlag, München, Wien.

[Stevens 1992] Stevens, W.R. (1992). *Programmieren von UNIX - Netzen, Grundlagen, Programmierung, Anwendung.* Hanser-Verlag, München, Wien; Prentice-Hall, London.

[Stolpe et al. 1997] Stolpe, R.; Homburg, C.; Deppe, M. (1997). Finanzierungsantrag und Ergebnisbericht zum DFG-Schwerpunktprogramm *Integrierte Steuerungssysteme mit harten Zeitbedingungen.* Teilprojekt: *Unterstützung des Entwurfs mechatronischer Systeme von der funktionellen Modularisierung bis zur Hardware-in-the-Loop-Simulation.* Universität Paderborn.

[Stolpe et al. 2000] Stolpe, R.; Deppe, M.; Zanella, M. (2000). *Rapid-Prototyping von verteilten, hierarchischen Regelungen am Beispiel eines Fahrzeugs mit hybridem Antriebsstrang.* Zeitschrift it&ti, 42. Jahrgang, Heft 2, pp. 54-58.

[Stolpe 2004] Stolpe, R. (2004). *Verteilte kommunizierende mechatronische Funktionsmodule - Von der mechatronisch funktionalen Modularisierung bis zur verteilten HIL-Realisierung.* Dissertation am Fachbereich Maschinentechnik, Universität Paderborn.

[Stumpe 1996] Stumpe, M. (1996). *Vergleichende Untersuchung von Integrationsverfahren zur numerischen Lösung von gewöhnlichen nichtlinearen Differentialgleichungen.* Diplomarbeit im Fachbereich Maschinentechnik, Universität Paderborn.

[Teich 1997] Teich, J. (1997). *Digitale Hardware / Software Systeme, Synthese und Optimierung.* Springer Verlag, Berlin/Heidelberg.

[Timmel 1980] Timmel, G. (1980). *Ein stochastisches Suchverfahren zur Bestimmung der optimalen Kompromißlösungen bei statischen polykriteriellen Optimierungsaufgaben.* Wiss. Zeitung TH Ilmenau, 6, pp. 159-174.

[Toepper 2002] Toepper, S. (2002). *Die mechatronische Entwicklung des Parallelroboters TRIPLANAR.* Dissertation am Fachbereich Maschinentechnik, Universität Paderborn, VDI-Verlag, Düsseldorf.

[Towsley et al. 1990] Towsley, D.; Rommel, G.; Stankovic, J.A. (1990). *Analysis of Fork-Join Program Response Times on Multiprocessors.* IEEE Trans. on Parallel and Distributed Systems 1, 3.

[Ungerer 1993] Ungerer, T. (1993). *Datenflussrechner.* Teubner-Verlag, Stuttgart.

[Van Laarhoven 1987] Van Laarhoven, P. J. M. (1987). *Simulated Annealing: Theory and Applications.* D. Reidel, Dordrecht.

[Vehicle Autonomous Systems 2002] International Journal of Vehicle Autonomous Systems. (2002), Volume 1, No. 1, The Open University, Milton Keynes, UK.

[Walther 1984] Walther, H. (1984). *Ten applications to graph theory.* D. Reidel, Dordrecht.

[Wältermann 2000] Wältermann, P. (2000). *Der serielle Hybridantrieb - Vom rechnergestützten Entwurf bis zur Hardware-in-the-Loop-Realisierung.* Dissertation am Fachbereich Maschinentechnik, Universität Paderborn, VDI-Verlag, Düsseldorf.

[Wismer 1971] Wismer, D. A. (1971). *Optimization Methods for Large Scale Systems - with Applications.* McGraw Hill, New York, London.

[Wittler 2003] Wittler, G. (2003). *Integrative Modellierung von Gestalt und dynamischem Verhalten beim Entwurf mechatronischer Systeme*. Dissertation am Fachbereich Maschinentechnik, Universität Paderborn, VDI-Verlag, Düsseldorf.

[Zanella et al. 2001] Zanella, M.; Koch, T.; Scharfeld, F. (2001). *Development and Structuring of Mechatronic Systems, Exemplified by the Modular Vehicle X-mobile.* 2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2001), Villa Olmo, Como.