Objektorientierte Modellierung mechatronischer Systeme am Beispiel geregelter Roboter

Dissertation

zur

Erlangung des Grades

Doktor-Ingenieur

der

Fakultät für Maschinenbau

der Ruhr-Universität Bochum

von

Martin Otter

geboren in Reutlingen

Dissertation eingereicht am:	26. November 1993
Tag der mündlichen Prüfung:	17. Februar 1994
Erster Referent:	Prof. Dr.-Ing. Georg Grübel
Zweiter Referent:	Prof. Dr. techn. Karl Heinz Fasol
Dritter Referent:	Prof. Dr.-Ing. Manfred Hiller

Das Promotionsverfahren wurde unter dem bis zum 18. April 1994 geführten Familiennamen "Dipl.-Ing. Martin **Bartanus**" durchgeführt.

Vorwort

Diese Arbeit gibt eine Einführung in die neuartigen, objektorientierten Methoden, die auf dem Gebiet der multidisziplinären Modellierung entwickelt wurden. Die Grundlagen für eine rechnergestützte, multidisziplinäre Modellierung dynamischer Systeme wurden schon Ende der 70er Jahre von Hilding Elmqvist erarbeitet. Zu diesem Zeitpunkt waren jedoch die Rechner für diese neue Vorgehensweise noch nicht leistungsfähig genug. In den letzten Jahren hat die objektorientierte Modellierung eine Renaissance erfahren, weil zum einen das enge dynamische Zusammenspiel von Teilkomponenten unterschiedlicher Fachgebiete, wie z.B. in der Mechatronik, immer wichtiger wird, und zum anderen jetzt auch schon mittelgroße PCs die notwendige Speicher- und Rechen-Kapazität besitzen, um eine objektorientierte Modellbildungstechnik praktisch anwenden zu können.

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Robotik und Systemdynamik bei der Deutschen Forschungsanstalt für Luft- und Raumfahrt (DLR) in Oberpfaffenhofen. Dem Leiter der Abteilung Entwurfsorientierte Regelungstechnik Prof. Dr.-Ing. G. Grübel gilt mein besonderer Dank, daß er diese Arbeit initiierte, nachhaltig förderte und als Hauptreferent im Promotionsverfahren an der Fakulät für Maschinenbau der Ruhr-Universität Bochum betreute.

Mein Dank gilt auch folgenden Personen:

Prof. Francois E. Cellier, University of Arizona, und Dr. Hilding Elmqvist, Dynasim AB Schweden, danke ich für die hervorragende, äußerst produktive Zusammenarbeit in den letzten zwei Jahren auf dem Gebiet der objektorientierten Modellierung, die ohne Electronic Mail so nicht möglich gewesen wäre.

Meinen Kollegen bei der DLR Dr. Ing. J. Bals, Dr. Ing. R. Finsterwalder, Dipl. Ing. J. Franke, Dr. C. Führer, Dr. Ing. H. D. Joos und Dr. Ing. A. Lewald danke ich für die angenehme Zusammenarbeit in einem echten Team und für die wissenschaftliche Unterstützung in Einzelfragen.

Dipl. Ing. M. Hocke vom Rechenzentrum der Universtät Stuttgart, sowie Dr. Ing. A. Daberkow und Dr. Ing. G. Leister vom Institut für Mechanik B der Universität Stuttgart danke ich für die Zusammenarbeit bei der Entwicklung eines objektorientierten Datenmodells für Mehrkörpersysteme im Rahmen des DFG-Schwerpunktprogrammes "Dynamik von Mehrkörpersystemen".

Prof. Dr. techn. K. H. Fasol, Ruhr-Universität Bochum und Prof. Dr. Ing. M. Hiller, Universität-GH Duisburg, danke ich für die Übernahme der Koreferate, sowie dem Dekan des Fachbereichs Maschinenbau der Ruhr-Universität Bochum Prof. G. Wagner danke ich für sein Entgegenkommen in einer verwaltungstechnischen Verfahrensvorschrift.

München, im Juli 1994 Martin Otter

Inhalt

Kapitel 1

Einführung

1.1 Umfeld und Ziel der Arbeit

Im Mittelpunkt dieser Arbeit steht die *multidisziplinäre* Modellbildung technischer, dynamischer Systeme, wobei der Schwerpunkt auf mechatronischen Systemen liegt. Es werden Systeme betrachtet, deren Einzelkomponenten mit Modellbildungsverfahren unterschiedlicher Ingenieur-Fachgebiete beschrieben werden, wobei das Gesamtverhalten wesentlich durch das Zusammenwirken dieser Einzelkomponenten bestimmt wird. Beispielsweise wird zur Simulation mechatronischer Systeme, wie Roboter, Satelliten oder aktive Fahrzeugkomponenten, die Modellierung mechanischer Konstruktionen, elektrischer Schaltungen, regelungstechnischer Wirkkomponenten und sonstiger physikalischer Effekte, wie Reibung, Lose, Aufheizung, benötigt. Durch Simulation soll das Systemverhalten analysiert werden, bzw. es sind Entwürfe durchzuführen, die auf Simulationen basieren.

Rein *mechanische Systeme* werden am besten mit Mehrkörperprogrammen wie ADAMS, DADS [Smit90], NEWEUL [Kreu90], SD-FAST [Rose86, Rose87] oder SIMPACK [Rulk90] modelliert, da diese eine der Anwendung angepaßte Spezifikation erlauben, zum Beispiel durch elementare Angaben: "starrer Körper", "verformbarer Körper", "Gelenk", "Kraftgesetz". Der anspruchvollste Teil dieser Programme besteht aus einem Formalismus, mit dem aus der Modellbeschreibung, zugehörige Differentialgleichungen oder differential-algebraische Gleichungen automatisiert erstellt werden.
Elektrische Schaltungen werden am besten mit SPICE [Nage75] oder einer seiner Derivate wie PSPICE modelliert, da die Eingabe zum Beispiel durch elementare Angaben: "Widerstand", "Kapazität", "MOS-Feldeffekttransistor", "CMOS-Operationsverstärker" und andere, sehr anwendungsnah ist. Das Know-How dieser Programme besteht in großen Bausteinbibliotheken elektrotechnischer Elemente mit unterschiedlich detaillierten mathematischen Modellen, der Zusammenschaltung einer sehr großen Anzahl dieser Elemente zu elektrischen Schaltungen, sowie der numerischen Behandlung der hier auftretenden speziellen differential-algebraischen Gleichungen.
Regelungstechnische Komponenten werden am besten mit einem grafischen Blockschaltbild-Editor, der lineare und nichtlineare Funktionsblöcke enthält, modelliert, da ein Blockschaltbild der grundlegende Abstraktionsmechanismus der Regelungstechnik ist. Verfügbare Blockschaltbild-Editoren sind z.B. EASY-5 [Easy88], SIMULINK [Math92] oder SYSTEM_BUILD [Shah85].

Physikalische Effekte, die direkt durch mathematische Gleichungen beschrieben werden, und bei denen ereignisabhängige Unstetigkeiten auftreten können, werden am besten mittels einer *allgemeinen Simulationssprache*, wie dem "Industriestandard" ACSL [Mitc91], Desire [Korn89] oder SIMNON [Elmq75, Elmq90] modelliert.

Alle diese Programmpakete haben den Nachteil, daß Komponenten außerhalb des vorgesehenen Fachgebiets *nicht*, oder nur sehr unzureichend, modelliert werden können. Zum Beispiel sind die Systeme ACSL oder SIMULINK vollkommen ungeeignet um die Mechanik eines 6-freiheitsgradigen Roboters zu modellieren, da diese Pakete die *Ein/Ausgangsgleichungen* eines Modells als Eingabe fordern. Die Hauptschwierigkeit besteht aber gerade in der Ermittlung dieser Gleichungen. Für eine solche Aufgabenstellung wurden die Mehrkörperprogramme entwickelt. Andererseits gibt es bisher kein verfügbares, allgemeines Mehrkörperprogramm das es erlauben würde, *diskrete* Komponenten zu simulieren, wie z.B. den mit einem Mikroprozessor realisierten Regler eines Roboters. Diese Aufgabe wäre dagegen mit ACSL oder SIMULINK sehr einfach und übersichtlich zu lösen. Schließlich sei noch darauf hingewiesen, daß die von einer Anwendung abstrahierten *allgemeinen* Simulationspakete, wie ACSL, SIMULINK oder SYSTEM_BUILD, in der Regel nur *explizite Differentialgleichungen* lösen können. In den Ingenieurdisziplinen, wie Mechanik, Elektrotechnik Hydraulik, werden Systeme jedoch oft in "natürlicher" Weise durch differential-*algebraische* Gleichungen beschrieben. In diesem Fall sind die "allgemeinen" Pakete nicht direkt anwendbar, es sei denn, daß es durch Umformungen gelingt, das Problem auf explizite Differentialgleichungen zurückzuführen.

Zusammenfassend kann festgehalten werden, daß es in vielen Anwendungen, wie z.B. in der Robotik oder der Fahrzeugtechnik (ABS-Bremse, aktive Federung etc.) unumgänglich geworden ist, Komponenten unterschiedlicher Fachgebiete *gemeinsam* zu simulieren. Hierzu gibt es bislang jedoch keine geeigneten Programmpakete. Man behilft sich zur Zeit mit problemspezifischen ad-hoc Lösungen.

H. Elmqvist hat in einer grundlegenden Arbeit [Elmq78] schon 1978 einen gangbaren Weg zur multidisziplinären Modellierung aufgezeigt und mit der hierfür entwickelten ersten *objektorientierten* Modellierungssprache Dymola eine rechnergestützte Modellbildungsumgebung realisiert. Das Grundprinzip besteht darin, physikalische Systeme möglichst direkt auf Modellobjekte abzubilden. Dabei wird keine Kausalität festgelegt, sondern es werden nur *Beziehungen* zwischen Objekten, bzw. beschreibenden Variablen, definiert. Objekte können entsprechend der *physikalischen* Verkopplungen miteinander verschaltet werden. Dies wird unter anderem durch die Unterstützung von "Across"- und "Through"-Variablen [Cann67] erreicht. Aus Anwendersicht ist damit eine "natürliche" Modellierung möglich, die dem jeweiligen Fachgebiet angepaßt ist. Die gemeinsamen Eigenschaften von Grundkomponenten eines Fachgebietes können in wiederverwendbaren Bibliotheken gespeichert und dann in entsprechenden Modellen eingesetzt werden[1].

Nach der Modelldefinition eines Systems wird festgelegt, welche Aufgabenstellung vorliegt, d.h. welche Größen bekannt und welche unbekannt sind. Aufgrund des deklarativ beschriebenen Modells und der gegebenen Aufgabenstellung, werden die Modellgleichungen dann automatisch in eine geeignete, numerisch effizient ausführbare Form transformiert,

[1] Bis jetzt fehlten jedoch Modell-Bibliotheken, deren Leistungsumfang mit fachgebietsspezifischen Programmpaketen vergleichbar ist. Es gab nur kleine Experimental-Bibliotheken für einfache 1-dimensionale Systeme. Cellier hat einige davon in [Cell91] beschrieben.

z.B. bei Simulationsproblemen in die Zustandsform. Nachdem ein Modell in die Zustandsform überführt wurde, ist es nicht mehr schwierig, die Zustandsgleichungen in einer für verschiedene *Simulationsumgebungen* spezifischen Form bereitzustellen.

Mit der obigen Vorgehensweise wird eine klare Dreiteilung einer Simulationsaufgabe erreicht: Im Rahmen der *Modellierung* wird ein Modell in einer anwendernahen Form spezifiziert. Die *Modellbildung* hat die Aufgabe, eine solche Definition mittels entsprechender Transformationsalgorithmen in eine effizient auswertbare Standardform (z.B. die Zustandsdarstellung) zu überführen. In einer *Simulations*umgebung wird diese Standardform mit einem Integrator numerisch ausgewertet, d.h. "gelöst".

In Kapitel 2 und in Kapitel 3 werden die Eigenschaften und Algorithmen zusammengestellt, um rein kontinuierliche, dynmische Systeme, beziehungsweise unstetige und strukturvariable Systeme, objektorientiert modellieren zu können. Diese beiden Kapitel bilden die Grundlage für die darauf folgenden fachgebietsspezifischen Abschnitte.

In den Kapiteln 4, 5, 6 werden bekannte Modellierungsverfahren aus drei wichtigen Bereichen der Mechatronik, (Mehrkörpersysteme, Antriebsstränge und regelungstechnische Systeme), in eine objektorientierte Beschreibungsform umformuliert. Dabei werden nur die *lokalen* Eigenschaften von Komponenten durch Gleichungen beschrieben. Die (physikalische) Verschaltung dieser Komponenten und die Transformation auf Zustandsform wird mit den fachgebietsunabhängigen Werkzeugen der objektorientierten Modellierungssprachen vorgenommen. Hiermit wird es möglich, Modelle aus diesen Fachgebieten anwendernah in einer objektorientierten Modellierungssprache zu erstellen. Eine neue Vorgehensweise bei der Modellbildung *strukturvariabler* Systeme erlaubt darüber hinaus die effiziente und numerisch sichere Behandlung von z.B. mechanischen Systemen, die durch eine große Zahl möglicher Konfigurationen bestimmt sind. Dies wird an einem Roboter demonstriert, der aufgrund reibungsbehafteter Gelenke 64 unterschiedliche Modellstrukturen besitzt, die jeweils durch eine unterschiedliche Zahl von Zustandsgrößen beschrieben werden.

Um die Leistungsfähigkeit der objektorientierten Modellierung und Modellbildung zu demonstrieren, werden die vorgestellten Methoden im Kapitel 7 auf ein komplexes mechatronisches System, den Industrieroboter Manutec r3, angewendet. Die Modellparameter dieses Roboters, wie Massen, Trägheitsmomente, Reibkennlinien, Motorkenndaten, wurden bei der DLR durch Einzelmessungen ermittelt. Im Modell werden alle wichtigen physikalischen Effekte erfaßt: die Mehrkörperdynamik; die Antriebsstränge inklusive Reibung, Elastizität, Dämpfung und Lose; die Dynamik der Motoren inklusive Stromregler; die Tachofilter und die Kaskadenregler. Alle Komponenten werden in ihrer "natürlichsten" Form modelliert und entsprechend der physikalischen Verkopplungen und der informatorischen Wirkflüsse zusammengeschaltet. Danach wird das Gesamtmodell mit den besprochenen Algorithmen automatisch in den Zustandsraum transformiert. Die Verifizierung des Modells erfolgt anhand ausgewählter Simulationen, die mit Messungen verglichen werden.

1.2 Modellierung aus Sicht verschiedener Fachdisziplinen

In diesem Abschnitt werden Methoden und Programmpakete diskutiert, die in unterschiedlichen Fachdisziplinen zur Modellierung eingesetzt werden. Diese Zusammenstellung zeigt die Stärken und Schwächen, sowie die Gemeinsamkeiten verschiedener Modellierungstechniken und bildet eine Grundlage zur Einschätzung der darüber hinausgehenden Möglichkeiten objektorientierter Modellierungssprachen.

Modellbeschreibung mit Simulationssprachen

Es wurde schon sehr frühzeitig erkannt, daß allgemeine Werkzeuge zum fachgebietsübergreifenden Modellieren benötigt werden. Hierzu wurden die *allgemeinen Simulationssprachen* entwickelt. Bekannte kommerzielle Systeme sind ACSL [Mitc91], Desire [Korn89] oder SIMNON [Elmq75, Elmq90]. Allgemeine Simulationssprachen setzen voraus, daß die komponentenbeschreibenden *Gleichungen* gegeben sind, und daß diese Gleichungen durch einfaches Sortieren in eine korrekte Abarbeitungsreihenfolge gebracht werden können. Der Vorteil dieser Sprachen besteht darin, daß die Gleichungen sehr direkt eingegeben werden können, und daß ein Codegenerator das oft mühsame Sortieren der Gleichungen übernimmt. Die Modelleingabe ist stark an den Simulationsablauf geknüpft. Zum Beispiel hat ein ACSL Modell die folgende Struktur (Schlüsselwörter von ACSL sind **fettgedruckt**):

PROGRAM
  INITIAL
    Anweisungen, die *vor* einer Integration ausgeführt werden.
  END

  DYNAMIC
    DERIVATIVE
      Anweisungen, die *immer* ausgeführt werden.
    END
    DISCRETE
      Anweisungen, die *nur* an Ereignispunkten ausgeführt werden.
    END
    Anweisungen, die *nur* an Kommunikationspunkten ausgeführt werden.
  END

  TERMINAL
    Anweisungen, die *nach* einer Integration ausgeführt werden.
  END
END

Mit speziellen Anweisungen (z.B. `SCHEDULE`-Anweisung in ACSL) können Ereignisse definiert werden. Wenn ein Ereignis eintritt, wird ein spezielles Codestück in der Discrete-Section ausgeführt. Danach wird die Integration neu gestartet. Damit können Unstetigkeiten im Modell oder diskrete Systeme, wie z.B. eine Verschaltung von kontinuierlicher Strecke und diskretem Regler, auf numerisch sichere Art behandelt werden.

Nach der Modelleingabe wird die Modelldefinition üblicherweise in ein Unterprogramm einer Programmiersprache umgesetzt, welches mit einem entsprechenden Compiler in Maschinencode übersetzt wird. Das übersetzte Unterprogramm wird in eine Simulationsumgebung

eingebunden, mit der Simulationsexperimente durchgeführt und die Ergebnisse eines Simulationslaufes grafisch veranschaulicht werden können.

Die Verwendung einer allgemeinen Simulationssprache erleichtert die Modell*implementierung* auf einem Rechner, sowie das Ausführen von *Simulationsexperimenten*. Nachteilig ist, daß die *Gleichungen* der zu simulierenden Komponenten so eingegeben werden müssen, daß ein Modell durch einfaches Sortieren der Anweisungen in eine Zustandsraumbeschreibung überführt werden kann. Bei fast allen *physikalischen* Systemen, z.B. in der Mechanik oder in der Elektrotechnik, besteht aber bei größeren Systemen eine Hauptschwierigkeit gerade darin, diese Gleichungen aus der (anwendernahen) Modelldefinition abzuleiten.

Modellbeschreibung mit Blockschaltbild-Editoren

Ähnlich wie die allgemeinen Simulationssprachen, sollen Blockschaltbild-Editoren und -Simulatoren eine fachgebietsübergreifende Modellbildung und Simulation unterstützen. Bei solchen Systemen wird jedoch wesentlich mehr Wert auf die Benutzerfreundlichkeit gelegt. Der zentrale Teil besteht aus einem grafischen Editor um Ein/Ausgangsblöcke grafisch einzugeben und zu verbinden. Bekannte kommerzielle Systeme sind EASY-5 [Easy88], SIMULINK [Math92] und SYSTEM_BUILD [Shah85].

Blockschaltbild-Editoren gehen von der informatorischen Sicht rückwirkungsfreier Ein/Ausgangsblöcke aus, wie sie in der Regelungstechnik verwendet werden. Üblicherweise wird eine große Zahl vordefinierter Funktionsblöcke zur Verfügung gestellt, z.B. Blöcke, die durch eine Übertragungsfunktion beschrieben werden, und Signalgenerator-Blöcke. Bei manchen Systemen ist es möglich, daß Anwender neue Blocktypen einführen können indem die Gleichungen des Blocks in einer Programmiersprache, wie C, oder einer speziellen Sprache, wie Matlab, bereitgestellt werden.

Funktionell sind allgemeine Simulationssprachen und Blockschaltbild-Editoren sehr ähnlich. Ein Block entspricht einem Satz von Gleichungen mit definierten Ein- und Ausgangsgrößen, sowie Zustandsvariablen. Bevor simuliert werden kann, wird die Abarbeitungsreihenfolge der Blöcke durch (automatisches) *Sortieren* ermittelt. Dies entspricht dem Sortieren der Gleichungen bei den Simulationssprachen. Es gibt Schwierigkeiten, wenn algebraische Schleifen auftreten, d.h. wenn keine eindeutige Sortierreihenfolge bestimmt werden kann. Die näherungsweise Lösung solcher Probleme, z.B. durch einen speziellen Iterationsblock oder durch eine zusätzliche Dynamik in der Schleife, ist oft wenig befriedigend.

Der Vorteil von Blockschaltbild-Editoren ist der Komfort bei der Eingabe und bei der Simulation. Auch ein unerfahrener Benutzer kann rasch ein Modell erstellen und simulieren. Nachteilig ist, ähnlich wie bei allgemeinen Simulationssprachen, daß es meist schwierig und aufwendig ist, ein *physikalisches* Systemmodell in ein Blockschaltbild zu überführen. Regelungstechnische Modelle liegen meist direkt als Blockschaltbilder vor, sodaß dies der Hauptanwendungsbereich von Blockschaltbild-Editoren ist. Große, detaillierte Systemmodelle sind in Blockschaltbilddarstellung unübersichtlich; ein "Zoomen" in unterschiedliche Detaillierungsebenen kann schnell unübersichtlich werden. Häufig werten Blockschaltbild-Editoren, wie z.B. SIMULINK, die Modellgleichungen während einer Simulation interpretativ aus, und sind deswegen für größere Modelle in der Regel deutlich langsamer als Systeme die Modelle in Maschinencode übersetzen, wie z.B. ACSL oder EASY-5.

Formalisierte Modellbildung elektrischer Schaltungen

Aufgrund ihrer technischen Bedeutung gab es schon sehr früh spezielle Systeme zur Simulation elektrischer Schaltungen. Das, auch heute noch, wichtigste Programmpaket für die analoge Schaltungssimulation ist SPICE, das 1975 von Nagel [Nage75] entwickelt wurde und kostenlos als "public domain" Programm weitergegeben wird. Die Bedeutung von SPICE ergibt sich vor allem daraus, daß die Hersteller von integrierten Schaltungen zu ihren Hardware-Komponenten entsprechende SPICE-Modelle mitliefern. Das heißt, ein elektrischer Baustein kann sofort in einer SPICE-Schaltung simuliert werden.

SPICE-Modelle werden durch Angabe vordefinierter Schaltungselemente, den Parametern dieser Elemente, sowie deren Verschaltung definiert. Als Beispiel ist in Bild 1.1 die Definition eines einfachen elektrischen Schaltkreises im SPICE-Format angegeben. In der ersten

```
R1   1   2     100
R2   1   3      20
C    2   0   1.0E-6
L    3   0   1.5E-3
V    1   0     220   0
```

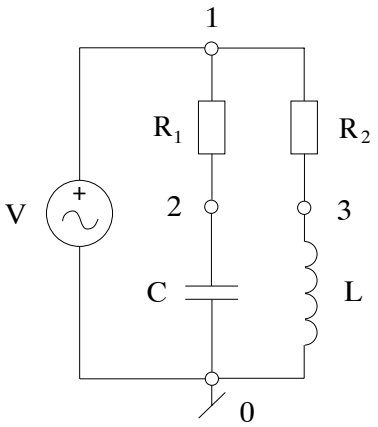


Bild 1.1: Definition eines elektrischen Schaltkreises im SPICE Format.

Spalte steht der Elementname. Dieser muß mit einem Kennbuchstaben für den Typ des Elements beginnen (der Kennbuchstabe ist fettgedruckt). In der zweiten und dritten Spalte ist angegeben, zwischen welchen Knotennummern das Element angebracht ist. In den letzten Spalten sind die aktuellen Parameterwerte für das jeweilige Element aufgeführt (z.B. der Widerstands-Wert).

Zu SPICE werden eine große Anzahl (kommerzieller) Pre- und Postprozessoren angeboten. Mit solchen Programmpaketen wird ein elektrischer Schaltkreis grafisch definiert, indem Icons für elektrische Bauteile auf dem Bildschirm plaziert und entsprechend verbunden werden. Im Gegensatz zu einem Blockschaltbild-Editor, sind hier die zu verbindenden Komponenten *keine* rückwirkungsfreien Blöcke, sondern elektrische Bauelemente mit Energieflüssen. Ausgehend von der grafischen Definition wird eine Datei erzeugt, die das Modell im SPICE Format enthält. Nach einer Simulation mit SPICE können die Simulationsergebnisse mittels Postprozessoren komfortabel angezeigt werden.

Mit der Definition der benutzten Elemente, sowie deren (physikalischer) Verschaltung, erzeugt SPICE automatisch die Gleichungen eines Schaltkreises mit der *modifizierten Knotenpunktsanalyse* (siehe z.B. [Chua87, McCa88]). Die Gleichungen der elektrischen Komponenten werden *zuerst* diskretisiert und *danach* zusammengeschaltet, d.h. es werden die Kirchhoff'schen Regeln angewandt. Dabei entsteht ein großes, dünnbesetztes, nichtlineares Gleichungssystem, das bei jedem Integrationsschritt gelöst werden muß. Die Unbekannten in diesem Gleichungssystem sind die Potentiale der Knoten, sowie Spannungen und Ströme von aktiven Elementen.

Um einen Eindruck von der Gleichungsstruktur zu geben, wird noch kurz das *"Sparse-Tableau"* Verfahren besprochen, welches in einigen elektrischen Schaltungsprogrammen eingesetzt wird (siehe z.B. [Chua87, McCa88]). Hier werden die Potentiale $\mathbf{v}$ aller Knoten, die Spannungsabfälle $\mathbf{u}$ über alle Elemente, und die Ströme $\mathbf{i}$ durch alle Elemente als Unbekannte verwendet. Jeder Schaltkreis kann dann durch ein Gleichungssystem mit folgender Struktur beschrieben werden:

$$\mathbf{0} = \mathbf{A}\,\mathbf{i}(t) \tag{1.1a}$$

$$\mathbf{u}(t) = \mathbf{A}^T\,\mathbf{v}(t) \tag{1.1b}$$

$$\mathbf{0} = \mathbf{h}\,(\frac{d\mathbf{u}(t)}{dt}, \mathbf{u}(t), \frac{d\mathbf{i}(t)}{dt}, \mathbf{i}(t), t)\ . \tag{1.1c}$$

Die konstante Matrix $\mathbf{A}$ ist die reduzierte Inzidenzmatrix, die die Verschaltungsstruktur eines Schaltkreises widergibt[2]. Gleichung (1.1a) beruht auf der Kirchhoff'schen Stromregel und gibt an, daß die Summe der Ströme in einem Knoten Null ist. Gleichung (1.1b) berechnet die Spannungsabfälle über alle Elemente mit Hilfe der Knotenpotentiale, und Gleichung (1.1c) enthält die *lokalen* Gleichungen der Elemente (z.B. Ohm'sches Gesetz). Das Gleichungssystem (1.1) ist ein großes, dünnbesetztes, nichtlineares differential-algebraisches Gleichungssystem. In elektrischen Schaltungsprogrammen ist es unüblich, das System auf Zustandsform zu transformieren. Stattdessen wird z.B. das differential-algebraische Gleichungssystem (1.1) direkt mit einem Spezialintegrator gelöst.

Elektrische Schaltungsprogramme sind hervorragend geeignet um große, elektrische Schaltungen zu modellieren und zu simulieren. Es ist in eingeschränktem Umfang möglich, auch nicht-elektrische Komponenten zu modellieren, indem ein Systemmodell zuerst in einen äquivalenten elektrischen Schaltkreis überführt wird und dieser Schaltkreis dann simuliert wird. Regelungstechnische Ein/Ausgangsblöcke können nicht modelliert werden. Weiterhin ist es bei diesen Programmen auch nicht üblich, ereignisabhängige Modelle zu unterstützen, z.B. die Verschaltung eines digitalen Reglers mit einer kontinuierlichen Strecke. Die in der Praxis auftretenden schaltenden Elemente, wie Diode oder Thyristor werden durch sehr genaue stetige Modelle beschrieben, bei denen das ideal-schaltende Verhalten "abgeschliffen" ist. Zum Beispiel wird eine Diode als Widerstand mit nichtlinearer Kennlinien beschrieben, die bei "offener" Stellung einen großen und bei geschlossener Stellung einen kleinen Widerstandswert besitzt. Dies führt selbst bei einfachen Schaltkreisen schnell auf *steife* Differentialgleichungen und damit zu hohen Rechenzeiten.

Formalisierte Modellbildung mechanischer Systeme

Systeme, wie Roboter, Satelliten, Fahrzeuge, können durch idealisierte mechanische Modelle beschrieben werden. Wenn die Verformungen von Körpern vernachlässigt werden, wird für die Modellbildung die Theorie der starren Mehrkörpersysteme (abgekürzt MKS) verwendet. Es gibt eine Vielzahl von Mehrkörperprogrammen, die sich in ihrer Benutzerfreundlichkeit, der Funktionalität und der Art der Gleichungserstellung unterscheiden. Bekannte kommerzielle Pakete sind ADAMS, DADS [Smit90], NEWEUL [Kreu90], SD-FAST [Rose86, Rose87], SIMPACK [Rulk90].

[2] A_{ij} ist $+1/-1$ wenn der j-te Zweig den i-ten Knoten verläßt bzw. in den i-ten Knoten eintritt. Ansonsten ist das Element Null.

Im Gegensatz zu elektrischen Schaltungsprogrammen, hat sich bei MKS-Programmen keine Standardeingabe durchgesetzt. Die zugrunde liegende Abstraktion ist jedoch immer dieselbe: Ein Mehrkörpersystem wird in *starre Körper*, *Gelenke* (auch Übertragungsmechanismen genannt) und *Kraftelemente* aufgeteilt. Ein starrer Körper hat Trägheitseigenschaften, wie Masse und Trägheitstensor. Weiterhin können auf einem Körper spezielle Punkte markiert werden, an denen Gelenke oder Kraftelemente angebracht werden können. Gelenke sind ideale Elemente, die die Bewegungsmöglichkeiten von Körpern einschränken. Zum Beispiel erlaubt ein einachsiges Drehgelenk nur die Rotation um eine Achse eines Körpers 1 um einen Körper 2. Kraftelemente sind ideale Elemente, die aufgrund von Relativbewegungen eine Kraft und/oder ein Moment erzeugen und diese Kraft, bzw. dieses Moment, zwischen zwei Körperpunkten angreifen lassen (z.B. eine Feder).

Zur Modellierung wird ein mechanisches System in Elemente dieser drei Grundtypen aufgeteilt. Die Elemente werden dann entsprechend der physikalischen Verschaltungsstruktur zusammengeschaltet. Mit Hilfe mechanischer Prinzipien, z.B. den Lagrange'schen Gleichungen oder dem Prinzip der virtuellen Arbeit, werden MKS auf ein Gleichungssystem der folgenden Struktur transformiert:

$$\dot{\mathbf{p}}(t) = \mathbf{v}(t) \tag{1.2a}$$

$$\mathbf{M}(\mathbf{p}(t),t)\,\dot{\mathbf{v}}(t) = \mathbf{h}(\mathbf{p}(t),\mathbf{v}(t),t) + \mathbf{G}^T(\mathbf{p}(t),t)\,\boldsymbol{\lambda}(t) \tag{1.2b}$$

$$\mathbf{0} = \mathbf{g}(\mathbf{p}(t),t) \qquad (\mathbf{G} = \frac{\partial \mathbf{g}}{\partial \mathbf{p}})\ . \tag{1.2c}$$

Hierbei sind $\mathbf{p}$ die verallgemeinerten Lagekoordinaten mit denen die Lage eines MKS eindeutig gekennzeichnet wird (z.B. die absoluten Koordinaten eines Punktes auf jedem Körper oder spezielle Gelenkkoordinaten, wie der Drehwinkel eines Drehgelenks). Die erste zeitliche Ableitung der Lagekoordinaten $\mathbf{p}$ sind die verallgemeinerten Geschwindigkeiten $\mathbf{v}$. Sind die Lagekoordinaten nicht unabhängig voneinander, so bestehen zwischen ihnen algebraische Beziehungen (1.2c). Dies bewirkt zusätzliche, unbekannte Zwangskräfte $\boldsymbol{\lambda}$. Enthält ein System kinematische Schleifen, kann man, von Ausnahmen abgesehen, keine voneinander unabhängigen Lagekoordinaten angeben und Gleichung (1.2c) ist mit zu berücksichtigen.

Das Gleichungssystem (1.2) ist ein unangenehmes differential-algebraisches Gleichungssystem, da es keine direkte Gleichung für $\boldsymbol{\lambda}$ gibt. Die Zwangskräfte sind nur indirekt durch Gleichung (1.2c) gegeben. Dieses Gleichungssystem ist ein sogenanntes "höheres Index System", welches nicht mit Standardintegrationsmethoden gelöst werden kann. In Kapitel 2.4 ab Seite 31 werden solche Arten von Gleichungssystemen näher behandelt.

Bislang muß man bei der Verwendung von Mehrkörperprogrammen immer einen Kompromiß eingehen: Werden MKS-Programme wie ADAMS oder DADS benutzt, so kann man fast jedes rein kontinuierliche MKS modellieren und simulieren. Die Rechenzeiten sind in der Regel aber sehr hoch. Mit speziellen MKS-Programmen kann mit deutlich geringeren Rechenzeiten simuliert werden, aber bei einem unter Umständen größeren Definitionsaufwand des Modells (z.B. mit dem Programmpaket Mobile [Kecs88, Kecs93b]).

Generell unbefriedigend ist die Situation, wenn unstetige Elemente, wie digitale Regler, Reibung oder Lose, behandelt werden müssen, oder wenn Stöße zu modellieren sind. Kommerzielle Programmpakete bieten hier kaum Unterstützung an. Für spezielle Anwendungen gibt es dazu an den Hochschulen Programme im Experimentierstadium.

Wie bei elektrischen Schaltungsprogrammen, ist es auch mit Mehrkörperprogrammen nur

bedingt möglich, Komponenten aus anderen Fachgebieten zu modellieren und zu simulieren. Üblicherweise wird von einem MKS-Programm eine Unterprogrammschnittstelle für Kraftelemente angeboten. Da ein Kraftelement auch Differentialgleichungen enthalten kann, ist es im Prinzip möglich, nicht-mechanische Komponenten zu modellieren, indem die rechte Seite einer Komponenten-Differentialgleichung als Unterprogramm zur Verfügung gestellt wird. Wegen des geringen Komforts, ist eine solche Lösung allenfalls bei kleinen Systemmodellen akzeptabel.

Meta-Modellbildung mit Bondgraphen

Die Bondgraph Methode wurde 1961 von Paynter [Payn61] eingeführt und vor allem von Karnopp und Rosenberg [Rose83, Karn90] weiterentwickelt. Wichtige neue Ergebnisse, wie Multi-Bondgraphen, stammen von Breedveld [Bree84]. Eine verständliche und kompakte Einführung ist in [Cell91] zu finden. Mit der Bondgraph Methode wird versucht, eine einheitliche Beschreibungsform *physikalischer* Systeme zu erhalten, die unabhängig von einer bestimmten Fachgebietssicht ist.

Ein grundlegendes Merkmal physikalischer Systeme ist das Fließen von Energie (= Leistung), wobei an jeder Stelle in einem System der Energieerhaltungssatz gültig ist. Erstaunlicherweise wird "Leistung" in jedem Fachgebiet durch ein Produkt zweier charakteristischer Größen dieses Fachgebiets gebildet. In der Bondgraph Literatur werden diese beiden Größen als *Effort* Variable e und *Flow* Variable f bezeichnet. Die an einer Schnittstelle des Systems fließende Energie P ist das Produkt dieser beiden Variablen, wobei e und f den Systemzustand an der Schnittstelle kennzeichnen. In Tabelle 1.1 (aus [Cell91]) sind für einige Fachgebiete die Effort- und Flow-Variablen angegeben. Der Energiefluß in einem

	Effort	Flow
elektrisch	Potential	Strom
mechanisch	Kraft	Geschwindigkeit
hydraulisch	Druck	Volumenstrom
chemisch	chemisches Potential	molarer Fluß
thermodynamisch	Temperatur	Entropiefluß

Tabelle 1.1: Effort- und Flow-Variablen physikalischer Systeme.

System wird durch einen Halbpfeil gekennzeichnet, wobei auf der einen Seite die Effort-Variable und auf der anderen Seite die Flow-Variable der fließenden Energie angetragen ist, siehe Bild 1.2. In jedem physikalischen System gibt es Komponenten, die Energieströme aufteilen, speichern oder umwandeln. In der Bondgraph Methodik werden dafür einige Grundkomponenten abstrahiert, die in jedem Fachgebiet anzutreffen sind. Solche Elemente sind in Bild 1.2 zusammen mit ihrer grafischen Darstellung angegeben. Die *0-* und *1-Junction* entsprechen einem Energieverteiler, wobei entweder die Effort- oder die Flow-Variable konstant gehalten wird. Die jeweils korrespondierende Variable ergibt sich

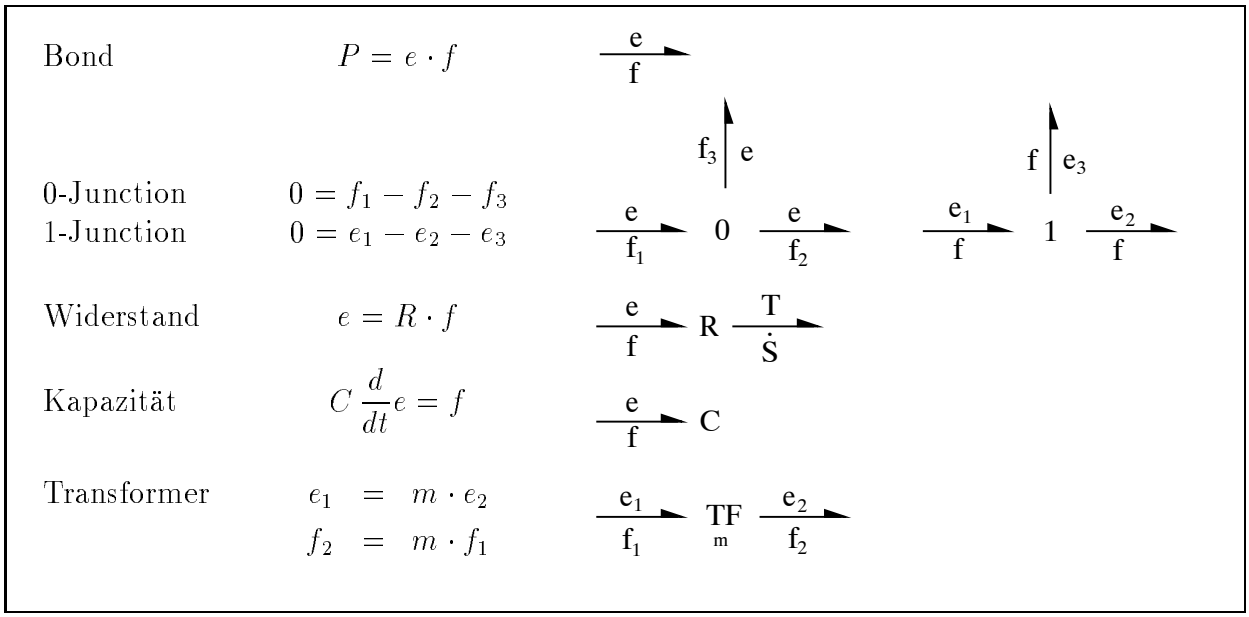


Bild 1.2: Elemente von Bondgraphen.

automatisch aus dem Energieerhaltungssatz. Typische weitere Elemente sind der *Widerstand* und die *Kapazität*. Der Widerstand ist ein Element, das Energie in Wärme umsetzt, z.B. als elektrischer Widerstand oder als viskose Reibung. Der auf der rechten Seite angetragene Halbpfeil beim Widerstand in Bild 1.2 gibt den Wärmestrom an. Hierbei ist T die Temperatur des Widerstandes und $\dot{S}$ der Entropiestrom. Die Kapazität ist eine ideale, verlustlos arbeitende Komponente, die Energie speichert, z.B. als elektrischer Kondensator oder als Feder. Schließlich verteilt der *Transformer* die Energie auf neue Weise auf die Effort- und Flow-Variable, z.B. als elektrischer Transformator oder als Getriebe.

Als Beispiel für einen Bondgraph ist in Bild 1.3 noch einmal der elektrische Schaltkreis von Seite 6 zusammen mit seinem Bondgraph gezeigt. Die Grobstruktur des Bondgraphs entspricht der Verschaltungsstruktur des elektrischen Kreises. Da keine Energie in denjenigen Leitungen fließt, die mit dem Nullpotential verbunden sind ($P = e \cdot f$ und $e = 0$), können

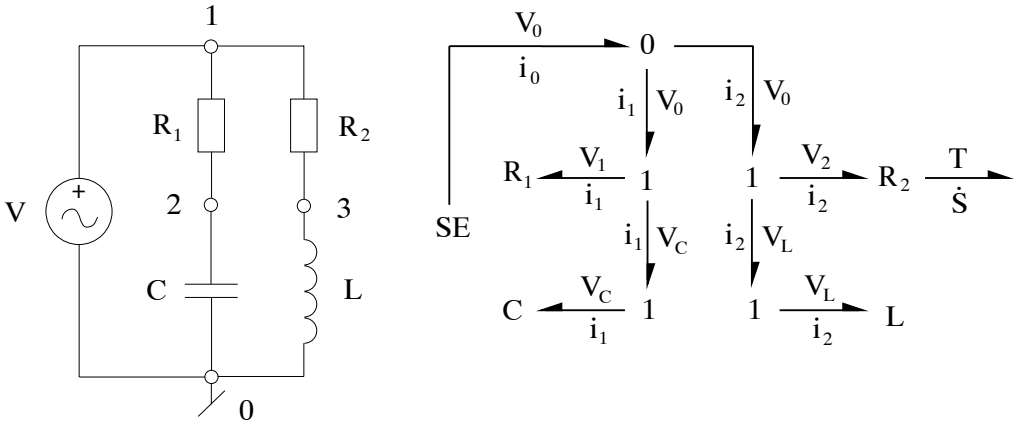


Bild 1.3: Bondgraph eines einfachen elektrischen Schaltkreises.

die entsprechenden Bonds weggelassen werden. Der Bondgraph macht z.B. deutlich, daß ein Teil der Energie in den Widerstand R_2 und ein Teil zum nächsten Element fließt. Die in den Widerstand fließende Energie ist verloren, da sie in Wärme umgesetzt und über den

rechten Halbpfeil des Widerstandes R_2 in die Umgebung abgeführt wird (aus Platzgründen wurde dieser Energiefluß am Widerstand R_1 weggelassen). In der Kapazität und in der Induktivität wird die Energie gespeichert. Der Bondgraph könnte noch vereinfacht werden, in dem die 1-Junctions bei der Kapazität und der Induktivität weggelassen werden.

Die Bondgraph Methode hat den Vorteil, daß physikalische Systeme mit einer einheitlichen Notation und Philosophie behandelt werden. Durch die abstrahierten Grundkomponenten ergeben sich die bekannten Analogien zwischen Fachgebieten, die ein Einarbeiten in eine andere Disziplin erleichtern (z.B. Temperatur $\Leftrightarrow$ elektrisches Potential, Entropie $\Leftrightarrow$ elektrische Ladung). Wenn in einem Modell mehrere Fachgebiete eine Rolle spielen, hilft die Bondgraph Methode dabei, die Übersicht zu behalten und auf den Energiesatz zu achten.

Leider gibt es bislang keine genügend mächtigen Programmpakete, um die Bondgraph Methode auf größere Systeme anwenden zu können. Dies liegt sicher auch mit daran, daß es im allgemeinen schwierig ist ein Bondgraph Modell in ein Zustandsraummodell zu überführen. Im Gegensatz zu einem Block beim Blockschaltbild, haben die Elemente eines Bondgraphs keine eingebaute (Ein/Ausgangs-) Kausalität. Erst aus der Verschaltung ergibt sich, nach welcher Variablen eine Gleichung aufgelöst wird. Weiterhin können hier auch strukturell singuläre Systeme auftreten (siehe Kapitel 2.4), was bei Blockschaltbildern nicht möglich ist. Ein Problem in der Anwendung von Bondgraphen ist immer auch, daß die meisten technischen Systeme *nicht nur* aus energiedurchströmten Komponenten bestehen. Zum Beispiel kann ein digitaler Regler nicht als Bondgraph beschrieben werden, da mit diesem Element kein Energiestrom, sondern ein Informationsstrom, verbunden ist.

Modellbildung mit objektorientierten Programmiersprachen

In den letzten Jahren wurden neue *Programmier*sprachen entwickelt, die durch objektorientierte Sprachelemente eine wesentlich mächtigere Art der Programmierung erlauben als rein prozedurale Sprachen. Bisher war es unkomfortabel, die Gleichungen eines Modells ohne Hilfsmittel direkt in einer Programmiersprache zu implementieren. Eine objektorientierte Programmiersprache wie C++ [Stro91] verspricht hier Abhilfe.

Dies hat insbesondere Kecskeméthy [Kecs88, Kecs93, Kecs93b] mit der Realisierung des "Baukastensystems" Mobile gezeigt, mit dem komplexe Mehrkörpersysteme modelliert werden können. In Mobile werden vordefinierte Klassen in der Programmiersprache C++ zur Verfügung gestellt, um mechanische Systeme zu beschreiben. Ein Mehrkörpersystem wird mit Hilfe von Objekten dieser Klassen hierarchisch aufgebaut. Mit ebenfalls zur Verfügung gestellten Operationen auf ein Mehrkörpersystem-Objekt, können unterschiedliche Aufgabenstellungen, wie z.B. ein Simulationsexperiment, behandelt werden.

Im Gegensatz zu sonstigen Mehrkörperprogrammen wird hier keine Eingabedatei erstellt um ein System zu definieren, sondern es wird ein anwendungsspezielles C++ Programm geschrieben. Mit den zur Verfügung gestellten Objekt-Klassen ist dies einfach möglich. Diese Vorgehensweise verbindet Flexibilität mit Komfort. Zum Beispiel kann ein so definiertes System leicht um Komponenten erweitert werden, die der Entwickler nicht vorgesehen hatte, was bei sonstigen (numerischen) Mehrkörperprogrammen in diesem Umfang nicht möglich ist. Weiterhin kann das Programm auch leicht in andere C++ Umgebungen eingebettet werden.

Mit der *direkten* Verwendung einer Programmiersprache gibt es jedoch immer Probleme, wenn unterschiedliche, bereits bestehende Komponenten zusammengeschaltet werden sollen. Zur Verdeutlichung sind in Bild 1.4 schematisch zwei Systeme gezeigt, die beide mit derselben Programmiersprache implementiert sind, in Zustandsraumdarstellung vorliegen, und zusammengeschaltet werden sollen. Beispielsweise könnte Block 1 ein Mehrkörpersystem und Block 2 ein Regler für das Mehrkörpersystem sein. Ein Integrator erwartet, daß bei

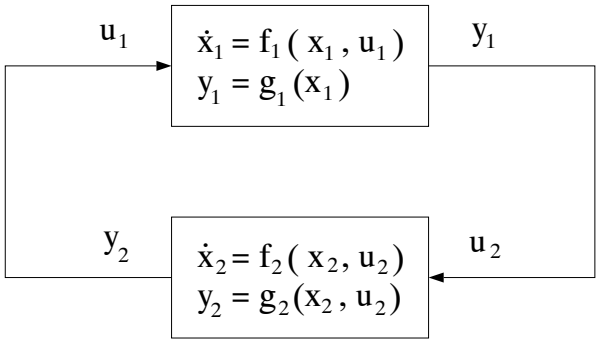


Bild 1.4: Probleme bei der Verschaltung von Modellen.

gegebenen Zustandsgrößen $\mathbf{x}_1, \mathbf{x}_2$ die Ableitungen der Zustandsgrößen berechnet werden. Wenn zur Berechnung der Gleichungen eines Blocks jeweils *ein* Unterprogramm vorliegt, so können die gesuchten Größen *nicht* berechnet werden, da Block 1 die Größe $\mathbf{u}_1$ benötigt, diese aber von Block 2 berechnet wird, der wiederum die zu berechnende Größe $\mathbf{y}_1$ vom Block 1 braucht. Die Berechnung kann nur durchgeführt werden, wenn die Gleichungen der beiden Blöcke auf mehrere Unterprogramme verteilt sind, die dann in der folgenden Reihenfolge aufgerufen werden (gegeben ist $\mathbf{x}_1, \mathbf{x}_2$):

$$
\begin{aligned}
\mathbf{y}_1 &= \mathbf{g}_1(\mathbf{x}_1) \\
\mathbf{u}_2 &= \mathbf{y}_1 \\
\mathbf{y}_2 &= \mathbf{g}_2(\mathbf{x}_2, \mathbf{u}_2) \\
\mathbf{u}_1 &= \mathbf{y}_2 \\
\dot{\mathbf{x}}_1 &= \mathbf{f}_1(\mathbf{x}_1, \mathbf{u}_1) \\
\dot{\mathbf{x}}_2 &= \mathbf{f}_2(\mathbf{x}_2, \mathbf{u}_2)
\end{aligned}
$$

Eine solche Vorgehensweise ist aber schon nicht mehr möglich, wenn auch der Ausgang $\mathbf{y}_1$ direkt vom Eingang $\mathbf{u}_1$ abhängt. Dann könnte eine algebraische Schleife vorliegen, muß aber nicht. Dies kann nur durch eine Untersuchung der Feinstruktur der Gleichungen festgestellt werden. In der chemischen Prozeßtechnik wurden auf diese Weise Anlagen simuliert. Hierbei wurde jede Komponente (z.B. ein Wärmeaustauscher) durch ein Unterprogramm beschrieben. Mit entsprechenden Algorithmen wurde ermittelt, in welcher Reihenfolge die Unterprogramme aufzurufen sind und wann Iterationen über Unterprogramm-Aufruffolgen (= algebraische Schleifen) notwendig sind. Aufgrund der oben erläuterten Schwierigkeiten ist man davon aber wieder abgekommen (siehe z.B. [Mah90]).

Zusammenfassend kann festgehalten werden, daß objektorientierte Programmiersprachen die Realisierung fachspezifischer Programme deutlich vereinfachen und zu flexibler verwendbaren Modellen führen. Der direkte Einsatz für eine fachübergreifende Modellierung ist jedoch eingeschränkt.

1.3 Notwendigkeit der multidisziplinären Modellierung geregelter Roboter

Im letzten Abschnitt wurde eine Reihe unterschiedlicher Arten der Modellierung diskutiert. Es zeigt sich, daß die Modellierung von Komponenten unterschiedlicher Fachrichtungen in einer gemeinsamen Modellbildungsumgebung heutzutage nur eingeschränkt möglich ist. Die allgemeinen Umgebungen, d.h. die Simulationssprachen und Blockschaltbild-Editoren, sind nicht mächtig genug, um ein physikalisches System anwendernah modellieren zu können. Die fachspezifischen Programme sind auf das jeweilige Fachgebiet zugeschnitten und erlauben nur in eingeschränktem Maße und auf unkomfortable Weise die Modellierung fachfremder Komponenten.

In vielen technischen Bereichen ist es jedoch unabdingbar geworden, Komponenten unterschiedlicher Ingenieurdisziplinen in einer gemeinsamen Umgebung zu analysieren und zu entwerfen. Ein typisches Beispiel aus der Mechatronik ist ein Industrieroboter. In Bild 1.5 sind die Hauptkomponenten eines Roboters schematisch zusammengestellt. Ein Roboter

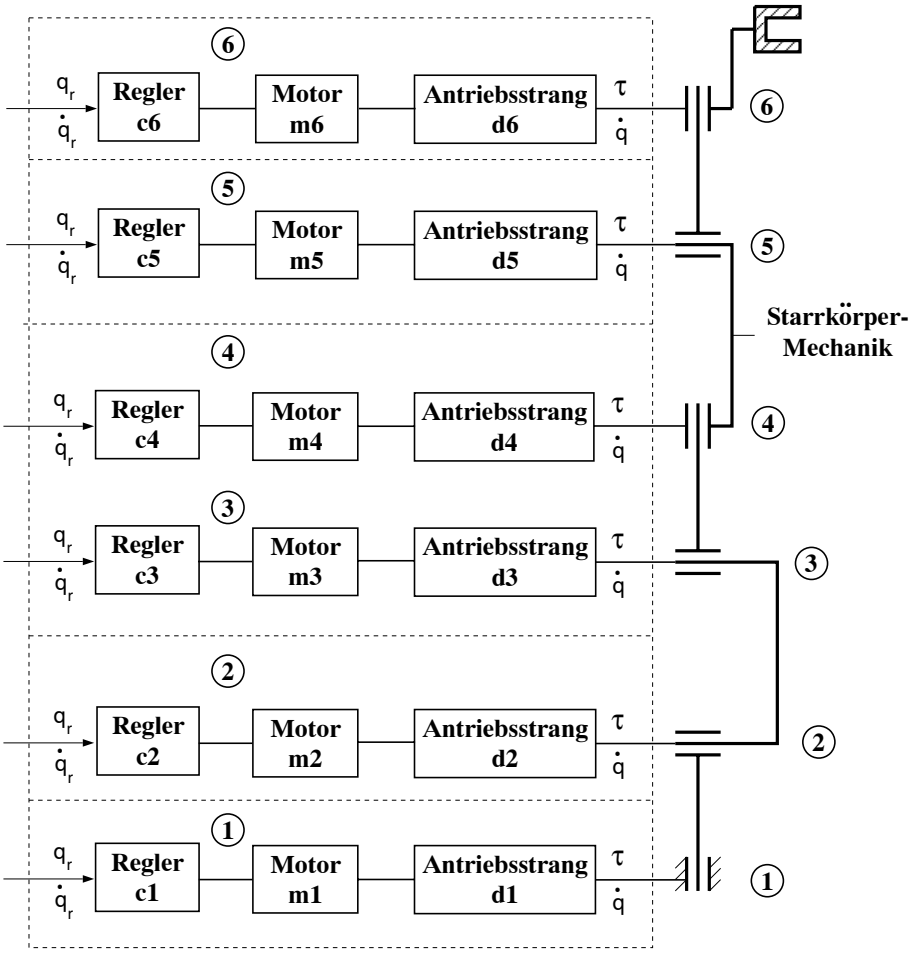


Bild 1.5: Gesamtmodell-Struktur des Roboters Manutec r3.

besteht aus einer mechanischen Grundstruktur von Armen und Gelenken. Die Gelenke werden von Elektromotoren über Untersetzungsgetriebe angetrieben. Die aktuelle Stellung der Motorläufer, und damit indirekt der Gelenkstellungen, wird gemessen und über dezentrale

Regler mit der Sollvorgabe verglichen. Abweichungen zur Sollvorgabe werden auf die Stromregler der Motoren geschaltet. Für eine realistische Modellierung werden hier die folgenden Fachdisziplinen benötigt:

- Die *Mechanik*, zur Modellierung der mechanischen Struktur als Mehrkörpersystem und zur Modellierung der Getriebe als Antriebsstrang, um die speziellen Eigenschaften von Antriebssträngen auszunutzen.
- Die *Elektrotechnik*, zur Modellierung der Elektromotoren und der Motoransteuerungen durch elektrische Schaltkreise.
- Die *Regelungstechnik*, zur Modellierung der dezentralen Gelenkregler.

Teile der Gelenkregler sind digital realisiert. Weiterhin tritt Coulomb'sche Reibung und Lose in den Antriebssträngen auf. Solche Effekte können nur mit Hilfe ereignisabhängiger Modelle numerisch sicher beschrieben werden.

Es ist kaum möglich, ein solches Gesamtmodell mit den im letzten Abschnitt besprochenen Modellierungsumgebungen zu erstellen: Die Mechanik des Roboters ist schon hinreichend kompliziert, sodaß zur Modellbildung auf jeden Fall ein Mehrkörperprogramm benötigt wird. Mit einem symbolischen Mehrkörperprogramm könnte entsprechender Code erzeugt und mit Hand in ein ACSL-Programm eingebettet werden, um vor allem die unstetigen Elemente mit ACSL beschreiben zu können. Im Falle von Reibung ist jedoch die Kopplung der mechanischen Gleichungen und der Ereignisbehandlung so eng, daß die ACSL Sprachelemente nicht ausreichen. Die Gleichungen der Gelenkregler und der Elektromotoren müssten mit Hand erzeugt werden, da keine Blockschaltbild-Editoren oder elektrischen Schaltungsprogramme verfügbar sind, mit denen portabler Code zur Einbettung in andere Programme erzeugt werden kann.

In diese Arbeit steht die multidisziplinäre Modellbildung im Mittelpunkt. Der oben diskutierte Roboter wird dabei als Repräsentant einer Anwendungsklasse der Mechatronik angesehen, an dem entschieden wird, welche Fachdisziplinen und welche Modellbildungstechniken besprochen werden. Die aufgeführten Verfahren gehen jedoch teilweise deutlich über die bei einem Roboter benötigten Komponenten hinaus.

1.4 Das Modellbildungswerkzeug Dymola

Für eine multidisziplinäre Modellbildung ist es nicht sinnvoll, eine spezielle Fachdisziplin in den Mittelpunkt zu stellen, und die Methoden und Programme dieses Fachgebiets um fachfremde Komponenten zu ergänzen. Es ist aussichtsreicher, allgemeine Werkzeuge als Grundlage zu nehmen, die auf fachspezifische Fragestellungen spezialisiert werden können.

H. Elmqvist hat in einer grundlegenden Arbeit [Elmq78] schon 1978 einen gangbaren Weg zur multidisziplinären Modellierung aufgezeigt und mit der hierfür entwickelten ersten *objektorientierten* Modellierungssprache Dymola eine rechnergestützte Modellbildungsumgebung realisiert. Das Grundprinzip besteht darin, physikalische Systeme möglichst direkt auf Modellobjekte abzubilden. Dabei wird keine Kausalität festgelegt, sondern es werden nur *Beziehungen* zwischen Objekten, bzw. beschreibenden Variablen, definiert. Objekte können

entsprechend der *physikalischen* Verkopplungen miteinander verschaltet werden. Aus Anwendersicht ist damit eine "natürliche" Modellierung möglich, die dem jeweiligen Fachgebiet angepaßt ist. Die gemeinsamen Eigenschaften von Grundkomponenten eines Fachgebietes können in wiederverwendbaren Bibliotheken gespeichert und dann in entsprechenden Modellen eingesetzt werden. Dymola ist seit 1992 als kommerzielles Produkt verfügbar.

Eine weitere, nachfolgend entwickelte, objektorientierte Modellierungssprache ist Omola [Ande90, Ande92, Nils89]. Die Grundkonzepte und der zentrale algorithmische Kern von Omola sind an Dymola angelehnt. Jedoch ist die Syntax der Sprache vollkommen anders und mehr der heute bei objektorientierten Programmiersprachen üblichen Notation angepaßt. Prinzipielle Unterschiede gibt es bei der Ereignisbehandlung und dem Lösen von Gleichungssystemen. Omola stellt Modelle nur als differential-algebraisches Gleichungssystem (abgekürzt DAE[3]) dar, während Dymola versucht ein Modell in die Zustandsform zu überführen, da die meisten Integratoren von dieser Beschreibungsform ausgehen. Nur wenn das nicht möglich ist, überführt Dymola ein Modell in eine DAE.

Eine eingeschränktere, objektorientierte Modellierungssprache ist MAST, vom Simulationssystem Saber [Mant92]. MAST und Saber haben ihre Wurzeln in der Modellierung und Simulation elektrischer Schaltungen. Um hierbei auch nicht-elektrische Systeme einbeziehen zu können, wurde MAST entwickelt. Die Sprache erlaubt es z.B., einfache 1-dimensionale mechanische oder thermische Systeme in objektorientierter Weise zu modellieren und entsprechend der physikalischen Verkopplungen zu verschalten. Die Stärken von MAST und Saber liegen jedoch in der Modellierung von großen elektrischen Schaltungen, wobei dann diesselben Modellbildungstechniken auf nicht-elektrische Teilsysteme angewandt werden. Ähnliche Sprachelemente wie in MAST sind auch für die nächste Version des IEEE 1076 Standards VHDL vorgesehen. Bis 1997 soll die bisher rein diskrete Hardware Beschreibungssprache VHDL um Sprachelemente erweitert werden, mit denen analoge elektrische Schaltungen, aber auch sonstige physikalische Systeme, objektorientiert beschrieben werden können [Vach93].

Diese Arbeit basiert auf den Konzepten der objektorientierten Modellierung, wie sie in den obigen Modellierungssprachen enthalten sind. Insbesondere wird auf der grundlegenden Arbeit von Elmqvist [Elmq78] und weiterführender Arbeiten von Elmqvist und Cellier [Cell91, Cell93, Elmq93a] aufgebaut, wie sie in Dymola realisiert sind.

Konsequenterweise werden deshalb auch alle Beispiele in der "Dymola-Sprache" notiert. Dies hat mehrere Vorteile:

- Durch die Grammatik und die semantischen Regeln der Dymola-Sprache liegt ein (durch den Compiler der Sprache überprüfbarer) Rahmen vor, in dem ein Sachverhalt kurz, prägnant und exakt formuliert werden kann.
- Alle Ideen können in der Dymola-Umgebung sofort am Rechner verifiziert werden. Damit kann zugleich auch deren praktische Nützlichkeit untersucht werden. Dies ist ein wichtiger Punkt: Es ist für Ingenieure wenig befriedigend, wenn (nur) eine ausgefeilte Theorie angeboten wird, aber die (Rechner-) Werkzeuge fehlen um diese Theorie auf praktische Probleme anwenden zu können.

[3] DAE steht für *Differential-Algebraic Equations.*

- Die Transformationsalgorithmen, um ein Modell von einer anwendernahen Beschreibung in eine numeriknahe Zustandsbeschreibung umzuformen, sind ein zentraler Punkt objektorientierter Modellierungssprachen. Um die Praktikabilität solcher Algorithmen zu untersuchen, müssen Rechnerexperimente an realistischen Modellen durchführbar sein. Dies ist mit Dymola in bester Weise gegeben, da Dymola entsprechende Algorithmen enthält und direkt Rechencode im neutralen DSblock-Modellformat [Otte92] (siehe auch Bild 1.6), sowie im Format der Simulationssysteme ACSL, DESIRE, SIMNON oder SIMULINK, erzeugt.

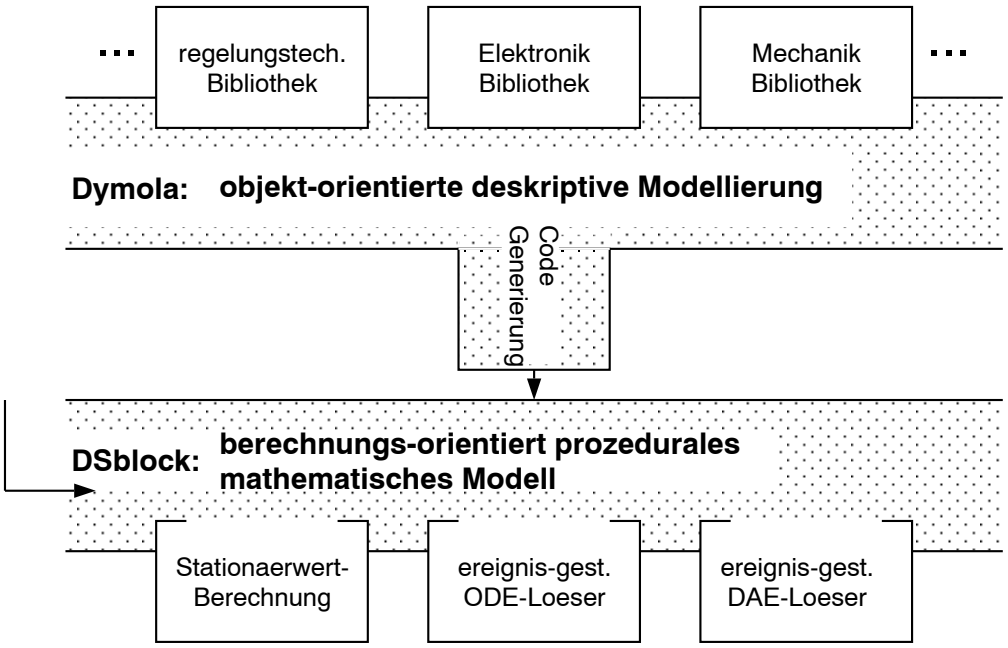


Bild 1.6: Die Rolle von Dymola als neutrale Modellbildungsumgebung zwischen anwendernaher, fachspezifischer Modelldefinition und rechnernahem numerikspezifischem Gleichungslöser

Die in dieser Arbeit beschriebene Vorgehensweise kann generell mit Hilfe objektorientierter Modellierungssprachen eingesetzt werden, und nicht nur speziell mit Dymola.

Kapitel 2

Objektorientierte Modellierung kontinuierlicher Systeme

In diesem Kapitel wird erläutert, wie kontinuierliche, dynamische Systeme objektorientiert modelliert werden können. Weiterhin werden Algorithmen besprochen, mit denen Modelle aus einer objektorientierten Beschreibungsform in eine für die rechnerische Auswertung geeignete Form überführt werden können. Dies ist in der Regel die Zustandsform. Die Vorgehensweise basiert auf [Elmq78, Cell91, Cell93, Elmq93a].

2.1 Grundkonzepte

Ein physikalisches Objekt wird durch zwei Eigenschaften charakterisiert: zum einen physikalische Gesetze, die *Zusammenhänge* zwischen physikalischen Größen des Objekts definieren und zum anderen die Art, wie diese Größen mit der Umgebung des Objekts in *Wechselwirkung* treten können. Diese beiden Eigenschaften eines Objekts werden in seiner *Klassen*-Beschreibung festgelegt. Eine *Klasse* legt die gemeinsamen Eigenschaften "gleichartiger" Objekte fest. In Bild 2.1 sind als Beispiel die Klassen zur Beschreibung elektrischer Widerstände und Kapazitäten aufgeführt, wie sie in der Standard-Bibliothek von Dymola für elektrische Bauteile vorliegen.

Die aufgeführten elektrotechnischen Elemente sind zwar unterschiedlich, haben aber die gemeinsame Eigenschaft, daß gleiche Verbindungs-Möglichkeiten zur Umgebung gegeben sind. Aus diesem Grund wird die *Oberklasse TwoPin* eingeführt, in der diese gemeinsame Eigenschaft definiert wird. Die Klassen der elektrischen Elemente sind dann *Unterklassen* von *TwoPin* und erben alle Eigenschaften von *TwoPin*.

Die Klasse *TwoPin* beschreibt elektrische Elemente, die zwei "Drahtenden" besitzen, mit denen ein Element mit anderen elektrischen Elementen verbunden werden kann. Die Verbindungsstelle eines Objekts zur Umgebung wird als **Cut** bezeichnet und folgendermaßen beschrieben:

cut *Cut-Name* (*Across-Variablen* / *Through-Variablen*) .

Wenn ein Objekt mit anderen Objekten verbunden wird, so wird eine Verbindungsstelle

```
model class TwoPin
        cut   a   (Va / i)
        cut   b   (Vb / -i)
    main cut  mc [a,b]
    main path mp < a - b >
    local  u

    u = Va - Vb
end

model class (TwoPin) Resistor
    parameter  R

    u = R * i
end

model class (TwoPin) Capacitor
    parameter  C

    C * der(u) = i
end
```

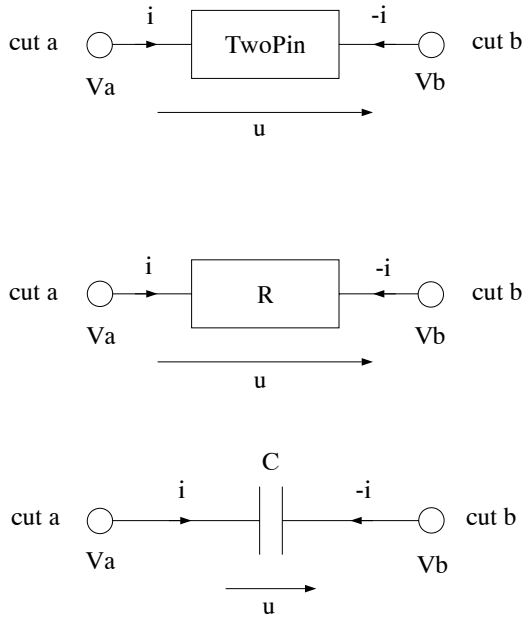


Bild 2.1: Modellklassen für elektrische Schaltungselemente.

durch *Objekt-Name:Cut-Name* identifiziert. In der Klammer einer Cut-Definition werden die Variablen aufgeführt, die über diese Verbindungsstelle mit der Umgebung in Kontakt treten. Diese Variablen werden auch als **Terminal**-Variablen bezeichnet, im Unterschied zu **Local**-Variablen, die nur lokal in einem Objekt Verwendung finden. Man beachte, daß mit Terminal-Variablen (in der Regel) *keine* Richtung verbunden ist. Es wird also nicht festgelegt, welche Variable Eingangs- und welche Ausgangsgröße ist. Dies ist eine notwendige Voraussetzung für die objektorientierte Modellierungstechnik, da nur die *Eigenschaften* eines Objekts beschrieben werden sollen. Die Signalrichtung ergibt sich *aus der Art, wie ein Objekt verschaltet ist, und welche Aufgabenstellung vorliegt.*

Es werden zwei Arten von Variablen unterschieden: *Across*- und *Through*-Variablen. Diese Abstraktion ist schon lange bekannt, siehe z.B. [Cann67]. Dymola verwendet die Verbindungseigenschaft der beiden Variablentypen als eingebaute Regel, um eine Verbindung von Objekten zu interpretieren, siehe auch Bild 2.2: Korrespondierende *Across*-Variable an einer Verbindungsstelle werden *gleichgesetzt*, während *Through*-Variable zu *Null aufsummiert* werden. Across- und Through-Variablen haben ähnliche Eigenschaften wie Effort- und Flow-Variablen, wie sie auf Seite 9 im Zusammenhang mit Bondgraphen erläutert wurden. Insbesondere ist das Produkt der jeweils korrespondierenden Variablen gleich dem an dieser Stelle auftretenden Energiefluß. Der einzige Unterschied besteht darin, daß es für Effort- und Flow-Variable jeweils zwei vollkommen symmetrische Verbindungselemente gibt (0- und 1-Junction), während für Across- und Through-Variable nur jeweils eine Verbindungsart definiert ist.

Die Cuts in der Klasse *TwoPin* in Bild 2.1 definieren die elektrischen Potentiale an den beiden Verbindungsstellen (Va, Vb) als Across-Variablen und den, über diese Stellen *einfließenden*, elektrischen Strom $(i, -i)$ als Through-Variablen. Die Verschaltungsregeln für Across- und Through-Variablen entsprechen dann den Kirchhoff'schen Gesetzen: An einem

Bei einer Verschaltung werden *Across* Variablen gleichgesetzt und
Through Variablen zu Null aufsummiert.

in der Bibliothek : **cut** *a* (*Va* / *i*)

im Modell : **node** *n*
connect *A:a* **at** *n*, *B:a* **at** *n*, *C:a* **at** *n*

Gleichungen : $A.Va = B.Va$
$B.Va = C.Va$
$0 = A.i + B.i + C.i$

Allgemeine Regel : *Leistung* = *"Across"* · *"Through"*
= *Spannung* · *Stromstärke*
= $Geschwindigkeit^T$ · *Kraft*
= *Temperatur* · *Entropiefluß*
= ...

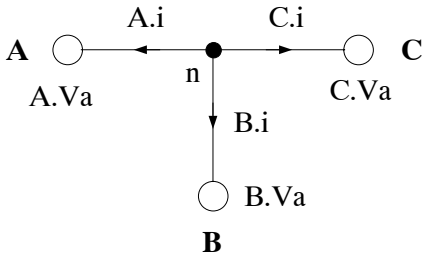


Bild 2.2: Physikalische Verschaltungsregeln.

elektrischen Verbindungsknoten sind die Potentiale gleich, während sich die Ströme zu Null aufaddieren. Die Anweisungen "**main cut ...**" und "**main path ...**" legen fest, wie eine Verbindungs-Definition zu interpretieren ist, wenn aus Bequemlichkeitsgründen der Name der an der Verbindung beteiligten Cuts weggelassen wird. Zum Beispiel sind die folgenden Anweisungen vollkommen äquivalent, um den Cut a eines elektrischen Schaltungselements C an einer Stelle $n1$ und den Cut b an einer Stelle $n2$ anzubringen:

connect *C:a* **at** $n1$, *C:b* **at** $n2$
connect C **at** $(n1, n2)$ {wegen "**main cut**"}
connect C **from** $n1$ **to** $n2$ {wegen "**main path**"}
connect $n1$ **to** C **to** $n2$ {wegen "**main path**"} .

Die physikalischen Gesetze der elektrotechnischen Bauteile sind in Unterklassen der Klasse TwoPin definiert, siehe Bild 2.1. Mit einer **Parameter**-Anweisung werden die Systemkonstanten eines Modells deklariert. Zum Beispiel ist der Parameter R die Größe des elektrischen Widerstands eines Objekts der Klasse *Resistor*. Die Werte von Parametern können z.B. vor einem Simulationslauf modifiziert werden.

Die physikalischen Gesetzmäßigkeiten zwischen Variablen werden in Form von *Gleichungen* (*nicht* in Form von *Zuweisungen*) angegeben. Dies ist zwangsläufig eine Folge der Forderung, daß in einer Klassenbeschreibung nicht festgelegt ist, welche Variablen Eingangsgrößen und welche Variablen Ausgangsgrößen sind. Es ist deswegen hier auch noch nicht bekannt, nach welcher der Variablen eine Gleichung aufgelöst werden muß. Das Ohm'sche Gesetz in der Klasse *Resistor* könnte deswegen genauso gut in der Form "$0 = u - R * i$" angeschrieben werden. Erst bei der Codeerzeugung wird festgestellt, nach welcher Variablen eine Gleichung aufzulösen ist. Wenn *diese* Variable *linear* in der Gleichung auftritt, wird die Gleichung mittels Symbolmanipulation nach der gewünschten Variable umgeformt. Die Zeitableitung in einer Gleichung wird mit dem Operator **der**[1] beschrieben. Zum Beispiel lautet die Gleichung für eine Kapazität: $C * \mathbf{der}(u) = i$.

Der Einsatz der oben diskutierten elektrotechnischen Bauteile wird an einem einfachen elek-

[1] **der** ist die Abkürzung von *derivative*.

trischen Schaltkreis aus [Cell91], Bild 2.3, erläutert. Der rechte Teil von Bild 2.3 zeigt den Schaltkreis, der linke Teil zeigt die entsprechende Modellbeschreibung.

```
@el.lib  {benutze die Bibliothek für elektrische Bauteile}
model circuit
    submodel (Vsource)   U0
    submodel (Resistor)  R1 (R=100),  R2(R=20)
    submodel (Capacitor) C  (C=1.E-6)
    submodel (Inductor)  L  (L=1.5E-3)
    submodel (Ground)    g

    node    n0, n1, n2, n3
    input   u
    output  y1, y2

    connect g  at n0       U0 at (n1,n0),
            R1 at (n1,n2), C  at (n2,n0),
            R2 at (n1,n3), L  at (n3,n0)

    u  = U0.V0
    y1 = C.u
    y2 = L.i
end
```

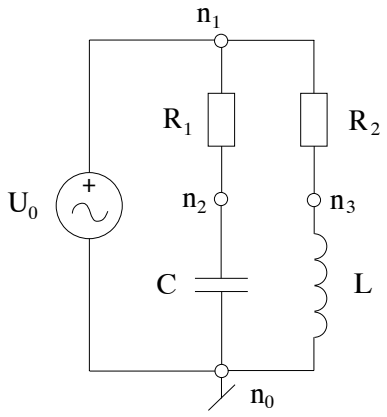


Bild 2.3: Einfacher elektrischer Schaltkreis.

Mit der ersten Anweisung wird die Bibliothek für elektrische Bauteile verfügbar gemacht (der @ Operator legt fest, daß die Datei *el.lib* in das Modell aufzunehmen ist). Danach werden alle benötigten Objekte mit den Anweisungen "**submodel** ..." deklariert. Die allgemeine Syntax hierfür ist:

submodel *(Klassenname) Objektname (Parameter)* .

Zum Beispiel wird mit der Anweisung "**submodel** (*Resistor*) *R1* (*R=100*)" ein neues Modell der Klasse *Resistor* angelegt. Dieses Modell erhält den Namen *R1* und der Modellparameter *R* der Klasse *Resistor* erhält den Wert 100. Durch diese Anweisung wird also ein Widerstand von 100Ω definiert.

Um die Verbindungsdefinition der elektrischen Bauteile zu vereinfachen, werden einige "Knotenpunkte", die **node** genannt werden, eingeführt (siehe Bild 2.3). Weiterhin werden durch die **input** und **output** Anweisungen das Eingangssignal *u* und die Ausgangssignale *y1, y2* deklariert. In den letzten drei Gleichungen des Modells wird festgelegt, daß *u* die Spannung der Spannungsquelle, *y1* der Spannungsabfall an der Kapazität *C* und *y2* der Strom in der Induktivität *L* ist. Schließlich werden mit der **connect** Anweisung alle deklarierten Objekte zusammengeschaltet.

Wie in Bild 2.3 zu sehen ist, wird die physikalische Realität, d.h. der elektrische Schaltkreis, durch die Definition entsprechender Objekte und deren Verschaltung sehr direkt abgebildet. Ziel neuer Modellbibliotheken ist es, eine solche direkte physikalische Abbildung auch für andere Fachgebiete anzubieten. Das obige Modell könnte sofort in einem größeren Modell als Sub-Modell eingesetzt werden. Dymola erlaubt eine beliebige hierarchische Verschachtelung von Modellen und Modell-Klassen.

2.2 Basis-Algorithmen

Im vorherigen Abschnitt wurde erläutert, wie ein physikalisches Modell mit Hilfe einer Klassen-Bibliothek realitätsnah aufgebaut werden kann. In diesem Abschnitt wird diskutiert, wie ein solches Modell in die minimale Beschreibungsform einer Zustandsdarstellung übergeführt werden kann. Für den praktischen Einsatz ist es dann natürlich entscheidend, daß die Zustandsform nicht nur für kleine "Spielmodelle", sondern auch für größere, praxisrelevante Modelle effizient berechnet werden kann. Die einzelnen Schritte der Transformation in die Zustandsform werden anhand des elektrischen Schaltkreises aus Abschnitt 2.1 erläutert.

Das Basis-Gleichungssystem

Zuerst werden automatisch die Gleichungen *aller* beteiligten Objekte als Kopien der Gleichungen aus den entsprechenden Klassenbeschreibungen aufgestellt. Zusätzlich werden alle explizit angegebenen Gleichungen aufgenommen (z.B. *u=U0.V0* in Bild 2.3). Zum Schluß werden die durch **connect** Anweisungen implizierten Beziehungen zwischen Variablen gleichungsmäßig formuliert. Für den Schaltkreis von Bild 2.3 führt diese Vorgehensweise auf folgende Gleichungen[2]:

U0.	U0.u	= U0.Va − U0.Vb
	U0.V0	= U0.u
R1.	R1.u	= R1.Va − R1.Vb
	R1.R∗R1.i	= R1.u
R2.	R2.u	= R2.Va − R2.Vb
	R2.R∗R2.i	= R2.u
C.	C.u	= C.Va − C.Vb
	C.C∗C.deru	= C.i
L.	L.u	= L.Va − L.Vb
	L.L∗L.deri	= L.u
g.	gV	= 0

circuit.	u	= U0.V0
	y1	= C.u
	y2	= L.i
	R1.Vb	= C.Va
	C.i	= R1.i
	R2.Vb	= L.Va
	L.i	= R2.i
	R2.Va	= U0.Va
	R1.Va	= R2.Va
	0	= U0.i + R2.i + R1.i
	U0.Vb	= g.V
	L.Vb	= U0.Vb
	C.Vb	= L.Vb

In der linken Spalte sind jeweils die Objekte angegeben in deren Klassenbeschreibung die Gleichung definiert ist, die in der rechten Spalte aufgeführt ist. Wie in objektorientierten Sprachen üblich, wird z.B. die Variable Va des Objekts C durch den Namen $C.Va$ gekennzeichnet. Das obige Gleichungssystem ist die *Basis*, von der aus alle Transformationen erfolgen.

Die Formulierung der Modellbildungsaufgabe

Das bis jetzt aufgestellte Gleichungssystem legt nur Beziehungen zwischen Variablen fest. Es wurde jedoch noch nicht bestimmt, *was* zu berechnen ist. Oder anders ausgedrückt:

[2] Bei der Code-Erzeugung wird der Operator **der** durch eine entsprechende neue Variable ersetzt, z.B. **der**(u) durch *deru*.

Aufgrund der Aufgabenstellung muß festgelegt werden, welche Variablen bekannt und welche Variablen unbekannt sind.

Üblicherweise geht man davon aus, daß ein dynamisches System in der Zustandsform simuliert werden soll, d.h., daß ein Anfangswertproblem des folgenden expliziten Differential-Gleichungssystems zu lösen ist:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{2.1}$$
$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}, t) \ . \tag{2.2}$$

Hierbei sind $\mathbf{x}$(t) die Zustandsgrößen, $\mathbf{u}$(t) sind die bekannten Eingangsgrößen, $\mathbf{y}$(t) sind die Ausgangsgrößen und t ist die Zeit. Integratoren verlangen, daß die Funktionen $\mathbf{f}$ und $\mathbf{g}$ zur Verfügung gestellt werden. Das heißt, bei bekannten $\mathbf{x}$, $\mathbf{u}$ und t sind die unbekannten Größen $\dot{\mathbf{x}}$ und $\mathbf{y}$ zu berechnen.

Aus diesem Grund wird standardmäßig angenommen, daß alle mit **input** deklarierten Größen, sowie alle Variablen w, bei denen Ableitungen nach der Zeit auftreten, *bekannte* Größen sind. Alle anderen Variablen sollen damit berechnet werden, insbesondere die Ableitungen **der**(w) und die als **output** deklarierten Variablen. Für den obigen Schaltkreis soll eine Simulation durchgeführt werden. Dafür legt diese Voreinstellung fest, daß $u, C.u$ und $L.i$ bekannt sind, und daß $der(C.u), der(L.i), y1$ und $y2$ zu berechnen sind.

Die Voreinstellung kann leicht geändert werden, indem *explizit* angegeben wird, welche Variablen bekannt und welche unbekannt sind. Zum Beispiel könnte bei den Gleichungen (2.1) gefordert werden, daß alle Ableitungen Null sind ($\dot{\mathbf{x}} = \mathbf{0}$), d.h. $\dot{\mathbf{x}}$ sind bekannte Größen, und daß alle Zustandsgrößen $\mathbf{x}$ unbekannt sind. In diesem Falle würde das Modell für eine Stationärwertberechnung aufbereitet werden. Es könnte auch die Aufgabe gestellt werden, das inverse dynamische Modell zu bestimmen. Dazu werden $\dot{\mathbf{x}}$ und $\mathbf{x}$ als bekannt, und die Eingangsgrößen $\mathbf{u}$ als unbekannt angesehen.

Ausgehend von den Basisgleichungen des Modells, können für jede beliebige Aufgabenstellung die entsprechenden Auswerte-Gleichungen abgeleitet werden. Wenn von der Voreinstellung abgewichen wird, können natürlich leicht Fehler gemacht werden, wobei dann z.B. die Zahl der Gleichungen mit der Zahl der Unbekannten nicht mehr übereinstimmt. Mit den im folgenden beschriebenen Algorithmen werden aber solche Fehler erkannt.

Elimination identischer Variablen

Durch die Verschaltungsregeln für Across-Variablen werden sehr viele triviale Gleichungen der Form $a = b$ eingeführt. In einem ersten Schritt werden alle solche Gleichungen entfernt und z.B. a in allen restlichen Gleichungen durch b ersetzt. Im obigen Schaltkreisbeispiel führt das zu dem folgenden, vereinfachten Gleichungssystem:

```
U0.     U0.u          = U0.Va - U0.Vb
R1.     R1.u          = R2.Va - C.Va
        R1.R*R1.i     = R1.u
R2.     R2.u          = R2.Va - L.Va
        R2.R*L.i      = R2.u
C.      C.u           = C.Va - L.Vb
        C.C*C.deru    = R1.i
L.      L.u           = L.Va - L.Vb
```

```
                L.L*L.deri  = L.u
g.              g.V         = 0
circuit.        0           = U0.i + L.i + R1.i
                y1          = C.u
                y2          = L.i
```

Transformation auf Blockdreiecksform

Jede Modellbildungsaufgabe kann als das Lösen eines (in der Regel großen und dünnbesetzten) nichtlinearen Gleichungssystems der Form $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ angesehen werden, wobei $\mathbf{x}$ die unbekannten, zu berechnenden, Größen sind. Zur Vereinfachung der Schreibweise werden dabei die *bekannten* Größen nicht mehr explizit in der Funktion aufgeführt. Die Grundidee zur effizienten Lösung dieses Gleichungssystems besteht darin, das System durch Permutationen der Unbekannten und der Gleichungen in eine Form zu überführen, in der möglichst "wenig" Abhängigkeiten zwischen den Gleichungen bestehen und das Gleichungssystem dann durch eine Vorwärtsrekursion gelöst werden kann.

Hierzu wird eine sogenannte *Occurrence* Matrix aufgestellt[3], siehe z.B. [Mah90]. Alle Elemente dieser Matrix sind entweder 0 oder 1. Das Element (i, j) ist 1, wenn die Unbekannte x_j in der Gleichung f_i auftritt, anderenfalls ist es 0. Als Beispiel ist in Gleichung (2.3) ein einfaches Gleichungssystem mit der zugehörigen Occurrence Matrix Occ_1 angegeben:

$$\begin{array}{lcl} f_1(x_1, x_3) & = & 0 \\ f_2(x_2) & = & 0 \\ f_3(x_1, x_2) & = & 0 \end{array} \qquad Occ_1 = \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \left[\begin{array}{ccc} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{array}\right] \end{array} . \tag{2.3}$$

Durch Permutationen der Gleichungen und der Unbekannten, d.h. der Zeilen und der Spalten der Occurrence Matrix, kann dieses Gleichungssystem auf die folgende Form gebracht werden:

$$\begin{array}{lcl} f_2(x_2) & = & 0 \\ f_3(x_1, x_2) & = & 0 \\ f_1(x_1, x_3) & = & 0 \end{array} \qquad Occ_2 = \begin{array}{c} \begin{array}{ccc} x_2 & x_1 & x_3 \end{array} \\ \left[\begin{array}{ccc} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{array}\right] \end{array} . \tag{2.4}$$

In dieser Form kann zuerst x_2 aus f_2, dann x_1 aus f_3 und schließlich x_3 aus f_1 berechnet werden, d.h. es werden drei voneinander unabhängige Gleichungssysteme jeweils der Ordnung 1 gelöst, statt eines nichtlinearen Gleichungssystems der Ordnung 3. Wenn darüberhinaus x_2 in f_2, x_1 in f_3 und x_3 in f_1 nur *linear* auftreten, können die einzelnen Gleichungssysteme sofort symbolisch nach der Unbekannten umgeformt werden und das ursprüngliche Problem ist gelöst.

Verallgemeinert kann diese zentrale Aufgabenstellung so formuliert werden: Gesucht sind Permutationsmatrizen $\mathbf{P}$ (für die Gleichungen) und $\mathbf{Q}$ (für die Unbekannten), so daß die Occurrence Matrix $\mathbf{A}$ auf die folgende Blockdreiecksform gebracht wird, wobei die Blöcke $\mathbf{B}_{ii}$ auf der Diagonalen eine minimale Dimension besitzen sollen (alle Elemente im oberen Dreieck im folgenden Gleichungssystem sind Null):

[3] Natürlich wird in einer Programm-Realisierung aus Speicherplatz- und Effizienzgründen die Occurrence Matrix nicht direkt aufgestellt, sondern durch verzeigerte Listen realisiert. In Dymola wird die Occurrence Matrix durch einen Bipartit-Graphen, eine spezielle Listenstruktur, beschrieben.

$$\mathbf{P\,A\,Q} = \begin{bmatrix} \mathbf{B}_{11} & & & & \\ \mathbf{B}_{21} & \mathbf{B}_{22} & & & \\ \mathbf{B}_{31} & \mathbf{B}_{32} & \mathbf{B}_{33} & & \\ . & . & . & . & \\ \mathbf{B}_{m1} & \mathbf{B}_{m2} & . & . & \mathbf{B}_{mm} \end{bmatrix} . \tag{2.5}$$

Das ursprüngliche Problem (Lösen eines Gleichungssystems der Ordnung n) wird dadurch auf das sukzessive Lösen von m Gleichungssystemen mit niedrigeren Ordnungen $\dim(\mathbf{B}_{11})$, $\dim(\mathbf{B}_{22})$, ..., $\dim(\mathbf{B}_{mm})$ zurückgeführt.

Erstaunlicherweise ist die Blockdreiecksform (2.5) mit *minimalen* Dimensionen der Diagonalblöcke im "wesentlichen" eindeutig und kann durch effiziente Algorithmen ermittelt werden. Im "wesentlichen" eindeutig bedeutet, daß sich alle Blockdreiecksformen, bei denen sich die Diagonalblöcke $\mathbf{B}_{ii}$ nicht weiter in kleinere Diagonalblöcke aufteilen lassen, durch folgende Transformation ineinander überführen lassen: Zeilen-Permutationen, die *nur* die Zeilen *eines* Diagonalblockes betreffen, Spalten-Permutationen, die *nur* die Spalten *eines* Diagonalblockes betreffen, sowie symmetrische Permutationen, die die Blöcke umordnen (wenn z.B. $\mathbf{B}_{21} = \mathbf{0}$, können die Blöcke $\mathbf{B}_{11}$ und $\mathbf{B}_{22}$ auch vertauscht werden). Für einen Beweis dieser Eigenschaft siehe z.B. [Duff86]. Die Transformation auf Blockdreiecksform geschieht üblicherweise in zwei Schritten:

In einem ersten Schritt wird die Occurrence-Matrix $\mathbf{A}$ durch Zeilen-Permutationen $\mathbf{PA}$ so umgeformt, daß jedes Diagonalelement eine Eins enthält. Dies ist immer möglich, wenn die Jacobi-Matrix $\partial\mathbf{f}/\partial\mathbf{x}$ nicht strukturell singulär ist. Wie strukturelle Singularitäten behandelt werden, wird in Abschnitt 2.4 ab Seite 31 besprochen. Hier wird zunächst angenommen, daß die Jacobi-Matrix (strukturell) regulär ist. Eine solche Umformung wird als *"output set assignment"* oder auch als *"finding a transversal"* bezeichnet. Algorithmische Details können in [Duff86, Pant88, Mah90] gefunden werden[4]. Der Rechenaufwand zur Ermittlung dieser Form ist höchstens $O(n\tau)$, wobei n die Dimension der Occurrence Matrix ist, d.h. die Anzahl der Gleichungen, und τ die Anzahl der Elemente ist, die den Wert 1 haben. In der Regel ist der Rechenaufwand aber deutlich geringer [Duff86].

In einem zweiten Schritt wird durch *symmetrische* Zeilen- und Spalten-Permutationen $\mathbf{Q}^T(\mathbf{PA})\mathbf{Q}$ auf die gewünschte Blockdreiecksform transformiert. Dies geschieht mit Tarjans bekanntem Algorithmus [Tarj72] zur Ermittlung der *strong components* eines gerichteten Graphen. Der Rechenaufwand von Tarjans Algorithmus ist $O(n) + O(\tau)$.

Üblicherweise sind in einer Gleichung nur wenige Unbekannte enthalten. Deswegen ist τ meist ein (kleines) Vielfaches der Anzahl der Gleichungen n. Der Rechenaufwand zur Umformung auf Blockdreiecksform ist deswegen im ungünstigsten Fall $O(n^2)$. In vielen praktischen Fällen ist er jedoch deutlich kleiner. Eigene Experimente zeigen, daß Dymola selbst bei 20000 Gleichungen die Transformation auf Blockdreiecksform innerhalb weniger Sekunden durchführt.

Wenn alle Diagonal-Blöcke der Blockdreiecksform die Dimension 1 haben und alle Variablen *auf der Diagonalen* nur linear in diesen Gleichungen auftreten, dann kann explizit nach allen Unbekannten aufgelöst werden und das Problem ist gelöst. Modelle, bei denen dies nicht

[4] In [Pant88] wird der vollständige Algorithmus kompakt in 13 Zeilen Pseudocode beschrieben und auf einfache Weise bewiesen.

zutrifft, werden im nächsten Abschnitt ab Seite 26 behandelt.

Wird das oben skizzierte Vorgehen auf das Schaltkreisbeispiel angewendet, ergeben sich die folgenden Gleichungen:

g.	g.V	= 0
U0.	U0.Va	= g.V + u
C.	C.Va	= g.V + C.u
R1.	R1.u	= U0.Va − C.Va
	R1.i	= R1.u/R1.R
circuit.	U0.i	= − (L.i + R1.i)
R2.	R2.u	= R2.R∗L.i
C.	C.deru	= R1.i/C.C
R2.	L.Va	= U0.Va − R2.u
L.	L.u	= L.Va − g.V
	L.deri	= L.u/L.L
circuit.	y1	= C.u
	y2	= L.i

In der linken Spalte sind wiederum die Objekte angegeben, in deren Klassenbeschreibungen die Gleichungen definiert sind, die in der rechten Spalte auftretenden. Dadurch sieht man deutlich die Umsortierung des ursprünglichen Gleichungssystems und das Auflösen der einzelnen Gleichungen nach den jeweiligen Unbekannten. Man beachte, daß die Gleichung des Widerstands R1 nach dem Strom (R1.i = R1.u/R1.R) aufgelöst wurde, während die Gleichung des Widerstands R2 nach der Spannung (R2.u = R2.R∗L.i) aufgelöst wurde. Das heißt, eine Gleichung derselben Klasse tritt, aufgrund der unterschiedlichen Verschaltung, in zwei verschieden aufgelösten Formen auf.

Elimination nicht benötigter Gleichungen

Standardmäßig wird angenommen, daß alle Parameter, d.h. alle Modellkonstanten, zur Laufzeit eines Simulationsexperiments modifiziert werden können. Deswegen werden alle Parameter bei der Code-Erzeugung wie bekannte (zeitabhängige) Variable behandelt. Für größere Modelle ist dieses Vorgehen ungeeignet: Zum einen, weil es im Laufzeitsystem unübersichtlich wird, hunderte oder tausende von Konstanten modifizieren zu können und zum anderen, weil die Effizienz des Modells gesteigert werden kann, wenn die numerischen Werte der Konstanten schon bei der Code-Erzeugung ausgenutzt werden. Die letzte Eigenschaft wird besonders deutlich bei den in Kapitel 4 betrachteten Mehrkörpersystemen, da hier viele Modellkonstanten 0 oder 1 sind.

Aus diesem Grund kann in Dymola definiert werden, welche Parameter bei der Modelltransformation durch ihre numerischen Werte ersetzt werden sollen (z.B. in der Form: "expandiere alle Parameter mit Ausnahme von ..."). In der Code-Generierungsphase werden dann z.B. die speziellen numerischen Parameter-Werte 0 und 1 ausgenutzt. Wenn z.B. "$a = b * (c + d)$" ist und der Parameter b den Wert 0 besitzt, so wird die komplette Gleichung eliminiert und a wird bei jedem weiteren Auftreten durch 0 ersetzt.

Bei vielen Aufgabenstellungen werden nicht alle, in einem Modell enthaltenen, Gleichungen benötigt. Weiterhin werden bei der Gleichungsaufstellung eventuell Zwischengrößen berechnet, die für die eigentliche Aufgabenstellung nicht benutzt werden. Wenn z.B. die Gleichung "$a = b * (c + d)$" eliminiert wird, weil der Parameter $b = 0$ ist und die Zwischengrößen c

und d im folgenden nicht mehr benötigt werden, so können zumindest auch die Definitionsgleichungen für c und d entfernt werden. Es zeigt sich, daß beide dieser Fälle bei Mehrkörpersystemen häufig auftreten. Deswegen werden optional *alle* Gleichungen entfernt, die nicht für die Berechnung der als **output** deklarierten Variablen, der Ableitungen der Zustandsgrößen oder sonstiger *explizit* gewünschter Variablen, benötigt werden. Diese Vorgehensweise ist an das in [Otte88] beschriebene symbolische Verfahren zur Erzeugung effizienter Bewegungsgleichungen von Mehrkörpersystemen angelehnt.

Wenn im Schaltkreisbeispiel alle Parameter, mit Ausnahme der Kapazität und der Induktivität, expandiert werden und nur die Ausgangsgrößen und die Ableitungen der Zustandsgrößen berechnet werden sollen, dann werden die in Bild 2.4 angegebenen Gleichungen erzeugt.

(bekannte Variablen: C.u, L.i, u)

R1.	R1.u	= u − C.u
	R1.i	= 0.010000∗R1.u
C.	C.deru	= R1.i/C.C
R2.	R2.u	= 20.000000∗L.i
	L.Va	= u − R2.u
L.	L.deri	= L.Va/L.L
circuit.	y1	= C.u
	y2	= L.i

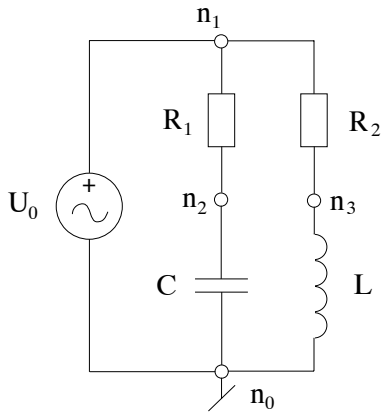


Bild 2.4: Für den Schaltkreis generierte Zustandsform.

Ausgehend vom objektorientierten Modell des Schaltkreises in Bild 2.3 auf Seite 20, wurden mehrere Gleichungs-Zwischenstufen gezeigt und erläutert, bis schließlich die gewünschte Endform erreicht wurde. Normalerweise werden diese Zwischengleichungen nur intern gebildet. Der Anwender "sieht" nur das Ausgangsmodell und die aufgeführten Endgleichungen. Für das Verständnis der folgenden Kapitel ist es jedoch wichtig zu wissen, auf welche Weise ein Modell in die Zustandsdarstellung übergeführt wird. Deshalb wurden hier die Zwischenstufen explizit gezeigt.

2.3 Modelle mit differential-algebraischen Gleichungssystemen

In diesem Abschnitt werden Modellbeschreibungen behandelt, die neben Differentialgleichungen auch algebraische Gleichungssysteme enthalten. Unter anderem wird das allgemeine *Tearing*-Verfahren eingeführt, welches zur Behandlung von Mehrkörpersystem-Modellen in Kapitel 4.3 benötigt wird.

Modelle, die nicht explizit auflösbar sind, führen zu einer Blockdreiecksform, bei der einige der Diagonalblöcke entweder eine Dimension größer als 1 besitzen und/oder die entsprechende Variable nicht-linear in die Gleichungen des Diagonalblocks eingeht. Diese Gleichungssysteme, sowie die Unbekannten in den Gleichungssystemen, werden bei der Transformation auf Blockdreiecksform automatisch ermittelt.

Lineare algebraische Gleichungen im Modell

Wenn die Unbekannten in ein Gleichungssystem nur *linear* eingehen, löst Dymola das Gleichungssystem explizit auf. Zur Zeit stehen zwei Methoden zur Verfügung:

Standardmäßig wird ein lineares Gleichungssystem bei der Code-Erzeugung *symbolisch* nach den Unbekannten aufgelöst. Hierfür wird eine "intelligente" Variante der Cramer'schen Regel benutzt. Insbesondere werden immer automatisch Zwischengrößen gebildet, die in den folgenden Ausdrücken mitbenutzt werden. Es ist nicht möglich einen effizienteren Algorithmus einzusetzen, da bei der Code-Erzeugung keine numerischen Werte der Systemmatrix bekannt sind. Deswegen kann insbesondere keine Pivot-Suche durchgeführt werden. Dieses Verfahren ist bei kleineren, vollbesetzten Matrizen und bei mittelgroßen, dünnbesetzten Matrizen sinnvoll anwendbar.

Optional werden lineare Gleichungssysteme *numerisch* gelöst. Hierzu wird zuerst ein lineares Gleichungssystem der Form "$\mathbf{Ax} = \mathbf{b}$" aufgestellt. Dies geschieht bei der Code-Erzeugung durch symbolische Wertezuweisungen an die Elemente der Matrix $\mathbf{A}$ und des Vektors $\mathbf{b}$. Danach wird das Gleichungssystem mit Unterprogrammen der LINPACK-Bibliothek [Dong79], basierend auf einem Gaußverfahren mit Spalten-Pivotsuche und Konditionsschätzung, gelöst. Dieses Verfahren ist bei vollbesetzten Matrizen sinnvoll anwendbar.

Es wäre vorteilhaft, auch ein spezielles numerisches Verfahren zur Lösung dünnbesetzter Matrizen anzubieten ("Sparse Matrix Solver"), da die Gleichungssysteme oft dünnbesetzt sind. Es ist beabsichtigt einen "general purpose" Löser, z.B. `Y12M` von Zlatev, Wasniewski und Schaumburg oder `HA28` von Duff aus der Harwell Bibliothek, dafür einzusetzen. Im Unterschied zu einem Löser für vollbesetzte Matrizen werden bei diesen Unterprogrammen nur alle Nicht-Null Elemente der Matrix in einem Vektor gespeichert sowie in zwei Integer-Vektoren die zugehörigen Matrizenindizes mitgeteilt. Der "Sparse Matrix Solver" nutzt dann die Null/Nicht-Null Struktur der Matrix bei der Lösung durch einen Gauß-Algorithmus aus, siehe z.B. [Duff86].

Nichtlineare algebraische Gleichungen im Modell

Wenn die Unbekannten in einem Gleichungssystem *nicht-linear* auftreten, überführt Dymola ein Simulationsmodell in ein differential-algebraisches Gleichungssystem (DAE[5]) der folgenden Form:

$$\mathbf{0} = \mathbf{f}(t, \mathbf{u}, \mathbf{x}, \dot{\mathbf{x}}) \tag{2.6}$$

$$\mathbf{y} = \mathbf{g}(t, \mathbf{u}, \mathbf{x}) \ . \tag{2.7}$$

Ein DAE-Löser wie DASSL [Petz82, Bren89] stellt zu jedem Zeitpunkt die Werte t, $\mathbf{x}$ und $\dot{\mathbf{x}}$ zur Verfügung und erwartet die Berechnung des Residuums $\mathbf{f}$. Aus diesem Grund werden die in einem nichtlinearen algebraischen Gleichungssystem auftretenden Variablen im Vektor der Deskriptorvariablen $\mathbf{x}$ aufgeführt. Dann wird Code zur Berechnung des Residuums der nichtlinearen Gleichungen erzeugt.

Wenn eine Überführung in eine DAE nicht erwünscht ist, oder wenn kein Simulationsproblem vorliegt, sondern z.B. ein Stationärwertproblem, kann das nichtlineare Gleichungssystem di-

[5] DAE steht für *Differential Algebraic Equations.*

rekt mit einem nichtlinearen Gleichungslöser behandelt werden. Es ist aber nicht praktikabel einen Gleichungslöser mit der üblichen Standard-Schnittstelle zu benutzen: Bei solchen Unterprogrammen wird nämlich verlangt, daß das Residuum des nichtlinearen Gleichungssystems in einem Unterprogramm berechnet wird. Ein Zeiger auf dieses Unterprogramm ist beim Aufruf des Lösers mitzugeben (`EXTERNAL` Anweisung in Fortran). Bei der Code-Erzeugung mit einem Programm wie Dymola ist es zu kompliziert, für jedes nichtlineare Gleichungssystem ein eigenes Unterprogramm zu generieren (automatische Namensvergabe für die Unterprogramme, Bereitstellung *aller* Variablen im Unterprogramm etc.).

Wesentlich einfacher gestaltet sich die Benutzung eines Lösers mit "reverse communication". Hier wird der Löser innerhalb einer *Schleife* aufgerufen. Immer wenn der Löser einen neuen Funktionswert benötigt, wird das Unterprogramm verlassen. Im *rufenden* Programm wird der Funktionswert berechnet und der Löser wird wieder aufgerufen. Dieser setzt dann seine Arbeit an der zuvor verlassenen Stelle fort.

In Dymola wird für diesen Zweck das Unterprogramm `RPNLZ1` aus der RASP-Bibliothek [Grue91] verwendet. `RPNLZ1` basiert auf den beiden Unterprogrammen `HYBRD` und `HYBRJ` von Garbow, Hillstrom und More aus der MINPACK-Bibliothek, wobei diese Unterprogramme auf "reverse communication" umgestellt wurden. Es wird ein Newton/Raphson-Verfahren mit einem Broyden Rang-1 Update[6] eingesetzt. Globale Konvergenz wird durch eine Begrenzung des Newton-Schritts erreicht.

Der Einsatz eines nichtlinearen Gleichungslösers in der rechten Seite einer *Differentialgleichung* muß wohlüberlegt sein, da z.B. die Genauigkeit des nichtlinearen Lösers und die Genauigkeit des Integrators aufeinander abgestimmt sein müssen[7]. Bei größeren nichtlinearen Gleichungssystemen ist es meist besser, eine generische DAE-Formulierung zu benutzen, da ein DAE-Löser mehr Informationen über die Vergangenheit benutzt um damit die Zukunft zu extrapolieren. Außerdem kann ein DAE-Löser durch seine Schrittweitenkontrolle während der Integration dafür sorgen, daß das Newton-Verfahren im Integrator schnell genug konvergiert. Bei kleineren nichtlinearen Gleichungssystemen in einem nichtsteifen Differentialgleichungssystem, kann aber die explizite Verwendung eines nichtlinearen Gleichungslösers vorteilhaft sein.

Lösung dünnbesetzter Gleichungssysteme durch Tearing

Die (linearen oder nichtlinearen) Gleichungssysteme in physikalischen Modellen sind oft groß und dünnbesetzt, d.h. in jede Gleichung gehen nur wenige der unbekannten Variablen ein. Solche Gleichungssysteme können effizient mit Methoden gelöst werden, die die Null/Nicht-Null Struktur (bezüglich des Auftretens von Variablen) eines Systems ausnutzen. "Sparse Matrix" Methoden für lineare Gleichungssysteme sind z.B. in [Duff86] beschrieben. Eine alternative Methode transformiert "große" dünnbesetzte Gleichungssysteme in "kleine" vollbesetzte Systeme, welche mit einem Löser für vollbesetzte Gleichungssysteme behandelt werden. Diese Verfahrensklasse wird als *Tearing* bezeichnet. Sie wurde von Rubin [Rubi62]

[6] Hierbei wird die Jacobi-Matrix nur einmal (symbolisch oder numerisch) berechnet und dann nach jedem Schritt vom Unterprogramm durch einen Rang-1 Update modifiziert. Wenn der Löser feststellt, daß die so approximierte Jacobi-Matrix zu ungenau geworden ist, wird diese erneut berechnet.

[7] Es macht z.B. wenig Sinn, wenn der nichtlineare Löser die Lösung auf 10 Stellen genau ermittelt, während der Integrator nur auf 4 Stellen genaue Ergebnisse liefert. Andererseits ist es kritisch, wenn die Lösung vom nichtlinearen Löser ungenauer als die vom Integrator ist.

und von Kron [Kron63] eingeführt. Tearing wird vor allem in der chemischen Prozeßtechnik verwendet [Mah90], ist jedoch z.B. auch bei Mehrkörpersystemen vorteilhaft einsetzbar.

Tearing für ein *nichtlineares* Gleichungssystem "$\mathbf{f}(\mathbf{x}) = \mathbf{0}$" besteht darin, Permutations-Matrizen $\mathbf{P}$ und $\mathbf{Q}$ zu finden, so daß die Unbekannten und das Gleichungssystem in zwei Teile aufgespaltet werden

$$\mathbf{x} = \mathbf{P} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}; \qquad \mathbf{f} = \mathbf{Q} \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix} \tag{2.8}$$

und das permutierte Gleichungssystem eine Form erhält

$$\mathbf{f}_1(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{0} \tag{2.9a}$$
$$\mathbf{f}_2(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{0}\ , \tag{2.9b}$$

wobei $\mathbf{f}_1$ *explizit* nach $\mathbf{x}_1$ auflösbar ist. Das heißt, bei bekanntem $\mathbf{x}_2$ muß $\mathbf{x}_1$ eindeutig (rekursiv) berechenbar sein, wie durch die folgende Gleichung ausgedrückt[8]:

$$\mathbf{x}_1 = \mathbf{f}_1^*(\mathbf{x}_1, \mathbf{x}_2)\ . \tag{2.10}$$

Nichtlineare Gleichungslöser arbeiten auf die folgende Weise: Es muß ein Unterprogramm zur Verfügung gestellt werden, das bei gegebenem $\mathbf{x}$ das Residuum $\mathbf{f}$ berechnet. Der Gleichungslöser bricht ab, wenn das Residuum "klein" genug geworden ist. Ein Gleichungssystem in der Form (2.9b, 2.10) benötigt nur $\mathbf{x}_2$, um zuerst $\mathbf{x}_1$ und dann das Residuum $\mathbf{f}_2$ zu berechnen. Anstatt ein Gleichungssystem der Ordnung $\dim(\mathbf{x})$ zu lösen, muß nur ein Gleichungssystem der Ordnung $\dim(\mathbf{x}_2)$ gelöst werden. Die Permutations-Matrizen sollten demnach so gewählt werden, daß $\mathbf{x}_2$ eine möglichst geringe Dimension besitzt. Tearing ist vorteilhaft, da man mit *symbolischen* Verfahren auf die Form (2.9b, 2.10) transformieren kann, und da wesentlich mehr nichtlineare Gleichungslöser für vollbesetzte als für dünnbesetzte Systeme verfügbar sind.

Tearing für ein *lineares* Gleichungssystem "$\mathbf{A}\mathbf{x} = \mathbf{b}$" ist ein Spezialfall von nichtlinearem Tearing. Hier müssen Permutations-Matrizen $\mathbf{P}$ und $\mathbf{Q}$ gefunden werden, so daß die Unbekannten und das Gleichungssystem wie folgt aufgespaltet werden:

$$\mathbf{P}\,\mathbf{A}\,\mathbf{Q} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \mathbf{P}\,\mathbf{b} \quad \Rightarrow \quad \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}\ . \tag{2.11}$$

$\mathbf{A}_{11}$ muß eine *untere Dreiecksmatrix* mit nicht-verschwindenden Diagonalelementen sein. Aufgrund dieser Eigenschaft ist $\mathbf{A}_{11}$ regulär und Gleichung (2.11) kann auf das folgende Gleichungssystem transformiert werden:

$$(\mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12})\mathbf{x}_2 = \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{b}_1 \tag{2.12a}$$
$$\mathbf{x}_1 = \mathbf{A}_{11}^{-1}(\mathbf{b}_1 - \mathbf{A}_{12}\mathbf{x}_2)\ . \tag{2.12b}$$

Da $\mathbf{A}_{11}$ eine untere Dreiecksmatrix ist, kann die Transformation auf Gleichung (2.12) auf einfache Weise *symbolisch* durchgeführt werden. Natürlich wird hierbei die Inverse von $\mathbf{A}_{11}$ nie

[8] In Gleichung (2.10) sind nicht alle Abhängigkeiten erlaubt, da sonst $\mathbf{x}_1$ nicht explizit berechenbar ist. Die formale Bedingung lautet, daß die Occurrence-Matrix von $\mathbf{f}_1^*$ bezüglich $\mathbf{x}_1$ eine untere Dreiecksmatrix sein muß, bei der alle Elemente auf und über der Diagonalen den Wert 0 besitzen.

explizit gebildet. Zum Beispiel wird der Term $\mathbf{X} = \mathbf{A}_{11}^{-1}\mathbf{A}_{12}$ durch "Lösen" des Gleichungssystems $\mathbf{A}_{11}\mathbf{X} = \mathbf{A}_{12}$ ermittelt. Dies ist eine einfache Vorwärtsrekursion, da $\mathbf{A}_{11}$ untere Dreiecksform besitzt. Gleichung (2.12) hat gegenüber der Ausgangsgleichung den Vorteil, daß nur noch ein Gleichungssystem der Ordnung $\dim(\mathbf{x}_2)$ anstelle eines Gleichungssystems der Ordnung $\dim(\mathbf{x})$ gelöst werden muß.

Der Rechenaufwand der Algorithmen zum Lösen linearer oder nichtlinearer vollbesetzter Gleichungssysteme ist mindestens proportional zu $O(n^3)$, wobei n die Systemordnung ist. Wenn durch Tearing die Systemordnung n wesentlich reduziert werden kann *und* die Transformation auf diese niedrigere Systemordnung "billig" ist, dann wird die Lösung des Gleichungssystems wesentlich effizienter, da $(n_2)^3 \ll n^3$ ($n_2 = \dim(\mathbf{x}_2)$). Die Transformation auf die niedrigere Systemordnung ist meist "billig", da die Systemmatrizen dünnbesetzt sind. Beispielsweise wird in Kapitel 4.3 gezeigt, daß bei einem wichtigen Fall des Tearing von Mehrkörpersystemen, die Transformation auf Gleichung (2.12) proportional zu $O(n_2^2)$ ist. Es ist dagegen natürlich ohne weiteres möglich, daß der direkte Einsatz von "Sparse Matrix" Methoden auf die ursprüngliche Gleichung der Ordnung n zu einer effizienteren Lösung führt als durch Tearing. Der umgekehrte Fall kann jedoch auch auftreten.

Um das Tearing praktisch durchführen zu können, müssen zwei Probleme gelöst werden. Zum einen müssen die Tearing-Variablen $\mathbf{x}_2$ und die Tearing-Gleichungen $\mathbf{f}_2$ bestimmt werden. Zum anderen muß auf die oben aufgeführten Gleichungssysteme niedrigerer Ordnung transformiert werden. Das letztere Problem ist einfach, wenn die Tearing-Variablen und die Tearing-Gleichungen festliegen: Zum Beispiel muß im nichtlinearen Fall nur das Gleichungssystem (2.9a), bei *bekannt angenommenem* $\mathbf{x}_2$, auf (explizit lösbare) Blockdreiecksform transformiert werden. Bei korrekten Tearing-Variablen und -Gleichungen ist dies immer möglich. Es werden dieselben Algorithmen verwendet, die auch zur Transformation auf Blockdreiecksform des kompletten Modells eingesetzt werden. Im linearen Fall ist die Transformation etwas aufwendiger.

Das erste, wesentlich schwierigere, Problem kann auch so formuliert werden: Finde Tearing-Variablen $\mathbf{x}_2$, so daß die Dimension von $\mathbf{x}_2$ minimal ist. Nach [Mah90] ist das so formulierte Problem NP-complete[9], und kann im wesentlichen nur durch Ausprobieren aller Möglichkeiten gelöst werden. In der Literatur gibt es eine große Anzahl heuristischer Algorithmen zur Lösung dieses Problems. Zum Beispiel werden in [Mah90] zwölf unterschiedliche Verfahren aufgeführt.

Zur Zeit gibt es in Dymola keinen Algorithmus um die Tearing-Variablen und -Gleichungen zu bestimmen. Es wird jedoch ein Sprachelement angeboten, mit dem der Anwender explizit gewünschte Tearing-Variablen und -Gleichungen festlegen kann. Nach einer Prüfung auf Zulässigkeit, wird auf die obigen Gleichungssysteme niedrigerer Ordnung transformiert. Für Mehrkörpersysteme und für elektrische Schaltungen reicht diese Eigenschaft aus, da aufgrund der Systemstruktur leicht Aussagen über eine günstige Wahl von Tearing-Variablen und -Gleichungen gemacht werden können.

Beispielsweise werden in elektrischen Schaltungsprogrammen zweierlei Arten von Methoden benutzt, um die Gleichungen für einen (meist sehr großen) elektrischen Schaltkreis aufzustellen und danach mit einem "Sparse-Matrix" Verfahren zu lösen (siehe z.B. [Chua87, McCa88]):

[9] Die Bezeichnung NP-complete steht für non-polynomial complete.

- Das *"Sparse-Tableau"*-Verfahren führt auf dieselben Gleichungen, die mittels der Klassen in Bild 2.1 auf Seite 18 erzeugt werden. Unbekannte sind hier die Potentiale aller Knoten, die Spannungsabfälle über alle Elemente und die Ströme durch alle Elemente.

- Die *modifizierte Knotenpunktsanalyse* (modified nodal analysis), welche z.B. in SPICE, dem wichtigsten Programmpaket für analoge elektrische Schaltkreise, verwendet wird, benutzt nur die Potentiale aller Knoten, sowie die Ströme durch "stromkontrollierte Elemente", als unbekannte Größen. Alle anderen Variablen werden vorab eliminiert. Mittels Tearing kann von den *"Sparse-Tableau"* Gleichungen auf die Gleichungen der Knotenpunktsanalyse transformiert werden. Die Tearing-Variablen sind die Unbekannten der Knotenpunktsanalyse. Die Tearing-Gleichungen sind die für die Through-Variablen (= Ströme durch die Elemente) erzeugten Gleichungen an Knotenpunkten (Summe aller Ströme ist Null an einem Knoten). Wenn z.B. die beiden Potentiale (= Tearing-Variablen) an einem Widerstand bekannt sind, kann sowohl der Spannungsabfall über den Widerstand, als auch der Strom durch den Widerstand (mittels des Ohm'schen Gesetzes) explizit berechnet werden. Der berechnete Strom gehört zum obigen Vektor $\mathbf{x}_1$ der nur noch bei der Stromsummenbildung in einer Gleichung, der Tearing-Gleichung, auftritt.

2.4 Modelle mit singulären differential-algebraischen Gleichungssystemen

Eine Transformation auf Blockdreiecksform, von deren Existenz in den letzten Abschnitten ausgegangen wurde, ist nur möglich, wenn die Jacobi-Matrix des Basis-Gleichungssystems regulär ist. Diese in Kapitel 2.2 getroffene Annahme wird jetzt fallengelassen. Als Beispiel ist in Bild 2.5 ein System aus [Elmq78] zu sehen, für das diese Annahme nicht zutrifft. Das Problem in diesem Beispiel liegt darin, daß jeder der beiden Kondensatoren den jeweiligen Spannungsabfall (u_1, u_2) als Zustandsgröße einführt. Durch die Zusammenschaltung der beiden Kapazitäten werden diese beiden Zustandsgrößen jedoch gleichgesetzt, d.h. das System hat in Wirklichkeit nur eine einzige Zustandsgröße anstatt zwei. Formal kann diese Eigenschaft durch die folgenden Gleichungen des Schaltkreises beschrieben werden:

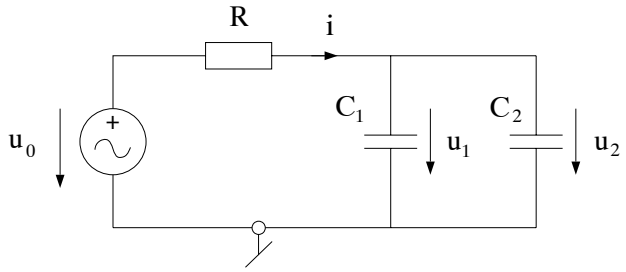


Bild 2.5: Modell mit höherem Index.

$$\begin{aligned} u_0 &= R\,i + u_1 \\ i &= C_1\,\dot{u}_1 + C_2\,\dot{u}_2 \\ u_2 &= u_1 \end{aligned} \qquad Occ = \begin{array}{c} \begin{matrix} i & \dot{u}_1 & \dot{u}_2 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array} . \tag{2.13}$$

Bei einem Simulationsproblem sind $i, \dot{u}_1$ und $\dot{u}_2$ unbekannt und die anderen Größen, d.h. u_0, u_1 und u_2, sind bekannt. Es gibt drei Gleichungen für die drei Unbekannten. Jedoch beschreibt die dritte Gleichung eine Beziehung zwischen den bekannten Größen u_1 und u_2, d.h. diese beiden Variablen sind nicht mehr unabhängig voneinander. Da in der dritten Gleichung keine der *Unbekannten* auftritt, "fehlt" eine Gleichung. Dies drückt sich durch eine Nullzeile in der Occurrence-Matrix *Occ* aus. Es ist deswegen nicht möglich, die Occurrence-Matrix auf Blockdreiecksform mit von Null verschiedenen Diagonalblöcken zu transformieren. Die Jacobi-Matrix des Systems ist damit strukturell singulär.

Die *allgemeine* Theorie zur Behandlung solcher Systeme wurde erst in den letzten Jahren entwickelt und ist noch nicht abgeschlossen. Für spezielle Systeme, z.B. Mehrkörpersysteme, sind Lösungsverfahren schon länger bekannt. Eine gute Übersicht über die Ergebnisse bis 1989 ist in [Bren89] zu finden.

Im folgenden wird diese Fragestellung näher untersucht. Betrachtet werden Anfangswertprobleme differential-algebraischer Gleichungssysteme (abgekürzt DAE[10]) in der allgemeinen Form:

$$\mathbf{f}\left(t, \mathbf{y}(t), \dot{\mathbf{y}}(t)\right) = \mathbf{0} \quad . \tag{2.14}$$

Die allgemeine Lösungstheorie linearer DAEs mit konstanten Koeffizienten ist schon lange bekannt [Gant59] und beruht auf dem Kronecker-Index eines singulären Matrizenbüschels. Die Verallgemeinerung des Kronecker-Index auf nichtlineare DAEs der Form (2.14) basiert auf folgendem nichtlinearen Gleichungssystem

$$\begin{array}{rcccccc}
0 & = & \mathbf{f} & = & \mathbf{f}_0(t, \mathbf{y}, \dot{\mathbf{y}}) & & \\
0 & = & \dfrac{d}{dt}\mathbf{f} & = & \mathbf{f}_1(t, \mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}) & = & \dfrac{\partial \mathbf{f}}{\partial \mathbf{t}} + \dfrac{\partial \mathbf{f}}{\partial \mathbf{y}}\dot{\mathbf{y}} + \dfrac{\partial \mathbf{f}}{\partial \dot{\mathbf{y}}}\ddot{\mathbf{y}} \\
 & & \vdots & & & & \vdots \\
0 & = & \dfrac{d^j}{dt^j}\mathbf{f} & = & \mathbf{f}_j(t, \mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(j+1)}) & = & \ldots + \dfrac{\partial \mathbf{f}}{\partial \dot{\mathbf{y}}}\mathbf{y}^{(j+1)}
\end{array} \tag{2.15}$$

in den separaten, abhängigen Variablen $\dot{\mathbf{y}}, \ldots, \mathbf{y}^{(j+1)}$ als Funktion der unabhängigen Variablen $\mathbf{y}$ und t (siehe z.B. [Bren89]). Man beachte, daß bei der folgenden Analyse dieses Gleichungssystem davon ausgegangen wird, daß die Ableitungen von $\dot{\mathbf{y}}$ als algebraische Variablen betrachtet werden, die unabhängig voneinander sind. Wenn $\frac{\partial \mathbf{f}}{\partial \dot{\mathbf{y}}}$ singulär ist, ist es nicht möglich nach $\mathbf{y}^{(j+1)}$ aufzulösen. Falls jedoch für endliches j nach $\dot{\mathbf{y}}$ aufgelöst werden kann, wird definiert:

> *Angenommen die differential-algebraische Gleichung (2.14) ist über dem gewünschten Zeitintervall eindeutig lösbar. Dann ist der* **differentielle Index** *(engl.: differential index) von (2.14) die kleinste Zahl j, so daß man aus den ersten $j+1$ Gleichungen des Systems (2.15) $\dot{\mathbf{y}}$ eindeutig als stetige Funktion von $\mathbf{y}$ und t bestimmen kann.*

Explizite und implizite (reguläre) Differentialgleichungen haben einen differentiellen Index = 0. Der differentielle Index einer DAE ist eine wichtige Struktureigenschaft, die z.B.

[10] DAE steht für *Differential-Algebraic Equations.*

angibt, wie oft eine DAE differenziert werden muß, um sie (zumindest lokal) in eine explizite Differentialgleichung transformieren zu können. Von Hairer, Lubich und Roche wurde in [Hair89] eine andere Definition des Index einer DAE angegeben, der zur Unterscheidung des differentiellen Index als Störungsindex bezeichnet wird (siehe auch [Gear90, Eich92]):

> *Der* **Störungsindex** *(engl.: perturbation index) der DAE (2.14) ist die kleinste Zahl j, so daß die Differenz zwischen der Lösung von (2.14) und der Lösung der gestörten DAE*
>
> $$\mathbf{f}\left(t, \hat{\mathbf{y}}(t), \dot{\hat{\mathbf{y}}}(t)\right) = \boldsymbol{\epsilon}(t) \tag{2.16}$$
>
> *über ein endliches Intervall $t \in [0, t_e]$ durch einen Ausdruck der Form*
>
> $$\begin{aligned} \max \|\hat{\mathbf{y}}(t) - \mathbf{y}(t)\| \;\leq\; C \Bigg(& \|\hat{\mathbf{y}}(0) - \mathbf{y}(0)\| + \max \Big\| \int_0^{t_e} \boldsymbol{\epsilon}(\tau) d\tau \Big\| + \max \|\boldsymbol{\epsilon}(t)\| \\ & + \max \|\dot{\boldsymbol{\epsilon}}(t)\| + \ldots + \max \|\boldsymbol{\epsilon}^{(j-1)}(t)\| \Bigg) \end{aligned} \tag{2.17}$$
>
> *abgeschätzt werden kann.*

Der Störungsindex spielt für die numerische Lösung einer DAE eine wichtige Rolle, da er den Einfluß unvermeidlicher Rundungsfehler in der Auswertung der Funktion $\mathbf{f}$ widerspiegelt: Kleine Rundungsfehler $\boldsymbol{\epsilon}(t)$ wirken sich bis zur (j-1)-ten Ableitung $\boldsymbol{\epsilon}^{(j-1)}$ in der Lösung $\mathbf{y}(t)$ aus. Das heißt, auch wenn der Rundungsfehler klein ist, können seine Ableitungen groß sein und damit zu einer stark fehlerbehafteten Lösung führen. Bei der Diskretisierung durch ein numerisches Verfahren wird die Ableitung $\boldsymbol{\epsilon}^{(j-1)}$ durch "numerische Differentiation" ersetzt, so daß ein Term der Form $O(\boldsymbol{\epsilon})/h^{j-1}$ auftritt. Je *kleiner* die Schrittweite h gewählt wird, um so größer wird der Einfluß dieses Fehlerterms! Dies ist ein Grund für die numerischen Schwierigkeiten bei der *direkten* Behandlung von DAEs mit Störungsindex ≥ 2.

Es stellt sich die Frage, welcher Zusammenhang zwischen dem differentiellen Index und dem Störungsindex besteht. Gear hat in [Gear90] die folgende Beziehung abgeleitet:

$$\text{differentieller Index} \leq \text{Störungsindex} \leq \text{differentieller Index} + 1.$$

Gear hat weiter bewiesen, daß der differentielle Index und der Störungsindex identisch sind, wenn bei einer DAE die Variablen $\dot{\mathbf{y}}$ nur linear auftreten und die Terme in denen $\dot{\mathbf{y}}$ auftritt als totales Differential interpretiert werden können. Ein Spezialfall hiervon sind semi-explizite DAEs (siehe Gleichung (2.25)).

In der Literatur werden beide Index-Definitionen benutzt. Diese gleichzeitige Verwendung unterschiedlicher Definitionen für den Index einer DAE ist für den Anwender unbefriedigend und verwirrend. In dieser Arbeit wird deswegen die Definition des differentiellen Index so modifiziert, daß dieser mit dem Störungsindex übereinstimmt.

Im Gegensatz zur Definition des differentiellen Index, geht die hier verwendete, neue Indexdefinition nicht von der DAE (2.14), sondern von der folgenden Funktion (2.18) aus, die die Struktur der DAE detaillierter beschreibt:

$$\mathbf{f}\left(t, \mathbf{x}(t), \dot{\mathbf{x}}(t), \mathbf{w}(t)\right) = \mathbf{0} \quad . \tag{2.18}$$

In (2.18) treten alle Ableitungen von $\mathbf{x}$ auch wirklich auf, d.h. es gibt mindestens ein i und ein j, so daß $\partial f_i / \partial \dot{x}_j \neq 0$. Alle nicht differenziert auftretenden Unbekannten werden

im Vektor der algebraischen Variablen $\mathbf{w}$ zusammengefaßt. Die Funktion $\mathbf{f}$ besteht aus $\dim(\mathbf{x}) + \dim(\mathbf{w})$ Gleichungen für die $\dim(\mathbf{x})$ Unbekannten $\dot{\mathbf{x}}$ und die $\dim(\mathbf{w})$ Unbekannten $\mathbf{w}$. Mit (2.18) kann das folgende nichtlineare Gleichungssystem

$$\begin{aligned} 0 &= \mathbf{f} &&= \mathbf{f}_1(t, \mathbf{x}, \dot{\mathbf{x}}, \mathbf{w}) \\ 0 &= \frac{d}{dt}\mathbf{f} &&= \mathbf{f}_2(t, \mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}, \mathbf{w}, \dot{\mathbf{w}}) \\ &\vdots \\ 0 &= \frac{d^{(j-1)}}{dt^{(j-1)}}\mathbf{f} &&= \mathbf{f}_j(t, \mathbf{x}, \dot{\mathbf{x}}, \ldots, \mathbf{x}^{(j)}, \mathbf{w}, \dot{\mathbf{w}}, \ldots, \mathbf{w}^{(j-1)}) \end{aligned} \tag{2.19}$$

in den separaten, abhängigen Variablen $\dot{\mathbf{x}}, \ldots, \mathbf{x}^{(j)}, \mathbf{w}, \ldots, \mathbf{w}^{(j-1)}$ als Funktion der unabhängigen Variablen $\mathbf{x}$ und t aufgestellt werden. Wie bei der Definition des differentiellen Index, werden auch hier alle höheren Ableitungen von $\dot{\mathbf{x}}$ und $\mathbf{w}$ als voneinander unabhängige, algebraische Variable angesehen. Die neue Index Definition lautet:

> *Angenommen die differential-algebraische Gleichung (2.18) ist über dem gewünschten Zeitintervall eindeutig lösbar. Dann ist der* **differential-algebraische Index** *(engl.: differential algebraic index) von (2.18) die kleinste Zahl j, so daß man aus den ersten j Gleichungen des Systems (2.19) die Größen* $\dot{\mathbf{x}}$ *und* $\mathbf{w}$ *eindeutig als stetige Funktionen von* $\mathbf{x}$ *und* t *bestimmen kann. Wenn (2.18) keine Funktion von* $\mathbf{w}$ *ist und wenn* $\partial\mathbf{f}/\partial\dot{\mathbf{x}}$ *regulär ist, dann wird der Index als Null definiert.*

Im Anhang A.1 wird gezeigt, daß für jede beliebige DAE der modifizierte differentielle Index gleich dem Störungsindex ist. Damit gelten alle in der Literatur bekannten Eigenschaften des Störungsindex sofort auch für den modifizierten differentiellen Index. Insbesondere gilt, daß die direkte numerische Lösung einer DAE mit Index ≥ 2 problematisch ist. Im restlichen Teil dieser Arbeit wird der Störungsindex bzw. der modifizierte differentielle Index verwendet.

Der modifizierte differentielle Index kann dazu verwendet werden, numerische Lösungsverfahren für *allgemeine* DAEs zu verbessern. Ein Beispiel ist der bekannte Integrator DASSL von Petzold [Petz82, Bren89]. Dort wird die DAE (2.14) diskretisiert, indem $\dot{\mathbf{y}}$ durch einen Rückwärts-Differenzenquotienten ersetzt wird. Im einfachsten Fall, bei der Verwendung des "Backward-Euler" Verfahrens, geht (2.14) über in:

$$\mathbf{f}\left(t_n, \mathbf{y}_{n+1}, \frac{\mathbf{y}_{n+1} - \mathbf{y}_n}{h}\right) = \mathbf{0} \quad . \tag{2.20}$$

Hierbei ist $\mathbf{y}_{n+1}$ der im aktuellen Schritt zu berechnende neue Wert, $\mathbf{y}_n$ ist der im vorherigen Schritt berechnete Wert und h ist die Schrittweite. Die Gleichung (2.20) wird mit einem Newton-Verfahren gelöst, das zu folgender Iterationsvorschrift führt:

$$\left(\frac{\partial\mathbf{f}}{\partial\mathbf{y}} + \frac{1}{h}\frac{\partial\mathbf{f}}{\partial\dot{\mathbf{y}}}\right)_n \triangle\mathbf{y}_{n+1} = \mathbf{f}_n \quad . \tag{2.21}$$

Hierbei ist $\triangle\mathbf{y}_{n+1} = \mathbf{y}_{n+1} - \mathbf{y}_n$. Ein Problem tritt auf, wenn die Schrittweite h sehr klein gewählt wird, da dann in der Jacobi-Matrix der Term $\partial\mathbf{f}/\partial\dot{\mathbf{y}}$ überwiegt. Dies sieht man

besonders deutlich, wenn die Gleichung mit h multipliziert wird:

$$\left(h\frac{\partial \mathbf{f}}{\partial \mathbf{y}} + \frac{\partial \mathbf{f}}{\partial \dot{\mathbf{y}}} \right)_n \triangle \mathbf{y}_{n+1} = h\mathbf{f}_n \quad . \tag{2.22}$$

Wenn nun in (2.22) die Schrittweite Null wird, so geht die Jacobi-Matrix in $\partial \mathbf{f}/\partial \dot{\mathbf{y}}$ über. Dies ist unproblematisch, wenn die DAE den Index 0 besitzt, da die Matrix $\partial \mathbf{f}/\partial \dot{\mathbf{y}}$ dann regulär ist. Wenn jedoch eine DAE vorliegt, die einen höheren Index besitzt, z.B. den Index 1, ist diese Matrix singulär[11]. Eine Schrittweitensteuerung wird dann Schwierigkeiten haben, da ein *Verkleinern der Schrittweite* die Kondition der Jacobi-Matrix des Newton-Verfahrens verschlechtert. Wird das Newton-Verfahren nicht auf die DAE (2.14), sondern auf die DAE (2.18) angewandt, ergibt sich die folgende Iterationsvorschrift:

$$\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}} + \frac{1}{h}\frac{\partial \mathbf{f}}{\partial \dot{\mathbf{x}}} \vdots \frac{\partial \mathbf{f}}{\partial \mathbf{w}} \right]_n \left[\begin{array}{c} \triangle \mathbf{x}_{n+1} \\ \triangle \mathbf{w}_{n+1} \end{array} \right] = \mathbf{f}_n \quad . \tag{2.23}$$

Mit $\mathbf{y} = [\mathbf{x};\ \mathbf{w}]$ ist die Jacobi-Matrix in (2.23) identisch zur vorigen Jacobi-Matrix in (2.21). Jedoch ist in (2.23) die DAE-Struktur genauer bekannt, sodaß die Newton-Iteration leicht umgeformt werden kann:

$$\left[h\frac{\partial \mathbf{f}}{\partial \mathbf{x}} + \frac{\partial \mathbf{f}}{\partial \dot{\mathbf{x}}} \vdots \frac{\partial \mathbf{f}}{\partial \mathbf{w}} \right]_n \left[\begin{array}{c} \triangle \mathbf{x}^*_{n+1} \\ \triangle \mathbf{w}_{n+1} \end{array} \right] = \mathbf{f}_n \tag{2.24a}$$

$$\triangle \mathbf{x}_{n+1} = h \triangle \mathbf{x}^*_{n+1} \quad . \tag{2.24b}$$

Wenn jetzt in (2.24) die Schrittweite h klein wird, so verschwindet wieder der erste Term der Jacobi-Matrix, aber aufgrund der Definition des modifizierten differentiellen Index, ist die *verbleibende Matrix regulär, wenn die DAE den Index 1 (oder 0) besitzt!* Durch diese einfache Maßnahme kann also bei Integratoren wie DASSL erreicht werden, daß auch bei Index-1 DAEs die Jacobi-Matrix des Newton-Verfahrens gegen eine reguläre Matrix konvergiert, wenn die Schrittweite Null wird.

Anhand zweier Beispiele soll der Indexbegriff verdeutlicht werden:

- Systeme mit *Störungsindex = 1:*
  Die semi-explizite differential-algebraische Gleichung

$$\dot{\mathbf{x}}(t) = \mathbf{f}_1\left(t, \mathbf{x}(t), \mathbf{w}(t)\right) \tag{2.25a}$$

$$\mathbf{0} = \mathbf{f}_2\left(t, \mathbf{x}(t), \mathbf{w}(t)\right) \tag{2.25b}$$

  hat dann und nur dann den Störungsindex 1, wenn

$$\frac{\partial \mathbf{f}_2}{\partial \mathbf{w}} \text{ regulär ist} \quad . \tag{2.26}$$

  Dies folgt aus dem Satz über implizite Funktionen, da nur mit der Eigenschaft (2.26) die Gleichung (2.25b) eindeutig nach $\mathbf{w}$ aufgelöst werden kann. Die so erhaltene Funktion $\mathbf{w}(\mathbf{x}, t)$ kann in (2.25a) eingesetzt werden und man erhält auch $\dot{\mathbf{x}}$ als Funktion von $\mathbf{x}$ und t.

[11] Man kann *immer* eine Schrittweite h finden, so daß die Jacobi-Matrix in (2.22) regulär ist, da (2.22) ein Matrizenbüschel in h darstellt, welches nur für endlich viele Werte von h singulär ist. Es ist jedoch schwierig eine Schrittweitenstrategie zu finden, wenn ein Verkleinern der Schrittweite problematisch ist.

- Systeme mit *Störungsindex = 2:*
  Die folgende spezielle, semi-explizite differential-algebraische Gleichung

$$\dot{\mathbf{x}}(t) = \mathbf{f}_1\left(t, \mathbf{x}(t), \mathbf{w}(t)\right) \tag{2.27a}$$
$$\mathbf{0} = \mathbf{f}_2\left(t, \mathbf{x}(t)\right) \tag{2.27b}$$

  ist kein Index 1 System, da die Jacobi-Matrix (2.26) eine Nullmatrix und daher singulär ist. Einmalige Differentiation der algebraischen Gleichung führt auf

$$\begin{aligned}\mathbf{0} &= \frac{d\mathbf{f}_2}{dt} = \frac{\partial \mathbf{f}_2}{\partial \mathbf{x}}\dot{\mathbf{x}} + \frac{\partial \mathbf{f}_2}{\partial t} \\ &= \frac{\partial \mathbf{f}_2(\mathbf{x}, t)}{\partial \mathbf{x}}\mathbf{f}_1(\mathbf{x}, \mathbf{w}, t) + \frac{\partial \mathbf{f}_2(\mathbf{x}, t)}{\partial t} \quad .\end{aligned} \tag{2.28}$$

  Die Gleichungen (2.27a) und (2.28) sind semi-explizite differential-algebraische Gleichungen und haben deswegen den Index 1, wenn

$$\frac{\partial}{\partial \mathbf{w}}\frac{d\mathbf{f}_2}{dt} = \frac{\partial \mathbf{f}_2}{\partial \mathbf{x}}\frac{\partial \mathbf{f}_1}{\partial \mathbf{w}} \text{ regulär ist.} \tag{2.29}$$

  Die Gleichung (2.27) hat damit den Störungsindex 2, wenn die Bedingung (2.29) zutrifft. Ansonsten hat die Gleichung einen höheren Index.

Im Gegensatz zu expliziten Differentialgleichungen müssen die *Anfangswerte* einer DAE (2.18) gewisse *Konsistenzbedingungen* erfüllen, um die Existenz einer eindeutigen Lösung zu garantieren [Pant88, Bren89, Eich92]. Es reicht nicht aus, daß die DAE zum Anfangszeitpunkt t_0 erfüllt ist, d.h.

$$\mathbf{f}\left(t_0, \mathbf{x}(t_0), \dot{\mathbf{x}}(t_0), \mathbf{w}(t_0)\right) = \mathbf{0} \ . \tag{2.30}$$

Stattdessen müssen die Anfangswerte $\mathbf{x}_0 = \mathbf{x}(t_0), \mathbf{w}_0 = \mathbf{w}(t_0)$ alle j Gleichungen (2.19) erfüllen, die bei der Bestimmung des Störungsindex auftreten. Zum Beispiel müssen die Anfangswerte $\mathbf{x}_0, \mathbf{w}_0$ des Index-2 Systems (2.27) so gewählt werden, daß die Gleichungen (2.27b, 2.28) erfüllt sind. Der Anfangswert $\dot{\mathbf{x}}(t_0)$ ergibt sich dann aus Gleichung (2.27a).

Die Forderung nach konsistenten Anfangsbedingungen hat eine wichtige Konsequenz: *Gleichgültig* welches Lösungs-Verfahren für das Anfangswertproblem einer DAE benutzt wird, es werden die j Gleichungen (2.19) benötigt, um zumindest konsistente Anfangsbedingungen bestimmen zu können.

Wie aus dem Index 2 Beispiel (2.27) zu ersehen ist, müssen nicht unbedingt alle Gleichungen der Funktion $\mathbf{f}$ differenziert werden, um $\dot{\mathbf{x}}, \mathbf{w}$ als Funktion von $\mathbf{x}, t$ angeben zu können. Pantelides [Pant88] hat hierzu einen (sehr kompakten) Algorithmus angegeben, mit dem die *minimale* Anzahl notwendiger Differentiationen für jede Gleichung des Ausgangs-Gleichungssystems (2.18) bestimmt werden kann. Dies bedeutet, daß sowohl der Störungsindex einer DAE als auch alle Gleichungen, die von konsistenten Anfangsbedingungen eingehalten werden müssen, durch ein Programm automatisch ermittelt werden können

Die Grundidee des Algorithmus von Pantelides ist einfach. Der Algorithmus basiert auf einer Erweiterung des *"output set assignments"*, das auf Seite 24 angesprochen wurde. Die "output set assignment" Transformation ist der erste Schritt, um eine Occurrence-Matrix

auf Blockdreiecksform zu transformieren. Hierzu wird die Occurence-Matrix durch Zeilen-Permutationen so umgeformt, daß jedes Diagonalelement eine Eins enthält. Wenn die Transformation nicht vollständig durchgeführt werden kann, hat die DAE einen (strukturellen) Index > 1. Beim "output set assignment" werden diejenigen Gleichungen identifiziert, die zu einer strukturellen Singularität führen. Mit dem Pantelides Algorithmus werden im wesentlichen gerade diese Gleichungen differenziert (siehe auch Anhang A.2). Der Rechenaufwand des Pantelides-Algorithmus ist nur $O(n\tau)$, wobei n die Zahl der *End*gleichungen und τ die Gesamtzahl der Variablen in allen Gleichungen ist (= Anzahl der Eins-Elemente in der Occurence-Matrix), die das erzeugte Endsystem (2.19) beschreiben.

Der Fehler in der Lösung, welcher sich im Störungsindex widerspiegelt, kann in einer DAE für verschiedene Variablen unterschiedliche Größenordnungen besitzen. Zum Beispiel wurde in [Hair89] gezeigt, daß in der obigen semi-expliziten DAE (2.27) vom Index 2 die erste Ableitung des Rundungsfehlers nur in die Lösung der algebraischen Variablen $\mathbf{w}$ eingeht, nicht jedoch in die Lösung der Variablen $\mathbf{x}$. Diese Index-2 DAE könnte mit dem Index-1 Löser DASSL integriert werden, wenn die Schrittweitensteuerung für die Variablen $\mathbf{w}$ abgeschaltet wird (d.h. die relativen und absoluten Toleranzen von $\mathbf{w}$ werden auf einen hohen Wert, z.B. 10^8, gesetzt). Man muß sich jedoch bewußt sein, daß die Lösung von $\mathbf{w}$ stärker fehlerbehaftet sein kann, als die Lösung von $\mathbf{x}$. Dies ist unkritisch, wenn man am Lösungsverlauf von $\mathbf{w}$ nicht interessiert ist.

Man kann die Probleme, die sich im Störungsindex widerspiegeln, vermeiden. Hierzu werden bei der numerischen Lösung nicht nur die ursprüngliche DAE (2.18), sondern auch alle Ableitungen der DAE benutzt, die für konsistente Anfangsbedingungen erfüllt sein müssen und die z.B. durch den Pantelides-Algorithmus ermittelt werden können. Dann existieren einige unterschiedliche Methoden um die Lösung der DAE zu ermitteln. Diese Methoden sollen kurz am Beispiel des obigen Index-2 Beispiels skizziert werden. Die benötigten Gleichungen werden noch einmal zusammengestellt:

$$\dot{\mathbf{x}} = \mathbf{f}_1(t, \mathbf{x}, \mathbf{w}) \tag{2.31a}$$

$$\mathbf{0} = \mathbf{f}_2(t, \mathbf{x}) \tag{2.31b}$$

$$\mathbf{0} = \mathbf{f}_3(t, \mathbf{x}, \mathbf{w}) \quad \left(= \frac{\partial \mathbf{f}_2}{\partial \mathbf{x}} \mathbf{f}_1 + \frac{\partial \mathbf{f}_2}{\partial t}\right) . \tag{2.31c}$$

Die ursprünglichen Gleichungen sind (2.31a, 2.31b). Durch Differentiation von (2.31b) ergibt sich (2.31c). Gleichungen (2.31a, 2.31c) zusammen sind ein Index-1 System. Wenn die Integrationszeit nicht zu lange ist, kann dieses Index-1 System auch alleine integriert werden. Mit größer werdender Endzeit führen aber kleine Diskretisierungsfehler dazu, daß die numerische Lösung immer weiter von der Erfüllung der Gleichung (2.31b) "wegdriftet", da diese Gleichung während der Integration nicht explizit erfaßt wird. Deswegen werden in den folgenden Methoden immer alle Gleichungen zusammen berücksichtigt.

1. *Die "Dummy Derivative" Methode:*
   Diese Methode wurde unabhängig voneinander von Cellier/Elmqvist [Cell92, Cell93a] und von Mattsson/Söderlind [Matt92, Matt93] entwickelt, wobei Mattsson/Söderlind noch zusätzlich einen Algorithmus angeben, aus welchen Teilmengen der Deskriptorvariablen die Zustandsgrößen zu wählen sind. Spezialfälle sind schon länger bekannt, z.B. in der Mehrkörperdynamik unter dem Namen "coordinate partitioning" [Weha82].
   Die Grundidee ist einfach: Das Gleichungssystem soll in die Zustandsform überführt

werden. Aufgrund der Gleichung (2.31b) bestehen zwischen den "abgeleiteten" Größen $\mathbf{x}$ Zusammenhänge. Diese Größen sind also nicht mehr unabhängig voneinander. Mittels numerischer Verfahren oder auf andere Weise wird eine Aufteilung der Variablen $\mathbf{x}$ in zwei Teile gefunden:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}, \quad \text{wobei } \frac{\partial \mathbf{f}_2}{\partial \mathbf{x}_2} \text{ regulär ist.} \tag{2.32}$$

Die $(\dim(\mathbf{f}_1) - \dim(\mathbf{f}_2))$ voneinander unabhängigen Größen $\mathbf{x}_1$ werden als Zustandsvariablen verwendet. Die $\dim(\mathbf{f}_2)$ abhängigen Größen $\mathbf{x}_2$ *und* deren Ableitungen $\dot{\mathbf{x}}_2$ (= Dummy Derivatives) werden als voneinander unabhängige Unbekannte angesehen. Für das Simulationsproblem bestehen die Gleichungen (2.31) damit aus $\dim(\mathbf{f}_1)+2\cdot\dim(\mathbf{f}_2)$ Gleichungen und aus ebensovielen Unbekannten $(= \dot{\mathbf{x}}, \mathbf{x}_2, \mathbf{w})$. Bekannt, d.h. vom Integrator geliefert, ist $\mathbf{x}_1$. An den Integrator zurückgegeben wird $\dot{\mathbf{x}}_1$. Wenn die Gleichungen $\mathbf{f}_2, \mathbf{f}_3$ nichtlinear sind, können sie entweder mit den Methoden von Abschnitt 2.3 behandelt werden oder die Variablen $\dot{\mathbf{x}}_2, \mathbf{x}_2, \mathbf{w}$ (oder eine Untermenge davon), werden als "algebraische" Variablen einer Index-1 DAE behandelt.

Der Vorteil dieser Methode besteht darin, daß sie für alle DAE Systeme anwendbar ist, und daß Standard Index-0 oder Index-1 Löser benutzt werden können. Es ist jedoch im allgemeinen schwierig, die geforderte Aufteilung von $\mathbf{x}$ zu finden. Außerdem kann es sein, daß diese Aufteilung nicht während der gesamten Lösungstrajektorie beibehalten werden kann, und daß auf andere Zustandsgrößen gewechselt werden muß. Für die speziellen Index-3 Systeme der Mehrkörperdynamik gibt es recht ausgefeilte Methoden zur Bestimmung dieser Aufteilung und zum geeigneten Wechseln der Zustandsgrößen während der Simulation, z.B. [Weha82, Leis92]. Weiterhin gibt es hier auch Methoden um die Zustandsgrößen aufgrund einer "Mechanik-Sichtweise" zu ermitteln und die nichtlinearen Gleichungen $\mathbf{f}_2$ analytisch zu lösen [Woer88, Hill93, Kecs93, Kecs93a].

2. *Transformation auf eine spezielle DAE mit Störungsindex 2:*
   Nach Gear [Gear86], können die Gleichungen (2.31) durch Einführung von $\dim(\mathbf{f}_2)$ zusätzlichen Lagrange'schen Multiplikatoren $\boldsymbol{\mu}$ in die folgende äquivalente Index-2 DAE übergeführt werden:

$$\dot{\mathbf{x}} = \mathbf{f}_1 + \left(\frac{\partial \mathbf{f}_2}{\partial \mathbf{x}}\right)^T \boldsymbol{\mu} \tag{2.33a}$$

$$\mathbf{0} = \mathbf{f}_2 \tag{2.33b}$$

$$\mathbf{0} = \mathbf{f}_3 \,. \tag{2.33c}$$

In [Gear86] wird gezeigt, daß diese DAE dieselbe (exakte) Lösung (mit $\boldsymbol{\mu} = \mathbf{0}$) besitzt wie die Ausgangs-DAE (2.31), und daß die DAE (2.33) den Index 2 hat.

Im Unterschied zur Index-2 DAE (2.31) hängt jedoch jetzt die Lösung von $\boldsymbol{\mu}$ von der ersten Ableitung der Rundungsfehler ab, nicht mehr jedoch die Lösung von $\mathbf{w}$! Bei Verwendung eines Index-1 Lösers, wie DASSL, kann damit eine zuverlässige Lösung in $\mathbf{x}$ und $\mathbf{w}$ erwartet werden, nicht mehr jedoch in den (allerdings auch nicht interessierenden) Variablen $\boldsymbol{\mu}$. Gear hat in [Gear86] gezeigt, daß zur Lösung von (2.33) jedes stabile, implizite Mehrschrittverfahren benutzt werden kann, wenn im Diskretisierungsverfahren für *vergangene* Werte von $\boldsymbol{\mu}$ *immer* die exakte Lösung $\boldsymbol{\mu} = \mathbf{0}$ eingesetzt wird.

Dieses Verfahren ist robuster und einfacher einsetzbar als die Dummy Derivative Methode, weil *keine* Zustandsgrößen ausgewählt werden müssen. Nachteilig ist, daß die Systemordnung durch die Einführung der (eigentlich unnötigen) Lagrange'schen Multiplikatoren erhöht wird. D.h. es ist mit Effizienzeinbußen zu rechnen, wenn die Struktur der Gleichungen im Integrator nicht ausgenutzt wird.

In [Gear88] hat Gear dieses Verfahren auf beliebige DAEs verallgemeinert. Die Verallgemeinerung ist, abgesehen von Spezialfällen, leider nicht praktisch anwendbar, da z.B. vorausgesetzt wird, daß ein nichtlineares Gleichungssystem nach einer Variablen explizit aufgelöst werden kann. Die Methode ist jedoch theoretisch von Interesse, da damit nachgewiesen werden kann, daß jede DAE in eine gewöhnliche Differentialgleichung umgeformt werden kann. Am Ende dieses Abschnitts wird eine neue, andersgeartete Verallgemeinerung des Gear'schen Verfahrens erläutert und im Anhang bewiesen. Diese neue Methode kann (automatisiert) auf beliebige DAEs angewandt werden.

3. *Lösung als überbestimmte DAE:*
   Die DAE (2.31) besteht aus $\dim(\mathbf{f}_1)+2\cdot\dim(\mathbf{f}_2)$ Gleichungen für die $\dim(\mathbf{f}_1)+\dim(\mathbf{f}_2)$ Unbekannten $\mathbf{x}(t), \mathbf{w}(t)$, d.h. es gibt mehr Gleichungen als Unbekannte. Aufgrund der Herleitung der Gleichungen ist jedoch sichergestellt, daß die Gleichungen zueinander kompatibel und nicht widersprüchlich sind. Führer [Fueh88] schlägt vor, dieses überbestimmte Gleichungssystem direkt zu lösen. Durch kleine, unvermeidbare Fehler in der Diskretisierung geht jedoch die Eigenschaft verloren, daß die Gleichungen zueinander kompatibel sind. Führer löst das diskretisierte Gleichungssystem deswegen in einem "Least Square" ähnlichen Sinne. Praktisch wichtig ist, daß er auch eine robuste Implementierung dieses Verfahrens – ODASSL – anbietet. ODASSL ist eine Modifikation von DASSL, bei der im wesentlichen der "lineare Algebra"-Teil durch ein geeignetes "Least Square" ähnliches Verfahren ersetzt wurde. In [Fueh91] wird gezeigt, daß das ODASSL-Verfahren *identisch* mit dem Gear'schen Stabilisierungsverfahren ist, wenn $\mathbf{f}_2$ linear von $\mathbf{x}$ abhängt.

   Der Vorteil liegt hier vor allem in der Allgemeinheit des zur Verfügung gestellten Integrators. Mit der Variante ODASSLRT (= ODASSL mit RooT-Finder) können z.B. auch die im nächsten Kapitel 3 behandelten ereignisabhängigen Systeme gelöst werden. ODASSLRT ist damit wohl einer der allgemeinsten, heute *verfügbaren*, Spezial-Integratoren für DAEs mit höherem Index. Zur Zeit können jedoch nur eingeschränkte DAE-Klassen mit höherem Index behandelt werden. Zum anderen werden (noch) keine Struktureigenschaften der DAE ausgenutzt (z.B. daß $\mathbf{f}_3$ die Ableitung von $\mathbf{f}_2$ ist).

4. *Projektionsverfahren:*
   Hier besteht der Integrator im wesentlichen aus zwei Teilen: Aus einem "normalen" Index-1 Löser, mit dem die Index-1 DAE (2.31a, 2.31c) gelöst wird und aus einem Projektionsverfahren, mit dem die mit dem Index-1 Löser erhaltene Lösung nach jedem Integrationsschritt auf die durch Gleichung (2.31b) beschriebene Mannigfaltigkeit projieziert wird. Die Konvergenz dieser Verfahrensklasse, d.h. die Eigenschaft, daß sich nur die Komponenten des Fehlers fortpflanzen, die in der durch $\mathbf{f}_2$ definierten Mannigfaltigkeit liegen, wurde von Shampine [Sham86] für Einschritt-Verfahren, von Eich [Eich92] für (BDF) Mehrschritt-Verfahren und von Lubich [Lubi91] für Extrapolations-Verfahren gezeigt.

   Der Vorteil dieser Verfahrensklasse liegt wahrscheinlich in einer erhöhten Verfahrens-Effizienz. Nachteilig ist, daß es kaum *verfügbare*, robuste Integratoren für eine hin-

reichend allgemeine DAE-Klasse mit höherem Index gibt. Am allgemeinsten ist hier wohl MEXX von Lubich [Lubi91].

Mit Ausnahme der "Dummy Derivative" Methode, können die oben aufgeführten Verfahren zur Zeit nur bei DAEs mit speziellen Strukturen benutzt werden. Es ist jedoch möglich, das Verfahren von Gear so zu verallgemeinern, daß es *automatisiert* auf *alle DAEs* angewandt werden kann, die *strukturell* singulär sind. Wenn der Index nicht von speziellen numerischen Werten des Modells abhängt, sondern eine strukturelle Eigenschaft ist, kann mit dem Algorithmus von Pantelides [Pant88] die minimale Anzahl von Differentiationen jeder Gleichung des Ausgangsgleichungssystems (2.18) ermittelt werden. Dies führt auf das folgende Gleichungssystem (Details siehe Anhang A.2):

$$\dot{\mathbf{x}} = \mathbf{z}_0 \tag{2.34a}$$

$$\mathbf{0} = \mathbf{F}(t, \mathbf{x}, \mathbf{z}) = \begin{bmatrix} \mathbf{f}(t, \mathbf{x}, \mathbf{z}_1) \\ \mathbf{J}(t, \mathbf{x}, \mathbf{z}_a) \begin{bmatrix} \mathbf{z}_0 \\ \dot{\mathbf{z}}_a \end{bmatrix} + \mathbf{g}(t, \mathbf{x}, \mathbf{z}_a) \end{bmatrix} . \tag{2.34b}$$

Als Vorbereitung auf die nachfolgende Transformation wird in der ersten Zeile für die Ableitung von $\mathbf{x}$ ein anderer Name ($\mathbf{z}_0$) eingeführt. An allen Stellen, an denen $\dot{\mathbf{x}}$ auftritt, wird $\mathbf{z}_0$ verwendet. Danach wird das Ausgangsgleichungssystem $\mathbf{f}$ (2.18) aufgeführt. Hierbei wird für die beiden unbekannten Argumente $\mathbf{z}_0$ und $\mathbf{w}$ die Abkürzung $\mathbf{z}_1$ benutzt. Schließlich sind im unteren Teil der Funktion $\mathbf{F}$ die Gleichungen angegeben, die mit Hilfe des Pantelides Algorithmus erzeugt werden (durch maximal (j-1) maliges Ableiten ausgewählter Gleichungen von $\mathbf{f}$, wobei j der Störungsindex von $\mathbf{f}$ ist). Der Vektor $\mathbf{z}_a$ besteht aus einer Teilmenge von $\mathbf{z}_1$, sowie aus höheren Ableitungen von Elementen aus $\dot{\mathbf{x}}$ und $\mathbf{w}$. Das Argument $\mathbf{z}$ von $\mathbf{F}$ ist eine Zusammenfassung der Vektoren $\mathbf{z}_0, \mathbf{z}_a$ und $\dot{\mathbf{z}}_a$. Elemente, die z.B. sowohl in $\mathbf{z}_a$ als auch in $\dot{\mathbf{z}}_a$ auftreten, werden nur einmal in $\mathbf{z}$ aufgeführt.

(2.34b) ist ein überbestimmtes algebraisches Gleichungssystem in den unbekannten Variablen $\mathbf{z}$. Durch Einführen von "dim($\mathbf{g}$)" zusätzlichen unbekannten Variablen $\boldsymbol{\mu}$ und "(dim($\mathbf{z}$) - dim($\mathbf{f}$))" zusätzlichen Gleichungen, kann die überbestimmte semi-explizite DAE (2.34) in die folgende DAE überführt werden:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_0 \\ \mathbf{0} \end{bmatrix} + \mathbf{J}^T(t, \mathbf{x}, \mathbf{z}_a)\, \boldsymbol{\mu} \tag{2.35a}$$

$$\mathbf{0} = \mathbf{F}(t, \mathbf{x}, \mathbf{z}) . \tag{2.35b}$$

Im Anhang A.2 wird gezeigt, daß jede Lösung $\mathbf{x}(t),\ \mathbf{w}(t)$ der DAE (2.35) auch Lösung der Ausgangs-DAE (2.18) ist, und daß umgekehrt, jede Lösung von (2.18) Lösung von (2.35) ist. In beiden Fällen ist $\boldsymbol{\mu}$ gleich Null. Weiterhin wird gezeigt, daß die DAE (2.35) einen Störungsindex von 2 besitzt, wobei nur die Variablen $\boldsymbol{\mu}$ Index 2 Variablen sind, während alle anderen Variablen einen kleineren Index besitzen.

Genauso wie bei der speziellen DAE (2.33), kann damit die DAE (2.35) mit jedem stabilen impliziten Mehrschrittverfahren, z.B. DASSL, integriert werden. Man muß nur im Diskretisierungalgorithmus für vergangene Werte von $\boldsymbol{\mu}$ die exakte Lösung $\boldsymbol{\mu} = \mathbf{0}$ verwenden. Mit Hilfe dieser neuen, recht einfachen Transformation, die auf dem Pantelides Algorithmus basiert, kann also jede DAE automatisiert in eine Form überführt werden, die mit vorhandenen Integratoren gelöst werden kann! Aus Effizienzgründen sollte die obige DAE mit einem Integrator eng gekoppelt werden, um die spezielle Struktur der DAE gezielt auszunutzen.

Wird die obige Transformation auf die einfache semi-explizite DAE (2.31) von Seite 37 angewandt, ergibt sich das von Gear angegebene Gleichungssystem (2.33). Zur Verdeutlichung wird noch das folgende kompliziertere Index-2 System betrachtet, welches in ähnlicher Form z.B. in der Mehrkörperdynamik bei Rad-Schiene Kontaktproblemen auftritt:

$$\dot{\mathbf{x}} = \mathbf{f}_0\left(t, \mathbf{x}, \mathbf{w}_1, \mathbf{w}_2\right) \tag{2.36a}$$

$$\mathbf{0} = \mathbf{f}_1\left(t, \mathbf{x}, \mathbf{w}_1\right) \qquad \left(\frac{\partial \mathbf{f}_1}{\partial \mathbf{w}_1} \text{ ist regulär}\right) \tag{2.36b}$$

$$\mathbf{0} = \mathbf{f}_2\left(t, \mathbf{x}, \mathbf{w}_1\right) . \tag{2.36c}$$

Dieses Gleichungssystem kann mit den meisten verfügbaren Spezialintegratoren für höhere Index DAEs nicht gelöst werden, da die über die Gleichung (2.36b) bestimmten algebraischen Variablen $\mathbf{w}_1$ in der Index-2 Gleichung (2.36c) auftreten. Prinzipiell wäre es möglich, $\mathbf{w}_1$ mittels (2.36b) als Funktion von $\mathbf{x}$ und t anzugeben und in (2.36a, 2.36c) einzusetzen. Dann könnten Standardverfahren verwendet werden. Aufgrund der angenommenen nichtlinearen Funktion (2.36b) ist das nicht möglich.

Der Algorithmus von Pantelides stellt fest, daß die Gleichungen (2.36b, 2.36c) je einmal differenziert werden müssen. Mit der neuen Transformationsvorschrift (2.35) wird das Problem in das folgende Gleichungssystem transformiert:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_0 \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \frac{\partial \mathbf{f}_1^T}{\partial \mathbf{x}} & \frac{\partial \mathbf{f}_2^T}{\partial \mathbf{x}} \\ \frac{\partial \mathbf{f}_1^T}{\partial \mathbf{w}_1} & \frac{\partial \mathbf{f}_2^T}{\partial \mathbf{w}_1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}$$

$$\mathbf{0} = \begin{bmatrix} \frac{\partial \mathbf{f}_1}{\partial \mathbf{x}} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{w}_1} \\ \frac{\partial \mathbf{f}_2}{\partial \mathbf{x}} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{w}_1} \end{bmatrix} \begin{bmatrix} \mathbf{f}_0 \\ \mathbf{z}_1 \end{bmatrix} + \begin{bmatrix} \frac{\partial \mathbf{f}_1}{\partial t} \\ \frac{\partial \mathbf{f}_2}{\partial t} \end{bmatrix}$$

$$\mathbf{0} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix} .$$

Da die Ausgangsgleichung schon in semi-expliziter Form ist, wurde $\mathbf{z}_0$ aus (2.35) gleich durch die Funktion $\mathbf{f}_0$ ersetzt. Die unbekannten, voneinander unabhängigen Variablen sind: $\dot{\mathbf{x}},\ \mathbf{w}_1,\ \mathbf{w}_2,\ \mathbf{z}_1\,(=\dot{\mathbf{w}}_1),\ \boldsymbol{\mu}_1,\ \boldsymbol{\mu}_2$.

Nach dieser Zusammenstellung der wichtigsten Lösungsverfahren für DAEs mit höherem Index stellt sich die Frage, auf welche Weise solche Systeme mit Dymola behandelt werden können. Hierzu wurde in Dymola der Pantelides-Algorithmus eingebaut, siehe [Cell92, Cell93]. Wenn beim Aufstellen der Blockdreiecksform feststellt wird, daß das Ausgangsgleichungssystem strukturell singulär ist, werden die Gleichungen mit Hilfe des Pantelides-Algorithmus abgeleitet[12]. Als Ergebnis erhält man den Störungsindex der DAE sowie alle Gleichungen, die von konsistenten Anfangswerten erfüllt werden müssen. Auf dieses Gleichungssystem kann die (allgemeine) "Dummy Derivative" Methode angewandt

[12] Bei den symbolischen Ableitungen der Gleichungen werden alle partiellen Ableitungen, die durch die Kettenregel auftreten, als Zwischengrößen eingeführt. Dadurch wird das exponentielle Anwachsen von Termen vermieden, wie man es von allgemeinen Formelmanipulationssprachen wie Mathematica oder Maple kennt.

werden. Hierzu muß der Anwender festlegen, welche Variablen Zustandsgrößen sind. Das Modell wird dann entweder in Zustandsform oder in Residuen-Darstellung (Index-1 DAE) umgeformt.

Diese Vorgehensweise soll kurz an dem Einführungsbeispiel von Seite 31, der Parallelschaltung von 2 Kapazitäten, verdeutlicht werden. Dymola stellt hier eine DAE mit Index 2 fest und differenziert die Gleichung "$u_2 = u_1$" einmal. Dies ergibt das folgende Gleichungssystem:

$$\begin{aligned} u_0 &= R\,i + u_1 && (2.37a)\\ i &= C_1\,\dot{u}_1 + C_2\,\dot{u}_2 && (2.37b)\\ u_2 &= u_1 && (2.37c)\\ \dot{u}_2 &= \dot{u}_1\ . && (2.37d) \end{aligned}$$

Der Anwender legt fest, daß u_1 Zustandsgröße sein soll. Damit sind u_1 und die Eingangsgröße u_0 bekannte Variablen. Gleichungen (2.37) sind dann 4 Gleichungen für die 4 Unbekannten $i, u_2, \dot{u}_1, \dot{u}_2$, die problemlos gelöst werden können:

$$\begin{aligned} i &:= (u_0 - u_1)/R && (2.38a)\\ \dot{u}_1 &:= i/(C_1 + C_2) && (2.38b)\\ u_2 &:= u_1 && (2.38c)\\ \dot{u}_2 &:= \dot{u}_1\ . && (2.38d) \end{aligned}$$

Die Gleichungen (2.38c,2.38d) werden später bei der Codegenerierung entfernt, wie auf Seite 25 erläutert.

In diesem Kapitel wurden nur Modelle betrachtet, bei denen der Index des Systems strukturell festliegt und sich während der Integration nicht ändert. Alle hier vorgestellten Verfahren, z.B. der Algorithmus von Pantelides oder die Spezial-Integratoren für höhere Index DAEs, arbeiten nur unter dieser Voraussetzung. Diese Voraussetzung ist aber nicht immer erfüllt. In Kapitel 3.3 wird die wichtige Klasse strukturvariabler Systeme diskutiert, bei der sich der Index während einer Integration ändern kann und es wird dort gezeigt, wie solche Systeme behandelt werden können.

2.5 Diskussion

In Abschnitt 2.1 wurden die wenigen Sprachelemente erläutert, die benötigt werden, um auf deklarative Weise objektorientiert zu modellieren. Hierbei werden nur die modellbeschreibenden *Gleichungen* erstellt und je nach Aufgabenstellung so in eine geeignete Form transformiert, daß sie mit numerischen Standardverfahren gelöst werden können. Die objektorientierte Beschreibungsform, zusammen mit dem Konzept der Across- und Through-Variablen, erlaubt es, lokal abgegrenzte Modelle zu betrachten, die normalerweise direkt mit physikalischen Teilsystemen korrespondieren. Zur *Verschaltung* von Objekten dienen die beiden einzigen in der Natur vorkommenden Erhaltungsgesetze, nämlich Gleichheit oder Null-Summe von Variablen am Verbindungspunkt. Das (Modellierungs-) Wissen über ein Fachgebiet wird nun nicht, wie sonst in Softwarepakten üblich, im Programm "hart" codiert, sondern

über wiederverwendbare Bibliotheken zur Verfügung gestellt. Die Bibliotheken werden mit denselben Sprachelementen erstellt, wie sonstige Modelle auch. Dadurch wird es möglich, multidisziplinär vereinheitlicht zu modellieren, indem Modelle aus unterschiedlichen Fachgebieten direkt miteinander verschaltet werden können.

Die Dymola Modellierungssprache enthält alle Sprachelemente, die für zeitkontinuierliche Modellierungsaufgaben notwendig sind. Wünschenswert ist noch die generische Unterstützung von Matrizen, da bei einigen Fachgebieten, wie z.B. der Mehrkörperdynamik, die Formulierung physikalischer Gesetze mühsam wird, wenn nur skalare Variablen zur Verfügung stehen. Die bisherige Entwicklung von Dymola konzentrierte sich vor allem auf die Sprache und die Transformationsalgorithmen, weniger auf die Entwicklung leistungsfähiger fachspezifischer Bibliotheken. In den nächsten Abschnitten werden neue Bibliotheken für Mehrkörpersysteme, für Antriebsstränge und für regelungstechnische Blockdiagramme beschrieben.

Ein großer Teil dieses Kapitels wurde den, vom Anwender nicht sichtbaren, Transformations-Algorithmen gewidmet. Dies ist eine sehr wichtige Komponente, da die deklarative, objektorientierte Modellierung für realistische, große Modelle unbrauchbar wäre, wenn keine Algorithmen zur Verfügung stünden, die auch umfangreiche Modellgleichungen in eine numerisch effizient auswertbare Form überführen können, z.B. eine Zustandsdarstellung. In Dymola werden effiziente Algorithmen für *unstrukturierte* differential-algebraische Gleichungen angeboten. Dabei dürfen lineare oder nichtlineare algebraische Gleichungssysteme auftreten, sowie DAEs mit höherem Index. Diese Algorithmen sind gut geeignet für die Modellbehandlung aus Fachgebieten, für die keine spezielle Formalismenstruktur bekannt ist, z.B. elektrische Schaltungen, die chemische Prozeßtechnik oder regelungstechnische Blockschaltbilder.

Im Fachgebiet Mehrkörperdynamik stehen leistungsfähige Formalismen für die Modellgenerierung zur Verfügung, wobei spezielle Strukturen berücksichtigt werden[13]. Zum Beispiel ist beim Simulationsmodell für ein Mehrkörpersystem die Systemmatrix des linearen Gleichungssystems immer symmetrisch und positiv definit. Mit dieser Eigenschaft können vor allem dünnbesetzte Gleichungssysteme effizienter gelöst werden, weil die komplette Lösungsstruktur schon bei der Code-Generierung ermittelt werden kann, da keine Pivotsuche notwendig ist. Weiterhin treten in der Mehrkörperdynamik immer 3×1, 3×3, 6×1 oder 6×6 Matrizen auf. Dann ist es sinnvoll einen Algorithmus zu verwenden, der direkt mit 3×3 bzw. 6×6 Blockmatrizen arbeitet, anstatt nur mit Skalaren.

In Abschnitt 4.4 wird dieses Problem pragmatisch gelöst, indem eine Spezialbibliothek für das Simulationsproblem entwickelt wird, in der die bekannten generischen Eigenschaften von Mehrkörpersystemen ausgenutzt werden. Dies sollte jedoch nur eine temporäre Zwischenlösung sein, da die deklarative Eigenschaft der Modellierung verloren geht (diese Bibliothek kann *nur* zum Aufstellen eines Simulationsmodells benutzt werden). Man beachte jedoch, daß es auch in der Mehrkörperdynamik Probleme gibt, bei denen keine spezielle Struktur vorliegt, z.B. wenn normalkraftabhängige Reibung auftritt. Im Gegensatz zum hier verfolgten Ansatz, können die meisten MKS-Programme solche Probleme dann nicht mehr lösen.

[13] Eine gute Zusammenstellung ist in [Schi93] zu finden.

Vor kurzem wurde von Campbell und Gear festgestellt, daß die in diesem Kapitel erläuterte Definition des differentiellen Index und des Störungsindex in einigen Fällen erweitert werden muß. Zum einen kann der differentielle Index für leicht gestörte DAEs unterschiedlich sein, und zum anderen kann der Störungsindex von der betrachteten Lösung der DAE abhängen, d.h. von Anfangsbedingungen. Dies führt dazu, daß die Gear'sche Ungleichung (diff. Index $\leq$ Störungsindex $\leq$ diff. Index+1) in bestimmten Fällen falsch ist. Campbell und Gear haben deswegen die bisherige Index-Definition so modifiziert, daß der jeweilige Index definiert wird als "das Maximum der Indizes von einer Lösungsumgebung über einer Menge von Störungen". Mit dieser Änderung wird u.a. erreicht, daß die Gear'sche Ungleichung auch für pathologische Fälle zutrifft. In den hier betrachteten Anwendungen spielt diese Erweiterung keine Rolle, da die praktische Behandlung von höheren Index-DAEs auf dem Algorithmus von Pantelides [Pant88] beruht. Dieser ist sowieso nur bei Systemen einsetzbar, bei denen der Index strukturell festliegt. Für diese Systeme sind die "neue" und die "alte" Index-Definition identisch.

Kapitel 3

Objektorientierte Modellierung ereignisabhängiger Systeme

In Abschnitt 2.1 ab Seite 17 wurden die wenigen Sprachelemente erläutert, die benötigt werden, um kontinuierliche Modelle objektorientiert zu beschreiben. Danach wurden die wesentlichen Algorithmen besprochen, um ein so definiertes Modell in eine Zustandsform überzuführen. Abgesehen von einfachen Trivialmodellen, enthalten jedoch alle Modelle technischer Systeme Unstetigkeiten irgendeiner Art. Modelle, die sowohl kontinuierliche als auch unstetige bzw. diskrete Komponenten besitzen, werden als *ereignisabhängige* Systeme bezeichnet, da nicht-kontinuierliche Elemente zu Ereignissen führen, an denen sich Gleichungen unstetig ändern. In diesem Kapitel werden weitere (Dymola-) Sprachelemente eingeführt, um unstetige Modelle objektorientiert beschreiben zu können.

Dieses Kapitel, und insbesondere die hier vorgestellten neuen Ideen, beruhen auf der kürzlich veröffentlichten Arbeit von Elmqvist, Cellier und Otter [Elmq93b]. Die wesentliche Grundidee der recht revolutionären neuen Sprachelemente stammt von Elmqvist [Elmq92b]. Teile davon hatte er schon für das Prozeßleitsystem SattLine [Elmq92] entwickelt.

3.1 Unstetige Modellgleichungen

Die prinzipiellen Probleme, welche bei Unstetigkeiten (im Funktionswert oder in einer höheren Ableitung einer Funktion) auftreten, werden zuerst an Hand des einfachen Begrenzers von Bild 3.1 diskutiert. In einer Simulations-Sprache wie ACSL [Mitc91] könnte

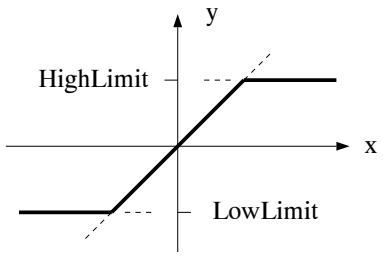


Bild 3.1: Begrenzer-Funktion.

ein Begrenzer auf die folgende Weise als (wiederverwendbares) Makro definiert werden:

```
macro Limiter (y, x, LowLimit, HighLimit)
    procedural (y, x)
        if     (x .le. LowLimit)  then
            y = LowLimit
        else if(x .ge. HighLimit) then
            y = HighLimit
        else
            y = x
        end if
    end
macro end
```

Der Begrenzer wird mittels einer Verzweigungsanweisung definiert. Leider wird eine solche naheliegende und einfache Beschreibungsform bei der Integration zu einem unbefriedigenden numerischen Verhalten führen. Der Grund liegt darin, daß die Schrittweitenkontrolle heutiger numerischer Integrationsverfahren auf der Annahme basieren, daß die Lösung $k+1$-mal stetig differenzierbar ist, wenn das Integrationsverfahren von der Ordnung k ist. An einer Unstetigkeitsstelle ist diese Voraussetzung nicht erfüllt, so daß die Fehlerkontrolle und damit das Ergebnis zweifelhaft ist. Darüberhinaus führt eine Unstetigkeitsstelle oft zu sehr kleinen Schrittweiten, da der Integrator eine sehr starke Änderung der Lösungskurve feststellt und deswegen die Schrittweite drastisch reduziert.

Eine "saubere" Lösung dieses numerischen Problems ist möglich und wurde schon 1979 von Cellier angegeben [Cell79]: Die Begrenzerfunktion besteht ja aus drei vollkommen stetigen und differenzierbaren Teilen. Man muß nun einfach verhindern, daß der Integrator über die nicht-differenzierbaren oder unstetigen Stellen "hinwegintegriert". Hierzu werden die drei Bereiche des Begrenzers stetig erweitert, so daß diese überlappend sind, wie in Bild 3.1 angedeutet. Während der numerischen Integration ist nur einer der drei Bereiche jeweils aktiv. Mittels einer Indikatorfunktion (auch Schaltfunktion, Monitorfunktion, zero-crossing function oder daemon genannt) wird der Umschaltpunkt von einem Bereich zum nächsten signalisiert. Der genaue Umschaltpunkt wird mit Hilfe einer Iterationsprozedur ermittelt. Dann wird die Integration angehalten, es wird auf den neuen, aktiven Bereich umgeschaltet und die Integration wird neu gestartet.

Die Indikatorfunktion wird so gewählt, daß ein Nulldurchgang der Funktion den Umschaltpunkt charakterisiert. Die effiziente Ermittlung des genauen Null-Durchgangs erfolgt üblicherweise mittels eines kombinierten Verfahrens, bestehend aus Intervallschachtelung (für die Robustheit) und Sekanten- oder Newton-Verfahren (für schnelle Konvergenz). Details sind z.B. in [Cell79, Eich92] zu finden. Einige *verfügbare* Integratoren besitzen diese sogenannte "Root-Finding" Eigenschaft, z.B. LSODAR von Petzold/Hindmarsh [Hind83] für explizite Differentialgleichungen oder DASSLRT von Petzold [Petz82, Bren89] für Index-1 DAEs.

In Simulationssprachen wie ACSL [Mitc91] wird das Anhalten der Integration an einem Umschaltpunkt als *Ereignis* (event) bezeichnet. Durch die Indikatorfunktion wird das Ereignis definiert. Beim Eintreten eines Ereignisses (= Nulldurchgang der Indikatorfunktion) wird die Integration angehalten und ein vorher definiertes Code-Stück, die *discrete section*, wird ausgeführt. Hier wird z.B. einfach der neue Bereich des Begrenzers festgelegt. Danach wird die Integration neu gestartet (= event restart).

Ein Ereignis, das direkt oder indirekt von Zustandsvariablen abhängt, wird als *Zustands*-Ereignis (state event) bezeichnet. Ein Ereignis, das *nur* von der Zeit abhängt, wird als *Zeit-*

Ereignis (time event) bezeichnet. Zeitereignisse können wesentlich effizienter abgehandelt werden als Zustandsereignisse, da der Zeitpunkt des Ereignisses im voraus bekannt ist. Man muß deswegen nur die Schrittweite des Integrators so anpassen, daß der gewünschte Zeitpunkt direkt getroffen wird. Dazu ist keine Iteration notwendig. Diese Möglichkeit ist bei allen Integratoren mit variabler Schrittweite gegeben. In Simulationssprachen wie ACSL gibt es Sprachelemente, mit denen Zustandsereignisse und Zeitereignisse definiert werden können.

In der Simulationsumgebung DSSIM [Otte91, Gaus91, Otte93b] wurde von Otter noch ein dritter Typ von Ereignissen eingeführt und in Integratoren wie DASSLRT eingebaut: das *Schritt*-Ereignis (step event). Bei einem Schrittereignis wird die Ereignisbedingung immer nach einem *abgeschlossenen* Integrationsschritt geprüft. Wenn die Ereignisbedingung eingetreten ist, wird die Integration sofort angehalten. Schrittereignisse können effizienter behandelt werden als Zustandsereignisse, da keine Iteration zur Bestimmung des genauen Ereignis-Zeitpunktes stattfindet. Dafür wird der Zeitpunkt an dem das Ereignis eintritt nicht exakt ermittelt. Schrittereignisse werden z.B. eingesetzt, um effizient zwischen verschiedenen Mengen von Zustandsgrößen zu schalten, wenn eine getroffene Wahl von Zustandsgrößen nicht über den gesamten Integrationsbereich beibehalten werden kann. Dies kann z.B. bei der in Abschnitt 2.4 erläuterten Dummy-Derivative Methode der Fall sein oder bei der Drei-Parametrisierung von Rotationsmatrizen in der Mehrkörperdynamik[1].

Nach diesen Vorbemerkungen kann gezeigt werden, wie der Begrenzer in Bild 3.1 numerisch zuverlässig realisiert werden kann. Dies soll mit der Notation der Simulationssprache ACSL geschehen. Mit der aktuellen ACSL Version 10 ist es nicht möglich, den Begrenzer als (wieder-verwendbares) Makro zu formulieren, deshalb wird dafür das folgende komplette ACSL-Programm benutzt:

```
program LimiterTest
   constant HighLimit = 1.0, LowLimit = -1.0
   logical  High, Low

   initial
      x    = 2 * sin(t)
      High = x .ge. HighLimit
      Low  = x .le. LowLimit
   end ! of initial

   dynamic
      derivative
         x = 2 * sin(t)

         if (High) then
            y = HighLimit
         elseif (Low) then
            y = LowLimit
         else
            y = x
         endif

         schedule HighEvent .xz. x - HighLimit
         schedule LowEvent  .xz. x - LowLimit
```

[1] Bei Rotationsmatrizen kann z.B. zwischen Eulerwinkeln und Cardanwinkeln geschaltet werden, da die Singularitäten der jeweiligen Drei-Parametrisierungen an unterschiedlichen Stellungen liegen. Details hierfür siehe z.B. [Schi86].

```
        end ! of derivative

        discrete HighEvent
            High = .not. High
        end ! of discrete HighEvent

        discrete LowEvent
            Low  = .not. Low
        end ! of discrete LowEvent

        termt t .ge. 10.0
    end ! of dynamic
end ! of program
```

Das obige Programm löst die numerischen Probleme, die aufgrund der Nicht-Differenzierbarkeit des Begrenzers auftreten. *High* und *Low* sind logische Variable, die ihren Wert nur an Ereigniszeitpunkten ändern. Sie werden in der **initial** section initialisiert, die einmal am Start der Integration durchlaufen wird. Zwei Indikatorfunktionen in Form von **schedule** Anweisungen überwachen die Bereichsgrenzen des Begrenzers. Wenn eine der beiden Funktionen das Vorzeichen wechselt, wird der Zeitpunkt des Nulldurchgangs lokalisiert, die Integration wird angehalten und die entsprechende **discrete** section wird ausgeführt. Hier wird in den nächsten Bereich umgeschaltet und die Integration wird neu gestartet.

Eine Schwäche der obigen Notation ist die vollkommene Mißachtung jeglichen objektorientierten Prinzips. Ein und dasselbe physikalische Objekt, d.h. der Begrenzer, wird an drei unterschiedlichen Stellen im Simulationsprogramm definiert. Für das Beispiel ist das noch akzeptabel, denn das Programm ist einfach und verständlich. Wenn ein Modell jedoch eine Vielzahl von Unstetigkeiten enthält, wie es in jedem realistischeren technischen Modell der Fall ist, dann wird ein solches Simulationsmodell schnell unübersichtlich und fehleranfällig bei Programmänderungen.

Das obige Beispiel enthält zwei versteckte, diffizile Probleme:

1. Was geschieht, wenn eine Indikatorfunktion zu Beginn der Integration oder an einem Ereignispunkt *exakt* Null ist? Integratoren mit einer Root-Finding Eigenschaft gehen davon aus, daß ein Ereignis *nur* eintritt, wenn eine Indikatorfunktion, ausgehend von einem Nicht-Null Wert, entweder exakt Null wird oder das Vorzeichen wechselt. Sei z_{i-1} der Wert einer Indikatorfunktion z im letzten Schritt und z_i der Wert am Ereignispunkt. Dann kann diese Eigenschaft mathematisch durch folgende Beziehung ausgedrückt werden:

   $$z_{i-1} \cdot z_i \leq 0 \textbf{ und } z_{i-1} \neq 0$$

   Wenn eine Indikatorfunktion zu Beginn der Integration exakt Null ist, führt deshalb z.B. DASSLRT einen kleinen Schritt aus, um ein Vorzeichen der Funktion festlegen zu können. Wenn die Indikatorfunktion nach dem kleinen Schritt immer noch Null ist, bricht DASSLRT mit einer Fehlermeldung ab. Ein solches Verhalten der Indikatorfunktion kann natürlich auftreten, ein Abbruch ist jedoch unerwünscht.

   Abgesehen davon, gibt es im obigen ACSL-Programm (und in jeder anderen Simulationsumgebung) noch zwei problematische Situationen: Angenommen, der Anfangswert von x ist gleich *HighLimit*. In der **initial** section wird damit *High* auf *.True.* gesetzt. Wenn x jetzt in den nächsten Schritten kleiner als *HighLimit* wird, tritt aufgrund der oben erläuterten Eigenschaft *kein* Ereignis auf und der Bereich des Begrenzers ist falsch gesetzt. Dasselbe Verhalten kann beim Neustart nach einem Ereignis

auftreten, wenn die Indikatorfunktion exakt Null war. Im obigen Programm wird ja immer sofort in den anderen Bereich gewechselt ($High = .not.\ High$). Es ist jedoch möglich, daß x nach dem Neustart wieder in den Bereich zurückgeht, in dem es vor dem Ereignis war. Auch dann tritt kein neues Ereignis mehr auf und eine der logischen Variablen $High$ oder Low ist wiederum falsch gesetzt.

2. Was geschieht, wenn mehrere Ereignisse zufällig zum selben Zeitpunkt auftreten, also gleichzeitig unterschiedliche **discrete** sections abzuarbeiten sind? Solch ein kritischer Fall kann z.B. auftreten, wenn im obigen ACSL-Programm noch eine **schedule** Anweisung mit einer zugehörigen **discrete** section (diese habe den Namen $xEvent$) aufgenommen wird, in der sich x unstetig verändert. Im obigen Beispiel würden dann Probleme auftreten, da $HighLimit$ den Wert von $High$ unabhängig vom aktuellen Wert von x (der ja wiederum von $xEvent$ modifiziert werden kann) ändert. Dieses Problem kann aber vermieden werden, wenn $HighEvent$ folgendermaßen modifiziert wird (der Fall, daß die Indikatorfunktion exakt Null ist, sei hier ausgeschlossen):

   discrete $HighEvent$
   $High = x$ **.ge.** $HighLimit$
   $Low = x$ **.le.** $LowLimit$
   end ! *of discrete* $HighEvent$

   Jetzt tritt allerdings das neue Problem auf, daß die *Reihenfolge* der Abarbeitung der **discrete** sections entscheidend ist. Wenn z.B. zuerst die $HighEvent$ section abgearbeitet wird, wird $High$ entsprechend dem aktuellen Wert von x gesetzt. Wenn dann aber die $xEvent$ section abgearbeitet wird, wird x unstetig modifiziert und der Wert von $High$ muß nicht mehr zu dem aktuellen x-Wert kompatibel sein. Aus diesem Grund muß hier auf jeden Fall immer zuerst die $xEvent$ und dann die $HighEvent$ section ausgeführt werden. In ACSL werden die **discrete** sections, die an demselben Ereignispunkt aktiv sind, in der Reihenfolge abgearbeitet, in der sie im Programmtext stehen. Deshalb muß die $xEvent$ section vor der $HighEvent$ section aufgeführt werden.

   Man sieht auch hier wieder: von Objektorientiertheit keine Spur. Nur weil von der Aufgabenstellung her zusätzlich ein anderes Ereignis zum gleichen Zeitpunkt auftreten kann, müssen die Gleichungen des Begrenzers umformuliert werden. Dabei muß dann auch noch darauf geachtet werden, in welcher Reihenfolge die **discrete** sections angegeben werden. Angenommen es gibt n verschiedene **discrete** sections und potentiell können alle zugehörigen Ereignisse zu demselben Zeitpunkt auftreten, dann müssen bis zu 2^n Varianten überprüft werden, um sicherzustellen, daß die Reihenfolge der **discrete** sections korrekt ist!

Die obige Diskussion des Beispiels zeigt, daß es für den "Normal-Anwender" fast nicht möglich ist, ein in allen Situationen korrektes Modell einer so einfachen Komponente, wie es ein Begrenzer ist, zu realisieren, wenn dafür nur eine "nackte" Ereignisbehandlung zur Verfügung steht (was heutzutage der Normalfall ist).

Mit Dymola können die obigen Probleme verblüffend einfach und elegant durch die Verwendung von *unstetigen Gleichungen* (discontinuous equations) gelöst werden:

```
< expression > = if      < condition 1 > then < expression 1 >
                 else if < condition 2 > then < expression 2 >
                 ...
                 else                        < expression n >
```

Zur Beschreibung von Begrenzer-Objekten kann damit eine Klasse *Limiter* definiert werden:

```
model class Limiter
    terminal x, y
    parameter LowLimit, HighLimit
        y = if      x > HighLimit then HighLimit
            else if x < LowLimit  then LowLimit
            else x
end
```

Die Beschreibung mit "unstetigen Gleichungen" sieht ähnlich aus wie der erste Versuch auf Seite 46 mit einem ACSL-Makro. Die Syntax ist zwar ähnlich, die Semantik ist jedoch verschieden. Bei der Code-Erzeugung werden die Bedingungen des **if**-Ausdrucks in Indikatorfunktionen umgewandelt. Ein Ereignis tritt ein, wenn die aktive Bedingung falsch wird. Dann wird der neue aktive Zweig des **if**-Ausdrucks festgelegt und die Integration wird neu gestartet. D.h. der **if**-Ausdruck wird numerisch vollkommen korrekt abgehandelt[2].

Aus Anwendersicht gibt es jetzt keinerlei Probleme mehr: Der **if**-Ausdruck legt eindeutig fest, wann welcher Zweig auszuführen ist; wie dies erreicht wird ist Sache des Compilers. Auch wenn unterschiedliche unstetige Gleichungen zum selben Zeitpunkt zu einem Ereignis führen, gibt es keine Schwierigkeiten, da *unstetige Gleichungen wie alle anderen Gleichungen behandelt werden.* Insbesondere werden sie mit den anderen Gleichungen zusammen *sortiert*, d.h. auf Blockdreiecksform transformiert. Dadurch ergibt sich *automatisch* die "richtige" Ordnung der "discrete sections", also der **if**-Ausdrücke.

Die Code-Erzeugung für einen **if**-Ausdruck ist relativ einfach. Der **if**-Ausdruck wird in ein entsprechendes Konstrukt der Zielsprache übersetzt. Dabei wird jede Bedingung durch eine logische Variable ersetzt und die Relationen in den Bedingungen werden in Indikatorfunktionen umgewandelt. Zum Beispiel ergibt die Relation $x > HighLimit$ die Indikatorfunktion $z = x - HighLimit$. Die logischen Variablen bleiben während einer Integration konstant und werden nur an einem Ereignispunkt modifiziert.

Die einzige wesentliche, offene Frage ist: Wie werden Indikatorfunktionen behandelt die exakt Null sind? Dieses Problem kann folgendermaßen gelöst werden: Für jede Indikatorfunktion z werden in Wirklichkeit zwei Indikatorfunktionen zp, zn eingeführt. Üblicherweise ist $zp = z,\ zn = z$, d.h. mit Ausnahme der Verdoppelung der Zahl an Indikatorfunktionen wird nichts geändert. Wenn jedoch die ursprüngliche Indikatorfunktion exakt Null ist oder Null wird ($z = 0$), dann werden die beiden neu eingeführten Indikatorfunktionen leicht modifiziert: $zp = z + eps, zn = z - eps$. Damit wird ein schmales Intervall um Null gelegt, wobei es wichtig ist, eps auf einen Wert zu setzen, der unabhängig von spezifischen Eigenschaften eines Modells ist. Dies ist möglich, wenn eps auf eine Zahl in der Nähe der kleinsten Maschinenzahl gesetzt wird, z.B. $eps = 10^{-38}$. In diesem Fall wird der "Root-Finder" des Integrators den Zeitpunkt zum Verlassen dieses Intervalls so wählen, daß das *Zeit*-Intervall

[2] Optional kann ein **if**-Ausdruck auch direkt in eine Verzweigungsanweisung der Zielsprache übersetzt werden, ohne daß dann eine explizite Ereignisbehandlung stattfindet.

zwischen dem letzten Ereignis und dem neuen Ereignis in derselben Größenordnung liegt wie das *Zeit*-Intervall, innerhalb dessen Grenzen sonst ein Ereignis ermittelt wird. Solange sich z in dem kleinen Intervall um Null befindet, wird es als Null angesehen und es wird der entsprechende Zweig des **if**-Ausdrucks gewählt. Wenn z das Band verläßt, tritt ein Ereignis ein und zp und zn werden wieder zurück auf z gesetzt. Jetzt ist z entweder positiv oder negativ und der korrekte Zweig des **if**-Ausdrucks ist festgelegt.

Mit dieser Vorgehensweise treten die oben für ein ACSL-Programm diskutierten Probleme nicht mehr auf. Zum Beispiel sei $x = HighLimit$, wenn die Integration gestartet wird. Dann ist die Indikatorfunktion $z = x - HighLimit$ exakt Null. Da $z = 0$ ist, ist der **else**-Zweig beim Integrationsstart aktiv und es wird ein kleines Intervall um $z = 0$ gelegt. Solange sich z in diesem Intervall befindet, bleibt der **else**-Zweig aktiv. Wenn das Intervall verlassen wird, tritt ein Ereignis auf und der neue Wert von z legt eindeutig fest, welcher Zweig des **if**-Ausdrucks jetzt zu wählen ist, also aktiv wird.

Wenn eine Indikatorfunktion nur von der Zeit abhängig ist und die Indikatorgleichung explizit nach der Zeit aufgelöst werden kann, wird kein Code für Zustandsereignisse erzeugt, sondern für die effizienter behandelbaren Zeitereignisse. Es sei z.B. die Bedingung $Time > Tevent$ gegeben, wobei $Tevent$ eine Konstante ist. Dann wird zuerst die Indikatorfunktion $z = Time - Tevent$ erzeugt. Da die Funktion $z = 0$ explizit nach der Zeit aufgelöst werden kann ($Time = Tevent$), kann Code für ein Zeitereignis erzeugt werden.

3.2 Instant-Modellgleichungen

Mit den "unstetigen Gleichungen" können noch nicht alle möglichen unstetigen Effekte beschrieben werden. Es werden auch Mechanismen benötigt, um *plötzliche* Modelländerungen zu erfassen, die mehr bewirken als nur eine Bereichsumschaltung für einen **if**-Ausdruck. Hierfür wurden in [Elmq93b] die *Instant-Gleichungen* (instantaneous equations) eingeführt, die folgende allgemeine Struktur besitzen:

when < *condition* > **then**
< *equations* >
endwhen

Die Gleichungen < *equations* > innerhalb des **when**-Rumpfs werden nur an Zeitpunkten ausgeführt, an denen die Bedingung < *condition* > wahr *wird*. Wie bei **if**-Ausdrücken tritt bei einem solchen Zeitpunkt ein Ereignis ein, d.h. der Compiler wandelt die < *condition* > in geeignete Indikatorfunktionen um, die dann ein Ereignis auslösen, wenn die Bedingung wahr *wird*. Dann wird der **when**-Rumpf ausgeführt und die Integration wird neu gestartet. Als Beispiel ist in Bild 3.2 eine Hysterese-Funktion dargestellt, die mit einer Instant-Gleichung definiert wird:

Der Ausgang y ändert seinen Wert nur, wenn x größer als H oder kleiner als $-H$ wird. Man beachte, daß der **if**-Ausdruck innerhalb eines **when**-Rumpfs keine zusätzlichen Indikatorfunktionen erzeugt, da ein **when**-Rumpf nur an Ereignispunkten ausgewertet wird.

Auch *Differenzengleichungen*, wie sie zur Formulierung von digitalen Regelgesetzen dienen, können mittels einer Instant-Gleichung definiert werden. Hierbei charakterisiert der Opera-

local $Start = True$
when $x > H$ **or** $x < -H$ **or** $Start$ **then**
y = **if** $x > H$ **then** 1 **else** -1
endwhen
new(Start) = False

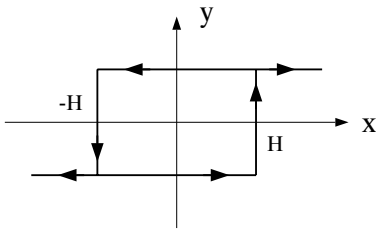


Bild 3.2: Hysterese-Funktion.

tor **new**(x) den *nächsten* Wert der diskreten Zustandsvariablen[3] x, d.h. den Wert den x *nach* Abarbeitung des aktuellen Ereignisses haben wird. Ein Beispiel ist:

when $Time >= NextTime$ **then**
new$(NextTime) = NextTime + SamplingRate$
new$(x) + a * x \quad = b * u$
endwhen

Auch ein **when**-Rumpf wird nicht durch Zuweisungen, sondern durch Gleichungen beschrieben. Wenn z.B. feststellt wird, daß Gleichung "**new**$(x)+a*x = b*u$" nach **new**(x) aufgelöst werden muß, dann wird die Gleichung entsprechend symbolisch umgeformt.

Es ist auch möglich, daß sich aufgrund einer Bedingung der Wert einer kontinuierlichen Variablen plötzlich ändert. Der Stoß im Modell eines springenden Balls kann z.B. folgendermaßen definiert werden:

when $Height <= 0.0$ **then**
init$(Velocity) = -c * Velocity$
endwhen

Weiter ist wichtig, daß Ereignisse innerhalb eines Objekts, und auch zwischen unterschiedlichen Objekten, *synchronisiert* werden können. Zum Beispiel soll ein Ereignis in einem Objekt ein anderes Ereignis in einem zweiten Objekt zur Folge haben. Diese Synchronisation von Ereignissen kann sehr einfach und elegant durch die Verwendung *Boole'scher Variablen* und *Boole'scher Gleichungen* erreicht werden. Eine Boole'sche Variable kann nur die Werte *False* oder *True* annehmen und wird entweder explizit auf einen dieser Werte gesetzt oder implizit durch eine Boole'sche Gleichung festgelegt. Ein Ereignis tritt ein, wenn eine Boole'sche Variable ihren Wert von $False$ nach $True$ oder von $True$ nach $False$ ändert, weil eine solche Änderung eine Unstetigkeit im Modell zur Folge haben kann. Zum Beispiel wird in der folgenden Boole'schen Gleichung

$$a = b \textbf{ and } c \textbf{ or } v > 0$$

die Boole'sche Variable a in Abhängigkeit anderer Boole'scher Variablen b und c sowie einer Relation $(v > 0)$ gesetzt. Anders ausgedrückt, ein durch a charakterisiertes Ereignis wird ausgelöst, wenn die spezifischen Bedingungen an die "Ereignisse" b und c sowie die Relation $v > 0$ erfüllt sind. Eine Synchronisation von Ereignissen *zwischen* Objekten wird einfach dadurch erreicht, daß entsprechende Boole'sche Variablen als **terminal** Variablen deklariert werden, d.h. sie besitzen in den zusammengeschalteten Objekten denselben Wert. Boole'sche Variable können dabei nur als *Across*-Variablen benutzt werden, da nur das Gleichsetzen Boole'scher Variablen an einer Verbindungsstelle Sinn macht.

[3]Eine diskrete Variable ändert ihren Wert *nur* an Ereignispunkten.

Wie **if**-Gleichungen, werden auch **when**-Gleichungen und Boole'sche Gleichungen *genauso wie alle anderen Gleichungen behandelt.* Insbesondere werden sie auch mit den anderen Gleichungen zusammen sortiert, d.h. auf Blockdreiecksform transformiert. Dadurch ergibt sich *automatisch* die "richtige" Ordnung des Codes an Ereignispunkten. Es ist ja auch hier möglich, daß zwei unterschiedliche **when**-Gleichungen zum selben Zeitpunkt ein Ereignis auslösen. In einem solchen Fall kann die Reihenfolge, in der die jeweiligen **when**-Rümpfe ausgewertet werden, wichtig sein. Durch das Sortieren wird garantiert, daß die korrekte Reihenfolge benutzt wird oder zumindest macht eine Fehlermeldung auf Probleme aufmerksam, wenn z.B. unbeabsichtigt eine algebraische Schleife zwischen **when**-Gleichungen auftritt.

Bei der Codeerzeugung werden (wie bei den **if**-Ausdrücken) für jede Relation immer *zwei* Indikatorfunktionen erzeugt, um identisch verschwindende Indikatorfunktionen korrekt abhandeln zu können. Da der **new**-Operator auch auf Boole'sche Variablen angewendet werden kann, ist es möglich, daß zu einem Ereignispunkt eine Boole'sche Variable einen *neuen* Wert *nach* dem aktuellen Ereignis erhält. Dies löst dann sofort ein neues Ereignis aus, ohne daß die Integration neu gestartet wurde. Im Code wird dies dadurch realisiert, daß die Modellgleichungen zu einem Ereignispunkt iterativ solange durchlaufen werden, bis sich keine Boole'sche Variable mehr ändert. Dieses Verhalten tritt vor allem bei den, im nächsten Abschnitt besprochenen, strukturvariablen Systemen auf, und hier insbesondere bei Coulomb'scher Reibung (siehe Kapitel 4.5 ab Seite 114).

Auf den ersten Blick scheinen **when**-Gleichungen große Ähnlichkeiten mit den **discrete**-sections in CSSL-Simulationssprachen, wie ACSL, zu haben. Ein entscheidender Unterschied besteht jedoch darin, daß an einem Ereignispunkt das gesamte Modell, d.h. sowohl die "kontinuierlichen" als auch die "diskreten" Teile, in der sortierten Reihenfolge abgearbeitet werden und nicht ausschließlich nur der Code der aktiven **discrete** sections. Dies erfüllt die Forderung, daß ein korrektes Modell auch dann gewährleistet werden soll, wenn mehrere unterschiedliche Ereignisse zufällig (oder beabsichtigt) zum selben Zeitpunkt auftreten. Ein **when**-Rumpf A kann eine kontinuierliche Variable x verändern; ein nachfolgender "kontinuierlicher" Code-Teil kann aus x eine Variable y berechnen, die wiederum in einem weiteren **when**-Rumpf B zur Berechnung einer Variablen z benutzt wird. Werden die beiden **when**-Gleichungen A und B am selben Ereignispunkt aktiv, wäre das Ergebnis falsch, wenn nur die beiden **when**-Rümpfe ausgeführt werden würden!

Zusammenfassend kann festgehalten werden, daß in Dymola kein Sprachelement zur (expliziten) Definition von Ereignissen verwendet wird, im Gegensatz zu sonstigen Modellierungssystemen, wie z.B. in ACSL [Mitc91], Omola [Ande92] oder MAST [Mant92]. Auch bei der für 1997 geplanten Erweiterung des VHDL-Standards, gehen vorläufige Sprachdefinitionen von dem traditionellen Ereignismodell aus, siehe z.B. [Vach93]. Stattdessen werden *höhere Sprachelemente* für die Definition unstetiger Gleichungen und Instant-Gleichungen zur Verfügung gestellt, die durch Boole'sche Variablen und Boole'sche Ausdrücke kontrolliert werden. Der *Compiler* bildet diese Sprachelemente auf geeignete Zeit- und Zustands-Ereignisse in der Zielsprache ab. Ereignisse sind dann nur "low level" Elemente zur Steuerung von Integratoren, werden aber bei der Modelldefinition nicht benutzt. Dies macht die Modellierung ereignisabhängiger Systeme wesentlich komfortabler und sicherer.

3.3 Strukturvariable Modelle

Die in den beiden letzten Abschnitten eingeführten Sprachelemente für ereignisabhängige Modelle werden jetzt dazu verwendet, um *allgemeine*, strukturvariable Modelle – eine wichtige und nicht-triviale Problemklasse ereignisabhängiger Systeme – zu beschreiben. *Spezielle* Arten strukturvariabler Systeme in der Mechanik, insbesondere Systeme mit Reibung oder Stößen, werden seit einiger Zeit untersucht [Loet81, Haug86, Pfei88, Gloc93, Gloc93a]. Für bestimmte Teilprobleme gibt es Demonstrationsprogramme an den Hochschulen, z.B. [Gloc93a]. Die im folgenden erläuterte Methodik kann in allen Fachgebieten, nicht nur in der Mechanik, eingesetzt werden. Weiterhin ist sie in einem allgemeinen Modellierungssystem wie Dymola direkt praktisch anwendbar.

Der ideale elektrische Schalter

Ein Ereignis kann so drastische Auswirkungen haben, daß damit eine Strukturänderung des Modells bewirkt wird. Dies soll an Hand eines *elektrischen Schalters* verdeutlicht werden.

Ein elektrischer Schalter hat dieselben Verbindungseigenschaften wie ein Widerstand oder eine Kapazität und kann deswegen auch von der, in Kapitel 2.1 auf Seite 18, eingeführten Klasse *TwoPin* durch Vererbung abgeleitet werden. Er wird jedoch durch ein ziemlich eigenartiges physikalisches Gesetz beschrieben: Ein idealer Schalter hat zwei Stellungen. Wenn der Schalter *offen* ist, fließt durch das Schalterelement *kein Strom* i. Wenn der Schalter *geschlossen* ist, ist kein Spannungsabfall u vorhanden. Das heißt, in einer Stellung gibt es eine Gleichung zur Bestimmung des Stroms, aber keine Gleichung zur Ermittlung des Spannungsabfalls, während es in der zweiten Stellung eine Gleichung zur Bestimmung des Spannungsabfalls, aber keine Gleichung zur Ermittlung des Stroms gibt.

Man könnte diese Art von Problemen dadurch lösen, daß für *jede* Stellung des Schalters ein (Gesamt-) Modell erstellt wird, und daß von einem Modell zum anderen umgeschaltet wird, wenn sich die Schalterstellung ändert. Diese Art der Lösung ist jedoch im allgemeinen nicht praktikabel: Angenommen ein elektrischer Schaltkreis habe 10 Schalter (z.B. ideale Dioden), dann sind alle Kombinationen von Schaltstellungen möglich und es wären $2^{10} = 1024$ unterschiedliche Modelle notwendig.

Ein elektrischer Schalter wird als *TwoPin*-Element angesehen, das die zusätzliche *Boole'sche* Terminal-Variable *Open* besitzt, um die Schalterstellung zu kennzeichnen. Hierbei soll "$Open = True$" den offenen und "$Open = False$" den geschlossenen Schalter kennzeichnen. Dann kann ein Schalter mittels einer einzigen Gleichung beschrieben werden, die den Strom i, den Spannungsabfall u und die Schalterstellung *Open* miteinander verknüpft:

```
model class (TwoPin) Switch
    terminal Open
    0 = if Open then i else u
end
```

In **if**-Ausdrücken kann nach einer unbekannten Variablen aufgelöst werden, wenn diese in allen Zweigen des jeweiligen Ausdrucks linear auftritt und nicht in den Verzweigungsbedingungen vorkommt. Das Resultat einer solchen Transformation ist jedoch nicht besonders

übersichtlich. Aus Gründen der besseren Verständlichkeit wird deswegen in den folgenden Beispielen eine alternative Formulierung benutzt, in der die Boole'sche Variable *Open* durch eine diskrete Variable[4] *OpenSw* ersetzt wird, die entweder 0 oder 1 sein kann:

model class $(TwoPin)$ $Switch$
  terminal $OpenSw$
  $0 = OpenSw * i + (1 - OpenSw) * u$
end

Als Beispiel wird der elektrische Schalter im Schaltkreis von Bild 3.3 betrachtet.

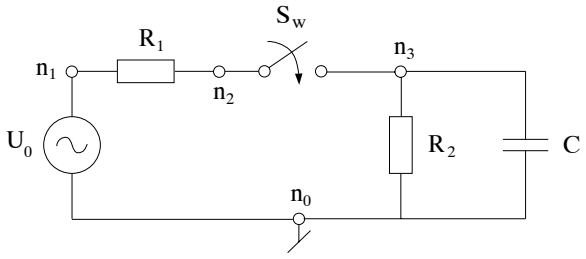


Bild 3.3: Schaltkreis mit Schalter.

Der Schalter soll zu bestimmten Zeitpunkten geöffnet und geschlossen werden. Ein Modell für dieses System sieht folgendermaßen aus:

model *circuit*
  submodel $(Vsource)$ $U0$
  submodel $(Resistor)$ $R1(R{=}20),\ R2(R{=}500)$
  submodel $(Capacitor)$ $C(C{=}1.0E\text{-}6)$
  submodel $(Switch)$ Sw
  submodel $(Ground)$ g

  input u
  output y
  node $n0,\ n1,\ n2,\ n3$

  connect $U0$ **from** $n1$ **to** $n0$, $R1$ **from** $n1$ **to** $n2$,
    Sw **from** $n2$ **to** $n3$, $R2$ **from** $n3$ **to** $n0$,
    C **from** $n3$ **to** $n0$, g **at** $n0$

  $Sw.OpenSw =$ **if** $Time < 0.0120$ **then** 1 **else**
    if $Time < 0.0144$ **then** 0 **else**
    if $Time < 0.0180$ **then** 1 **else**
    if $Time < 0.0228$ **then** 0 **else** 1

  $U0.V0 = u$
  $y \quad = C.u$
end

Daraus werden die folgenden, *sortierten Gleichungen* erzeugt, die die Blockdreiecksform des Schaltkreises darstellen[5]:

circuit.	[Sw.OpenSw]	= if Time < 0.0120 then 1 else ...
g.	[g.V]	= 0
U0.	u	= [R1.Va] − g.V

[4] Zur Erinnerung: Eine diskrete Variable ändert ihren Wert nur zu Ereignispunkten.

[5] Im nächsten Transformations-Schritt werden die Gleichungen nach den Variablen aufgelöst, die durch eckige Klammern in den sortierten Gleichungen gekennzeichnet sind.

```
  C.          C.u          = [Va] - g.V

| R1.         R1.R*R1.i    = [R1.u]
|             R1.u         = R1.Va - [Sw.Va]
| Sw.         [Sw.u]       = Sw.Va - C.Va
|             0            = Sw.OpenSw*[R1.i] + (1 - Sw.OpenSw)*Sw.u

  circuit.    0            = R1.i + [U0.i]
  R2.         [R2.u]       = C.Va - g.V
              R2.R*[R2.i]  = R2.u
  circuit.    R1.i         = [C.i] + R2.i
  C.          C.C*[C.deru] = C.i
  circuit.    [y]          = C.u
```

Die zu einem algebraischen Gleichungssystem gehörenden Gleichungen werden durch einen vertikalen Strich "|" markiert. Man sieht, daß sich die Schaltergleichungen innerhalb des linearen Gleichungssystems befinden. An dieser Stelle ist es vollkommen gleichgültig, daß die Schaltergleichung eine ungewöhnliche Form hat. Es liegt nur ein lineares Gleichungssystem vor, das z.B. symbolisch aufgelöst werden kann:

```
Solved System of Equations
        Q105  = Sw.OpenSw - 1
        Q106  = Sw.OpenSw + R1.R*Q105
        Q107  = R1.R*Q105
        Q108  = R1.R*Q105
        R1.u  = (Q107*R1.Va - Q108*C.Va)/Q106
        Q109  = R1.R*Q105
        Sw.Va = (Sw.OpenSw*R1.Va + Q109*C.Va)/Q106
        Sw.u  = (Sw.OpenSw*R1.Va - Sw.OpenSw*C.Va)/Q106
        R1.i  = (Q105*R1.Va - Q105*C.Va)/Q106
End of system of simultaneous equations
```

Es ist leicht zu überprüfen, daß die Determinante Q106 in beiden Schalterstellungen ungleich Null ist. Das obige Gleichungssystem ist also regulär und es gibt keinerlei Probleme mit diesem Zustandsraummodell. Ein Simulationsergebnis dieses Beispiels ist in Bild 3.4 zu sehen. Im linken Teil ist der zeitliche Verlauf der Spannung U0.V0 der Spannungsquelle (= Kurve 1) und der Spannungsabfall C.u über der Kapazität (= Kurve 2) aufgetragen. Man sieht, daß die Kapazität nach dem Schließen des Schalters versucht, so schnell wie möglich der von der Spannungsquelle aufgezwungenen Spannung zu folgen. Deswegen tritt auch ein

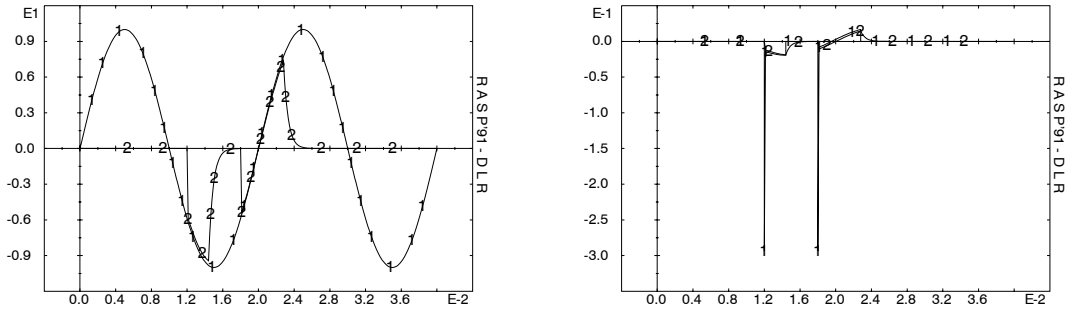


Bild 3.4: Simulationsergebnis des Schaltkreises mit Schalter.

Knick ein, wenn die Spannung an der Kapazität die Spannung der Spannungsquelle erreicht hat. $C.u$ kann wegen des Widerstands R_1 der Spannung $U0.V0$ jedoch nur angenähert folgen. Wenn der Schalter wieder geöffnet wird, entlädt sich der Kondensator nach einer

Exponentialfunktion, die durch R_2 und C bestimmt wird. Im rechten Teil von Bild 3.4 ist der zeitliche Verlauf des Stroms $R1.i$ (= Kurve 1) durch den Widerstand $R1$ und der Strom $R2.i$ (= Kurve 2) durch den Widerstand $R2$ aufgetragen. Beim Schließen des Schalters treten aufgrund der starken Spannungsänderung im Kondensator Stromspitzen in der Kapazität, und deswegen auch im Widerstand $R1$ auf.

Es stellt sich die Frage, ob der elektrische Schalter in allen Situationen so problemlos eingesetzt werden kann wie im obigen Beispiel. Zu diesem Zweck werde ein Schalter parallel zu einer Kapazität geschaltet. Der Spannungsabfall u über die Kapazität ist dann eine natürliche Zustandsgröße. Für das Simulationsproblem ist u deswegen immer eine bekannte Variable und die Schaltergleichung muß nach der einzigen verbleibenden Unbekannten, dem Strom i durch den Schalter, aufgelöst werden:

$$i = \frac{OpenSw - 1}{OpenSw} \cdot u \ . \tag{3.1}$$

Gleichung (3.1) ist jedoch *nur* gültig, solange der Schalter offen ist, da bei geschlossenem Schalter eine Division durch Null auftritt. Dieses Ergebnis ist physikalisch vollkommen verständlich. Angenommen, der Schalter werde geschlossen und der Spannungsabfall u an der Kapazität wäre zu diesem Zeitpunkt ungleich Null. Dann ändert sich u unstetig auf $u = 0$, nachdem der Schalter geschlossen ist. Diese unstetige Änderung von u ist ein Spannungsstoß, der in den idealen Elementen zu einem "unendlich" großen Strom führt. In der Realität ist ein innerer Widerstand des Kondensators und ein Leitungswiderstand vorhanden. Diese Widerstände sorgen dafür, daß die in der Kapazität gespeicherten Ladungen (in endlicher Zeit) abfließen, und daß erst dann der Spannungsabfall an der Kapazität Null wird. Wenn zumindest einer dieser Widerstände im Modell berücksichtigt wird, tritt keine Division durch Null mehr auf. Eine ähnliche Schwierigkeit ergibt sich, wenn ein Schalter in Reihe mit einer idealen Induktivität geschaltet wird.

Es kann natürlich auch sein, daß der Schalter *nur* geschlossen wird, wenn u gleich Null ist. Dies ist z.B. möglich, wenn der Schalter mit einer Diode realisiert wird. Dann tritt kein "unendlich" großer Strom mehr auf. Trotzdem wird durch Null dividiert. Der Grund liegt darin, daß sich der Störungsindex des Modells ändert. Angenommen der Index ist bei geöffnetem Schalter entweder 0 oder 1. Bei geschlossenem Schalter kann u jedoch keine Zustandsgröße mehr sein, da der Schalter die Gleichung $u = 0$ einführt. D.h. der Index des Systems ist jetzt zumindest 2. Hier liegt also ein Beispiel vor, bei dem sich der Index des Systems während der Simulation ändert.

Wird von der Erhöhung des Störungsindex einmal abgesehen, so führt eine korrekte Verwendung eines Schalters *immer* zu einem algebraischen Gleichungssystem, in dem die Schaltergleichung enthalten ist und in dem sowohl u als auch i unbekannte Variable sind (siehe auch das Beispiel von Bild 3.3). Dies ist natürlich nur eine notwendige Bedingung. Es ist ohne weiteres möglich, daß bei Verwendung mehrerer Schalter für bestimmte Schalterstellungen das Gleichungssystem singulär wird, was dann jedoch auf ein nichtphysikalisches Modell, bzw. unzulässige physikalische Vereinfachungen, hinweist.

Man beachte, daß die bei falscher Benutzung eines Schalters auftretende Singularität *keine strukturelle* Eigenschaft ist, da die Singularität nur für einen bestimmten numerischen Wert einer Variablen auftritt, die zu einem Ereignispunkt den kritischen Wert annimmt. Eine sol-

che spezielle Art der Singularität kann trotzdem bei der Transformation auf Blockdreiecksform festgestellt werden: Solche Probleme können auftreten, wenn in der Blockdreiecksform nach einer Variablen in einem **if**-Ausdruck aufgelöst werden soll, wobei diese Variable nicht in *allen* Zweigen auftritt. Dann kann diese Variable nicht bestimmt werden, wenn ein "unpassender" Zweig aktiv wird. Eine Berechnung ist nur möglich, wenn der **if**-Ausdruck zu einem algebraischen Gleichungssystem gehört. Deswegen sind die folgenden beiden Überprüfungen notwendig:

1. Wenn eine Gleichung in der Blockdreiecksform nicht zu einem algebraischen Gleichungssystem gehört und nach einer Variablen in einem **if**-Ausdruck aufzulösen ist, muß diese Variable in allen Zweigen des **if**-Ausdrucks auftreten.

2. Wenn in der Blockdreiecksform ein algebraisches Gleichungssystem vorhanden ist, in dem **if**-Ausdrücke auftreten, so muß dieses Gleichungssystem genauer analysiert werden. Hierzu wird das Gleichungssystem für jede mögliche Kombination der aktiven Zweige der **if**-Ausdrücke jeweils auf Blockdreiecksform transformiert. Da jetzt immer nur die Variablen eines Zweigs und nicht mehr alle Variablen betrachtet werden, kann das algebraische Gleichungssystem eventuell in kleinere Gleichungssysteme zerfallen. Alle Blockdreiecksformen müssen dann zu (strukturell) regulären Formen führen. Diese Vorgehensweise kann rechenaufwendig werden, wenn ein Gleichungssystem sehr viele **if**-Ausdrücke enthält.

Zusammenfassend kann festgehalten werden, daß strukturvariable Modelle durch Gleichungen mit variabler Kausalität beschrieben werden. Sind n verschiedene "Schaltstellungen" unabhängig voneinander möglich, so gibt es 2^n Modelle unterschiedlicher Kausalität. Wenn keine unzulässigen physikalischen Vereinfachungen gemacht wurden, wird ein solches strukturvariables Modell zu einem oder zu mehreren algebraischen Gleichungssystemen führen, in denen die Gleichungen variabler Kausalität enthalten sind. Die unterschiedlichen "Schaltstellungen" führen zu einer unterschiedlichen Null/Nicht-Null Struktur in diesen Gleichungssystemen.

Die ideale Diode

Der ideale elektrische Schalter kann dazu verwendet werden, um eine *ideale Diode* zu beschreiben. Eine ideale Diode ist dadurch charakterisiert, daß der Strom i durch die Diode niemals negativ und der Spannungsabfall u niemals positiv ist, siehe Bild 3.5. Der offene

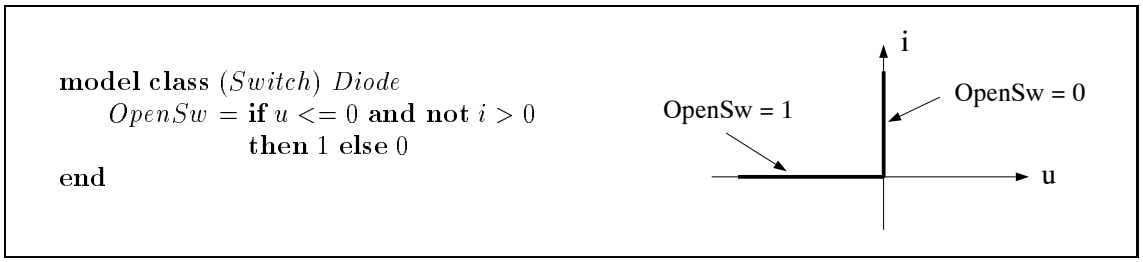


Bild 3.5: Modell und Charakteristik der idealen Diode.

Schalter wird geschlossen, wenn u positiv wird und der geschlossene Schalter wird geöffnet,

wenn i negativ wird. Der Punkt $u = 0,\ i = 0$ kann einer der beiden Schalterstellungen zugeordnet werden und wird hier willkürlich zur offenen Schalterstellung gezählt. Das Modell der idealen Diode ist in Bild 3.5 angegeben. In diesem Fall tritt das Schalten aufgrund numerischer Werte interner Variablen auf. Die Klasse *Diode* wird von der Klasse *Switch* durch Vererbung abgeleitet, d.h. die Eigenschaften des elektrischen Schalters sind auch in der Diode enthalten. Der Wert der Schaltervariablen *OpenSw* wird in der Dioden-Klasse abhängig von u bzw. i gesetzt. Am Beispiel der einfachen Gleichrichter-Schaltung mit Last in Bild 3.6 wird die Verwendung der idealen Diode näher untersucht. Die Schaltung wird

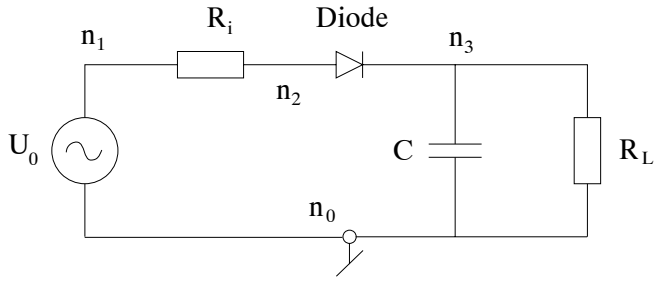


Bild 3.6: Einfacher Gleichrichter mit Last.

durch das folgende Modell beschrieben:

```
model Rectifier
    submodel (Vsource)   U0
    submodel (Diode)     Diode
    submodel (Resistor)  Ri(R = 20), RL(R = 500)
    submodel (Capacitor) C(C = 0.0001)
    submodel (Ground)    g
    input  u

    connect U0    at (n1,n0), Ri  at (n1,n2),
            Diode at (n2,n3), RL  at (n3,n0),
            C     at (n3,n0), g   at n0
    u = U0.V0
end
```

Bei der Umformung auf Blockdreiecksform ergibt sich

```
| Ri.          Ri.R*Ri.i          = [Ri.u]
|              Ri.u               = Ri.Va - [Diode.Va]
| Diode.       [Diode.u]          = Diode.Va - C.Va
|              [Diode.OpenSw]     = if Diode.u <= 0 and not U0.i > 0 then 1 else 0
|              0                  = Diode.OpenSw*[Ri.i] + (1 - Diode.OpenSw)*Diode.u
```

Im Gegensatz zum vorherigen Beispiel des elektrischen Schalters, ist dies ein unangenehmes, nichtlineares Gleichungssystem. Das Gleichungssystem ist nichtlinear, da die beiden Unbekannten Diode.OpenSw und Ri.i in der letzten Gleichung miteinander multipliziert werden, und da zudem noch die beiden Unbekannten Diode.u und U0.i in der **if**-Bedingung auftreten. Mit Standardverfahren kann dieses Gleichungssystem nicht gelöst werden, da zum einen die Unbekannte Diode.OpenSw keine kontinuierliche Variable ist, sondern nur einen von zwei Werten annehmen kann, und da zum anderen der **if**-Ausdruck eine Unstetigkeit einführt.

Das Problem kann gelöst werden, indem die Eigenschaft ausgenutzt wird, daß Diode.OpenSw den Wert nur an Ereignispunkten ändert. *Während* der Integration ist Diode.OpenSw deswegen konstant und *bekannt* und geht nicht in das Gleichungssystem ein. An Ereignispunkten

liegt jedoch genau das obige Gleichungssystem vor. Das Gleichungssystem wird durch eine Fixpunktiteration gelöst, wobei die Variable Diode.OpenSw als Iterationsvariable benutzt wird.

Die Umsetzung dieser Lösungsstrategie erfolgt mit einer Ersetzung von OpenSw durch eine Boole'sche *Zustands*variable *Open*. Hierzu wird die Boole'sche Fassung des elektrischen Schalters benötigt:

```
model class (TwoPin) Switch
    terminal Open
    0 = if Open then i else u
end
```

Die Klasse *Diode* wird jetzt durch Einführung des **new**-Operators modifiziert:

```
model class (Switch) Diode
    new(Open) = u ≤ 0 and not i > 0
end
```

Open ist damit eine Boole'sche Zustandsvariable, also eine *bekannte* Größe[6]. An Ereignispunkten ergibt sich **new**(*Open*) aufgrund der aktuellen Werte des Spannungsabfalls an der Diode und des Stroms durch die Diode (man beachte, daß im Modell **new**(*Open*) und *Open* immer zwei unterschiedliche Variablen sind). Am Ende der Ereignisbehandlung fügt der Compiler den "Update" der Boole'schen Zustandsvariablen hinzu, d.h. $Open = \mathbf{new}(Open)$. Durch die unstetige Änderung von *Open* wird sofort wieder ein Ereignis ausgelöst und das Modell noch einmal ausgewertet. Die gesamte Prozedur wird solange wiederholt, bis der neu berechnete Wert von **new**(*Open*) gleich dem aktuellen Wert von *Open* ist. Dies ist im obigen einfachen Modell schon bei der zweiten Iteration der Fall. Wenn mehrere Dioden oder andere unstetige Elemente zum selben Zeitpunkt schalten, können mehrere Iterationsdurchläufe notwendig sein. Die Konvergenz der Iteration kann nur für einfache Spezialfälle, nicht jedoch im allgemeinen Fall garantiert werden. Man beachte, daß es für dieses Problem kaum bessere numerische Verfahren gibt, da einige der Unbekannten in dem nichtlinearen Gleichungssystem Boole'sche Variablen sind.

Boole'sche Zustandsvariablen werden vor dem Start einer Simulation standardmäßig mit *False* initialisiert. Weiterhin wird der Integrationsstart wie ein Ereignis behandelt. Das heißt, auch hier findet eine Iteration statt, um die korrekte Schaltstellung der Diode beim Beginn der Integration zu ermitteln.

Das Ergebnis einer Simulation ist in Bild 3.7 zu sehen. Im linken Teil ist der zeitliche Verlauf der Spannung u der Spannungsquelle (= Kurve 1) und der Spannungsabfall $RL.u$ (= Kurve 2) über der Last aufgetragen. Im rechten Teil von Bild 3.7 ist der zeitliche Verlauf des Stroms $Ri.i$ (= Kurve 1) durch den Innenwiderstand der Spannungsquelle und der Strom $RL.i$ (= Kurve 2) durch die Last aufgetragen. Wenn der Strom durch die Diode Null wird, wird der Dioden-Schalter geöffnet. Bei Vernachlässigung des Innenwiderstandes der Spannungsquelle tritt dieser Fall dann ein, wenn sich u und $RL.u$ schneiden, da dann der Spannungsabfall über die Diode Null ist. Da ein Innenwiderstand vorhanden ist, tritt das Schalten kurz vor diesem Schnittpunkt auf. Bei geöffnetem Schalter entleert sich der Kondensator, so daß durch den Lastwiderstand weiterhin ein Strom fließt, bis der

[6] Alle Variablen, auf die der **der** Operator oder der **new** Operator angewendet werden, benötigen einen Anfangswert beim Start der Integration und werden deswegen standardmäßig als *bekannte* Größen bei der Auswertung der Modellgleichungen angesehen.

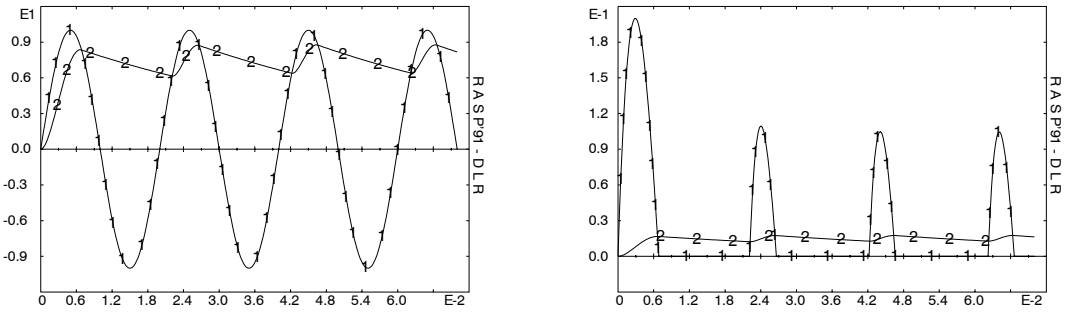


Bild 3.7: Simulationsergebnis des einfachen Gleichrichterkreises.

Dioden-Schalter wieder geschlossen wird. Als Ergebnis erhält man einen etwas unruhigen Gleichstrom durch den Lastwiderstand.

Beschreibung von Schaltvorgängen durch endliche Automaten

Abgesehen von der Konvergenz-Problematik bei der Fixpunktiteration tritt im obigen Dioden-Modell noch eine andere Schwierigkeit auf. Diese Schwierigkeit ist nicht spezifisch für die Diode, sondern tritt in ähnlicher Form in den meisten strukturvariablen Modellen auf, und zwar vollkommen unabhängig davon, wie die Strukturvariabilität letztendlich behandelt wird. Deshalb wird dieses Problem hier genauer diskutiert.

Aus Bild 3.6 auf Seite 59 liest man die Gleichungen für den Diodenstrom und die Diodenspannung in beiden Schaltstellungen ab:

$$\begin{array}{lll} Open = True: & Diode.i = 0 \quad , & Diode.u = u - C.u \\ Open = False: & Diode.i = (u - C.u)/Ri.R \; , & Diode.u = 0 \quad . \end{array}$$

Angenommen, *Open* sei zunächst $True$ und die Diodenspannung wird größer als Null, d.h. es tritt ein Ereignis ein und *Open* wird $False$. Aufgrund der Diodengleichung ist garantiert, daß am Schaltpunkt die Diodenspannung klein ist, aber größer als Null:

$$Diode.u = u - C.u = eps > 0 \quad .$$

Nach dem Schalten der Diode ist $Open = False$, die Diodenspannung wird identisch Null, und der Diodenstrom berechnet sich zu:

$$Diode.i = (u - C.u)/Ri.R = eps/Ri.R \quad .$$

Weil der Widerstand $Ri.R$ immer positiv ist und weil $eps > 0$ ist, ist $Diode.i$ bei exakter Rechnung ebenfalls positiv. Wenn jedoch *eps* genügend klein und $Ri.R$ genügend groß ist, wird durch die Division eine Zahl entstehen, die kleiner als die kleinste darstellbare Zahl ist. Damit wird $Diode.i$ *exakt* Null. Die Diodengleichung (siehe Seite 60) legt jedoch fest, daß der Schalter geöffnet wird, wenn der Diodenstrom kleiner oder gleich Null wird. Somit tritt sofort wieder ein Ereignis ein, und die Diode schaltet in die ursprüngliche Stellung zurück. Dort ist aber $Diode.u$ immer noch positiv und der Schalter wird wieder geschlossen, d.h. es entsteht eine Iterationsschleife, die nie abbricht.

Dies ist ein generelles Problem: Nach einem Schaltvorgang werden Indikatorfunktionen eventuell anders berechnet, so daß durch kleine numerische Fehler eine schon in der Nähe

von Null befindliche Indikatorfunktion exakt Null wird oder gar das Vorzeichen wechselt. Dies kann vor allem dann auftreten, wenn Indikatorfunktionen durch Lösen algebraischer Gleichungssysteme berechnet werden. Diese Schwierigkeit kann durch Einführung einer kleinen Hysterese abgeschwächt werden, da dann bei Ungenauigkeiten in der Rechnung nicht sofort zurückgeschaltet wird. Mit einer geschickten Realisierung wird die Hysterese *nur* aktiv, wenn der oben beschriebene kritische Fall eintritt, so daß diese großzügig dimensioniert werden kann.

Im Dioden-Beispiel wird die Schaltstruktur durch die Einführung einer solchen Hysterese komplizierter. Um die Übersichtlichkeit auch bei einer komplexeren Schaltstruktur zu bewahren ist es sinnvoll, diese durch einen *deterministischen endlichen Automaten* (abgekürzt: DEA) zu beschreiben und dann automatisch in eine Modellbeschreibung umzuwandeln. Die Eigenschaften eines DEA sind z.B. in [Aho88] beschrieben. Im linken Teil von Bild 3.8

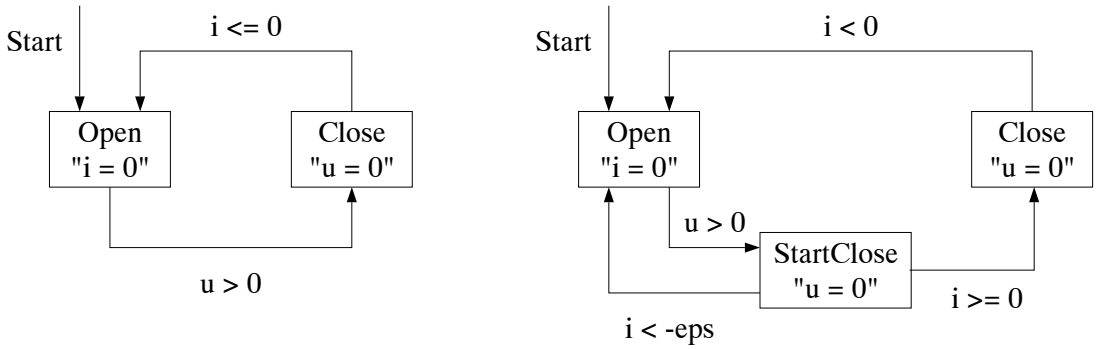


Bild 3.8: Übergangsdiagramme zweier Diodenmodelle.

ist das Übergangsdiagramm des DEA des bisherigen Diodenmodells zu sehen. Der DEA besteht aus den zwei *Zuständen "Open"* und *"Close"*. Beim Start einer Simulation wird (willkürlich) mit dem Zustand *"Open"* begonnen. Der *Übergang* von einem Zustand zum nächsten wird eindeutig durch den Wert der Diodenspannung u bzw. des Diodenstroms i charakterisiert. Wenn die Diode z.B. offen ist, befindet sich der DEA im Zustand *"Open"*. Wird die Diodenspannung größer als Null, so schaltet der DEA in den Zustand *"Close"*.

Im rechten Teil von Bild 3.8 ist das Übergangsdiagramm der Diode nach Einführung einer Hysterese zu sehen. Dieser DEA unterscheidet sich vom ursprünglichen DEA durch einen zusätzlichen Schaltzustand *"StartClose"*. Wenn die Diode geschlossen wird, so wird nicht direkt in den Zustand *"Close"* geschaltet, sondern zuerst in diesen Zwischenzustand. Der Zustand *"StartClose"* wird fast genauso wie der Zustand *"Close"* behandelt. Insbesondere wird die Diodenspannung auf Null gesetzt. Bei exakter Rechnung sollte der Diodenstrom in diesem Zustand positiv sein. In diesem Fall wird sofort auf den Zustand *"Close"* weitergeschaltet. Ist der Diodenstrom jedoch aufgrund numerischer Ungenauigkeiten negativ, so wird erst dann wieder zurück in den Zustand *"Open"* geschaltet, wenn der Strom einen gewissen Grenzwert $-eps$ unterschritten hat. Ansonsten bleibt die Diode im Zustand *"StartClose"*. Durch geeignete Wahl von eps kann immer erreicht werden, daß die oben geschilderte Iterationsschleife nicht auftritt. Im Gegensatz zum ursprünglichen Diodenmodell wird der Punkt $u = 0, i = 0$ sowohl der offenen als auch der geschlossenen Diode zugeordnet. Damit werden die meisten numerischen Ungenauigkeiten schon korrekt abgehandelt.

Ein Modell kann automatisiert aus dem Übergangsdiagramm eines DEA gewonnen werden. Hierzu wird für jeden DEA-Zustand eine entsprechende Boole'sche Variable eingeführt und

die folgende Transformationsregel wird auf jeden Zustand im Übergangsdiagramm angewendet:

```
new(state) = pre-state-1 and in-condition-1 or
             pre-state-2 and in-condition-2 or
                        ...                or
             state and not (out-condition-1 or
                            out-condition-2 ...)
```

Damit ergibt sich z.B. für das ursprüngliche Diodenmodell die folgende Form:

```
model class Diode
    local Open = True, Close = False

    0 = if Open then i else u
    new(Close) = Open and u > 0  or  Close and not i <= 0
    new(Open)  = Close and i <= 0  or  Open and not u > 0
end
```

Dieses Modell kann vereinfacht werden, da *"Close"* immer gleich "**not** *Open*" ist:

```
model class Diode
    local Open = True

    0 = if Open then i else u
    new(Open) = not Open and i <= 0  or  Open and u <= 0
end
```

Auf dieselbe Weise gewinnt man aus dem rechten Übergangsdiagramm von Bild 3.8 ein numerisch robusteres (ideales) Diodenmodell:

```
model class Diode
    parameter eps = 1.E-10
    local     Open = True, Close = False, StartClose

    0 = if Open then i else u

    new(Open)  = Close and i < 0 or
                 StartClose and i < -eps or
                 Open and not u > 0
    new(Close) = StartClose and i >= 0 or
                 Close and not i < 0
    StartClose = not (Open or Close)
end
```

Zum Abschluß soll, unter Verwendung des Diodenmodells mit Hysterese, noch ein etwas komplexeres Beispiel behandelt werden. In Bild 3.9 ist ein Gleichrichterkreis mit Last aus [Tiet86] mit einer Brückenschaltung aus 4 (idealen) Dioden zu sehen. Der Schaltkreis hat insgesamt $2^4 = 16$ verschiedene Schaltstellungen. Wenn die Schalter aller Dioden offen sind, ist der Lastwiderstand R_L zusammen mit der Kapazität C ein separater Schaltkreis, der nicht mehr mit dem Schaltkreis verbunden ist, der die Spannungsquelle enthält. Dies führt zu Schwierigkeiten, da dann eine Gleichung "fehlt" und die Matrix des linearen Gleichungssystems singulär wird. Nach einem Vorschlag von Elmqvist, wird deswegen der (sehr große) Widerstand R_b eingeführt, welcher den tatsächlich vorhandenen hochohmigen Kontakt des

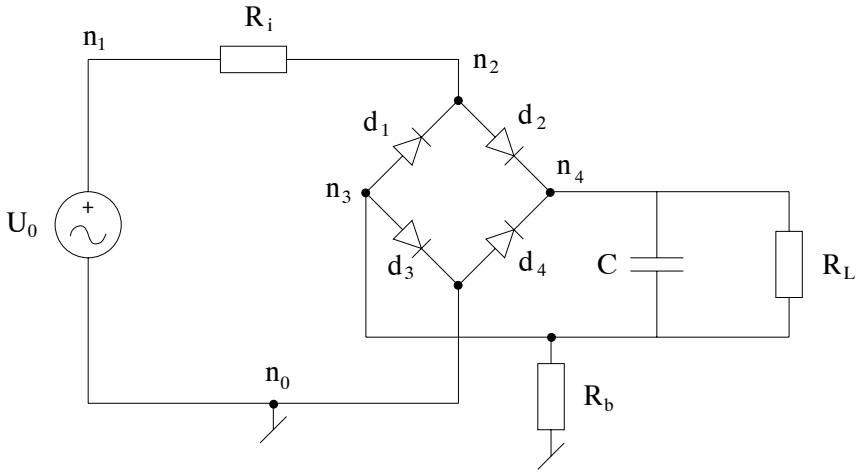


Bild 3.9: Brückengleichrichter mit Last.

Lastschaltkreises mit der "Erde" beschreibt. Auf die Integration hat dieser Widerstand praktisch keinen Einfluß. Die Brückenschaltung kann folgendermaßen modelliert werden:

```
model Graetz
    submodel (Vsource)   U0
    submodel (Diode)     d1, d2, d3, d4
    submodel (Resistor)  Ri(R = 20), RL(R = 500), Rb(R = 1.E6)
    submodel (Capacitor) C(C = 0.0001)
    submodel (Ground)    g

    input u
    node  n0, n1, n2, n3, n4

    connect U0 at (n1,n0),  Ri at (n1,n2),  d1 at (n3,n2),
            d2 at (n2,n4),  d3 at (n3,n0),  d4 at (n0,n4),
            C  at (n4,n3),  RL at (n4,n3),  Rb at (n3,n0),  g at n0
    u = U0.V0
end
```

Das Ergebnis einer Simulation ist in Bild 3.10 zu sehen. Im linken Teil des Bilds ist der zeitliche Verlauf der Spannung u der Spannungsquelle (= Kurve 1) und der Spannungsabfall $RL.u$ über der Last (= Kurve 2) aufgetragen. Im rechten Teil des Bilds ist der zeitliche

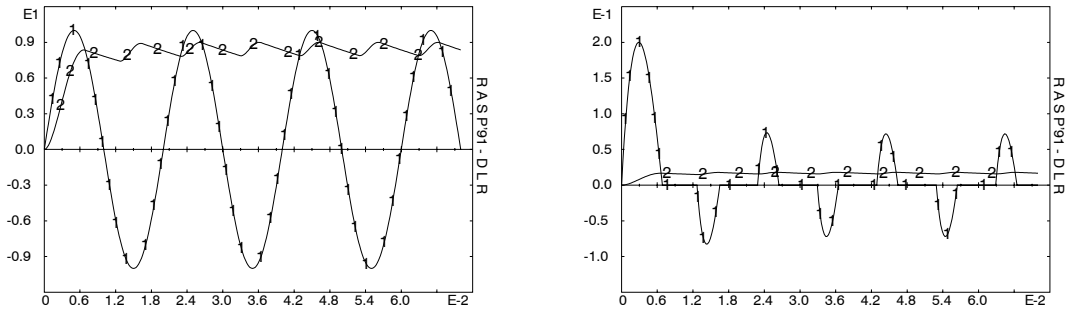


Bild 3.10: Simulationsergebnis des Brückengleichrichters.

Verlauf des Stromes $Ri.i$ durch den Innenwiderstand der Spannungsquelle (= Kurve 1) und der Strom $RL.i$ durch die Last (= Kurve 2) aufgetragen. Ein Vergleich mit dem Simulationsergebnis des einfachen Gleichrichterkreises in Bild 3.7 auf Seite 61 zeigt, daß die Brückenschaltung zu einem etwas ruhigeren Verhalten der Last führt.

3.4 Diskussion

Jedes physikalische System kann bei genügend genauer Modellierung durch ein kontinuierliches Modell beschrieben werden. Dies kann jedoch zur sehr steifen, und damit in der Simulation sehr rechenzeitintensiven, Differentialgleichungen führen; abgesehen davon, daß der Aufwand zur Ermittlung der Modellparameter groß ist. Durch Einführen schaltender Elemente kann die Realität oft genügend genau approximiert werden. Die Simulation-Rechenzeit kann im Vergleich zu einer detaillierten, "steifen", Formulierung um Größenordnungen geringer sein.

In heute verfügbaren Modellierungsumgebungen[7], werden unstetige Elemente mit Hilfe von Ereignissen modelliert. Eine Unstetigkeit, oder eine Strukturänderung, löst ein Ereignis aus, die Integration wird angehalten, ein vordefiniertes Codestück wird ausgeführt und die Integration wird neu gestartet. Diese Vorgehensweise löst die numerischen Probleme, führt jedoch zu Schwierigkeiten, wenn mehrere Ereignisse gleichzeitig zum selben Zeitpunkt auftreten, oder wenn Indikatorfunktionen am Beginn der Integration, oder an einem Ereignispunkt, identisch verschwinden.

Mit den vor kurzem in [Elmq93b] vorgeschlagenen Sprachelementen erfolgt die Definition schaltender Elemente auf einem höheren Niveau, mittels *unstetiger* Gleichungen und mittels *Instant*-Gleichungen, die durch Boole'sche Variable und Ausdrücke kontrolliert und synchronisiert werden. Diese neuen Gleichungstypen werden genauso wie kontinuierliche Gleichungen behandelt und insbesondere mit auf Blockdreiecksform transformiert. Damit wird garantiert, daß die Reihenfolge aller Gleichungen auch an Ereignispunkten korrekt ist. Bei der Codegenerierung werden die neuen Sprachelemente auf geeignete Zeit- und Zustands-Ereignisse in der Zielsprache abgebildet, wobei auch verschwindende Indikatorfunktionen zufriedenstellend abgehandelt werden.

Die neuen Sprachelemente erlauben es, eine große Klasse von strukturvariablen Systemen auf einfache Weise zu modellieren. Eine variable Modellstruktur führt immer auf ein algebraisches Gleichungssystem mit variabler Null/Nicht-Null Struktur der Elemente des Gleichungssystems. Zur Zeit wird in Dymola immer das komplette Gleichungssystem gelöst. Ein Effizienzgewinn kann erwartet werden, wenn z.B. für jede Schaltstellung ein spezieller Code des Gleichungssystems ermittelt und erzeugt wird. Dies stößt jedoch an Grenzen, wenn zuviele Schaltelemente in dasselbe Gleichungssystem eingehen.

Die komfortable und sichere Definition von schaltenden Elementen mit den erläuterten Sprachelementen ist ein Fortschritt, der die Modellierung solcher Komponenten vereinfacht. Das Konzept hat noch ein paar (unkritische) Schwachstellen, die verbessert werden sollten. Zum Beispiel können unstetige Gleichungen nur durch **if**-Ausdrücke beschrieben werden, die jeweils *genau eine* Gleichung pro **if**-Zweig besitzen. Aus Anwendersicht ist es wichtig, daß in jedem **if**-Zweig auch mehrere Gleichungen aufgeführt werden können. Es ist jedoch noch nicht klar, wie solche Gleichungstypen mit auf Blockdreiecksform zu transformieren sind. Weiterhin kann die Effizienz an einigen Stellen verbessert werden. Zum Beispiel ist es häufig nicht notwendig, daß die Iterationsschleife an einem Ereignispunkt über das gesamte Modell geht. Hier sollte die kleinstmögliche Iterationsschleife automatisch ermittelt werden.

[7] Zum Beispiel in ACSL [Mitc91], Omola [Ande92], MAST [Mant92] oder dem geplanten Analog-VHDL [Vach93].

Kapitel 4

Objektorientierte Modellierung von Mehrkörpersystemen

In diesem Kapitel wird beschrieben wie allgemeine Mehrkörpersysteme (abgekürzt: MKS) *objektorientiert* modelliert, und wie solche Modelle für *verschiedene Aufgabenstellungen* in numerisch lösbare Formen überführt werden können. Hierzu werden die im Kapitel 2.2 ab Seite 21 erläuterten allgemeinen Techniken eingesetzt. Weiterhin wird die Bondgraph-Methode auf MKS angewandt und verallgemeinert. Mit dieser Verallgemeinerung kann für jedes objektorientierte Modell eines MKS sofort ein äquivalenter Bondgraph angegeben werden. Der Energiefluß in einem MKS wird deswegen sowohl durch das objektorientierte MKS-Modell als auch durch den zugehörigen Bondgraph angegeben.

Die Theorie der Mehrkörpersysteme wurde seit Mitte 1960 entwickelt und ist für Starrkörpersysteme im wesentlichen abgeschlossen. Es existiert eine umfangreiche Literatur mit einer großen Anzahl an Vorschlägen zur effizienten Erstellung der Bewegungsgleichungen mechanischer Systeme. Eine Zusammenstellung der wichtigsten MKS-Programme ist in [Schi90] zu finden.

Kecskeméthy [Kecs88, Kecs93, Kecs93a] hat als erster MKS-Modelle und MKS-Algorithmen mittels der Programmiersprache C++ objektorientiert beschrieben. Ein objektorientiertes MKS-*Daten*modell zur Verwendung in einer Datenbank wurde von Otter, Hocke, Daberkow und Leister [Otte90, Otte93a] entwickelt und prototypisch mit Hilfe des RSYST-Datenbanksystems [Rueh90, Rueh93] implementiert [Hock93]. Die in der vorliegenden Arbeit verwendete objektorientierte Sicht eines MKS basiert auf diesen beiden Ansätzen. Ein weiterer, ebenfalls recht erfolgversprechender, objektorientierter MKS-Ansatz stammt von Anantharaman [Anan93].

F.E. Cellier hat schon in [Cell91] 1-dimensionale mechanische Systeme mit Hilfe von Dymola objektorientiert modelliert. H. Elmqvist hat prototypisch mit 2-dimensionalen MKS-Modellen innerhalb von Dymola experimentiert [Elmq92a]. Beide Vorgehensweisen sind nicht direkt auf allgemeine 3-dimensionale MKS verallgemeinerbar und werden deswegen hier nicht benutzt. Jedoch hat Elmqvist in [Elmq92a] schon wichtige prinzipielle Techniken basierend auf dem Tearing-Verfahren entwickelt, um ein objektorientiertes MKS-Modell für die wichtigsten Aufgabenstellungen in eine geeignete mathematische Form zu überführen.

Die in Kapitel 1.2 ab Seite 9 erläuterte Bondgraph-Methode wurde entwickelt, um physi-

kalische Systeme einheitlich darstellen zu können. Insbesondere können hiermit auch leicht 1-dimensionale mechanische Systeme modelliert werden. Seit längerem wird versucht, allgemeine 3-dimensionale MKS mit Bondgraphen zu beschreiben [Bos86, Karn90, Fahr91]. Diese Ansätze sind jedoch unbefriedigend, da Teile eines speziellen MKS-Algorithmus (z.B. die Wahl der Minimalkoordinaten) hier in den Bondgraph eingehen, und damit die rein physikalische Sichtweise verloren geht. In diesem Kapitel wird eine neue Darstellung von MKS-Bondgraphen angegeben, die diesen Nachteil vermeidet.

Der zentrale Teil der Bondgraph-Methode, d.h. die explizite Darstellung der Energieflüsse in einem physikalischen System, ist zum Systemverständnis generell hilfreich. Zum einen, weil damit in einer großen Zahl von Gleichungen der Überblick nicht so leicht verloren geht, zum anderen, weil man damit einfach die Ergebnisse einer Simulation überprüfen kann[1]. Weiterhin wird der Blick auf das wesentliche gelenkt: In welchen Größenordnungen liegt die übertragene Leistung? Wie ist diese auf die Bauteil-Abmessungen abgestimmt? Woher kommt die Energie, wo und in welcher Form wird sie im System gespeichert, und wie fließt sie aus dem System ab (z.B. in Form von Wärme)? Wie diese Information grafisch darstellt wird, als Bondgraph oder in einer anderen Form, ist demgegenüber letztendlich unwesentlich.

4.1 Einführungsbeispiel

In diesem Abschnitt werden anhand eines einfachen MKS die wichtigsten MKS-Klassen eingeführt und die prinzipielle Modellierungssicht verdeutlicht. Im nächsten Abschnitt werden diese Klassen im Detail besprochen.

Im linken Teil von Bild 4.1 ist schematisch ein Doppelpendel gezeigt. Dieses besteht aus zwei Starrkörpern, die über einfache Drehgelenke miteinander und mit der Umgebung verbunden sind. Einer der Starrkörper ist darüberhinaus noch mit einer Feder an die Umgebung angekoppelt.

Im rechten Teil von Bild 4.1 wird die objektorientierte Sichtweise des Doppelpendels grafisch veranschaulicht. Das Modell besteht aus mehreren Objekten die (mechanisch) starr miteinander verbunden sind. Zum Beispiel gibt es ein Inertialsystem *"i"* der Klasse Inertial, das über ein Drehgelenk *"r1"* der Klasse Revolute mit der masselosen Stange *"b1"* der Klasse Bar und mit dem massebehafteten Körper *"m1"* der Klasse Body verbunden ist. Die kleinen Querstriche im Bild verdeutlichen die Objektgrenzen. Ein MKS wird aus Grundelementen aufgebaut, die ihrerseits auf einigen wenigen Basisobjekten beruhen. Die Grundelemente werden (mechanisch) starr miteinander gekoppelt und bilden die mechanischen Eigenschaften des Systems nach. Ein solches Modell hat Ähnlichkeiten mit einem nichtlinearen Finite-Elemente Programm. Der rechte Teil von Bild 4.1 kann direkt in die folgende Modellbeschreibung überführt werden:

[1] Die Ergebnisse einer Simulation können überprüft werden, indem die Summe der Energieströme (= Leistungssumme) berechnet wird, die in die Cuts einer Komponente einfließen. Wenn in diesem Element keine Energie gespeichert wird, so muß diese Summe Null ergeben.

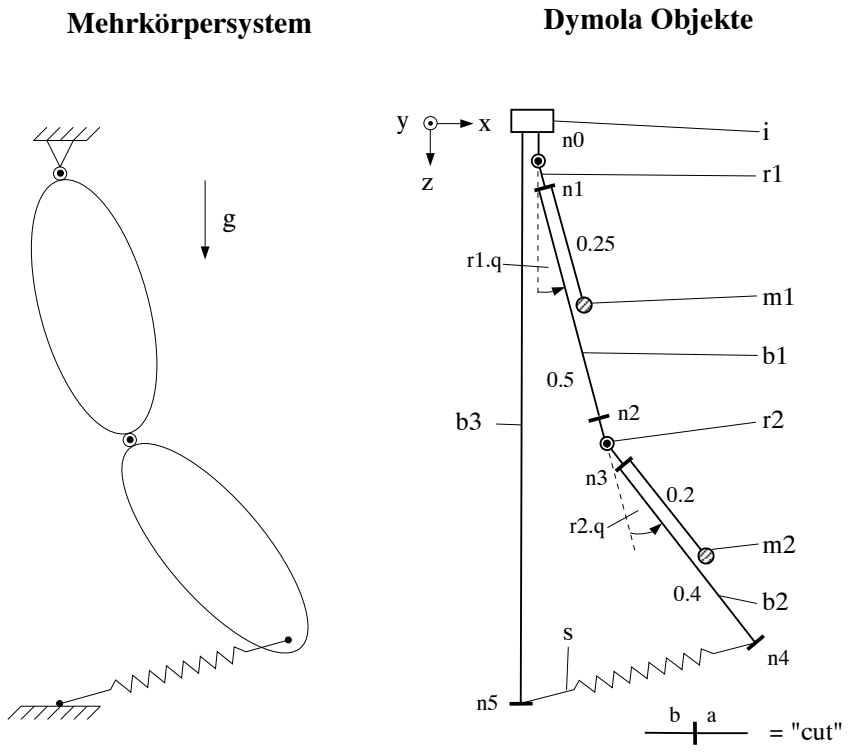


Bild 4.1: Objektorientiertes Modell eines Doppelpendels.

```
model DoublePendulum
    submodel (Inertial)  i   (ng3=1, g=9.81)
    submodel (Revolute) r1   (n2=1)  , r2 (n2=1)
    submodel (Bar)      b1   (r3=0.5), b2 (r3=0.4)
    submodel (Bar)      b3   (r3=1.3)
    submodel (Body)     m1 (m=1, r3=0.25)
    submodel (Body)     m2 (m=1, r3=0.2)
    submodel (Spring)   s    (c=1, s0=0.5)

    connect i   to r1 to b1 to r2 to b2,  i to b3,
            m1 at r1:b,
            m2 at r2:b,
            s   at (b3:b,b2:b)

    { alternativ kann die Verbindung definiert werden als:

      node n0, n1, n2, n3, n4, n5
      connect i    at n0,  r1 at (n0,n1),  b1 at (n1,n2),
              m1  at n1,  r2 at (n2,n3),  b2 at (n3,n4),
              m2  at n3,  b3 at (n0,n5),  s   at (n4,n5)   }
end
```

Als Kommentar in geschweiften Klammern ist in diesem Modell noch eine alternative Beschreibungsform der Objekt-Verbindungen angegeben. Objektgrenzen werden hier durch Knotenpunkte (**node**) charakterisiert, siehe Knoten $n_0, \ldots, n_5$ in Bild 4.1, die in den Verbindungsdefinitionen benutzt werden. Dies ist vor allem zur Beschreibung von MKS mit Schleifen vorteilhaft.

Im obigen Modell werden alle benötigten Objekte zuerst mit den **submodel** Anweisungen deklariert und damit instantiiert. Danach wird mit der **connect** Anweisung definiert, wie

diese Objekte miteinander verbunden sind.

Gelenke, wie Dreh- und Schubgelenke, sind masselose Elemente, die über zwei Schnittufer mit anderen mechanischen Elementen verbunden werden können. Dies sind in der Regel entweder masselose Stangen oder massebehaftete Körper. Eine direkte Verbindung zweier Gelenke miteinander ist natürlich auch möglich. Die beiden Schnittufer werden durch **cuts** mit den Namen *a* und *b* beschrieben. Der **main path** geht von Cut *a* zum Cut *b*, d.h. eine Anweisung der Form "**connect** *r1* **to** *b1*" ist gleichbedeutend mit "**connect** *r1:b* **at** *b1:a*". Die Anweisung "**connect** *s* **at** *(b3:b,b2:b)*" bedeutet, daß Cut *a* der Feder am Cut *b* des Objekts *b3* und Cut *b* der Feder am Cut *b* von *b2* zu befestigen ist.

Um die Abmessungen der Elemente zu definieren, wird das MKS in eine ausgezeichnete Stellung, die Referenzkonfiguration, gebracht. Beim Doppelpendel-Beispiel ist diese Stellung durch die senkrecht nach unten hängenden Körper charakterisiert. Zusätzlich muß ein beliebiges Koordinatensystem im Inertialsystem, d.h. ein Objekt der Klasse *Inertial*, eingeführt werden, wie im oberen Teil von Bild 4.1 angedeutet. Alle koordinatensystemabhängigen Größen (= Tensoren 1. und 2. Stufe) sind in der Referenzkonfiguration im Koordinatensystem des Inertialsystems anzugeben. Zum Beispiel wird das Objekt *b2* durch einen Ortsvektor charakterisiert, der vom Cut *a* zum Cut *b* der Stange zeigt und der in der Referenzkonfiguration die folgenden Koordinaten hat:

$$\mathbf{r}_{(n3,n4)} = \begin{bmatrix} r1 \\ r2 \\ r3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0.4 \end{bmatrix} m \ .$$

Die drei Koordinaten $r1$, $r2$, $r3$ sind in der MKS-Bibliothek als Parameter definiert und haben die Voreinstellung Null[2]. Alle physikalischen Parameter müssen im *kg-m-s* SI-System angegeben werden. Durch die Anweisung "**(submodel)** *Bar* $b2\,(r3 = 0.4)$" wird also eine masselose Stange mit der Länge 0.4 m definiert. In der Referenzkonfiguration liegt die Stange in Richtung der 3-Achse des Inertialsystems.

Auf die gleiche Weise wird die Drehachse eines Drehgelenks durch einen Vektor $\mathbf{n} = [n1, n2, n3]$ beschrieben, der in der Drehachse des Gelenks liegt. Dieser Vektor muß kein Einheitsvektor sein. Die Anweisung "**(submodel)** *Revolute* $r1\,(n2 = 1)$" definiert damit ein Drehgelenk mit dem Namen $r1$, dessen Drehachse in der Referenzkonfiguration in Richtung der 2-Achse des Inertialsystems zeigt. Im obigen, ebenen, Beispiel zeigt die Drehachse immer in Richtung dieser Achse. Bei einer räumlichen Bewegung ist das in der Regel aber nicht mehr der Fall.

Ein Objekt der Klasse *Body* wird durch den Vektor $\mathbf{r} = [r1, r2, r3]$ definiert, der vom Cut *a* des Objekts zum Massenmittelpunkt zeigt, sowie durch seine Masse m und durch den symmetrischen Trägheitstensor $\mathbf{I}$ bezüglich des Massenmittelpunkts (dargestellt im Inertialsystem). Die Koordinaten des Trägheitstensors werden durch die Parameter $I11$, $I22$, $I33$, $I21$, $I31$, $I32$ definiert. Schließlich wird eine Feder durch die Federkonstante c und die unausgelenkte Federlänge $s0$ beschrieben. Die Voreinstellung aller Parameter eines MKS ist, ohne Ausnahme, Null.

Mit den in den nächsten Abschnitten beschriebenen Elementen, kann das Doppelpendel von Bild 4.1 auf Seite 68 auch als Bondgraph dargestellt werden. Im linken Teil von Bild 4.2

[2] Wenn ein Parameter in der Deklaration nicht angegeben wird, wird seine Voreinstellung benutzt.

sind noch einmal die verschalteten Objekte zu sehen, aus denen das Doppelpendel aufgebaut ist. Für jedes Basis-Element des MKS gibt es korrespondierende Bondgraph-Elemente, wie im rechten Teil von Bild 4.2 zu sehen ist. Die einzelnen Elemente können leicht durch

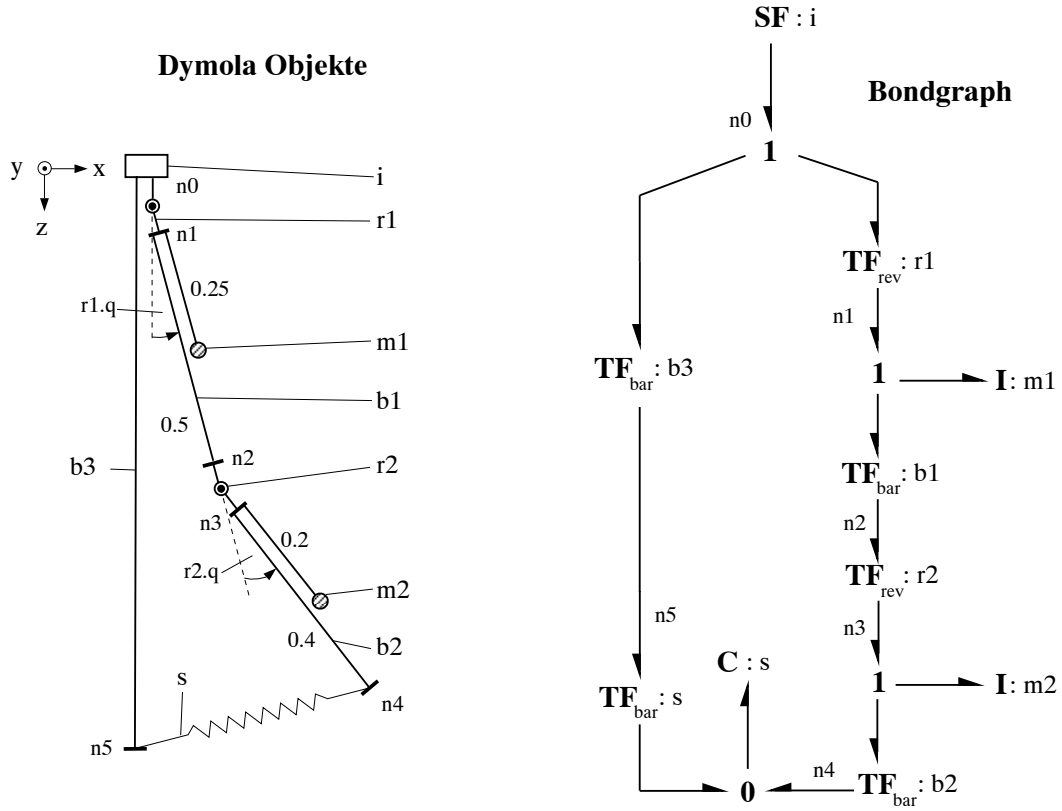


Bild 4.2: Bondgraph eines Doppelpendels.

die Knotenpunkte n_i identifiziert werden. Zum Beispiel wird ein Drehgelenk durch einen Transformer $\mathbf{TF}_{rev}$ dargestellt, da ein Gelenk nur die Effort- und Flow-Variablen (= Lage, Geschwindigkeit, Beschleunigung und Schnittkraft/moment) , am Cut b auf andere Weise aufteilt als am Cut a, ohne dabei den Energiefluß zu ändern. Eine Stange wird ebenfalls durch einen speziellen Transformer $\mathbf{TF}_{bar}$ beschrieben. Die Trägheitseigenschaften eines Starrkörpers werden durch eine verallgemeinerte Induktivität **I** dargestellt. Das Inertialsystem ist eine Flow-Source **SF**, da die kinematischen Flow-Variablen am Inertialsystem vorgegeben sind. Schließlich ist ein Feder-Kraftelement aus mehreren Bondgraph-Elementen zusammengesetzt, um zuerst mit einem Transformer die Größen der beiden Cuts in dasselbe Koordinatensystem zu transformieren und dann auf eine verallgemeinerte Kapazität **C** zu schalten.

4.2 Aufgabeninvariante Grundelemente

In diesem Abschnitt werden die Modellklassen eingeführt, die zur objektorientierten Modellierung von MKS benutzt werden. Alle MKS werden immer mittels dieser Klassen beschrieben, unabhängig von der Aufgabenstellung. In den nächsten Abschnitten wird gezeigt, wie ein so beschriebenes Modell in eine numerisch auswertbare Form überführt werden kann.

Eine direkte, objektorientierte Modellierung von MKS führt auf eine DAE mit Störungsindex 3 (siehe z.B. [Andr90]). Die "Zwangsgleichungen" müssen deswegen zweimal differenziert werden, um eine Index-1 DAE zu erhalten. In der Mechanik wird die in Kapitel 2.4 erläuterte allgemeine "dummy derivative" Technik schon lange benutzt, um das so erhaltene überbestimmte differential-algebraische Gleichungssystem zu lösen, z.B. durch Einführung von Minimalkoordinaten. Es ist deswegen zweckmäßig alle diese Gleichungen schon in die Basisklassen aufzunehmen, d.h. es werden jeweils separate Gleichungen für Position, Geschwindigkeit und Beschleunigung formuliert. Hierbei wird die Information, daß z.B. die Geschwindigkeit die Ableitung der Position ist, noch nicht mitgeteilt. Damit wird auch die Schwierigkeit vermieden, die sich bei einer Beschreibung eines Starrkörpers durch 3 Translations- und 3 Rotationskoordinaten ergibt: hier führt jede beliebige Wahl von Rotationskoordinaten (z.B. Eulerwinkel) in einer bestimmten Stellung zu einer Singularität.

Die Grundelemente in der Mechanik sind 3×1, 6×1 und 3×3 Matrizen. Zur Zeit werden von Dymola aber noch keine Matrizen unterstützt. In den hier entwickelten Mechanik-Bibliotheken werden deswegen alle Matrizenoperationen in expliziter Koordinatenschreibweise ausgeführt. Aus Gründen der Übersichtlichkeit wird im folgenden jedoch eine Matlab-ähnliche Matrizennotation benutzt.

Gleichungen in der Mechanik werden häufig in einer koordinateninvarianten Form angegeben. Dies ist zur Formulierung eines physikalischen Grundgesetzes auch zweckmäßig. Für eine konkrete Aufgabenstellung muß jedoch an irgendeiner Stelle festgelegt werden, in welchem Koordinatensystem mechanische Größen auszuwerten sind. Da mit den Mechanik-Klassen konkrete Aufgaben gelöst werden sollen, werden in den Klassenbeschreibungen Vektoren und Dyaden (= Tensoren 1. und 2. Stufe) immer bezüglich eines Koordinatensystems angegeben. Hierfür wird die folgende Konvention benutzt:

Die Koordinaten eines Vektors $\vec{h}^a$ bezüglich eines Koordinatensystems b werden in einer Spaltenmatrix ${}^b\mathbf{h}^a$ zusammengefaßt. Wenn $a = b$, so kann der Index b, der das Koordinatensystem kennzeichnet, auch weggelassen werden. Das heißt, $\mathbf{h}^a$ sind die Koordinaten eines Vektors $\vec{h}^a$ bezüglich des Koordinatensystems a. Der Index 0 bezeichnet immer das Inertialsystem.

In den Gleichungen wird der Operator **skew** und dessen inverser Operator **vec** benutzt. Wenn $\mathbf{h}$ eine 3×1 Spaltenmatrix und $\mathbf{H}$ eine schiefsymmetrische 3×3 Matrix ist, dann sind die beiden Operatoren folgendermaßen definiert:

$$\mathbf{H} = \mathbf{skew}(\mathbf{h}) = \mathbf{skew}(\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}) = \begin{bmatrix} 0 & -h_3 & h_2 \\ h_3 & 0 & -h_1 \\ -h_2 & h_1 & 0 \end{bmatrix} \quad ; \qquad \mathbf{h} = \mathbf{vec}(\mathbf{H}) \ .$$

Mit dem **skew** Operator kann z.B. das Kreuzprodukt zweier Vektoren $\mathbf{b}$ und $\mathbf{c}$ geschrieben werden als

$$\mathbf{a} = \mathbf{b} \times \mathbf{c} = \mathbf{skew}(\mathbf{b}) \cdot \mathbf{c} \quad .$$

Schließlich kennzeichnet die Matrix $\mathbf{E}$ eine Einheitsmatrix geeigneter Dimension.

4.2.1 Elementverbindungen

Jedes Mechanikelement hat zumindest eine oder zwei Schnittstellen, mit denen das Element mechanisch starr mit anderen Elementen verbunden werden kann. In Bild 4.3 werden die

verschiedenen Sichten einer solchen Schnittstelle dargestellt. Im linken Teil des Bilds ist

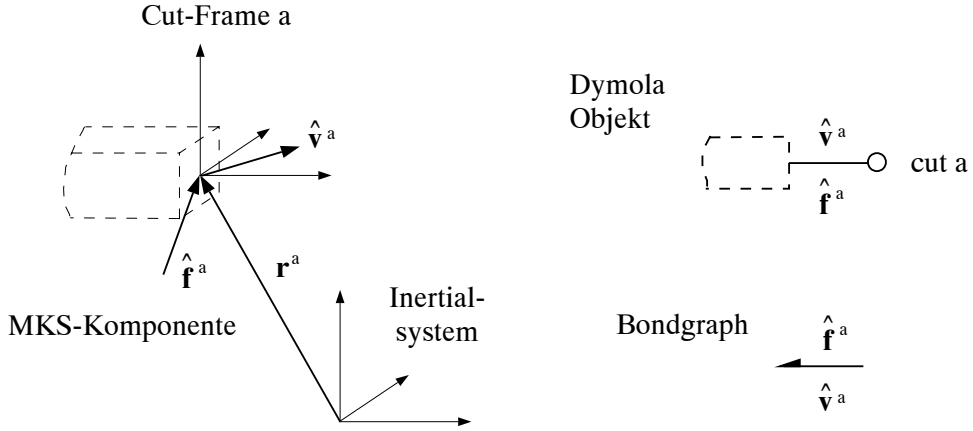


Bild 4.3: Definition eines "mechanischen" Cuts.

eine mechanische Schnittstelle mit dem Namen a zu sehen. In der Schnittebene befindet sich ein lokales Koordinatensystem, welches *fest* mit der Schnittebene verbunden ist und als Cut-Frame bezeichnet wird. Die Lage dieses Koordinatensystems wird automatisch dadurch definiert, daß es in der Referenzkonfiguration parallel zum Inertialsystem ist. Wenn zwei Objekte an einer solchen Schnittstelle zusammengeschaltet werden, ist damit auch sofort garantiert, daß die Koordinatensysteme der beiden Schnittstellen deckungsgleich sind.

Die (räumliche) Bewegung einer Schnittstelle a wird durch die Bewegung des zugeordneten Koordinatensystems beschrieben. Dessen Bewegung wird wiederum *eindeutig* durch eine 3×3 Drehmatrix ${}^0\mathbf{T}^a$, die einen Tensor vom Frame a in das Inertialsystem 0 transformiert, und durch den absoluten Ortsvektor $\mathbf{r}^a$ beschrieben, der vom Ursprung des Inertialsystems zum Ursprung von Cut-Frame a zeigt. Auf Geschwindigkeitsebene wird die Bewegung durch die absolute Winkelgeschwindigkeit $\boldsymbol{\omega}^a$ des Frames a und durch die absolute translatorische Geschwindigkeit $\mathbf{v}^a$ des Ursprungs dieses Frames beschrieben. Schließlich wird die Bewegung auf Beschleunigungsebene durch die zeitliche Ableitung der beiden Geschwindigkeitsvektoren, d.h. die absolute Winkelbeschleunigung $\boldsymbol{\alpha}^a$ und die absolute translatorische Beschleunigung $\mathbf{a}^a$, beschrieben. Die in der Schnittebene wirkenden Kräfte und Momente werden im Sinne des Schnittprinzips durch ein resultierendes Schnittmoment $\boldsymbol{\tau}^a$ und eine im Ursprung von Frame a angreifende resultierende Schnittkraft $\mathbf{f}^a$ beschrieben.

Zur Vereinfachung der Schreibweise werden die beiden Geschwindigkeiten, die beiden Beschleunigungen, das Schnittmoment und die Schnittkraft jeweils in einer 6×1 Spaltenmatrix zusammengefaßt:

$$\hat{\mathbf{v}}^a = \begin{bmatrix} \boldsymbol{\omega}^a \\ \mathbf{v}^a \end{bmatrix} ; \qquad \hat{\mathbf{a}}^a = \begin{bmatrix} \boldsymbol{\alpha}^a \\ \mathbf{a}^a \end{bmatrix} ; \qquad \hat{\mathbf{f}}^a = \begin{bmatrix} \boldsymbol{\tau}^a \\ \mathbf{f}^a \end{bmatrix} .$$

Der vollständige "Effekt" einer mechanischen Schnittstelle wird damit durch die Größen ${}^0\mathbf{T}^a, \mathbf{r}^a, \hat{\mathbf{v}}^a, \hat{\mathbf{a}}^a, \hat{\mathbf{f}}^a$ beschrieben. Wenn mehrere mechanische Elemente an einer Stelle miteinander verbunden werden, so stimmen die Cut-Frames an dieser Stelle überein. Da damit die Bewegung der Koordinatensysteme gleich ist, sind die kinematischen Größen ${}^0\mathbf{T}^a, \mathbf{r}^a, \hat{\mathbf{v}}^a, \hat{\mathbf{a}}^a$ *Across*-Variablen. Andererseits addieren sich die Schnittkräfte und Schnittmomente aufgrund des *actio=reactio* Prinzips zu Null auf. Die kinetischen Größen $\hat{\mathbf{f}}^a$ sind damit *Through*-Variablen. Wenn z.B. zwei Elemente a und b an einer Stelle verbunden

werden, so gilt $\hat{\mathbf{f}}^a + \hat{\mathbf{f}}^b = 0$, d.h. die Schnittkräfte und Schnittmomente sind an den beiden Schnittufern gleich groß, aber entgegengesetzt gerichtet.

Es gibt mehrere vernünftige Alternativen für die Koordinatensysteme, in denen die Größen eines mechanischen Cuts dargestellt werden können. Insbesondere ist dies das lokale Koordinatensystem des Cuts oder das globale Inertialsystem. Hier wird folgende Konvention benutzt: Der absolute Ortsvektor ${}^0\mathbf{r}^a$ vom Inertialsystem zum Ursprung eines Cuts wird im Inertialsystem angegeben, während alle anderen Cut-Größen im lokalen Koordinatensystem des Cuts dargestellt werden. Der Grund für diese Wahl ist der, daß der Ortsvektor vor allem bei kinematischen Analysen und bei der Berechnung von Animationsdaten benötigt wird. In solchen Aufgabenstellungen werden Vektoren normalerweise im Inertialsystem benötigt. Geschwindigkeiten, Beschleunigungen und Schnittkräfte treten dagegen in dynamischen Analyseproblemen auf. Die wichtigste Klasse dieser Anwendungen ist das Simulationsproblem, für das die Differentialgleichung des MKS benötigt wird. Eine detaillierte Analyse einer wichtigen Klasse von Mehrkörperalgorithmen in [Hill92] zeigt, daß eine Algorithmusformulierung in elementfesten Koordinatensystemen am effizientesten ist. Weiterhin ist es "natürlicher", Kräfte und Momente im entsprechenden Cut-Frame darzustellen, da die Richtungen der Kräfte und Momente mit geometrischen Eigenschaften des Elements assoziiert werden können. Natürlich kann bei Bedarf jeder beliebige Vektor auch in jedem beliebigen Koordinatensystem dargestellt werden, da die absolute Drehmatrix ${}^0\mathbf{T}^a$ zu jedem Cut-Frame vorliegt. Als Voreinstellung ist jedoch die oben erläuterte Wahl der Koordinatensysteme gewählt.

Formal ergeben sich die absolute Geschwindigkeit und die absolute Beschleunigung eines Cut-Frames a, beide dargestellt im Cut-Frame a, durch Differentiation des absoluten Ortsvektors und der absoluten Drehmatrix:

$$\hat{\mathbf{v}}^a = \begin{bmatrix} \mathbf{vec}\left({}^0\mathbf{T}^{a\,T}\,{}^0\dot{\mathbf{T}}^a\right) \\ {}^0\mathbf{T}^{a\,T}\,{}^0\dot{\mathbf{r}}^a \end{bmatrix} \; ; \qquad \hat{\mathbf{a}}^a = {}^0\hat{\mathbf{T}}^{a\,T}\,\frac{d}{dt}\,{}^0\hat{\mathbf{v}}^a \tag{4.1}$$

mit

$${}^0\hat{\mathbf{v}}^a = {}^0\hat{\mathbf{T}}^a\;\hat{\mathbf{v}}^a \; ; \qquad {}^0\hat{\mathbf{T}}^a = \begin{bmatrix} {}^0\mathbf{T}^a & \mathbf{0} \\ \mathbf{0} & {}^0\mathbf{T}^a \end{bmatrix} .$$

Diese Beziehungen zwischen Position, Geschwindigkeit und Beschleunigung werden aber aufgrund der "dummy derivative" Technik vorerst nicht benutzt.

Ähnlich wie es bei elektrischen Komponenten eine Oberklasse *TwoPin* gibt, um elektrische Elemente mit denselben Verbindungseigenschaften zu beschreiben, werden bei MKS die Oberklassen *MbsOneCut*, *MbsTwoCut* und *Interact* benutzt, um mechanische Elemente mit einer und mit zwei Schnittstellen zu definieren. Die Klasse *MbsOneCut* definiert Elemente, die *eine* mechanische Schnittstelle besitzen, und die mit den oben erläuterten Variablen der Schnittstelle mit anderen Objekten gekoppelt werden können. Die Klasse hat den folgenden Aufbau:

```
model class MbsOneCut
    main cut a ( Ta(3,3), ra0(3), va(6), aa(6) / fa(6) )
    terminal Pa

    Pa = fa' * va
end
```

Zum Beispiel wird durch $Ta(3,3)$ die 3×3 Matrix $Ta = {}^0\mathbf{T}^a$ definiert. Als zusätzliche, noch nicht besprochene Variable, ist im obigen Modell die Leistung P^a enthalten. P^a ist die *in* das Element fließende Energie. Diese kann z.B. benutzt werden um neu eingeführte Komponenten zu überprüfen, da die in ein Element fließende Energie gleich der Summe der abfließenden und der im Element gespeicherten Energie sein muß. Die Klasse *MbsTwoCut* definiert Elemente, die *zwei* mechanische Schnittstellen besitzen; sie ist entsprechend zur Klasse *MbsOneCut* aufgebaut:

```
model class MbsTwoCut
        cut   a   ( Ta(3,3), ra0(3), va(6), aa(6) / fa(6) )
        cut   b   ( Tb(3,3), rb0(3), vb(6), ab(6) / fb(6) )
    main cut   mc [a,b]
    main path  mp < a - b >

    terminal Pa, Pb

    Pa = fa' * va
    Pb = fb' * vb
end
```

Durch die "**main cut** ..." Definition kann die Verschaltung eines Elements c an zwei Knoten $n1, n2$ einfach mit "**connect** c **at** $(n1, n2)$" definiert werden. Dann wird der Cut *c:a* mit $n1$ und *c:b* mit $n2$ starr verbunden. Wie bei der Klasse *MbsOneCut* werden auch hier die über die beiden Cuts *einfließenden* Energieströme Pa und Pb berechnet.

Die Klasse *Interact* wird zur Modellierung masseloser mechanischer Elemente benutzt und ist äquivalent zur Klasse *TwoPin* für elektrische Bauteile. Durch die MKS-spezifischen Gegebenheiten ist der Aufbau der Klasse jedoch komplexer.

In der Klasse *TwoPin* für elektrische Elemente wird nicht nur die externe Verbindung definiert, sondern es werden auch die (sehr einfachen) Beziehungen zwischen den Across- und den Through-Variablen der beiden Cuts angegeben:

$$u = V_a - V_b; \qquad i = i_a = -i_b \ . \tag{4.2}$$

Zum einen wird also eine Relativgröße u eingeführt. Zum anderen besteht aufgrund einer "Kontinuitätsgleichung" (einfließender = ausfließender Strom) eine Beziehung zwischen den Through-Variablen der beiden Cuts. Die Relativgröße u wird benötigt, da bei allen Standardelementen (Widerstand, Kapazität, Spannungsquelle, etc.) die physikalischen Gesetze als Funktion der Relativgröße und eines Teils der Through-Variablen (i) angegeben werden. Es ist deswegen vorteilhaft, die Gleichungen (4.2) nicht in jeder Elementklasse zu wiederholen, sondern nur *einmal* in der Oberklasse *TwoPin* aufzuführen.

Bei MKS liegt dieselbe Situation vor. Die Relativgrößen, d.h. die Relativlage, die Relativgeschwindigkeit und die Relativbeschleunigung, werden in allen Standardelementen (Gelenke, Kraftelemente, Beobachtungselemente) zur Formulierung der entsprechenden physikalischen Gesetze benötigt. Weiterhin gibt es aufgrund einer Kräfte/Momente Bilanz eine Beziehung zwischen den Kräften und Momenten der beiden Cut-Frames, d.h. der Through-Variablen, welche wiederum unabhängig von einem spezifischen Element ist. Deswegen werden auch hier alle diese Gleichungen nur *einmal* in der Oberklasse *Interact* formuliert. Die Relativgrößen sind folgendermaßen definiert (siehe auch Bild 4.4):

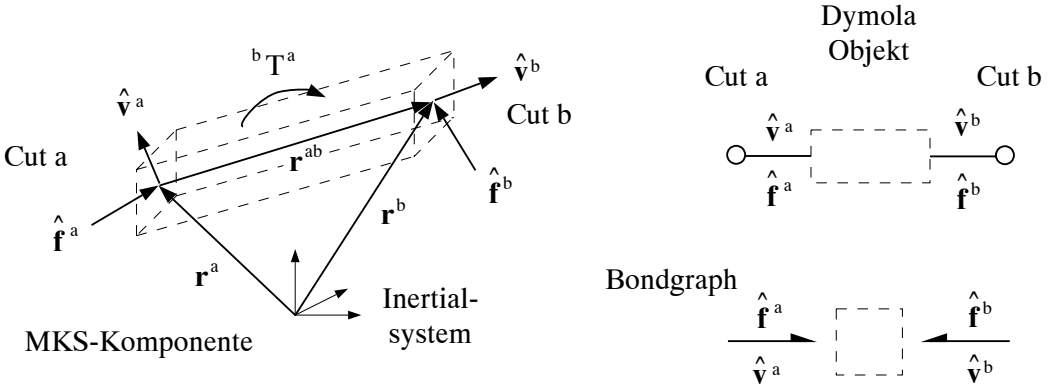


Bild 4.4: Ein allgemeines Element der Klasse *Interact*.

${}^b\mathbf{T}^a$ Relative Drehmatrix vom Cut-Frame a zum Cut-Frame b (= *Trel* in Klasse *Interact*).

${}^a\mathbf{r}^{ab}$ Relativer Ortsvektor vom Ursprung des Cut-Frames a zum Ursprung des Cut-Frames b, dargestellt im Cut-Frame a (= *rrela* in Klasse *Interact*).

${}^a\hat{\mathbf{v}}^{ab}$ Relative Winkelgeschwindigkeit und relative translatorische Geschwindigkeit, dargestellt im Cut-Frame a (= *vrela* in Klasse *Interact*).

$$ {}^a\hat{\mathbf{v}}^{ab} = \begin{bmatrix} {}^a\boldsymbol{\omega}^{ab} \\ {}^a\mathbf{v}^{ab} \end{bmatrix} = \begin{bmatrix} vec\left({}^b\dot{\mathbf{T}}^{a\,T}\, {}^b\mathbf{T}^a \right) \\ {}^a\dot{\mathbf{r}}^{ab} \end{bmatrix} \tag{4.3} $$

${}^a\hat{\mathbf{a}}^{ab}$ Relative Winkelbeschleunigung und relative translatorische Beschleunigung, dargestellt im Cut-Frame a (= *arela* in Klasse *TwoPin*).

$$ {}^a\hat{\mathbf{a}}^{ab} = \begin{bmatrix} {}^a\boldsymbol{\alpha}^{ab} \\ {}^a\mathbf{a}^{ab} \end{bmatrix} = \begin{bmatrix} {}^a\dot{\boldsymbol{\omega}}^{ab} \\ {}^a\dot{\mathbf{v}}^{ab} \end{bmatrix} \quad . \tag{4.4} $$

Die Relativgrößen werden sowohl im Cut-Frame a als auch im Cut-Frame b benötigt und werden folgendermaßen aus den absoluten Größen der beiden Cut-Frames bestimmt (die Herleitung dieser Gleichungen ist im Anhang A.3 zu finden):

$$ {}^b\mathbf{T}^a = {}^0\mathbf{T}^{b\,T}\, {}^0\mathbf{T}^a \tag{4.5a} $$

$$ {}^a\mathbf{r}^{ab} = {}^0\mathbf{T}^{a\,T}\left({}^0\mathbf{r}^b - {}^0\mathbf{r}^a \right) \tag{4.5b} $$

$$ {}^a\hat{\mathbf{v}}^{ab} = {}^b\hat{\mathbf{T}}^{a\,T}\, {}^b\hat{\mathbf{v}}^{ab} \tag{4.5c} $$

$$ {}^a\hat{\mathbf{a}}^{ab} = {}^b\hat{\mathbf{T}}^{a\,T}\, {}^b\hat{\mathbf{a}}^{ab} \tag{4.5d} $$

$$ {}^b\mathbf{r}^{ab} = {}^0\mathbf{T}^{b\,T}\left({}^0\mathbf{r}^b - {}^0\mathbf{r}^a \right) \tag{4.5e} $$

$$ {}^b\hat{\mathbf{v}}^{ab} = \hat{\mathbf{v}}^b - \mathbf{C}\hat{\mathbf{v}}^a \tag{4.5f} $$

$$ {}^b\hat{\mathbf{a}}^{ab} = \hat{\mathbf{a}}^b - \mathbf{C}\hat{\mathbf{a}}^a - \boldsymbol{\xi} \tag{4.5g} $$

$$ \mathbf{0} = \hat{\mathbf{f}}^a + \mathbf{C}^T\hat{\mathbf{f}}^b \tag{4.5h} $$

mit

$$ {}^b\hat{\mathbf{T}}^a = \begin{bmatrix} {}^b\mathbf{T}^a & \mathbf{0} \\ \mathbf{0} & {}^b\mathbf{T}^a \end{bmatrix} \tag{4.6a} $$

$$ \mathbf{C} = \begin{bmatrix} {}^b\mathbf{T}^a & \mathbf{0} \\ \mathbf{0} & {}^b\mathbf{T}^a \end{bmatrix} \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ -\mathbf{skew}({}^a\mathbf{r}^{ab}) & \mathbf{E} \end{bmatrix} \tag{4.6b} $$

$$ \boldsymbol{\xi} = \begin{bmatrix} {}^b\mathbf{T}^a & \mathbf{0} \\ \mathbf{0} & {}^b\mathbf{T}^a \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega}^a \times {}^a\boldsymbol{\omega}^{ab} \\ \boldsymbol{\omega}^a \times (\boldsymbol{\omega}^a \times {}^a\mathbf{r}^{ab} + 2\,{}^a\mathbf{v}^{ab}) \end{bmatrix} \quad . \tag{4.6c} $$

Die Gleichungen (4.5a – 4.5g) geben die Beziehungen zwischen den Relativgrößen des Elements und den Absolutgrößen der beiden Cut-Frames an. Zum Beispiel beschreibt (4.5f), daß sich die Relativ-Winkelgeschwindigkeit und Relativ-Geschwindigkeit ${}^b\hat{\mathbf{v}}^{ab}$ aus der Differenz der absoluten Geschwindigkeitsgrößen am Cut b und am Cut a ergeben. Hierbei werden die Größen des Cuts a mit einer Matrix multipliziert, in die der Abstand zwischen den beiden Cuts und die relative Drehmatrix eingehen. Gleichung (4.5h) ist eine Kräfte/Momente-Bilanz am Element unter der Voraussetzung, daß das Element masselos ist. In den obigen Gleichungen wird noch nicht festgelegt, welche Größen als bekannt und welche als unbekannt anzusehen sind. Mit (4.5) ergibt sich die Klasse *Interact* als Unter-Klasse von *MbsTwoCut* zu:

```
model class (MbsTwoCut) Interact
    cut Rel ( Trel(3,3), rrela(3), vrela(6), arela(6),
                         rrelb(3), vrelb(6), arelb(6) )
    local     C(6,6), xi(6)

    Trel  = Tb' * Ta
    rrela = Ta' * (rb0 - ra0)
    vrela = [ Trel' * vrelb(1:3)
              Trel' * vrelb(4:6) ]
    arela = [ Trel' * arelb(1:3)
              Trel' * arelb(4:6) ]

    rrelb = Tb' * (rb0 - ra0)
    vrelb = vb - C * va
    arelb = ab - C * aa - xi

    C     = [ Trel                  , zero(3,3)
              -Trel * skew(rrela)   , Trel      ]
    xi    = [ Trel * ( skew(wa) * wrela )
              Trel * ( skew(wa) * (skew(wa) * rrela + 2 * vrela) )
end
```

Der Operator **zero** erzeugt eine Nullmatrix und der Operator **skew** erzeugt eine schiefsymmetrische Matrix, wie weiter oben erläutert wurde. Man beachte, daß es hier nicht auf die Reihenfolge der angegebenen Gleichungen ankommt, da die Gleichungen bei der Transformation auf Blockdreiecksform sowieso umgeordnet werden.

Im rechten unteren Teil von Bild 4.3 auf Seite 72 ist die Bondgraphdarstellung eines mechanischen Cuts angegeben. Hier muß die Bondgraph Methode etwas erweitert werden, um den Besonderheiten einer DAE mit höherem Index gerecht zu werden. In einem Bondgraph werden über einen Bond nur die Effort- und Flow-Variablen übertragen, mit denen die Leistung gebildet wird. Bei höheren Index DAEs werden auch Integrale und Ableitungen dieser Variablen als selbständige Variablen benötigt ("dummy derivative" Methode). Aus diesem Grund wird jetzt zugelassen, daß solche Variablen auch über einen Bond transportiert werden können. Aus Gründen der Übersichtlichkeit werden diese zusätzlichen Variablen im Bondgraph aber nicht dargestellt. Am übertragenen Energiefluß ändert sich dadurch nichts. In Bild 4.3 sind am Bond deswegen nur die resultierende Schnittkraft und das resultierende Schnittmoment $\hat{\mathbf{f}}^a$, sowie die Geschwindigkeit und die Winkelgeschwindigkeit $\hat{\mathbf{v}}^a$ des Cut-Frames angetragen. Diese Größen werden so gewählt, daß eine positive Leistung einem Energiefluß *in* den Cut entspricht.

4.2.2 Kraftelemente

Kraftelemente, wie Federn, Dämpfer oder Motoren, sind mechanische Elemente mit zwei Cuts, die über diese Cuts ein eingeprägtes Moment und eine eingeprägte Kraft auf die beiden benachbarten mechanischen Elemente ausüben. Die Kraft und das Moment sind üblicherweise Funktionen der relativen kinematischen Größen zwischen den beiden Cuts. Die gemeinsamen Eigenschaften *aller* Kraftelemente werden in der Klasse *Force* definiert. Diese Klasse ist wiederum eine Unterklasse der Klasse *Interact*, da ein Kraftelement immer zumindest zwei mechanische Cuts besitzt. In Bild 4.5 ist eine schematische Darstellung eines allgemeinen Kraftelements, ein entsprechendes Dymola-Objekt und eine Bondgraph-Darstellung zu sehen.

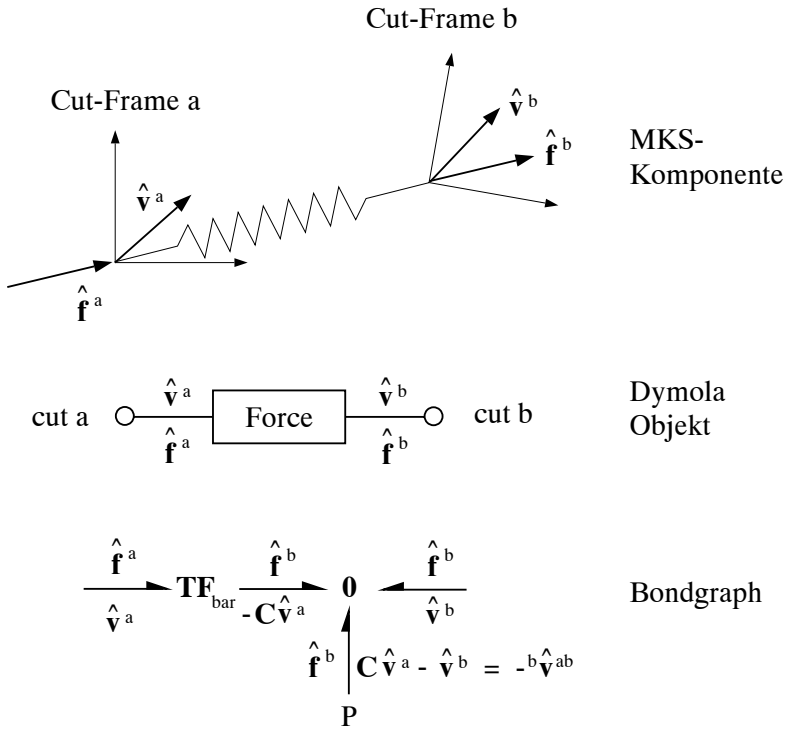


Bild 4.5: Ein allgemeines Kraftelement (Klasse *Force*).

Da die Klasse *Force* aus der Oberklasse *Interact* durch Vererbung abgeleitet wird, stehen alle in *Interact* definierten Beziehungen in einem Kraftelement zur Verfügung. In einem Kraftelement wird angenommen, daß die kinematischen Relativgrößen bekannt sind, und daß daraus das Schnittmoment und die Schnittkraft $\hat{\mathbf{f}}^b$ am Cut b als Funktion dieser Relativ-Größen berechnet werden. Die Schnittkraft $\hat{\mathbf{f}}^a$ ergibt sich dann aufgrund der schon in Klasse *Interact* aufgeführten Gleichung (4.5h)[3].

Fast alle gemeinsamen Eigenschaften von Kraftelementen sind schon in der Klasse *Interact* definiert worden. Es verbleibt nur noch die Berechnung der im Kraftelement erzeugten Leistung P, die z.B. auch von einer externen Quelle herrühren kann. Diese Leistung ergibt

[3]Natürlich werden auch hier wiederum nur Beziehungen zwischen Variablen definiert. Die angegebene Kausalität ist nur die am häufigsten benutzte. Bei einer Stationärwertberechnung sind aber z.B. die relativen Positionsgrößen sowie die Schnittkraft und das Schnittmoment gesucht, während die relativen Geschwindigkeits- und Beschleunigungsgrößen identisch Null, also bekannt, sind.

sich aus einer einfachen Bilanzgleichung unter Berücksichtigung der Gleichungen (4.5h, 4.5f):

$$\begin{aligned} 0 &= P + \hat{\mathbf{f}}^{a\,T}\,\hat{\mathbf{v}}^{a} + \hat{\mathbf{f}}^{b\,T}\,\hat{\mathbf{v}}^{b} \\ \Rightarrow P &= -\hat{\mathbf{f}}^{b\,T}\,\hat{\mathbf{v}}^{b} + (\hat{\mathbf{f}}^{b\,T}\,\mathbf{C})\hat{\mathbf{v}}^{a} \\ &= -\hat{\mathbf{f}}^{b\,T}\,(\hat{\mathbf{v}}^{b} - \mathbf{C}\hat{\mathbf{v}}^{a}) \\ &= -\hat{\mathbf{f}}^{b\,T}\,{}^{b}\hat{\mathbf{v}}^{ab}\ . \end{aligned} \tag{4.7}$$

Das heißt, die Leistung P ist gleich dem negativen Skalarprodukt aus der Schnittkraft und dem Schnittmoment am Cut b mit der relativen Geschwindigkeit und der relativen Winkelgeschwindigkeit. Damit ergibt sich die Klasse *Force* zu:

```
model class (Interact) Force
    terminal P

    P = -fb' * vrelb
end
```

In der Bondgraphdarstellung eines Kraftelements in Bild 4.5 werden zuerst die Größen des Cut-Frames a mittels eines *Transformers TF* auf den Cut-Frame b übertragen[4]. Dies bedeutet, daß die Wirkung von $\hat{\mathbf{f}}^{a}$ und $\hat{\mathbf{v}}^{a}$ auf den Cut-Frame b umgerechnet werden. Durch die *0-Junction*[5] werden die Relativgrößen berechnet. P ist die schon erläuterte Leistung, die entweder vom Kraftelement erzeugt wird oder von außen zufließt.

Es erweist sich als zweckmäßig, die zwei allgemeinen Unterklassen *DirForce* und *LineForce* der Klasse *Force* einzuführen. Mit der Klasse *DirForce* werden Kraftelemente beschrieben, die nur eine Kraft aber kein Moment erzeugen. Hierzu wird das im Cut-Frame b wirkende Moment auf Null gesetzt:

```
model class (Force) DirForce
    fb(1) = 0
    fb(2) = 0
    fb(3) = 0
end
```

Diese Klasse kann zum Beispiel benutzt werden, um die Windkraft auf ein Fahrzeug zu modellieren, indem noch ein Kraftgesetz für die drei Komponenten $fb(4), (5), (6)$ angegeben wird. Mit der Klasse *LineForce* werden Linienkräfte beschrieben, d.h. Kräfte, die immer auf der Verbindungslinie vom Ursprung des Cut-Frames a zum Ursprung des Cut-Frames b wirken. Deswegen hat eine Linienkraft nur noch eine Komponente f, die normalerweise vom relativen Abstand s und von der relativen Abstandsgeschwindigkeit $sd = \dot{s}$ der beiden Cut-Frames abhängt. Im Anhang A.4 werden die folgenden Beziehungen hergeleitet:

$$s = |{}^{b}\mathbf{r}^{ab}| \tag{4.8a}$$

$$\mathbf{n} = {}^{b}\mathbf{r}^{ab}/s \tag{4.8b}$$

$$\dot{s} = \mathbf{n}^{T}\,{}^{b}\mathbf{v}^{ab} \tag{4.8c}$$

$$\ddot{s} = \mathbf{n}^{T}\,{}^{b}\mathbf{a}^{ab} + ({}^{b}\mathbf{v}^{ab\,T}\,{}^{b}\mathbf{v}^{ab} - \dot{s}^{2})/s\ . \tag{4.8d}$$

[4] Der Transformer in der Mechanik wird in Abschnitt 4.2.3 ab Seite 80 besprochen.

[5] Einige Standardelemente von Bondgraphen, unter anderem die 0-Junction, wurden auf Seite 10 erläutert.

Hierbei ist **n** ein Einheitsvektor, der auf der Wirkungslinie der Kraft liegt. Die Klasse *LineForce* ergibt sich dann zu:

model class (*DirForce*) *LineForce*
  cut Force (s, sd, sdd / f)
  terminal $n(3)$

  $s = sqrt(rrelb(1) + rrelb(2) + rrelb(3))$
  $n = rrelb/s$
  $sd = n' * vrelb$
  $sdd = n' * arelb + (vrelb' * vrelb - sd * sd)/s$
  $fb(4) = f * n(1)$
  $fb(5) = f * n(2)$
  $fb(6) = f * n(3)$
end

Jetzt ist es sehr einfach ein spezielles Kraftelement einzuführen, da nur noch die Funktion $f = f(s, \dot{s})$ angegeben werden muß. Zum Beispiel wird eine Feder mit linearer Kennlinie durch die folgende Klasse beschrieben:

model class (*LineForce*) *Spring*
  parameter c, s0

  $f = c * (s - s0)$
end

Das Vorzeichen der Federkraft in der Klasse *Spring* kann leicht über eine Leistungsbilanz überprüft werden. Wenn die Feder gedehnt wird, muß Energie aufgewendet werden, die in das Federelement über die Cut-Frames einfließt und gespeichert wird. Die Leistung P zählt positiv, wenn das Kraftelement Energie an die Cut-Frames abgibt. P muß also beim Dehnen einer Feder negativ sein. Mit (4.7, 4.8c) ergibt sich:

$$\begin{aligned} P &= -\hat{\mathbf{f}}^{b\,T}\,{}^{b}\hat{\mathbf{v}}^{ab} \\ &= -\mathbf{f}^{b\,T}\,{}^{b}\mathbf{v}^{ab} \\ &= -(f\mathbf{n}^{T})\,{}^{b}\mathbf{v}^{ab} \\ &= -f\dot{s} \quad . \end{aligned}$$

Wenn zum Dehnen der Feder Energie zugeführt wird, so ist $s - s0 > 0$, und damit $f > 0$, sowie $\dot{s} > 0$. Die Leistung P ist demnach, wie gefordert, negativ und das Vorzeichen der Federkraft in der Klasse *Spring* ist korrekt.

Es zeigt sich, daß bei unterschiedlichen Elementen 1-dimensionale Kraftgesetze benötigt werden; zum Beispiel nicht nur als Spezialfall eines 3-dimensionalen Kraftelements, sondern auch bei Kräften in Gelenken, wobei die Kraft in Richtung der freien Bewegungsmöglichkeiten des Gelenks wirkt, oder bei Antriebssträngen. Um für alle diese Fälle nicht immer dasselbe Kraftgesetz in unterschiedlichen Varianten anbieten zu müssen, werden sogenannte *externe Kraftgesetze* eingeführt, die Unterklassen der folgenden Oberklasse sind:

```
model class ExtForce
    main cut ext (q, qd, qdd / f)
    terminal P
    P = f * qd
end
```

Ein externes Kraftgesetz hat nur einen Cut, über den die Lage q, die Geschwindigkeit $qd = \dot{q}$ und die Beschleunigung $qdd = \ddot{q}$ in "Richtung" der Kraftrichtung übergeben wird. Im Objekt wird mit diesen Variablen die Größe der Kraft f berechnet. Wie üblich werden die Vorzeichen der Terminalvariablen so gewählt, daß ein *einströmender* Energiefluß positiv zählt. Ein spezielles externes Kraftgesetz wird durch Vererbung von *ExtForce* abgeleitet. Zum Beispiel wird eine Feder folgendermaßen beschrieben:

```
model class (ExtForce) ExtSpring
    parameter c, s0

    f = c * (s - s0)
end
```

Um ein externes Kraftgesetz als Linienkraft zwischen zwei Körperpunkten eines 3-dimensional bewegten MKS angreifen zu lassen, wird die spezielle Klasse *ExtLineForce* angeboten. Diese ist entsprechend zur Klasse *LineForce* aufgebaut und hat nur noch einen zusätzlichen Cut, über den ein externes Kraftgesetz angeschlossen werden kann.

Neben der allgemeineren Verwendbarkeit hat ein externes Kraftgesetz noch den Vorteil, daß mehrere Kraftgesetze über dasselbe *ExtLineForce*-Objekt zwischen zwei Körperpunkten wirken können, wobei die Relativgrößen zwischen diesen beiden Punkten nur *einmal* berechnet werden. Würde man stattdessen unterschiedliche 3-dimensionale Kraftelemente zwischen den zwei Körperpunkten angreifen lassen, so würden die Relativgrößen von jedem Kraftelement erneut berechnet werden.

4.2.3 Ideale Gelenke

Ideale Gelenke, wie ein einachsiges Dreh- oder Schubgelenk oder ein Kugelgelenk, sind mechanische Elemente mit zwei Cuts, die die Relativbewegung der beiden Cuts zueinander einschränken. Es gibt zwei prinzipielle Möglichkeiten diese Eigenschaft zu beschreiben. Zum einen können zusätzliche Beziehungen zwischen den kinematischen Relativgrößen aufgestellt werden, zum anderen können die Relativgrößen als Funktion von *Minimalkoordinaten* des Gelenks definiert werden. Unter idealen Gelenken werden im weiteren Sinn alle masselosen Übertragungselemente mit zwei mechanischen Cuts verstanden, z.B. auch eine masselose Stange oder eine Koppelstange.

Die gemeinsamen Eigenschaften *aller* Gelenke werden in der Klasse *Joint* definiert. Diese Klasse ist wiederum eine Unterklasse der Klasse *Interact*, da ein Gelenk immer zumindest zwei mechanische Cuts besitzt. In Bild 4.6 sind eine schematische Darstellung eines allgemeinen Gelenks, ein entsprechendes Dymola-Objekt und eine Bondgraph-Darstellung angegeben.

Zuerst wird die auf Minimalkoordinaten basierende Beschreibung von Gelenken diskutiert. Danach werden Gelenke erläutert, die durch zusätzliche Zwangsbedingungen definiert wer-

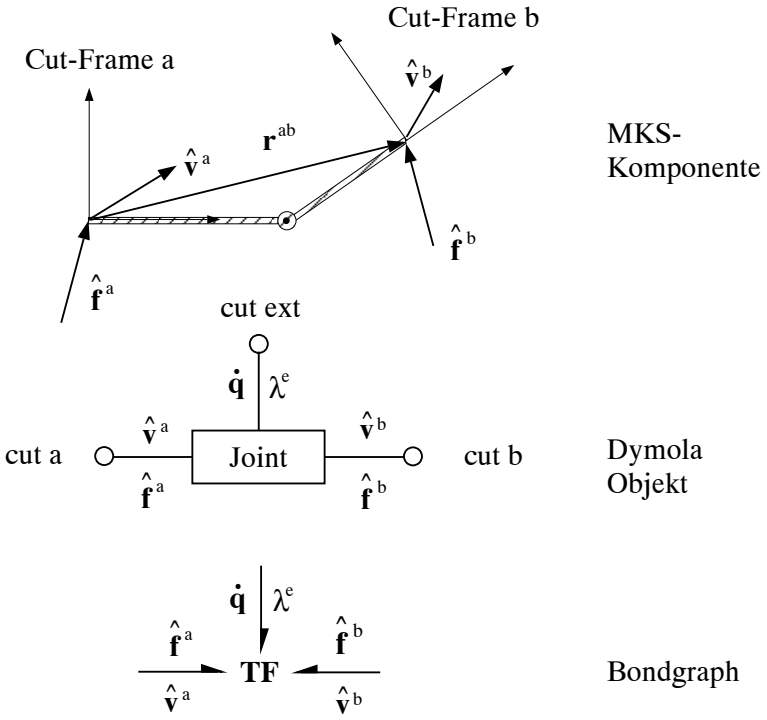


Bild 4.6: Ein allgemeines Gelenk (Klasse *Joint*).

den. Beide hier verwendete Darstellungen beruhen auf der allgemeinen Gelenkbeschreibung von Roberson und Schwertassek [Robe88].

Da die Klasse *Joint* von der Oberklasse *Interact* durch Vererbung abgeleitet wird, stehen alle in *Interact* definierten Beziehungen in einem Gelenk zur Verfügung. Es wird jetzt angenommen, daß die Relativgrößen nicht mehr unabhängig voneinander sind, sondern eindeutig als Funktion der f verallgemeinerten Minimalkoordinaten $\mathbf{q} = [q_1, q_2, \ldots, q_f]$ angegeben werden können. Hierbei ist f die Zahl der Freiheitsgrade (degrees of $f$reedom) des Gelenks, wobei f im Bereich $0 \leq f \leq 6$ liegen kann. Sowohl die relative Drehmatrix als auch der relative Ortsvektor zwischen den beiden Cut-Frames sind Funktionen von $\mathbf{q}(t)$. Die relative Geschwindigkeit und die relative Beschleunigung ergeben sich dann durch Differentiation dieser Beziehungen mit den Definitionsgleichungen (4.3, 4.4, 4.5c, 4.5d). Ein Gelenk ist damit durch Angabe der folgenden Funktionen für die Relativgrößen eindeutig charakterisiert:

$$
\begin{aligned}
{}^{b}\mathbf{T}^{a} &= {}^{b}\mathbf{T}^{a}(\mathbf{q}) && \text{(4.9a)}\\
{}^{b}\mathbf{r}^{ab} &= {}^{b}\mathbf{r}^{ab}(\mathbf{q}) && \text{(4.9b)}\\
{}^{b}\hat{\mathbf{v}}^{ab} &= \boldsymbol{\Phi}(\mathbf{q})\dot{\mathbf{q}} && \text{(4.9c)}\\
{}^{b}\hat{\mathbf{a}}^{ab} &= \boldsymbol{\Phi}(\mathbf{q})\ddot{\mathbf{q}} + \boldsymbol{\zeta}(\mathbf{q},\dot{\mathbf{q}}) \ . && \text{(4.9d)}
\end{aligned}
$$

Oft kommt es vor, daß zwischen den beiden Cut-Frames eines Gelenks ein Kraftelement vorhanden ist, z.B. der Antrieb eines Drehgelenks bei einem Roboter oder ein Dämpferelement in einem Gelenk. Würden Kraftelement und Gelenk unabhängig voneinander behandelt werden, müßten im Kraftelement die Relativgrößen zwischen den beiden Cut-Frames noch einmal berechnet werden, obwohl diese vom Gelenk schon zur Verfügung gestellt werden. Deswegen ist es zweckmäßig, bei der Beschreibung eines Gelenks schon ein allgemeines

Kraftelement vorzusehen, das zwischen den beiden Cut-Frames des Gelenks wirkt, wie im Bondgraph von Bild 4.7 gezeigt. Die Summe der Kräfte und Momente an den 1-Junctions,

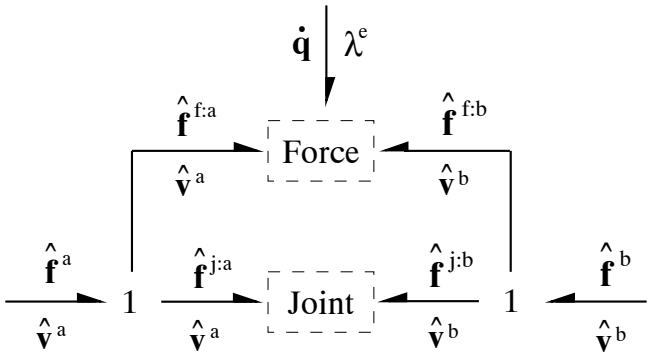


Bild 4.7: Gelenk mit zugehörigem Kraftelement.

d.h. an den beiden Schnittstellen des Gelenks, muß Null sein. Durch die Halbpfeile sind die positiven Richtungen der Energieflüsse gekennzeichnet. Da an einer Schnittstelle des Gelenks (= 1-Junctions) die kinematischen Größen für alle Elemente am Schnitt denselben Wert besitzen, charakterisieren die Halbpfeile auch die "positiven" Richtungen der Kräfte und Momente. Mit Gleichung (4.5h) ergibt sich:

$$\begin{aligned} \hat{\mathbf{f}}^a &= \hat{\mathbf{f}}^{f:a} + \hat{\mathbf{f}}^{j:a} \\ &= -\mathbf{C}^T \hat{\mathbf{f}}^{f:b} - \mathbf{C}^T \hat{\mathbf{f}}^{j:b} \\ &= -\mathbf{C}^T \hat{\mathbf{f}}^{f:b} - \mathbf{C}^T (\hat{\mathbf{f}}^b - \hat{\mathbf{f}}^{f:b}) \\ &= -\mathbf{C}^T \hat{\mathbf{f}}^b \ . \end{aligned} \tag{4.10}$$

Gleichung (4.10) ist identisch mit Gleichung (4.5h). D.h. man kann das Gelenk und das Kraftelement als *ein* Element der Klasse *Interact* ansehen, da die Kräftebilanz der Klasse *Interact* auch für das kombinierte Gelenk/Kraftelement Gültigkeit hat.

Technisch macht es nur Sinn die Kraft und das Moment des Kraftelements so angreifen zu lassen, daß diese in "Richtung" der freien Bewegungsrichtung des Gelenks wirken. In den anderen "Richtungen" gibt es keine Relativgeschwindigkeit, so daß keine Leistung übertragen werden kann. Aus diesem Grund wird angenommen, daß die Kraftelemente in Gelenken durch eine verallgemeinerte Kraft $\boldsymbol{\lambda}^e$ (der Index e steht für "eingeprägte" Kraft) beschrieben werden, so daß die vom Kraftelement auf das Gelenk übertragene Leistung $P = \boldsymbol{\lambda}^{e\,T} \dot{\mathbf{q}}$ ist. Normalerweise gilt $\boldsymbol{\lambda}^e = \boldsymbol{\lambda}^e(\mathbf{q}, \dot{\mathbf{q}})$. Zum Beispiel wirkt bei einem einachsigen Drehgelenk das antreibende Moment $\boldsymbol{\lambda}^e$ in Richtung der Drehachse, oder bei einem einachsigen Schubgelenk wirkt die antreibende Kraft $\boldsymbol{\lambda}^e$ in Richtung der Schubachse.

In Bild 4.6 auf Seite 81 ist das kombinierte Gelenk/Kraftelement dargestellt. Der zusätzliche Cut *ext* charakterisiert den Anschluß, über den die Energie des Kraftelements in das Gelenk einfließt. Wenn ein Gelenk genau einen Freiheitsgrad besitzt, kann an den Cut *ext* ein externes Kraftgesetz-Objekt angeschlossen werden. Externe Kraftgesetze wurden auf Seite 80 besprochen.

Durch die eingeschränkte Relativbewegung in einem Gelenk werden auch die möglichen Schnittkräfte und Schnittmomente der beiden Cut-Frames eingeschränkt. Die entsprechenden Gleichungen ergeben sich aus einer Leistungsbilanz, wobei die Summe der ein- und

ausfließenden Energieströme Null sein muß, da in einem idealen Gelenk keine Energie gespeichert wird. Mit (4.10, 4.5f, 4.9c) ergibt sich:

$$\begin{aligned}\sum P_i \;=\; 0 \;&=\; \hat{\mathbf{f}}^{a\,T}\,\hat{\mathbf{v}}^a \;+\; \hat{\mathbf{f}}^{b\,T}\,\hat{\mathbf{v}}^b \;+\; \boldsymbol{\lambda}^{e\,T}\,\dot{\mathbf{q}} \\ &=\; (-\hat{\mathbf{f}}^{b\,T}\,\mathbf{C})\hat{\mathbf{v}}^a \;+\; \hat{\mathbf{f}}^{b\,T}(\mathbf{C}\hat{\mathbf{v}}^a + \boldsymbol{\Phi}\dot{\mathbf{q}}) \;+\; \boldsymbol{\lambda}^{e\,T}\,\dot{\mathbf{q}} \\ &=\; (\boldsymbol{\lambda}^e + \boldsymbol{\Phi}^T\,\hat{\mathbf{f}}^b)^T\,\dot{\mathbf{q}} \;\;. \end{aligned} \tag{4.11}$$

Da $\mathbf{q}$ die Minimalkoordinaten des Gelenks sind, kann $\dot{\mathbf{q}}$ jeden beliebigen Wert annehmen, ohne dabei die Bewegungseinschränkungen des Gelenks zu verletzen. Da darüberhinaus Gleichung (4.11) immer gültig sein muß, muß der Klammerausdruck identisch verschwinden, d.h. es muß gelten

$$\mathbf{0} = \boldsymbol{\lambda}^e + \boldsymbol{\Phi}^T\,\hat{\mathbf{f}}^b \;\;. \tag{4.12}$$

Gleichung (4.12) entspricht dem d'Alembertschen Prinzip, das über eine lokale Leistungsbilanz abgeleitet wurde. Normalerweise wird zur Herleitung der Gleichung (4.12) das Prinzip der virtuellen Arbeit benutzt. Dies ist hier nicht notwendig, da die Gelenkgrößen *nicht explizit von der Zeit* abhängen. In diesem Fall sind die virtuellen Verschiebungen identisch mit den absoluten Zeitableitungen und damit gleich den möglichen (tatsächlich auftretenden) Geschwindigkeiten.

Die Gelenkbeschreibung ist trotzdem recht allgemein und enthält rheonome Gelenke, da nicht festgelegt ist was bekannte und was unbekannte Größen sind. Zum Beispiel können bei einem Drehgelenk der Drehwinkel und seine Ableitungen als Funktionen der Zeit vorgegeben werden. Die verallgemeinerte Kraft $\boldsymbol{\lambda}^e$ ist dann keine eingeprägte Kraft mehr, sondern ergibt sich aufgrund der Bewegung des Systems. Der Energiestrom $\boldsymbol{\lambda}^{e\,T}\,\dot{\mathbf{q}}$, der über den Cut *ext* einfließt, ist die Leistung, die benötigt wird, um die gewünschte Bewegung $\mathbf{q}(t)$ aufrecht zu erhalten[6].

Zur Verdeutlichung werde ein einachsiges Drehgelenk betrachtet. Wenn $\mathbf{n}$ die Drehachse des Drehgelenks ist, dann ist $\boldsymbol{\Phi} = [\mathbf{n}; \mathbf{0}]$. Gleichung (4.12) besagt in diesem Fall, daß die Projektion des Schnittmoments $\boldsymbol{\tau}$ auf die Drehachse des Gelenks entgegengesetzt gleich dem angreifenden, eingeprägten Moment λ^e im Gelenk ist, d.h. $\mathbf{n}^T\boldsymbol{\tau} = -\lambda^e$.

Es kommt vor, daß sich während einer Simulation die Zahl der Freiheitsgrade eines Gelenks ändert, weil z.B. die Reibung im Gelenk zum "Haften" führt, weil eine Kupplung gelöst wird, oder weil durch eine Bremse Freiheitsgrade blockiert werden. Diese Effekte können mit den in Kapitel 3 besprochenen Methoden beschrieben werden und werden in Kapitel 4.5 ab Seite 114 genauer behandelt. Die Gleichung, die das Blockieren der Freiheitsgrade eines Gelenks beschreibt, wird jedoch schon hier erläutert. Es wird dabei wie beim elektrischen Schalter auf Seite 54 vorgegangen.

Durch eine Boole'sche Variable *Locked* wird der Schaltzustand des Gelenks festgelegt. Wenn $Locked = True$ gilt, dann ist der entsprechende Freiheitsgrad des Gelenks blockiert, d.h. $q_i = const.$; bei $Locked = False$ ist der Freiheitsgrad aktiv. Beim Blockieren eines Gelenks wird eine neue Zwangskraft $\boldsymbol{\lambda}^l$ eingeführt, die in "Richtung" der gesperrten Bewegungsrichtung wirkt und die so bestimmt wird, daß die Bedingung $q_i = const.$ erfüllt ist. Verglichen

[6] Man sieht, daß das Prinzip der *virtuellen* Arbeit bzw. der *virtuellen* Leistung nur deswegen notwendig ist, weil ausschließlich die Leistung der *eingeprägten* Kräfte berücksichtigt wird. Sobald *alle* Leistungsanteile berücksichtigt werden, d.h. auch die Leistungen der *Zwangskräfte*, die in Richtung einer *vorgegebenen* Bewegung wirken, kann die *virtuelle* Leistung durch die *wirkliche* Leistung ersetzt werden.

mit dem elektrischen Schalter ist das eine etwas andere Situation, da der elektrische Schalter zwischen *zwei unterschiedlichen Gleichungen* schaltet. Dagegen wird beim Gelenk *eine* Gleichung und *eine* neue Variable hinzugefügt. Sowohl die Gleichung als auch die Variable verschwinden, wenn das Gelenk nicht mehr blockiert ist. Da Variablen in einer Simulationsumgebung nicht einfach entfernt werden können, wird stattdessen eine Dummy-Gleichung hinzugefügt, die der Zwangskraft einen eindeutigen (jedoch willkürlichen) Wert zuordnet, z.B. Null. Damit wird das Blockieren durch die Gleichung

$$0 = \textbf{if } Locked_i \textbf{ then } q_i - q0_i \textbf{ else } \lambda_i^l \tag{4.13}$$

beschrieben. Bei Gelenken liegt nun aber immer der in Kapitel 3.3 beschriebene, unangenehme Fall vor, der auch auftritt, wenn ein elektrischer Schalter parallel zu einer Kapazität oder in Reihe mit einer Induktivität geschaltet ist. Wenn zu einem beliebigen Zeitpunkt ein Gelenk blockiert wird, ist die Relativgeschwindigkeit in der Regel nicht Null. Nach dem Blockieren ist sie aber identisch Null. Dieser Effekt kann nur erreicht werden, wenn eine unendlich große Kraft, d.h. ein Kraftstoß, auftritt.

Es wird nun angenommen, daß das Schalten so erfolgt, daß entweder der Kraftstoß korrekt beschrieben wird, oder daß die Relativgeschwindigkeit beim Blockieren Null ist. Genau wie beim elektrischen Schalter müssen beide Variablen in der obigen Gleichung immer *unbekannt* sein, so daß diese Gleichung in einem algebraischen Gleichungssystem auftreten muß. Dies ist bei MKS jedoch nicht der Fall. Einige Positionsgrößen sind immer Zustandsgrößen. Die anderen Positionsgrößen werden dann eindeutig als Funktionen dieser Zustandsgrößen ausgerechnet. Deswegen ist in der obigen Gleichung q_i immer bekannt und die Gleichung wird immer nach λ_i^l aufgelöst. Wie beim Schalter führt dies zu einer Division durch Null, wenn das Gelenk blockiert. Anders ausgedrückt: Durch das Einführen der obigen Gleichung ändert sich der Störungsindex des Systems, da eine neue, zusätzliche Bedingung an die Zustandsgrößen des Systems gestellt wird, wodurch die Zahl der Zustandsgrößen verringert wird.

In Kapitel 2.4 ab Seite 31 wurde gezeigt, wie Systeme mit einem (konstant bleibenden) Störungsindex > 1 behandelt werden können. Alle dort aufgeführten Schwierigkeiten treffen natürlich in erhöhtem Maße auf eine DAE zu, deren Index sich während der Integration verändert. Der Index des Systems kann reduziert werden, wenn die Zwangsbedingung (4.13) zweimal differenziert wird, also die Gleichung

$$0 = \textbf{if } Locked_i \textbf{ then } \ddot{q}_i \textbf{ else } \lambda_i^l \tag{4.14}$$

benutzt wird, da (4.14) keine algebraische Beziehung zwischen Zustandsgrößen mehr darstellt. Wie in Kapitel 2.4 erläutert, erfordert eine korrekte Behandlung einer DAE mit höherem Index die gleichzeitige Berücksichtigung der Ausgangsgleichungen, sowie deren Ableitungen bis zur Ordnung $j-1$, wobei j der Störungsindex des Systems ist. Im obigen Fall müssen also die Gleichungen (4.13), sowie deren erste und zweite Ableitung während der Integration berücksichtigt werden.

In vielen Fällen, z.B. bei Haftreibung, bleibt ein Gelenk nur eine begrenzte Zeit im gesperrten Zustand. Dann genügt es während der Integration nur Gleichung (4.14) zu berücksichtigen, da das Wegdriften der Lösung von (4.13) und seiner ersten zeitlichen Ableitung "klein" bleibt, wenn die Blockierzeit genügend klein ist. Eine "saubere" Lösung ist möglich, wenn die gesperrte Gelenkkoordinate im ungesperrten Zustand eine Zustandsgröße ist, was z.B.

bei MKS in Baumstruktur immer erreicht werden kann. In diesem Fall kann im gesperrten Gelenkzustand eine Hilfsdifferentialgleichung eingeführt werden:

$$
\begin{aligned}
&\textbf{local } w,\ wd \\
&wd = \textbf{der}(w) \\
&qdd = \textbf{der}(wd) \\
&q \quad = \textbf{if } Locked \textbf{ then } q0 \quad \textbf{else } w \\
&qd \quad = \textbf{if } Locked \textbf{ then } 0 \quad \textbf{else } wd \\
&0 \quad = \textbf{if } Locked \textbf{ then } qdd \textbf{ else } \lambda_i^l \quad .
\end{aligned}
$$

In den mechanischen Gleichungen werden in jedem Schaltzustand die Größen q, qd $(= \dot{q})$, qdd $(= \ddot{q})$ verwendet. Auf Grund der obigen Gleichungen werden damit (4.13) und seine ersten beiden Ableitungen korrekt berücksichtigt. Im blockierten Zustand werden q und qd jedoch nicht mehr als Zustandsgrößen benutzt. Stattdessen werden die Hilfs-Zustandsgrößen w, wd $(= \dot{w})$ eingeführt, die mit der (nicht interessierenden) Dummy-Differentialgleichung $\dot{w} = wd$, $\dot{wd} = 0$ berechnet werden. Wie oben schon erwähnt, soll im folgenden jedoch angenommen werden, daß nur Gleichung (4.14) bei der Integration verwendet wird.

Mit (4.14, 4.12) wird ein allgemeines, blockierbares Gelenk durch die folgenden Gleichungen beschrieben:

$$\mathbf{0} = \boldsymbol{\lambda}^e + \mathbf{L}\boldsymbol{\lambda}^l + \boldsymbol{\Phi}^T \hat{\mathbf{f}}^b \tag{4.15a}$$

$$\mathbf{0} = \mathbf{L}\ddot{\mathbf{q}} + (\mathbf{E} - \mathbf{L})\boldsymbol{\lambda}^l \tag{4.15b}$$

wobei

$$\mathbf{L} = diag(L_{ii}); \quad L_{ii} = \textbf{if } Locked_i \textbf{ then } 1 \textbf{ else } 0 \ . \tag{4.16}$$

An dieser Stelle enthält die gemeinsame Oberklasse *Joint* aller Gelenke keine zusätzlichen Beziehungen, sondern dient nur als Platzhalter[7]:

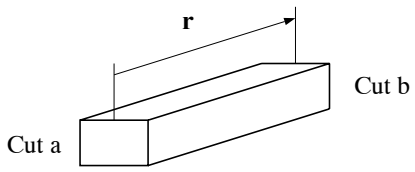
```
model class (Interact) Joint
      {Common superclass of all joints}
end
```

Ein Gelenk wird immer durch eine Unterklasse von *Joint* beschrieben. In der Gelenkklasse werden die Funktionen (4.9, 4.15) definiert.

Im folgenden werden die wichtigsten Gelenke besprochen. Ein einfacher Sonderfall stellt eine masselose Stange dar, die mit der Klasse *Bar* beschrieben wird. Dieses Element wird benutzt um Punkte auf einem Körper zu definieren, an denen Kraftelemente oder andere Gelenke angebracht werden (siehe dazu auch das Doppelpendel-Einführungsbeispiel auf Seite 67). Bei einer Stange sind die Relativgeschwindigkeit und die Relativbeschleunigung des Cut-Frames b relativ zum Cut-Frame a immer Null. Weiterhin sind die beiden Cut-Frames immer parallel zueinander, so daß die relative Drehmatrix gleich der Einheitsmatrix ist. Die einzige nichttriviale Größe ist der relative Abstand der beiden Cut-Frames, der immer konstant bleibt. Der Abstand wird durch den konstanten Ortsvektor vom Ursprung des Cut-Frames a zum Ursprung des Cut-Frames b beschrieben. Seine Koordinaten werden als Parameter vorgegeben. Damit hat die Klasse *Bar* den folgenden Aufbau:

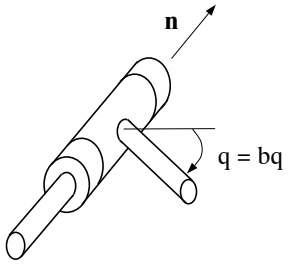
[7] Wie später erläutert wird, werden aus Effizienzgründen einige Berechnungen der Klasse *Interact* in der Klasse *Joint* vorgenommen.

```
model class (Joint) Bar
    parameter r1 = 0, r2 = 0, r3 = 0

    Trel  = eye(3,3)
    rrelb = [r1; r2; r3]
    vrelb = zero(6)
    arelb = zero(6)
end
```



Die beiden wichtigsten Gelenkklassen sind das einachsige Drehgelenk und das einachsige Schubgelenk, da viele andere Gelenke einfach aus einer Kombination dieser beiden Grundtypen aufgebaut werden können. Stellvertretend wird das Drehgelenk im Detail besprochen: Die Grundklasse *RevoluteBase* enthält alle benötigten Gleichungen des Drehgelenks. Aus Komfortgründen gibt es Spezialisierungen dieser Klasse.

```
model class (Joint) RevoluteBase
    local     bq, bqd, bqdd
    terminal n(3), frel, P

    P    = frel * bqd
    Trel = n * n' + (eye(3,3) - n * n') * cos(bq)
           -skew(n) * sin(bq)
    rrelb = zero(3)
    vrelb = [n * bqd; 0; 0; 0]
    arelb = [n * bqdd; 0; 0; 0]
    0    = frel + n' * fb(1:3)
end
```



Die Variable *bq* ist der Drehwinkel und die Variablen *bqd*, *bqdd* sind seine erste und zweite Ableitung. Die Größe n ist ein Vektor in Richtung der Drehachse des Gelenks. Diese wird nicht als Parameter, d.h. als Konstante, deklariert, da Dymola zur Zeit keine Operationen auf Parameter erlaubt. Wenn ein neues Gelenk definiert werden soll, das ein Drehgelenk enthält, so muß die Drehachse in der Regel aufgrund anderer Gegebenheiten *berechnet* werden. Ein Beispiel dafür ist weiter unten angegeben. Man beachte, daß die Drehachse n im Cut-Frame a dieselben Koordinaten hat wie im Cut-Frame b. Die Terminal Variable $frel$ ist ein Drehmoment des Gelenks, das in Richtung der Drehachse wirkt. Normalerweise ist dies ein eingeprägtes Moment. Wenn ein Gelenk gesperrt wird, kann es jedoch auch ein Zwangsmoment sein. Schließlich ist die Variable P der Energiefluß, welcher über das Drehmoment $frel$ in das Gelenk einfließt.

Die relative Drehmatrix $Trel = {}^b\mathbf{T}^a$ wird nach einer bekannten Formel aus Drehachse und Drehwinkel berechnet, siehe z.B. [Robe88]. Da alle Koordinatensysteme in der Referenzkonfiguration parallel zueinander sind, ist der Drehwinkel *bq* in dieser Lage Null. Der relative Ortsvektor $rrel = {}^b\mathbf{r}^{ab}$ ist Null, da sowohl der Ursprung des Cut-Frames a, als auch der Ursprung des Cut-Frames b auf der Drehachse liegen. Die relative Geschwindigkeit und die relative Beschleunigung ergeben sich sehr einfach mit Hilfe der Drehachse $\mathbf{n}$. Schließlich wird in der letzten Anweisung die Gleichung (4.12) aufgeführt.

Die gemeinsamen Eigenschaften aller Drehgelenk-Varianten werden in einer zweiten Hilfsklasse zusammengefaßt, die aus *RevoluteBase* durch Vererbung abgeleitet wird:

```
model class (RevoluteBase) RevoluteBase2
    parameter n1 = 0, n2 = 0, n3 = 0
    local     absn

    absn = sqrt(n1*n1 + n2*n2 + n3*n3)
    n(1) = n1/absn
    n(2) = n2/absn
    n(3) = n3/absn
end
```

In allen Drehgelenk-Varianten werden die Koordinaten $n1$, $n2$, $n3$ eines Vektors als Parameter vorgegeben, der in Richtung der Drehachse des Drehgelenks zeigt. Dieser Vektor wird normiert und dem in Klasse *RevoluteBase* definierten Vektor **n** zugewiesen. Es gibt 4 abgeleitete Klassen für verschiedene Varianten des Drehgelenks:

Revolute	Einfaches Drehgelenk.
RevoluteL	Blockierbares Drehgelenk.
RevoluteS	q, qd sind Zustandsvariablen, ansonsten wie *Revolute*.
RevoluteLS	q, qd sind Zustandsvariablen, ansonsten wie *RevoluteL*.

Diese Klassen sind wie folgt definiert:

```
model class (RevoluteBase2) Revolute
    cut ext (q, qd, qdd / f)

    bq    = q
    bqd   = qd
    bqdd  = qdd
    frel  = f
end

model class (Revolute) RevoluteS
    qd  = der(q)
    qdd = der(qd)
end

model class (RevoluteBase2) RevoluteL
    cut ext  (q, qd, qdd / f)
    cut extL (q, qd, qdd, Locked / fL, fLc)

    bq    = q
    bqd   = qd
    bqdd  = qdd
    frel  = f + fL + (if Locked then fLc else 0)
    0     = if Locked then qdd else fLc
end

model class (RevoluteL) RevoluteLS
    qd  = der(q)
    qdd = der(qd)
end
```

Über den neuen Cut *ext* kann ein externes Kraftgesetz an das Drehgelenk angeschlossen werden, welches das Gelenk über ein eingeprägtes Moment f antreibt. Man beachte, daß f eine Through-Variable ist. Die Variablen q, qd, qdd sind der Drehwinkel sowie seine erste und zweite Ableitung. Externe Kraftgesetze wurden auf Seite 80 besprochen. In den Klassen *RevoluteL* und *RevoluteLS* kann über den weiteren Cut *extL* ein externes Kraftgesetz angeschlossen werden, mit welchem das Gelenk *blockiert* werden kann. Externe Kraftgesetze mit dieser Eigenschaft müssen von der Klasse *ExtForceL* durch Vererbung abgeleitet werden:

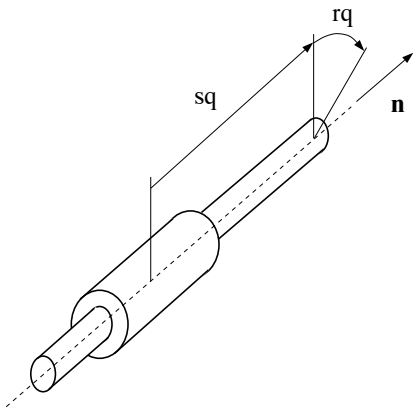
```
model class ExtForceL
    main cut ext (q, qd, qdd, Locked / fL, fLc)
    terminal f, P
    f = fL + fLc
    P = f * qd
end
```

Wie in der Klasse *RevoluteL* ist fL die im Gelenk wirkende (verallgemeinerte) eingeprägte Kraft $\boldsymbol{\lambda}^e$ und fLc die im Gelenk wirkende (verallgemeinerte) Zwangskraft $\boldsymbol{\lambda}^l$. Mit der Boole'schen Variablen *Locked* kann dasjenige Gelenk blockiert werden, an das das externe Kraftgesetz angeschlossen ist. In der Klasse *RevoluteL* wird die Blockiereigenschaft in der letzten Gleichung, entsprechend zu (4.15), beschrieben.

Die Klassen *RevoluteS* und *RevoluteLS* werden aus den Klassen *Revolute* und *RevoluteL* durch Vererbung abgeleitet. Durch Angabe der Beziehungen, daß qd die Ableitung von q und qdd die Ableitung von qd ist, legen die eingebauten Regeln von Dymola implizit fest, daß q, qd Zustandsvariablen sind. Die Auswahl einer geeigneten Variante richtet sich nach der Anwendung. Zum Beispiel wird die Klasse *RevoluteLS* verwendet, wenn ein Drehgelenk blockierbar sein soll und wenn der Drehwinkel und dessen erste Ableitung als Zustandsgrößen zu benutzen sind. Oft wird ein Drehgelenk benötigt, bei dem kein Drehmoment im Gelenk angreifen soll. Man muß dann nicht unbedingt das Drehmoment f des Cuts *ext* explizit auf Null setzen, da es in Dymola eine Option gibt, mit der *Through*-Variablen aller Cuts *automatisch* zu Null gesetzt werden, wenn diese nicht mit anderen Cuts verbunden sind, oder wenn diese *Through*-Variablen auf keine andere Weise von außerhalb des Objekts angesprochen werden.

Das einfache Schubgelenk ist ganz entsprechend aufgebaut. Hier wird als Basisklasse die Klasse *PrismaticBase* benutzt. Anstatt einen neuen Gelenktyp in der gleichen Weise wie ein Dreh- oder Schubgelenk einzuführen, kann dieser auch aus bestehenden Grundgelenken aufgebaut werden. Zum Beispiel wird mit der folgenden Klasse *CylinderS* ein Zylindergelenk aus einem Drehgelenk und aus einem Schubgelenk gebildet, wobei beide Gelenke dieselbe Achse besitzen:

```
model class (Joint) CylinderS
    submodel (PrismaticBasis) s
    submodel (RevoluteBasis)  r
    parameter n1 = 0, n2 = 0, n3 = 0
    local     sq, sqd, sqdd, rq, rqd, rqdd, absn

    connect a to s:a,  s:b to r:a,  r:b to b

    absn   = sqrt(n1*n1 + n2*n2 + n3*n3)
    s.n(1) = n1/absn
    s.n(2) = n2/absn
    s.n(3) = n3/absn
    r.n    = s.n

    s.frel = 0
    r.frel = 0

    sqd    = der(sq)
    rqd    = der(rq)
    sqdd   = der(sqd)
    rqdd   = der(rqd)
end
```



Die Klasse *CylinderS* wird aus der Klasse *MbsTwoCut* durch Vererbung abgeleitet. Die beiden Cuts a und b der Klasse *MbsTwoCut* stellen die Verbindung dieses Objekts nach außen dar. Das Drehgelenk r und das Schubgelenk s werden an diesen beiden Cuts mit der **connect** Anweisung befestigt. Das Zylindergelenk wird durch Angabe der Zylinderachse mit den Koordinaten $n1$, $n2$, $n3$ definiert. Der Vektor **n** in Richtung dieser Achse wird normiert und den Achsvektoren des Drehgelenks und des Schubgelenks zugewiesen. Am Ende der Klassenbeschreibung werden der Drehwinkel rq, die Schublänge sq und jeweils deren erste Zeitableitung als Zustandsgrößen definiert.

Die meisten technisch wichtigen Gelenke können, genauso wie das Zylindergelenk, auf einfache Weise aus Dreh- und Schubgelenken aufgebaut werden. Bei einigen wenigen Gelenktypen ist das aber nicht möglich. Diese müssen in derselben Weise wie das Dreh- und Schubgelenk neu definiert werden. Neben dem Schraubgelenk, sind das vor allem Gelenke, die eine räumliche Drehung um 3 Achsen erlauben, wobei diese Drehung durch Quaternionen (auch Eulerparameter genannt) beschrieben wird.

In manchen Fällen ist es günstiger, keine Minimalkoordinaten für einen neuen Gelenktyp einzuführen, sondern direkt Beziehungen zwischen den Relativgrößen sowie (Leistungs-) kompatible Beziehungen in den Schnittkräften und Schnittmomenten anzugeben. Zum Beispiel kann ein Kugelgelenk dadurch definiert werden, daß die translatorischen Relativgrößen Null sind, und daß ein Kugelgelenk keine Schnittmomente überträgt:

```
model class (Joint) SphereCut
    rrelb      = zero(3)
    vrelb(4:6) = zero(3)
    arelb(4:6) = zero(3)
    fb(1:3)    = zero(3)
end
```

In Bild 4.6 auf Seite 81 ist die Bondgraph-Darstellung eines allgemeinen Gelenks angegeben. Ein Gelenk ist ein Bondgraph *Transformer TF*, da keine Energie im Gelenk gespeichert wird und das Gelenk nur die Leistung unterschiedlich auf die Across- und die Through-Variablen verteilt. Allerdings hat ein gewöhnliches Transformerelement nur zwei Leistungs-Cuts. Ein dritter Cut wurde aus Gründen der Übersichtlichkeit eingeführt, weil die oft auftretende Gelenk/Kraftelement Kombination in ihrer Beschreibung recht komplex würde, wenn nur gewöhnliche Bondgraph Grundelemente benutzt würden. Außerdem ginge ohne den dritten Cut der gezeigte enge Zusammenhang (4.12) zwischen Gelenk und Kraftelement verloren.

4.2.4 Trägheitseigenschaften

Die Trägheitseigenschaften eines Starrkörpers werden durch seine Masse, seinen Massenmittelpunkt und seinen Trägheitstensor charakterisiert. Diese Eigenschaften können durch ein Objekt mit einem mechanischen Cut CM[8] beschrieben werden. Das Objekt stellt den Starrkörper dar, dessen Massenmittelpunkt sich im Cut CM befindet. Über den Cut kann der Starrkörper mit anderen mechanischen Elementen verbunden werden. Die Schnittkraft und das Schnittmoment im Cut stellen die resultierende Kraft und das resultierende Moment dar, das im Massenmittelpunkt angreift. Die Dynamik eines Starrkörpers wird mit dem Impuls- und Drallsatz auf die folgende Weise beschrieben (siehe z.B. [Robe88]):

$$\hat{\mathbf{f}}^{CM} = \hat{\mathbf{I}}^{CM}\,\hat{\mathbf{a}}^{CM} + \mathbf{b}^{CM} \tag{4.17}$$

mit

$$\hat{\mathbf{I}}^{CM} = \begin{bmatrix} \mathbf{I}^{CM} & \mathbf{0} \\ \mathbf{0} & m\,\mathbf{E} \end{bmatrix} \tag{4.18a}$$

$$\mathbf{b}^{CM} = \begin{bmatrix} \boldsymbol{\omega}^{CM} \times \mathbf{I}^{CM}\boldsymbol{\omega}^{CM} \\ \mathbf{0} \end{bmatrix} . \tag{4.18b}$$

Hierbei ist $\hat{\mathbf{f}}^{CM}$ die Schnittkraft und das Schnittmoment im Cut CM, $\mathbf{I}^{CM}$ ist der Trägheitstensor, m ist die Masse, $\hat{\mathbf{a}}^{CM}$ ist die Winkelbeschleunigung und die translatorische Beschleunigung, und $\boldsymbol{\omega}^{CM}$ ist die absolute Winkelgeschwindigkeit des Cut-Frames CM. Üblicherweise ist der Massenmittelpunkt kein ausgezeichneter Punkt, so daß fast immer eine masselose Stange (ein Objekt der Klasse *Bar*) benutzt werden muß, um den Starrkörper an einem Gelenk zu befestigen. Es erscheint deswegen zweckmäßig, einen Starrkörper generell an einer masselosen Stange zu befestigen und dann beide Teile als ein Element zu betrachten, siehe Bild 4.8.

Mit den Beziehungen (4.5) zur Umrechnung der Größen des Cuts b einer Stange auf den Cut a, sowie dem Impuls- und Drallsatz (4.17) bezüglich des Massenmittelpunkts, folgt nach kurzer Zwischenrechnung:

[8] CM steht für *Center of Mass*.

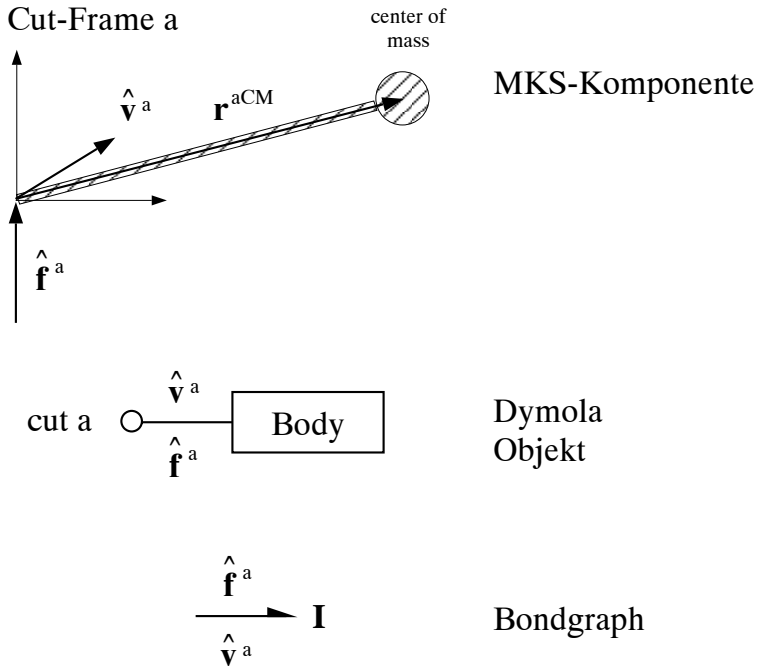


Bild 4.8: Trägheitseigenschaften eines Starrkörpers (Klasse *Body*).

$$\hat{\mathbf{f}}^a = \mathbf{I}^a \hat{\mathbf{a}}^a + \mathbf{b}^a \tag{4.19}$$

mit

$$\mathbf{I}^a = \begin{bmatrix} \bar{\mathbf{I}}^a & m \cdot \mathbf{skew}({}^a\mathbf{r}^{aCM}) \\ -m \cdot \mathbf{skew}({}^a\mathbf{r}^{aCM}) & m \cdot \mathbf{E} \end{bmatrix} \tag{4.20a}$$

$$\mathbf{b}^a = \begin{bmatrix} \boldsymbol{\omega}^a \times \bar{\mathbf{I}}^a \boldsymbol{\omega}^a \\ m \cdot \boldsymbol{\omega}^a \times (\boldsymbol{\omega}^a \times {}^a\mathbf{r}^{aCM}) \end{bmatrix} \tag{4.20b}$$

$$\bar{\mathbf{I}}^a = {}^a\mathbf{I}^{CM} - m \cdot \mathbf{skew}({}^a\mathbf{r}^{aCM}) \cdot \mathbf{skew}({}^a\mathbf{r}^{aCM}) \ . \tag{4.20c}$$

Hierbei ist ${}^a\mathbf{r}^{aCM}$ der Ortsvektor vom Cut-Frame a der Stange zum Massenmittelpunkt CM. ${}^a\mathbf{I}^{CM}$ ist der Trägheitstensor des Starrkörpers bezüglich des Massenmittelpunktes, dargestellt in einem zum Cut-Frame a parallel verschobenen Koordinatensystem. Gleichung (4.19) wird in Abschnitt 4.4 benötigt. Eine andere Formulierung rechnet zuerst mit den Gleichungen der Stange die Beschleunigung vom Schwerpunkt aus (4.5g), wendet den Impuls- und Drallsatz an (4.17) und transformiert die Schnittkraft und das Schnittmoment zum Cut-Frame a (4.5h):

$$\hat{\mathbf{a}}^{CM} = \begin{bmatrix} \boldsymbol{\alpha}^a \\ \mathbf{a}^a + \boldsymbol{\alpha}^a \times {}^a\mathbf{r}^{aCM} + \boldsymbol{\omega}^a \times (\boldsymbol{\omega}^a \times {}^a\mathbf{r}^{aCM}) \end{bmatrix} \tag{4.21a}$$

$$\hat{\mathbf{f}}^a = \begin{bmatrix} {}^a\mathbf{I}^{CM}\boldsymbol{\alpha}^a + \boldsymbol{\omega}^a \times {}^a\mathbf{I}^{CM}\boldsymbol{\omega}^a + {}^a\mathbf{r}^{aCM} \times m \cdot \mathbf{a}^{CM} \\ m \cdot \mathbf{a}^{CM} \end{bmatrix} \ . \tag{4.21b}$$

Die Klasse *Body* zur Beschreibung eines Starrkörpers wird aus der Klasse *MbsOneCut* durch Vererbung abgeleitet, da das Objekt *einen* mechanischen Cut besitzt. Mit (4.21) hat die Klasse den folgenden Aufbau:

```
model class (MbsOneCut) Body
    parameter m = 0, r1 = 0, r2 = 0, r3 = 0,
              I11 = 0, I22 = 0, I33 = 0, I21 = 0, I31 = 0, I32 = 0
    local     r(3), I(3,3)

    r         = [r1; r2; r3]
    I         = [ I11 I21 I31
                  I21 I22 I32
                  I31 I32 I33 ]
    fa(4:6)   = m * ( aa(4:6) - skew(r) * aa(1:3) + skew(va(1:3)) * ( skew(va(1:3)) * r ) )
    fa(1:3)   = I * aa(1:3) + skew(va(1:3)) * ( I * va(1:3) ) + skew(r) * fa(4:6)
end
```

4.2.5 Beobachtungsgrößen

Beobachtungsgrößen sind zusätzliche Elemente, mit denen kinematische Relativgrößen zwischen zwei Cut-Frames ermittelt werden. Sie werden immer dann benötigt, wenn diese Relativgrößen nicht implizit schon durch andere Elemente bestimmt werden. Ein Beobachtungs-Element wird z.B. benötigt, wenn ein Laser-Entfernungsmesser mitsimuliert werden soll, der die Entfernung eines Robotergreifers zur Umgebung mißt und diese Daten an einen Regler rückgekoppelt werden. Beobachtungsgrößen werden mit der Klasse *Sensor* beschrieben und durch Vererbung aus der Klasse *Interact* abgeleitet, da dort alle Relativgrößen schon berechnet werden. In der Klasse *Sensor* müssen nur noch die Schnittkraft und das Schnittmoment am Cut-Frame b zu Null gesetzt werden, da durch einen Sensor keine Kräfte und/oder Momente aufgebracht werden. Die Schnittkraft und das Schnittmoment am Cut a ergibt sich wegen (4.5h) auf Seite 75 dann ebenfalls zu Null. Die Klasse *Sensor* hat deswegen den folgenden Aufbau:

```
model class (Interact) Sensor
    fb = zero(6)
end
```

Manchmal werden nur spezielle Relativgrößen benötigt, z.B. der Abstand zwischen zwei Punkten. Deswegen ist es zweckmäßig auch Unterklassen einzuführen, z.B. die Klasse *LineSensor*, um den relativen Abstand s zwischen zwei Punkten, sowie die Abstands-Geschwindigkeit sd und -Beschleunigung sdd zu ermitteln. Hierzu werden die Gleichungen (4.8) von Seite 79 benutzt:

```
model class (Sensor) LineSensor
    terminal s, sd, sdd, n(3)

    s     = sqrt(rrelb' * rrelb)
    n     = rrelb/s
    sd    = n' * vrelb
    sdd   = n' * arelb + (vrelb' * vrelb - sd * sd)/s
end
```

4.2.6 Inertialsystem

In jedem MKS-Modell muß ein Inertialsystem, d.h. *genau ein* Objekt der Klasse *Inertial*, vorhanden sein. Das Inertialsystem ist das Gegenstück zur Erdung (Klasse *Ground*) bei einer elektrischen Schaltung. Ein Objekt der Klasse *Ground* setzt das Potential zu Null. Entsprechend setzt ein Objekt der Klasse *Inertial* die Across-Variablen seines mechanischen Cuts, d.h. die absolute Lage, Geschwindigkeit und Beschleunigung, zu Null.

Bei Mehrkörpersystemen kommt noch eine Besonderheit hinzu. Bei allen Anwendungen auf der Erdoberfläche muß die Gravitation berücksichtigt werden. Dies bedeutet, daß im Schwerpunkt aller Starrkörper eine entsprechende Gewichtskraft angreifen muß. Bei der Modellerstellung ist es jedoch lästig und fehleranfällig, bei jedem Körper explizit ein Kraftelement anzugeben, das die Gewichtskraft modelliert. Bei einigen MKS-Programmen wird deswegen vom Äquivalenzprinzip der allgemeinen Relativitätstheorie Gebrauch gemacht (siehe z.B. [Falk75]). Dieses besagt, daß ein homogenes Gravitationsfeld vollkommen äquivalent zu einem beschleunigten Bezugssystem ist. Ein Gravitationsfeld kann also durch ein beschleunigt bewegtes Inertialsystem ersetzt werden. Alle physikalischen Vorgänge (nicht nur mechanische) laufen in beiden Systemen vollkommen identisch ab. Es gibt keinen Versuch mit dem festgestellt werden kann, ob man sich in einem Gravitionsfeld oder einem beschleunigt bewegten Raum befindet. Für mechanische Systeme kann auch formal bewiesen werden, daß die beschreibenden Gleichungen für ein MKS identisch sind, gleichgültig ob die Gewichtskraft bei jedem Körper berücksichtigt wird oder ob das Inertialsystem beschleunigt bewegt wird. Deswegen wird hier die Gravitation durch ein mit $-\mathbf{g}$ translatorisch beschleunigtes Inertialsystem beschrieben, wobei $\mathbf{g}$ die Gravitationsbeschleunigung ist. Konsequenterweise müßte dann die Geschwindigkeit des Inertialsystems auf $-\mathbf{g}t$ und die Position auf $-\mathbf{g}t^2$ gesetzt werden. Dies ist jedoch weder notwendig noch zweckmäßig:

Im Grunde werden zwei Koordinatensysteme eingeführt: Das "nicht bewegte" Inertialsystem 1 bezüglich dessen Ursprung Position und Geschwindigkeit angegeben werden, sowie das "beschleunigt bewegte" (jedoch nicht rotierende) Inertialsystem 2, in dem der Impuls- und Drallsatz angeschrieben wird. Da die beiden Systeme nur parallel verschoben sind, sind die Winkelgeschwindigkeit und die Winkelbeschleunigung eines Körpers in beiden Systemen identisch. Positionen und Geschwindigkeiten, welche indirekt über die Kraftelemente in den Impuls- und Drallsatz eingehen, beziehen sich jedoch immer relativ auf das System 1 und nicht auf das System 2!

Die erläuterte Vorgehensweise hat den Vorteil, daß die Auswirkungen der Gravitation durch einmalige Angabe der Gravitionsbeschleunigung problemlos erfaßt werden kann. Dies ist auch effizient, da die Graviationsbeschleunigung nicht mehr explizit in jedes körperfeste Koordinatensystem transformiert werden muß (in dem ja der Impuls- und Drallsatz ausgewertet wird).

Nachteilig ist, daß sich die vom Programm berechneten (translatorischen) *Beschleunigungen* immer auf das beschleunigt bewegte Inertialsystem 2 beziehen. Wenn eine Absolutbeschleunigung gegenüber System 1 benötigt wird, muß deswegen noch die Gravitationsbeschleunigung zur berechneten Beschleunigung hinzuaddiert werden. Dies geschieht am komfortabelsten durch Einsatz eines Beobachtungselements (Klasse *Sensor*), mit dem die Relativbeschleunigung zwischen System 1 und dem gewünschten Koordinatensystem bestimmt wird. Auf alle anderen Größen, insbesondere Schnittkräfte oder Energieflüsse, hat die angegebene Vorgehensweise jedoch keinen Einfluß.

Das Inertialsystem wird damit durch die folgende Klasse beschrieben:

```
model class (MbsOneCut) Inertial
    parameter g = 0, ng1 = 0, ng2 = 0, ng3 = 0

    Ta      = eye(3,3)
    ra      = zero(3)
    va      = zero(6)
    aa(1:3) = zero(3)
    aa(4:6) = -g * [ng1; ng2; ng3]
end
```

i

In einer Bondgraph-Darstellung ist das Inertialsystem eine spezielle *Flow-Source*, da die Flow-Variablen, die den mechanischen Across-Variablen entsprechen, einen vorgegebenen, eingeprägten Wert erhalten.

4.2.7 Diskussion

Mit den in diesem Kapitel erläuterten Klassen kann jedes Mehrkörpersystem modelliert werden, und es können alle Gleichungen erzeugt werden, die ein MKS beschreiben. Insbesondere können auch MKS mit kinematischen Schleifen modelliert werden. In der zur Verfügung gestellten Bibliothek sind eine Reihe von Kraftelement- und Gelenktypen vorhanden. Wie in den Abschnitten 4.2.2 und 4.2.3 gezeigt, können leicht weitere Kraftelement- und Gelenkklassen erstellt werden, wenn diese für eine spezielle Anwendung benötigt werden. Alle besprochenen Klassen zusammengenommen und in einer Datei abgespeichert, bestehen aus ca. 200 Zeilen Code, Leerzeilen nicht mitgezählt. Die "reale" MKS-Bibliothek ist ungefähr um den Faktor 6 größer, da aufgrund der in Dymola noch fehlenden Unterstützung der Matrizenrechnung, alle Matrizen in Koordinatenschreibweise angegeben sind.

Es ist erstaunlich, daß es mit einer neutralen Modellierungssprache wie Dymola möglich ist, die Gleichungen für beliebige Mehrkörpersysteme in einer so kompakten Form anzugeben, denn die Dymola-Sprache selbst besitzt ja keinerlei Kenntnisse über MKS. Weiterhin wird automatisch eine einfache (File-) Eingabe eines MKS-Modells mit zur Verfügung gestellt. Wie am Beispiel des Doppelpendels auf Seite 68 gezeigt, ist diese Fileeingabe recht komfortabel[9]. Außerdem können jetzt MKS-Modelle leicht hierarchisch strukturiert werden und neue Kraftelemente und Gelenke können einfach von einem Anwender eingebracht werden, wie die Definition des Feder-Kraftelements auf Seite 79 zeigt.

Die neue, objektorientierte MKS-Beschreibung hat darüberhinaus den Vorteil, daß *alle* Modellierungsdetails eines MKS leicht verstanden werden können. Dies wird durch die Spezifikation *lokaler* Eigenschaften überschaubarer Objekte erreicht. Weiterhin führt die Verwendung des Vererbungsprinzips dazu, daß die meisten Gleichungen nur an genau einer Stelle definiert und erläutert werden. Demgegenüber gehen in bisherigen MKS-Programmen dieselben Gleichungen in identischer Form, oder nach einer anderen Variablen aufgelöst, an unterschiedlichen Stellen in ein Programm ein. Die einzigen komplexeren Formeln, zur Beschreibung der Relativkinematik (Klasse *Interact* auf Seite 76), werden in gleicher Form bei Gelenken, bei Kraftelementen und bei Beobachtungselementen eingesetzt.

[9] Seit einiger Zeit bieten MKS-Programmpakete wie RASNA oder SIMPACK eine sehr gute grafische Eingabe von MKS an. Diese ist natürlich gegenüber einer Fileeingabe generell komfortabler.

In die Klassenbeschreibungen wurden recht großzügig alle Gleichungen aufgenommen, die in irgendeiner Form von Nutzen sein können, z.B. der über Cuts in ein Element einfließende Energiestrom. Bei einem rein numerisch arbeitenden MKS-Programm kann so etwas aus Effizienzgründen nicht gemacht werden, ohne daß eine mehr oder minder komplexe Logik dafür sorgt, daß nicht benötigte Codeteile nicht durchlaufen werden. Bei einem symbolisch arbeitenden Programm wie Dymola ist das unkritisch, da zum Zeitpunkt der Codeerzeugung *alle* Gleichungen *entfernt* werden, die nicht zur Lösung der Aufgabenstellung benötigt werden.

In der konkreten Realisierung in der Dymola-Bibliothek *mbs.lib* wurden aus Effizienzgründen einige kleinere Modifikationen vorgenommen. Zum Beispiel muß bei Gelenken die Gleichung (4.5e) auf Seite 75 entweder nach ${}^0\mathbf{r}^b$ oder nach ${}^0\mathbf{r}^a$ aufgelöst werden. In beiden Fällen muß dazu die Matrix ${}^0\mathbf{T}^{b\,T}$ invertiert werden. Analytisch ist das unproblematisch, da ${}^0\mathbf{T}^b$ eine orthogonale Matrix ist und die Inversion damit durch das Transponieren der Matrix erreicht wird. Dymola kennt diese Information nicht und würde unnötigerweise ein lineares Gleichungssystem der Ordnung 3 lösen. Aus diesem Grund wurde diese Gleichung aus der Klasse *Interact* herausgenommen und, nach unterschiedlichen Variablen aufgelöst, in die Klassen *Joint, Force* und *Sensor* kopiert. Dies sollte jedoch nur eine temporäre Lösung sein, die überflüssig wird, wenn die Information über die Orthogonalität einer Matrix in einer Modell-Bibliothek mitgeteilt werden kann.

Gegenüber MKS-Programmen hat die hier vorgeschlagene Lösung den Nachteil, daß bei einer fehlerhaften Modellierung eines MKS die Fehlermeldungen nicht immer leicht zu interpretieren sind, da ein neutrales mathematisches Werkzeug wie Dymola nur *Gleichungen* kennt. Wenn z.B. vergessen wurde das Inertialsystem einzuführen und mit einem Element zu verbinden, so wird in der Fehlermeldung mitgeteilt, daß es nicht genügend Gleichungen für alle Variablen gibt und es werden diejenigen Variablen ausgedruckt, für die keine Berechnungsvorschriften zur Verfügung stehen. Aus einer solchen Meldung die Ursache des Fehlers zu erkennen, ist nicht immer einfach. Dazu müssen sicher noch bessere Mechanismen entwickelt werden. In der objektorientierten Sprache Omola [Ande90] gibt es hierfür einige Ansätze.

4.3 Lösen unterschiedlicher Aufgabenstellungen

Im letzten Abschnitt wurde gezeigt, wie ein Mehrkörpersystem objektorientiert modelliert wird und wie die Gleichungen erzeugt werden, die ein MKS beschreiben. Jetzt geht es um die zentrale Frage, wie die erzeugten Gleichungen für unterschiedliche Aufgabenstellungen so umgeformt werden können, daß die Endgleichungen mit numerischen Standardverfahren effizient lösbar sind.

Es werden hier nur *baumstrukturierte* MKS betrachtet. Die schwieriger zu handhabenden MKS mit kinematischen Schleifen werden in Abschnitt 4.6 ab Seite 123 diskutiert. Es werden 5 Aufgabenstellungen besprochen, die nach wachsendem Schwierigkeitsgrad angeordnet sind: Vorwärtskinematik, inverses dynamisches Problem, direktes dynamisches Problem, inverse Kinematik, Stationärwertberechnung.

Um die Lösungsmöglichkeiten besser demonstrieren zu können, werden alle Aufgabenstellungen anhand eines typischen 6-freiheitsgradigen Industrieroboters diskutiert. Hierzu wird

der Manutec r3 verwendet, der in Kapitel 7 ab Seite 164 im Detail besprochen wird. Mit der objektorientierten Modellierungstechnik ist es möglich, für *alle* Aufgabenstellungen vom selben Modell auszugehen. Deshalb wird der Roboter nur einmal an dieser Stelle angegeben und modelliert. Ein Bild des Roboters ist auf Seite 165 zu sehen. Die Mehrkörperdynamik des Roboters wird durch folgendes Modell beschrieben:

```
@mbs.lib          {verwende MKS-Bibliothek}
model robot
    submodel (Inertial)  i   (g = 9.81, ng3 = −1)

    submodel (Revolute) r1 (n3 = 1)  , r2 (n1 = 1)   , r3 (n1 = 1)
    submodel (Revolute) r4 (n3 = 1)  , r5 (n1 = 1)   , r6 (n3 = 1)
    submodel (Bar)      b3 (r3 = 0.5), b5 (r3 = 0.73), bL (r1 = rL1, r2 = rL2, r3 = rL3)

    submodel (Body) m1 (I33 = 1.16),  Load (m = mL)
    submodel (Body) m2 (m = 56.5, r1 = 0.172, r3 = 0.205  , I11 = 2.58   , I22 = 0.73 ,
                                                           I33 = 0.64   , I31 = −0.46)
    submodel (Body) m3 (m = 26.4, r1 = 0.064, r3 = −0.034, I11 = 0.279  , I22 = 0.413,
                                                           I33 = 0.245  , I31 = −0.07)
    submodel (Body) m4 (m = 28.2              , r3 = 0.32   , I11 = 1.67   , I22 = 1.67,
                                                           I33 = 0.081)
    submodel (Body) m5 (m = 5.2               , r3 = 0.023  , I11 = 0.0125, I22 = 0.0153,
                                                           I33 = 0.0081)
    submodel (Sensor) sensor

    parameter mL = 5, rL1 = 0.1, rL2 = 0, rL3 = 0
    terminal  q(6), qd(6), qdd(6), f(6)
    cut s

    connect i to r1 to r2 to b3 to r3 to r4 to b5 to r5 to r6 to bL,
            m1 at r1, m2 at r2, m3 at r3, m4 at r4, m5 at r5, load at bL,
            sensor at (i, Load), s at sensor:Rel

    q(1)  = r1.q
    q(2)  = r2.q
            ...
    qd(1) = r1.qd
            ...
    qdd(1)= r1.qdd
            ...
    f(1)  = r1.f
            ...
    f(6)  = r6.f
end
```

Die Referenzkonfiguration des Roboters ist dadurch charakterisiert, daß alle Arme des Roboters senkrecht nach oben stehen. In dieser Lage sind die Drehwinkel aller 6 Drehgelenke Null. Für die Drehgelenke wird die Klasse *Revolute* benutzt, da die Blockiereigenschaft der Gelenke nicht benötigt wird. Die Klasse *RevoluteS* wird nicht verwendet, weil hier die Drehwinkel schon als Zustandsgrößen definiert wären, was nur bei einer bestimmten Aufgabenstellung (der Lösung des direkten dynamischen Problems) zutrifft. Da für manche Aufgabenstellungen ein Beobachtungselement zwischen der Last und dem Inertialsystem benötigt wird, wird ein Objekt der Klasse *Sensor* vorgesehen. Durch die Anweisung "**connect** *s* **at** *sensor*:*Rel*" wird der Cut *Rel* des Beobachtungselements mit dem Cut *s* des Roboters verbunden. Die im Cut *Rel* vorliegenden Relativgrößen können damit auch außerhalb des Roboters angesprochen werden.

4.3.1 MKS-Algorithmen und objektorientierte Modellierung

Bevor die unterschiedlichen Aufgabenstellungen näher untersucht werden, soll hier kurz auf die etwas ungewohnte "Darstellung" von MKS-Algorithmen eingegangen werden.

Üblicherweise werden Algorithmen zur Berechnung der Kinematik oder der Dynamik eines MKS explizit angegeben. Zum Beispiel können die absoluten Positionsgrößen aller Körper eines baumstrukturierten Mehrkörpersystems mit dem folgenden Algorithmus in Pseudocode-Darstellung bestimmt werden:

Gegeben sind die relativen Ortsvektoren ${}^{i}\mathbf{r}_{rel}^{i,j}$ von einem Koordinatensystem i zu einem Koordinatensystem j, dargestellt im Koordinatensystem i, der relativen Drehmatrix ${}^{i}\mathbf{T}_{rel}^{j}$ vom Koordinatensystem j zum Koordinatensystem i, sowie der Funktion $\texttt{ref}(i)$ die den Index des Referenz-Koordinatensystems angibt, bezüglich dessen das Koordinatensystem i beschrieben ist. Weiterhin wird angenommen, daß die Indizes der Koordinatensysteme so angeordnet sind, daß "$\texttt{ref}(i) < i$".
Gesucht sind die absoluten Ortsvektoren ${}^{0}\mathbf{r}_{abs}^{0,i}$ vom Inertialsystem zu den Koordinatensystemen i, dargestellt im Inertialsystem, sowie die absoluten Drehmatrizen ${}^{0}\mathbf{T}_{abs}^{i}$ von den Koordinatensystemen i zum Inertialsystem. Die gesuchten Größen werden aus den gegebenen Größen auf die folgende Weise berechnet:

for i=1,nframe
  if $\texttt{ref}(i) = 0$ **then**
    $${}^{0}\mathbf{r}_{abs}^{0,i} = {}^{0}\mathbf{r}_{rel}^{0,i}$$
    $${}^{0}\mathbf{T}_{abs}^{i} = {}^{0}\mathbf{T}_{rel}^{i}$$
  else
    $${}^{0}\mathbf{r}_{abs}^{0,i} = {}^{0}\mathbf{r}_{abs}^{0,\texttt{ref}(i)} + {}^{0}\mathbf{T}_{abs}^{\texttt{ref}(i)} \cdot {}^{\texttt{ref}(i)}\mathbf{r}_{rel}^{\texttt{ref}(i),i}$$
    $${}^{0}\mathbf{T}_{abs}^{i} = {}^{0}\mathbf{T}_{abs}^{\texttt{ref}(i)} \cdot {}^{\texttt{ref}(i)}\mathbf{T}_{rel}^{i}$$
  endif
endfor

Auf Grund der vielen Indizes bei einer Variablen, ist dieser einfache Algorithmus schon relativ unübersichtlich. Die Indizes sind notwendig, um die Verschaltungstopologie der Einzelelemente (hier: Koordinatensysteme) zu beschreiben. Bei anderen Algorithmen, wie z.B. bei einem Algorithmus zur Erstellung der Bewegungsgleichungen, nimmt die Komplexität noch deutlich zu.

Bei der objektorientierten Modellierung von Mehrkörpersystemen wird kein MKS-Algorithmus explizit angegeben.

Stattdessen werden, genauso wie im vorigen Kapitel 4.2, *nur* die Gleichungen der Einzelkomponenten erstellt, und die Schnittstellen der Einzelkomponenten definiert. Die topologische Zusammenschaltung der Einzelkomponenten erfolgt dann automatisch mit einem objektorientierten Modellierungswerkzeug, das nicht MKS-spezifisch ist. Dieses erzeugt auch alle Verschaltungsgleichungen und transformiert die Gleichungen so, daß die gesuchten Größen berechnet werden können.

Auf Grund dieser Vorgehensweise ist es unnötig, noch einen expliziten Algorithmus zur Berechnung der gesuchten Größen anzugeben. Es genügt stattdessen, die Gleichungen der

Einzelkomponenten anzugeben und durch globale Überlegungen sicherzustellen, daß das Problem mit den verwendeten Komponenten wohl definiert ist (z.B. Zahl der Gleichungen = Zahl der Unbekannten).

MKS-Algorithmen unterscheiden sich oft nur darin, wie auftretende lineare Gleichungssysteme behandelt werden. In der objektorientierten Vorgehensweise werden solche linearen Gleichungssysteme identifiziert und mit unterschiedlichen Verfahren gelöst. Auch hier ist es unnötig jedesmal den kompletten Algorithmus zum Lösen eines linearen Gleichungssystems zu definieren, nur weil dieser speziell für ein MKS benötigt wird. Es genügt, anzugeben, wie das Modellierungswerkzeug das lineare Gleichungssystem lösen soll. Beim Tearing-Verfahren (siehe S. 28) können hierbei auch fachspezifische Strukturen "einfließen".

Auf Grund der obigen Überlegungen, werden in den folgenden Abschnitten keine MKS-Algorithmen abgeleitet, sondern es wird angegeben, wie diese vom Modellierungsprogramm automatisch ermittelt werden. Es wird gezeigt, daß diese automatisch ermittelten "MKS-Algorithmen" genauso effizient sind, wie die bekannten Standardverfahren.

4.3.2 Vorwärtskinematik

Bei einem baumstrukturierten System ist die Lage eines MKS eindeutig durch die Minimalkoordinaten aller Gelenke charakterisiert. Dies ist einfach einzusehen, da in einer Baumstruktur jeder Körper durch genau *ein* Gelenk mit seinem Vorgängerkörper in Richtung zur "Wurzel" des Baumes gekoppelt ist. Die Lage eines Gelenks, d.h. die Relativlage des Cut-Frames b relativ zum Cut-Frame a eines Gelenks, ist eindeutig durch seine f Minimalkoordinaten $\mathbf{q}$ definiert. Gleichzeitig ist damit die Lage jedes Körpers eindeutig in bezug auf seinen Vorgängerkörper festgelegt. Also ist die Lage des gesamten MKS eindeutig bekannt, wenn die Minimalkoordinaten aller Gelenke bekannt sind.

Die einfachste Aufgabenstellung besteht in der Berechnung aller *kinematischen* Größen eines MKS, und insbesondere der Absolutgrößen jedes Cut-Frames, wenn die Gelenk-Minimalkoordinaten und deren erste und zweite Ableitungen $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ gegeben sind. Die Gleichungen können auf explizit lösbare Blockdreiecksform transformiert werden (siehe auch Kapitel 2.2 ab Seite 23):

Die Bewegung des Inertialsystems ist bekannt. Damit sind die kinematischen Größen aller Cut-Frames bekannt, die direkt mit dem Inertialsystem gekoppelt sind. Angenommen, diese Cut-Frames definieren die Bewegung von "a"-Cuts. Da die Minimalkoordinaten aller Gelenke bekannt sind, sind alle Relativgrößen in jedem Gelenk bekannt. Daher können mit den Gleichungen (4.5) auf Seite 75 alle Absolutgrößen des Cuts b der Gelenke bestimmt werden, die direkt mit dem Inertialsystem gekoppelt sind. Die Gleichungen (4.5) können symbolisch nach den Cut b Größen aufgelöst werden. Jetzt sind aber auch alle Absolutgrößen der Cuts der Gelenke bekannt, die direkt mit den gerade behandelten Elementen gekoppelt sind. Es liegt damit dieselbe Situation vor, wie sie bei den mit dem Inertialsystem verbundenen Elementen vorlag. Auf die gleiche Weise können alle absoluten Größen dieser Elemente, dann die der darauffolgenden Elemente, usw., berechnet werden. Während der Abarbeitung wird jedes Element eines Modells einmal "besucht" und die Gleichungen (4.5) auf Seite 75 werden jeweils einmal ausgewertet. Der Algorithmus ist damit $O(n)$, wobei n die Zahl der Elemente ist. Q.E.D.

Bis jetzt wurde angenommen, daß alle Cut-Frames a mit dem Vorgängerkörper verbunden sind. Ist dies nicht der Fall, sind die Größen des Cut-Frames b bekannt und die Gleichungen (4.5) müssen nach den Größen des Cut-Frames a aufgelöst werden. Hierzu muß die Matrix $\mathbf{C}$ invertiert werden. Bei der Transformation auf Blockdreiecksform wird das entsprechende System von 6 Gleichungen automatisch ermittelt und symbolisch oder numerisch gelöst. Dies ist jedoch uneffizient, weil $\mathbf{C}$ aufgrund seiner besonderen Struktur analytisch sehr einfach invertiert werden kann. Auch hier wäre wiederum ein Sprachelement wünschenswert, mit der diese Eigenschaft der Matrix $\mathbf{C}$ definiert werden könnte. Solange das nicht der Fall ist, sollte bei der Modellierung darauf geachtet werden, daß alle Cut-Frames a mit dem Vorgängerkörper verbunden sind (und nicht die Cut-Frames b). Ansonsten sind unötige Operationen im erzeugten Code vorhanden.

Bei Robotern ist man in der Regel an der absoluten Position, Geschwindigkeit und Beschleunigung des Endeffektors interessiert, wobei die Minimalkoordinaten der Gelenke gegeben sind. Alle gesuchten Größen sollen dabei im Inertialsystem angegeben werden. Da die Absolutgeschwindigkeit und -Beschleunigung von Cut-Frames nicht im Inertialsystem, sondern im entsprechenden Cut-Frame dargestellt sind, wird hier am besten ein Beobachtungselement eingeführt, welches vom Inertialsystem zum Endeffektor zeigt. Die Relativgrößen dieses Elements, dargestellt im Cut-Frame a, sind die gewünschten Größen. Für den Manutec Roboter werden diese Variablen mit folgendem Modell ermittelt:

```
model rob1
    {benutze oben definiertes Robotermodell}
        @robot.dym

    {deklariere Eingangsgrößen (q, q̇, q̈) und Ausgangsgrößen (ᵇTᵃ, ᵃrᵃᵇ, ᵃv̂ᵃᵇ, ᵃâᵃᵇ)}
        input  q(6), qd(6), qdd(6)
        output T06(3,3), r06(3), v06(6), a06(6)

    {definiere Eingangs- und Ausgangsgrößen}
        q   = robot.q
        qd  = robot.qd
        qdd = robot.qdd

        T06 = robot:s.Trela
        r06 = robot:s.rrela
        v06 = robot:s.vrela
        a06 = robot:s.arela
end

{definiere Parameter, die nicht expandiert werden sollen}
    variable not evaluate robot.rL1, robot.rL2, robot.rL3
```

Durch die Festlegung der Eingangs- und Ausgangsgrößen mit der **input** und **output** Anweisung wird definiert, welche Variablen bekannt sind und welche Variablen berechnet werden sollen. Mittels einer Option wird erreicht, daß alle Gleichungen entfernt werden, die nicht zur Berechnung der Ausgangsgrößen benötigt werden.

Um es noch einmal zu betonen: Mit dem obigen Modell, zusammen mit den im letzten Abschnitt besprochenen Klassen, erstellt Dymola effizienten Code zur Berechnung der kinematischen Größen des Endeffektors. Dies ist möglich, da die Gleichungen der Aufgabenstellung auf eine explizit lösbare Blockdreiecksform führen.

4.3.3 Inverses dynamisches Problem

Das "inverse dynamische Problem" stammt aus der Robotik. Gesucht sind die verallgemeinerten Kräfte $\boldsymbol{\lambda}^e$ in allen Gelenken, die notwendig sind, um eine gewünschte Bewegung zu erzeugen. Die Bewegung wird durch Vorgabe der Minimalkoordinaten der Gelenke beschrieben. Die so berechneten Kräfte und Momente können von den Reglern der Motoren benutzt werden, um den Roboter auf einer gewünschten Sollbahn zu halten. Auch bei dieser Aufgabenstellung können die Gleichungen immer auf eine explizit lösbare Blockdreiecksform transformiert werden:

1. Da die Minimalkoordinaten bekannt sind, können, wie im vorigen Abschnitt besprochen, die kinematischen Größen aller Cut-Frames und aller relativen Größen problemlos berechnet werden.

2. Wenn alle kinematischen Größen bekannt sind, können die Kräfte und Momente aller Kraftelemente berechnet werden und damit die Schnittkräfte und Schnittmomente an denjenigen Elementen, an denen die Kraftelemente angekoppelt sind.

3. Starrkörper, d.h. alle Objekte der Klasse *Body*, besitzen nur einen Cut. Da die kinematischen Größen dieses Cuts schon berechnet wurden, können mit dem Impuls- und Drallsatz (4.21) auf Seite 91 die Schnittkraft und das Schnittmoment dieses Cuts berechnet werden.

4. Jetzt werden die Gelenke an den "Blättern" des MKS-Baums betrachtet. Am äußeren Cut-Frame b dieser Gelenke ist entweder ein Kraftelement und/oder ein Objekt der Klasse *Body* angebracht. In beiden Fällen sind die Schnittkraft sowie das Schnittmoment am Cut b bekannt. Mit Gleichung (4.5h) auf Seite 75 können die Schnittkraft und das Schnittmoment am Cut a ermittelt werden.

5. Werden alle abgearbeiteten "Blatt"-Objekte gedanklich entfernt, liegt wieder genau die gleiche Situation vor, d.h. die Schnittkräfte und Schnittmomente am Cut b der neuen "Blatt"-Objekte sind bekannt und die Kräfte und Momente des Cuts a können berechnet werden. Auf diese Weise ensteht eine Rückwärtsrekursion, von den Blättern zur Wurzel, bei der die Schnittkräfte und Schnittmomente aller Cuts bestimmt werden.

6. In einem letzten Schritt müssen ausgewählte Schnittkräfte und Schnittmomente nur noch mit Gleichung (4.12) von Seite 83 auf die freien Bewegungsrichtungen der Gelenke projiziert werden, um die gewünschten verallgemeinerten Gelenkkräfte $\boldsymbol{\lambda}^e$ zu erhalten.

Die gesuchten Größen können also durch Sortieren der Gleichungen, d.h. durch Transformation auf Blockdreiecksform, berechnet werden. Der Algorithmus ist $O(n)$, wobei n die Anzahl der im MKS-Modell enthaltenen Elemente ist, da die Gleichungen für jedes MKS-Element genau einmal ausgewertet werden. Von Luh, Walker und Paul wurde 1980 in [Luh80] ein Algorithmus zur Lösung des inversen dynamischen Problems angegeben. Es zeigt sich, daß die mit Dymola automatisch erzeugten (sortierten) Gleichungen identisch mit den Gleichungen dieses Algorithmus sind!

Für den Manutec Roboter werden die Gleichungen des inversen dynamischen Problems mit dem folgenden Modell ermittelt:

```
model rob2
  {benutze oben definiertes Robotermodell}
    @robot.dym

  {deklariere Eingangsgrößen (q, q̇, q̈) und Ausgangs-Größen (u = Drehmomente in den Gelenken)}
    input  q(6), qd(6), qdd(6)
    output u(6)

  {definiere Eingangs- und Ausgangsgrößen}
    q   = robot.q
    qd  = robot.qd
    qdd = robot.qdd
    u   = robot.f
end

{definiere Parameter, die nicht expandiert werden sollen}
  variable not evaluate robot.mL, robot.rL1, robot.rL2, robot.rL3
```

Dieses Modell ist sehr ähnlich zum Modell, das für die Lösung des direkten kinematischen Problems benutzt wurde. Unterschiedlich sind nur die Ausgangsgrößen, da beim inversen dynamischen Problem jetzt die Drehmomente in den Drehgelenken gesucht sind.

4.3.4 Direktes dynamisches Problem (Bewegungsgleichungen)

Das "direkte dynamische Problem", d.h. die Erstellung eines Simulationsmodells, ist die häufigste und wichtigste Aufgabenstellung. Bei MKS in Baumstruktur können die Minimalkoordinaten aller Gelenke, sowie deren erste Ableitung $(\mathbf{q}, \dot{\mathbf{q}})$, als Zustandsvariablen benutzt werden. Dies ergibt sich aus der schon beim direkten kinematischen Problem erläuterten Eigenschaft, daß die Lage aller Körper bei einem baumstrukturierten MKS eindeutig festliegt, wenn die Minimalkoordinaten der Gelenke gegeben sind.

Beim Simulationsproblem müssen die Ableitungen der Zustandsvariablen bestimmt werden. Deswegen ist $\ddot{\mathbf{q}}$ bei gegebenen $\mathbf{q}, \dot{\mathbf{q}}$ zu ermitteln. Es zeigt sich, daß dieses Problem nicht auf eine explizit lösbare Blockdreiecksform führt. Beim Manutec Roboter tritt z.B. ein Block auf der Diagonalen auf, der aus 255 dünnbesetzten, gekoppelten linearen Gleichungen besteht.

Bei gegebenen $\mathbf{q}, \dot{\mathbf{q}}$ können mittels der Vorwärtskinematik alle Positions- und Geschwindigkeitsgrößen berechnet werden. Da Kraftelemente in der Regel nur von Position und Geschwindigkeit abhängen, können auch noch die Schnittkräfte und Schnittmomente an den Kraftelementen bestimmt werden. Weitere direkte Berechnungen sind nicht mehr möglich. Zum Beispiel gibt es für ein Objekt der Klasse *Body* wegen des Impuls- und Drallsatzes (4.19) auf Seite 91, 6 Gleichungen für die 12 Unbekannten $\hat{\mathbf{f}}^a, \hat{\mathbf{a}}^a$. In der gleichen Weise gibt es bei Gelenken wegen der Gleichungen (4.5g, 4.5h, 4.9d, 4.12) $(18 + f)$ Gleichungen für die $(30 + f)$ Unbekannten $\hat{\mathbf{a}}^a, \hat{\mathbf{a}}^b, {}^b\hat{\mathbf{a}}^{ab}, \hat{\mathbf{f}}^a, \hat{\mathbf{f}}^b, \ddot{\mathbf{q}}$, wobei f die Zahl der Freiheitsgrade eines Gelenks ist. Dies ist der Grund dafür, daß ein großes dünnbesetztes Gleichungssystem entsteht, in das die Schnittkräfte und Schnittmomente, die absoluten und die relativen Beschleunigungen und Winkelbeschleunigungen, sowie die gesuchten zweiten Ableitungen der Gelenk-Minimalkoordinaten eingehen.

Wenn ein "Sparse Matrix Solver" in Dymola schon vorhanden wäre, könnte dieser benutzt werden, um das lineare Gleichungssystem zu lösen. Eine (verfügbare) Alternative besteht darin, das große, dünnbesetzte Gleichungssystem, mittels des in Kapitel 2.3 ab Seite 28

erläuterten Tearing-Verfahrens, auf ein kleines dichtbesetztes Gleichungssystem *symbolisch* zu transformieren und dieses kleine System numerisch mit LINPACK Unterprogrammen zu lösen.

Das einzige Problem beim Tearing-Verfahren besteht darin, geeignete Tearing-Variablen und -Gleichungen zu bestimmen. Diese können mit folgender Fragestellung anschaulich ermittelt werden (siehe Gleichungssystem (2.11) auf Seite 29):

> Welche *unbekannten* Variablen $\mathbf{x}_2$ müssen *bekannt* und welche *bekannten* Variablen $\mathbf{b}_2$ müssen *unbekannt* sein, damit die anderen unbekannten Variablen $\mathbf{x}_1$ sowie $\mathbf{b}_2$ berechnet werden können?

Genau das ist die Aussage der Gleichungen (2.11, 2.12). Vom inversen dynamischen Problem wissen wir, daß bei bekannten $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ die verallgemeinerten Gelenkkräfte $\boldsymbol{\lambda}^e$ bestimmt werden können. Bei Betrachtung dieses Algorithmus stellt man fest, daß dabei alle anderen Größen im MKS berechnet werden. Aus diesem Grund sind die Variablen $\ddot{\mathbf{q}}$ die *Tearing-Variablen* und die Gleichung (4.12) auf Seite 83, mit der die verallgemeinerten Gelenkkräfte $\boldsymbol{\lambda}^e$ berechnet werden können, ist die *Tearing-Gleichung*. Da es insgesamt f Tearing-Variablen gibt, wobei f die Summe der Freiheitsgrade aller Gelenke ist, wird das "große" dünnbesetzte Gleichungssystem auf ein "kleines" System mit f linearen Gleichungen reduziert. Dieses wichtige Ergebnis stammt von Elmqvist [Elmq92a], der es bei 2-dimensionalen Mehrkörpersystemen angewandt hat.

Eine Analyse zeigt, daß die Zahl der Rechenoperationen für diese Transformation $O(nf)$ ist, wobei n die Zahl der MKS-Elemente und f die Zahl der Freiheitsgrade des MKS ist. Die Lösung eines linearen Gleichungssystems der Ordnung f benötigt $O(f^3)$ Operationen. Aus diesem Grunde ist der gesamte Algorithmus zum Berechnen der Ableitungen der Zustandsgrößen proportional zu $O(f^3) + O(nf)$. Es stellt sich die Frage, wie der obige Algorithmus mit den bekannten Gleichungen und Algorithmen der Mehrkörperdynamik zusammenhängt.

Ein baumstrukturiertes MKS kann mit den Prinzipien der Mechanik auf die folgende Form transformiert werden (siehe z.B. [Robe88]):

$$\mathbf{M}(\mathbf{q}, t)\ddot{\mathbf{q}} = \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}, t) + \boldsymbol{\lambda}^e \ . \tag{4.22}$$

Hierbei ist $\mathbf{M}$ die $f \times f$ Massenmatrix, $\mathbf{q}$ sind die verallgemeinerten Minimalkoordinaten aller Gelenke und $\boldsymbol{\lambda}^e$ sind die verallgemeinerten Kräfte aller Gelenke. Es kann gezeigt werden, daß mit dem obigen Vorgehen durch den Tearing-Algorithmus genau Gleichung (4.22) erzeugt wird.

Von Walker und Orin wurden 1982 in [Walk82] mehrere Algorithmen angegeben, wie die Massenmatrix $\mathbf{M}$ und die rechte Seite $\mathbf{h}$ der Gleichung (4.22) unter Benutzung des Algorithmus von Luh/Walker/Paul [Luh80] bestimmt werden können. Wie schon erläutert, werden mit diesem Algorithmus die verallgemeinerten Gelenkkräfte $\boldsymbol{\lambda}^e$ bei bekannten $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ berechnet. Durch $(f+1)$ malige Anwendung des Algorithmus und mit geschickter Vorgabe von $\ddot{\mathbf{q}}$ können die beiden Matrizen aus (4.22) bestimmt werden:

$$\begin{array}{lcllcll} \ddot{\mathbf{q}} & = & \mathbf{0} & \Rightarrow & \boldsymbol{\lambda}^e & = & -\mathbf{h} \\ \ddot{\mathbf{q}} & = & \mathbf{e}_j & \Rightarrow & \lambda_i^e & = & M_{ij} \quad (\mathbf{h} = \mathbf{0}). \end{array} \tag{4.23}$$

Hierbei ist $\mathbf{e}_j$ der j-te Einheitsvektor. Zur Bestimmung der Massenmatrix $\mathbf{M}$ wird bei der Codeerzeugung dafür gesorgt, daß alle Kraftelemente und alle von $\dot{\mathbf{q}}$ abhängigen Terme ignoriert werden. Dies ist gleichbedeutend damit, daß $\mathbf{h}$ verschwindet. *Es zeigt sich, daß diese Art des Vorgehens identisch mit dem Tearing-Verfahren ist.* Um dies zu sehen, wird die Walker/Orin Methode auf das allgemeine lineare Tearing Problem angewandt:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} \quad . \tag{4.24}$$

Es wird angenommen, daß $\mathbf{A}_{11}$ eine *untere Dreiecksmatrix* mit nichtverschwindenden Diagonalelementen ist. Dann ist $\mathbf{A}_{11}$ regulär und Gleichung (4.24) kann symbolisch auf das folgende Gleichungssystem transformiert werden (vergleiche auch (2.12) auf Seite 29):

$$\hat{\mathbf{A}}\mathbf{x}_2 = \hat{\mathbf{b}} \tag{4.25}$$

mit

$$\hat{\mathbf{A}} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \tag{4.26a}$$

$$\hat{\mathbf{b}} = \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{b}_1 \quad . \tag{4.26b}$$

Die Matrizen $\hat{\mathbf{A}}, \hat{\mathbf{b}}$ können jetzt wie beim Walker/Orin Algorithmus durch geschickte Vorgabe von $\mathbf{x}_2$ und $(n_2 + 1)$ malige Berechnung von $\mathbf{b}_2$ aus (4.24) ermittelt werden:

$$\begin{array}{lcl} \mathbf{x}_2 = \mathbf{0} \quad \Rightarrow \quad \mathbf{b}_2 &=& \mathbf{A}_{21}\left(\mathbf{A}_{11}^{-1}\mathbf{b}_1\right) \\ \mathbf{x}_2 = \mathbf{e}_j \quad \Rightarrow \quad \mathbf{b}_2 &=& \mathbf{A}_{22}\mathbf{e}_j - \mathbf{A}_{21}\left(\mathbf{A}_{11}^{-1}\left(\mathbf{A}_{12}\mathbf{e}_j\right)\right) \quad (\mathbf{b}_1 = \mathbf{0}) \\ &=& \hat{\mathbf{A}}_{\bullet j} \quad . \end{array} \tag{4.27}$$

Hierbei ist $\hat{\mathbf{A}}_{\bullet j}$ die j-te Spalte der Matrix $\hat{\mathbf{A}}$. Man sieht, daß der Walker/Orin Algorithmus ein Spezialfall der Tearing-Methode ist, da durch die $(n_2 + 1)$ malige Berechnung von $\mathbf{b}_2$, bei unterschiedlichen Vorgabewerten für $\mathbf{x}_2$, exakt dieselben Ausdrücke wie beim Tearing bestimmt werden. Weiterhin kann die oben angegebene Abschätzung der Zahl der Rechenoperationen verifiziert werden. Die Lösung der inversen Dynamik benötigt $O(n)$ Operationen, wobei n die Zahl der MKS-Elemente ist. Dieser Algorithmus wird $(f + 1)$ mal angewandt um das "kleine" Gleichungssystem zu erhalten. Also ist die Gesamtzahl der Operationen proportional zu $O(nf)$.

Aufgrund der obigen Überlegungen werden beim Manutec Roboter die Bewegungsgleichungen mit dem folgenden Modell erzeugt:

```
model rob3
    {benutze oben definiertes Robotermodell}
        @robot.dym

    {deklariere Eingangsgrößen (u = Drehmomente in den Gelenken}
        input u(6)

    {definiere Eingangsgrößen und Zustandsgrößen}
        robot.qd  = der(robot.q)
        robot.qdd = der(robot.qd)
        u         = robot.f
end

{definiere Parameter, die nicht expandiert werden sollen}
```

variable not evaluate *robot.mL*, *robot.rL1*, *robot.rL2*, *robot.rL3*

{definiere Tearing-Variablen (*robot.qdd*) und Tearing-Gleichungen (*u*)}
variable tear *robot.qdd* [*u*]

Die Gleichung $robot.qd = \mathbf{der}(robot.q)$ besagt, daß *robot.qd* die Ableitung von *robot.q* ist. Wie schon erwähnt, legt die implizit in Dymola eingebaute Regel fest, daß *robot.q* dann Zustandsgröße ist. Entsprechendes gilt für die Gleichung $robot.qdd = \mathbf{der}(robot.qd)$. Die Tearing-Variablen und Tearing-Gleichungen werden mit der letzten Anweisung im obigen Modell definiert. Dymola bestimmt dann die $f \times f$ Massenmatrix, sowie die rechte Seite, und löst das Gleichungssystem numerisch mit einem LINPACK-Unterprogramm (= Gauß-Algorithmus mit Spalten-Pivotsuche und Konditionsschätzung).

Bis jetzt wurde davon ausgegangen, daß die Gelenke immer "beweglich" bleiben. Im objektorientierten MKS-Modell können Gelenke jedoch während der Simulation auch blockiert werden. Wenn dieser Fall eintreten kann, muß die obige Vorgehensweise leicht modifiziert werden: Wie auf Seite 85 erläutert, gibt es bei sperrbaren Gelenken die zusätzliche Zwangskraft $\boldsymbol{\lambda}^l$ und die zusätzliche Gleichung

$$\mathbf{0} = \mathbf{L}\ddot{\mathbf{q}} + (\mathbf{E} - \mathbf{L})\boldsymbol{\lambda}^l \quad . \tag{4.28}$$

Die Diagonalmatrix $\mathbf{L}$ gibt an, ob die Bewegungsrichtung eines Gelenks "beweglich" oder "gesperrt" ist. Beim Tearing wird nun $\boldsymbol{\lambda}^l$ als zusätzliche Tearing-Variable und Gleichung (4.28) als korrespondierende Tearing-Gleichung benutzt. Dadurch ändert sich das obige Vorgehen nur geringfügig. Da $\boldsymbol{\lambda}^l$ Tearing-Variable ist und nur in die beiden Gleichungen (4.15) auf Seite 85 eingeht, führt Tearing bei MKS mit blockierbaren Gelenken auf das folgende Gleichungssystem:

$$\begin{bmatrix} \mathbf{M}(\mathbf{q},t) & -\mathbf{L} \\ -\mathbf{L} & \mathbf{L}-\mathbf{E} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda}^l \end{bmatrix} = \begin{bmatrix} \boldsymbol{\lambda}^e + \mathbf{h}(\mathbf{q},\dot{\mathbf{q}},t) \\ \mathbf{0} \end{bmatrix} \quad . \tag{4.29}$$

Hierbei sind in $\mathbf{L}$ der Gleichung (4.29) die $\mathbf{L}$-Matrizen aller Gelenke und in $\boldsymbol{\lambda}^l$ die zusätzlichen Zwangskräfte aller Gelenke zusammengefaßt. Die Ordnung des linearen Gleichungssystems erhöht sich durch die Blockierbarkeit der Gelenke von f auf $2f$. Je nach Blockierzustand, liegt eine andere Null/Nichtnull Struktur der Massenmatrix von (4.29) vor.

Das allgemeine Tearing-Verfahren hat den Nachteil, daß nicht erkannt wird, daß die entstehende Massenmatrix $\mathbf{M}$ symmetrisch und positiv definit ist. Deswegen wird das lineare Gleichungssystem (4.22) mit einem Löser für allgemeine, *unsymmetrische* Matrizen gelöst. Dies hat mehrere unangenehme Konsequenzen: Zum einen ist die Zahl der Rechenoperationen zum Erstellen von (4.22) größer, da nicht erkannt wird, daß nur eine Hälfte der Massenmatrix berechnet werden muß. Zum anderen ist die Zahl der Rechenoperationen zur Lösung einer unsymmetrischen Matrixgleichung rund doppelt so groß wie zur Lösung einer symmetrischen Matrixgleichung. Schließlich ist die Lösung einer positiv definiten Matrixgleichung numerisch zuverlässiger möglich (es wird z.B. kein Pivoting benötigt) als die Lösung einer unsymmetrischen Matrixgleichung. Es ist noch unklar, wie mit dem allgemeinen Tearing-Verfahren zumindest die Eigenschaft der Symmetrie automatisch erkannt werden kann. In Kapitel 4.4 wird aus diesem Grund eine alternative Vorgehensweise besprochen, die diese Nachteile nicht besitzt.

4.3.5 Inverse Kinematik

Die "inverse Kinematik" wird vor allem in der Robotik benötigt und ist das "inverse" Problem zur Vorwärtskinematik in Abschnitt 4.3.2 auf Seite 98. Um die Soll-Bewegung eines Roboters zu definieren, ist es am komfortabelsten die Bewegung des Endeffektors vorzugeben, d.h. die absolute Drehmatrix vom Inertialsystem zum Endeffektor, den absoluten Ortsvektor, die absolute Geschwindigkeit und Winkelgeschwindigkeit, sowie die absolute Beschleunigung und Winkelbeschleunigung. Diese Größen werden von einem Bahnplanungsprogramm festgelegt, mit dem die Aufgabenstellung für den Roboter spezifiziert wird. Um daraus die Soll-Vorgaben für die Gelenkregler zu berechnen, müssen aus der Bewegung des Endeffektors die Minimalkoordinaten der Gelenke, sowie ihre erste und zweite Ableitung, berechnet werden. Diese Aufgabenstellung wird im folgenden behandelt.

Durch die Vorgabe der 3×3 Drehmatrix und des 3×1 Ortsvektors des Endeffektors bezüglich des Inertialsystems sind die 6 Minimalkoordinaten $\mathbf{q}$, bis auf Mehrdeutigkeiten der Roboterstellung, eindeutig festgelegt. Die 12 vorgegebenen Variablen, d.h. die 9 Elemente der Drehmatrix und die 3 Elemente des Ortsvektors, sind jedoch nicht unabhängig voneinander, da eine Drehmatrix durch drei voneinander unabhängige Koordinaten beschrieben werden kann. Die Vorgabe ist damit redundant und muß auf die 6 wesentlichen Gleichungen reduziert werden. Hierzu wird angenommen, daß die Vorgabematrix $\mathbf{T}_{des}$ nicht genau mit der Drehmatrix vom Endeffektor zum Inertialsystem ${}^0\mathbf{T}^6$ übereinstimmt. Die Differenz-Drehmatrix $\mathbf{T}_{diff}$ ergibt sich zu:

$$\begin{aligned}
\mathbf{T}_{diff} &= \mathbf{T}_{des}^T\ {}^0\mathbf{T}^6 && \text{(4.30a)}\\
&= \mathbf{n}\,\mathbf{n}^T + \left(\mathbf{E} - \mathbf{n}\,\mathbf{n}^T\right)\cos(\varphi) + \mathbf{skew}(\mathbf{n})\sin(\varphi) && \text{(4.30b)}\\
&\approx \mathbf{E} + \mathbf{skew}(\mathbf{n})\varphi \quad \text{(für kleines } \varphi\text{)} . && \text{(4.30c)}
\end{aligned}$$

Hierbei ist $\mathbf{n}$ die momentane Drehachse und φ der momentane Winkel um diese Drehachse, um damit das aktuelle, durch ${}^0\mathbf{T}^6$ gekennzeichnete Koordinatensystem in das durch $\mathbf{T}_{des}$ charakterisierte Koordinatensystem überzuführen (siehe z.B. [Robe88]). Wenn die beiden Koordinatensysteme schon dicht beieinander liegen, d.h. wenn der Winkel φ klein ist, kann $\mathbf{T}_{diff}$ linearisiert werden und es ergibt sich Gleichung (4.30c). Umgeformt folgt:

$$\mathbf{skew}(\mathbf{n})\varphi = \mathbf{T}_{des}^T\ {}^0\mathbf{T}^6 - \mathbf{E} . \tag{4.31}$$

Die Matrix $\mathbf{skew}(\mathbf{n})$ ist schiefsymmetrisch und hat nur drei wesentliche Elemente. Aus diesem Grund sind in dieser Lage die 3 voneinander unabhängigen Elemente gleich den 3 (oberen oder unteren) Nebendiagonalelementen der Matrix auf der rechten Seite des Gleichheitszeichens. Die 3 Diagonalelemente sind, bis zu Größen erster Ordnung entwickelt, Null. Sei $\mathbf{vec}^*$ identisch mit dem Operator $\mathbf{vec}$, wobei $\mathbf{vec}^*$ jedoch nicht nur auf schiefsymmetrische, sondern auch auf beliebige Matrizen angewendet werden kann. Dieser Operator bildet in derselben Weise wie $\mathbf{vec}$ aus den drei oberen Elementen einer Matrix einen Vektor. Damit können die 6 nicht-redundanten Vorgaben angegeben werden als:

$$\begin{aligned}
\mathbf{0} &= \mathbf{vec}^*\left(\mathbf{T}_{des}^T\ {}^0\mathbf{T}^6\right) && \text{(4.32a)}\\
{}^0\mathbf{r}_{des} &= {}^0\mathbf{r}^{06} , && \text{(4.32b)}
\end{aligned}$$

wobei die Einheitsmatrix $\mathbf{E}$, weil sie keine Außerdiagonal-Elemente besitzt, im Operator $\mathbf{vec}^*$ weggelassen werden kann.[10]

Werden die Gleichungen (4.32) zusätzlich zum Modell eingeführt, wird ein großes, dünnbesetztes, *nichtlineares* Gleichungssystem zur Bestimmung der Minimalkoordinaten der Gelenke $\mathbf{q}$ festgestellt. Das Gleichungssystem ist nichtlinear, da durch Drehgelenke der Sinus und Cosinus des Drehwinkels in den Drehmatrizen auftritt.

Mit dem in Kapitel 2.3 ab Seite 28 erläuterten Tearing-Verfahren für nichtlineare Systeme kann das obige Gleichungssystem in ein System von 6 nichtlinearen Gleichungen überführt werden. Aus Abschnitt 4.3.2 ab Seite 98 ist ja bekannt, daß bei gegebenen Gelenk-Minimalkoordinaten alle gewünschten Drehmatrizen und Ortsvektoren berechnet werden können. Deshalb sind $\mathbf{q}$ die Tearing-Variablen und die 6 Gleichungen (4.32) sind die Tearing-Gleichungen. Mittels Tearing wird das entsprechende nichtlineare Gleichungssystem zusammen mit Code zur Lösung dieses Systems durch ein Newton/Raphson-Verfahren erzeugt (siehe auch Seite 27).

Auf Geschwindigkeits- und Beschleunigungsebene wird ganz entsprechend vorgegangen. Hier gibt es keine Probleme mehr mit der Redundanz der Vorgaben. Man erhält jeweils ein großes, dünnbesetztes, *lineares* Gleichungssystem, das, entsprechend zu oben, mit den Tearing-Variablen $\dot{\mathbf{q}}$ bzw. $\ddot{\mathbf{q}}$ auf ein lineares System mit jeweils 6 Gleichungen transformiert werden kann. Diese beiden linearen Gleichungssysteme werden numerisch mit LINPACK Unterprogrammen gelöst.

Die obige Vorgehensweise kann in einem MKS-Programm effizienter gestaltet werden: Die beiden Gleichungssysteme auf Geschwindigkeits- und Beschleunigungsebene haben dieselbe Systemmatrix. Diese Matrix muß deswegen nur einmal berechnet werden, und es ist auch nur eine LU-Zerlegung notwendig, anstatt zwei wie oben. Weiterhin ist diese Systemmatrix eine angenährte Jacobi-Matrix für die 6 nichtlinearen Gleichungen auf Positionsebene und könnte damit vom nichtlinearen Löser beim Newton/Raphson Verfahren mitbenutzt werden.

Aufgrund der obigen Überlegungen wird beim Manutec Roboter das inverse kinematische Problem mit dem folgenden Modell gelöst:

model *rob4*
{benutze oben definiertes Robotermodell}
@robot.dym

{deklariere Eingangs- und Ausgangsgrößen}
input $Tdes(3,3),\ rdes(3),\ vdes(6),\ ades(6)$
output $q(6),\ qd(6),\ qdd(6)$
constant $c(3) = [0;0;0]$

{definiere Eingangs- und Ausgangsgrößen}
$c = \mathbf{vec}(Tdes' * robot::s.Tb)$
$rdes = robot::s.rb$
$Tdes' * vdes = robot::s.vb$
$Tdes' * ades = robot::s.ab$

$q = robot.q$
$qd = robot.qd$
$qdd = robot.qdd$

[10] Auf diese elegante Methode, nicht-redundante Vorgaben zu erhalten, hat mich Dr. Günter Leister, ehemals Mechanik B, Universität Stuttgart, aufmerksam gemacht.

end

{definiere Parameter, die nicht expandiert werden sollen}
variable not evaluate *robot.rL1*, *robot.rL2*, *robot.rL3*

{definiere Tearing-Variablen und Tearing-Gleichungen}
variable tear $q(1:3)$ [*c*]
variable tear $q(4:6)$ [*rdes*]
variable tear *qd* [*robot*::*s.vb*]
variable tear *qdd* [*robot*::*s.ab*]

4.3.6 Stationärwertberechnung

Bei der Stationärwertberechnung soll ein stationärer Zustand des Systems ermittelt werden, in dem die Ableitungen der Zustandsvariablen Null sind, d.h. das System in Ruhe ist. Bei einem System in Zustandsform

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) \\ \mathbf{y}(t) &= \mathbf{g}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) \end{aligned}$$

gibt es dabei zwei Aufgabenstellungen:

1. Gegeben: $t_0, \mathbf{u}(t_0), \dot{\mathbf{x}}(t_0) = 0$. Gesucht: $\mathbf{x}(t_0)$. Hierzu muß das folgende nichtlineare Gleichungssystem nach $\mathbf{x}$ gelöst werden:

$$\mathbf{0} = \mathbf{f}\left(t_0, \mathbf{x}, \mathbf{u}(t_0)\right) \ . \tag{4.33}$$

   Diese Aufgabenstellung ergibt sich dann, wenn um den Stationärwert $\mathbf{x}(t_0), \mathbf{u}(t_0)$ linearisiert werden soll. Man kann dieses Problem bei stabilen Systemen auch (uneffizienter) dadurch lösen, daß von einem Startwert ausgehend, "hinreichend" lange simuliert wird, bis das System im stationären Punkt zur Ruhe gekommen ist.

2. Gegeben: $t_0, \mathbf{y}(t_0), \dot{\mathbf{x}}(t_0) = 0$. Gesucht: $\mathbf{x}(t_0), \mathbf{u}(t_0)$. Hierzu muß das folgende nichtlineare Gleichungssystem nach $\mathbf{x}$ und $\mathbf{u}$ gelöst werden:

$$\begin{aligned} \mathbf{0} &= \mathbf{f}\left(t_0, \mathbf{x}, \mathbf{u}\right) && \text{(4.34a)} \\ \mathbf{y}(t_0) &= \mathbf{g}\left(t_0, \mathbf{x}, \mathbf{u}\right) \ . && \text{(4.34b)} \end{aligned}$$

   Für eine wohldefinierte Aufgabenstellung muß die Zahl der Ausgangsgrößen gleich der Zahl der Eingangsgrößen sein. Diese Problemformulierung ergibt sich aus der Forderung, daß die Ausgangsgrößen des Systems im eingeschwungenen Zustand einen vorgegebenen Wert annehmen sollen, und daß die hierfür notwendigen Stellgrößen $\mathbf{u}$ zu berechnen sind.

Bei Mehrkörpersystemen können beide Aufgabenstellungen als Spezialfälle der obigen Problemformulierungen auftreten. Es ist einfach, die erforderlichen nichtlinearen Gleichungssysteme zu erstellen. Zum Beispiel lautet die erste Aufgabenstellung wegen Gleichung (4.22) auf Seite 102:

$$\mathbf{0} = \mathbf{h}(\mathbf{q}, 0, t) + \boldsymbol{\lambda}^e \ . \tag{4.35}$$

Zur Ermittlung dieses Gleichungssystems wird wieder das (nichtlineare) Tearing-Verfahren eingesetzt. Hierzu werden die Gleichungen $\dot{\mathbf{q}} = \mathbf{0}, \ddot{\mathbf{q}} = \mathbf{0}$ zum MKS-Modell hinzugenommen. Wenn nun $\mathbf{q}$ bekannt wäre, könnten alle Größen und insbesondere $\boldsymbol{\lambda}^e$ berechnet werden, da dies gerade ein Spezialfall des inversen dynamischen Problems ist. Aus diesem Grund sind die Minimalkoordinaten der Gelenke $\mathbf{q}$ die Tearing-Variablen und die zugeordnete Gleichung (4.12) auf Seite 83 ist die entsprechende Tearing-Gleichung. Die Aufgabenstellung wird damit durch die folgende Vorgabe beschrieben:

$$\begin{aligned}&\dot{\mathbf{q}} = \mathbf{0}\\ &\ddot{\mathbf{q}} = \mathbf{0}\\ &\textbf{variable tear } \mathbf{q}\ [\boldsymbol{\lambda}^e]\end{aligned}$$

Wie schon in Kapitel 3 betont, gibt es bei jedem realitätsnahen Modell eines physikalischen Systems Unstetigkeiten irgendeiner Art. Traditionell werden diese Unstetigkeiten über eine Ereignisbehandlung erfaßt. Bei der Stationärwertberechnung gibt es aber dann Schwierigkeiten, da die obigen Aufgabenstellungen nur für rein kontinuierliche Systeme definiert sind.

Mit der in Kapitel 3 ab Seite 45 eingeführten, neuartigen Behandlung von ereignisabhängigen Modellen ist die Sachlage anders. Dort werden Unstetigkeiten nicht durch Ereignisse, sondern durch *Gleichungen* beschrieben. Diese Gleichungen werden genauso behandelt wie kontinuierliche Gleichungen. Insbesondere werden sie auch mit auf Blockdreiecksform transformiert. Der Unterschied zu kontinuierlichen Gleichungen besteht nur darin, daß die "diskreten" Gleichungen nur in bestimmten Situationen ausgewertet werden.

Die "diskreten" Gleichungen können in 4 unterschiedliche Typen eingeteilt werden:

1. *Unstetige* Gleichungen, beschrieben durch **if**-Ausdrücke der Form:

$$\begin{aligned}< \textit{expression} > = \ &\textbf{if} \quad\quad < \textit{condition } 1 > \textbf{ then } < \textit{expression } 1 >\\ &\textbf{else if} < \textit{condition } 2 > \textbf{ then } < \textit{expression } 2 >\\ &\ldots\\ &\textbf{else} \quad\quad\quad\quad\quad\quad\quad\quad < \textit{expression } n >\end{aligned}$$

2. *Instant*-Gleichungen, beschrieben durch **when**-Anweisungen, bei der die Bedingung nur von der Zeit abhängt:

$$\begin{aligned}&\textbf{when} < \textit{time-condition} > \textbf{ then}\\ &\quad\quad < \textit{equations} >\\ &\textbf{endwhen}\end{aligned}$$

3. *Boole'sche*-Gleichungen in der Form:

$$< \textit{Boolean variable} > = < \textit{Boolean expression} >$$

4. *Instant*-Gleichungen, beschrieben durch **when**-Anweisungen, mit einer beliebigen (nicht nur von der Zeit abhängigen) Bedingung:

$$\begin{aligned}&\textbf{when} < \textit{condition} > \textbf{ then}\\ &\quad\quad < \textit{equations} >\\ &\textbf{endwhen}\end{aligned}$$

Wenn nur die Gleichungstypen 1–3 in einem Modell auftreten, kann auch hier eine sinnvolle Aufgabenstellung für eine Stationärwertberechnung angegeben werden. Hierzu werden die "diskreten" Gleichungen wie beim Start einer Simulation behandelt:

1. Bei **if**-Ausdrücken und bei Boole'schen Gleichungen werden immer diejenigen Zweige verwendet, die aufgrund der aktuellen Bedingung zutreffen.

2. Die Gleichungen mit **when**-Anweisungen werden wie kontinuierliche Gleichungen behandelt, wenn die Zeitbedingung zutrifft. Ansonsten werden die Gleichungen ignoriert.

3. Für alle *diskreten* Zustandsvariablen $x1$, die in "Ruhe" sein sollen, werden zusätzliche Gleichungen der Form $\mathbf{new}(x1) = x1$ eingeführt. Damit wird gefordert, daß der neu berechnete Wert $\mathbf{new}(x1)$ gleich dem Vorgabewert $x1$ ist.

4. Für alle sonstigen diskreten und Boole'schen Zustandsvariablen $x2$ werden die normalerweise bei der Codegenerierung vom Compiler eingeführten Anweisungen der Form $x2 = \mathbf{new}(x2)$ nicht generiert, da $x2$ und $\mathbf{new}(x2)$ bei dieser Aufgabenstellung als voneinander unabhängige Variable betrachtet werden müssen.

Punkt 1 ist etwas kritisch, da hierdurch Unstetigkeiten in die Residuumberechnung des nichtlinearen Lösers eingeführt werden. Normalerweise setzt ein Löser voraus, daß diese Funktion einmal stetig differenzierbar ist. Für einen pragmatischen ersten Ansatz ist das Vorgehen jedoch akzeptabel. Besser wäre es natürlich, einen Löser so zu modifizieren, daß eine Unstetigkeit numerisch zuverlässig behandelt wird.

Das Vorgehen soll an einem einfachen Beispiel demonstriert werden. In Bild 4.9 ist das stark vereinfachte Modell eines Antriebsstrangs eines Roboters gezeigt. Die Strecke wird durch einen diskreten P-PI Kaskadenregler geregelt. Die Filterung des Tachosignals und das analoge Antialeasfilter des Reglers werden vernachlässigt. Die Aufgabe besteht darin, bei gegebenen Sollwerten $q_r, \dot{q}_r$, den Reglerzustand x zu ermitteln, so daß das Regelungssystem in Ruhe ist. Dies ist eine typische Aufgabenstellung, um einen definierten Zustand für

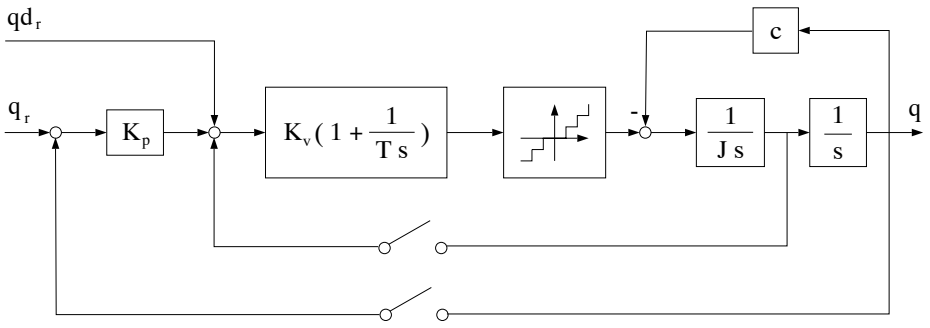


Bild 4.9: Digital geregelter Antriebsstrang.

den Start einer Simulation zu erhalten; sie könnte aber auch dazu verwendet werden, um im realen Regler den Reglerzustand beim Einschalten des Reglers zu bestimmen. Üblicherweise werden alle diskreten Zustandsvariablen beim Start einer Simulation auf Null gesetzt. Dann muß eine "gewisse" Zeit mit einem konstanten Sollwert simuliert werden, bis ein Ruhezustand, und damit ein definierter Ausgangspunkt, erreicht wird. Erst dann sollte die

eigentlich interessierende Simulation durchgeführt werden. Das Beispiel kann leicht auf einen allgemeinen Roboter mit einem digitalen Regelungssystem erweitert werden. Dann muß ein nichtlineares Gleichungssystem gelöst werden, in das kontinuierliche Modellteile und die diskreten Reglergleichungen eingehen.

Der einfache Antriebsstrang mit diskretem Regler kann durch das folgende Modell beschrieben werden:

```
model rob5
    input      rq, rqd
    parameter J, c, Kp, Kv , T, SampleTime
    local      q, qd, u, uv, x = 0, NextTime = 0

    {Strecke}
        der(q)      = qd
        J * der(qd) = u - c * q

    {Regler}
        when Time >= NextTime then
            new(NextTime) = Time + SampleTime
            uv            = (rqd - qd) + Kp * (rq - q)
            new(x)        = uv/T
            u             = Kv * (x + uv)
        endwhen
end
```

Der diskrete Regler wurde in Regelungs-Normalform angeschrieben. Die Gleichungen für die Stationärwert-Ermittlung werden mit der folgenden Modell- und Aufgabenbeschreibung erzeugt:

```
model stat
    {benutze Antriebsstrang-Modell}
        @rob5.dym

    {definiere Aufgabenstellung}
        der(rob5.q)  = 0
        der(rob5.qd) = 0
        Time         = 0

        when Time >= rob5.NextTime then
            new(rob5.x) = rob5.x
        endwhen
end

{definiere, daß q, qd keine Zustandsvariablen mehr sind}
    variable unknown rob5.q, rob5.qd
```

Da die Ableitungen der Variablen q und qd auftreten, nimmt Dymola standardmäßig an, daß diese Größen Zustandsvariablen und damit *bekannt* sind. Durch die Anweisung "**variable unkown** ..." wird festgelegt, daß entgegen der Voreinstellung beide Variablen unbekannt sind. Bei der Transformation auf Blockdreiecksform ergibt sich folgendes lineare Gleichungssystem in den 4 unbekannten Variablen q, uv, x, u:

```
| stat.      0    = u - c*[q]
|            [uv] = rqd + Kp*(rq - q)
|            [x]  = uv/T
|            [u]  = Kv*(x + uv)
```

Das lineare Gleichungssystem kann problemlos symbolisch oder numerisch aufgelöst werden. Die Determinate des Systems ist "Kp*(Kv/T + Kv) + c". Da alle Systemkonstanten positiv sind, ist diese Determinante immer ungleich Null und das obige Gleichungssystem ist eindeutig lösbar. Werden die Zustandsgrößen q und x beim Start der Simulation auf die berechneten stationären Werte gesetzt, bleibt das System in Ruhe, wenn sich die Eingangsgrößen *rq* und *rqd* nicht ändern. Für dieses einfache System kann der Stationärwert natürlich auch leicht mit Hand ausgerechnet werden. Mit dem obigen Verfahren kann man jedoch für jedes beliebige System den Stationärwert berechnen, auch wenn zusätzliche Abtastsysteme vorhanden sind.

4.4 Effiziente Bewegungsgleichungen für baumstrukturierte Systeme

Die wichtigste Aufgabenstellung bei Mehrkörpersystemen ist die Simulation eines MKS-Modells. In Abschitt 4.3.4 ab Seite 101 wurde gezeigt, wie mit dem objektorientierten MKS-Modell die Gleichungen für ein Simulationsmodell erzeugt werden können. Diese Vorgehensweise hat (bis jetzt noch) den Nachteil, daß immer die Tearing-Variablen und die Tearing-Gleichungen angegeben werden müssen. Außerdem werden bei der Lösung die Struktureigenschaften der Massenmatrix nicht ausgenutzt. In diesem Abschnitt wird gezeigt, wie die Bewegungsgleichungen von baumstrukturierten, strukturvariablen MKS ohne diese Nachteile effizient erstellt werden können.

Seit einiger Zeit gibt es eine neue Algorithmenklasse für Mehrkörpersysteme, die nicht mehr das "klassische" Gleichungssystem (4.22) auf Seite 102 mit der Massenmatrix erzeugt, sondern direkt die gewünschten verallgemeinerten Beschleunigungen $\ddot{\mathbf{q}}$ berechnet [Vere74, Arms79, Feat83, Bran86, Bae87, Weha88]. Für baumstrukturierte Mehrkörpersysteme sind diese Algorithmen besonders gut geeignet, da der Verfahrensaufwand hier proportional zu $O(n)$ ist, mit n gleich der Zahl der Freiheitsgrade des Systems. Im Gegensatz dazu sind alle Verfahren, die auf (4.22) basieren, proportional zu $O(n^3)$, da ein lineares Gleichungssystem mit n Gleichungen gelöst werden muß.

Es ist prinzipiell möglich ein $O(n)$-Verfahren direkt aus dem objektorientierten MKS-Modell in Abschnitt 4.2 zu erhalten. Hierzu muß ein spezieller "*Sparse Matrix Solver*" zum Lösen eines dünnbesetzten linearen Gleichungssystems eingesetzt werden, der nach einer "*frontal method*" [Duff86] arbeitet. In diesem Zusammenhang gibt es hier noch einige ungeklärte Fragen. Aus diesem Grund wird ein pragmatischer Weg beschritten, um die attraktive $O(n)$ Verfahrensklasse benutzen zu können.

Alle Klassen der MKS-Bibliothek werden mit (fast) identischen Objekt-Schnittstellen benutzt. Die Gleichungen in einer Klasse werden jedoch etwas umformuliert. Damit der Anwender diese Klassen in derselben Weise benutzen kann wie die bisherige MKS-Bibliothek *mbs.lib*, behalten die modifizierten Klassen ihre Namen bei, werden jedoch in einer neuen MKS-Bibliothek *mbssim.lib* abgelegt. Man kann deswegen mit demselben MKS-Modell unterschiedliche Gleichungen erzeugen, je nachdem welche der beiden Bibliotheken benutzt wird. Nachteilig ist, daß die neue Bibliothek *nur* zum Aufstellen der Bewegungsgleichungen und nicht für andere Aufgabenstellungen benutzt werden kann, im Gegensatz zur Bibliothek *mbs.lib*. Sie verstößt damit etwas gegen die deklarative Philosophie objektorientierter

Modellierungssprachen.

Auf den ersten Blick scheint es schwierig zu sein, einen MKS-Algorithmus mit einem fachneutralen Werkzeug wie Dymola auszudrücken, da eine objektorientierte Modellierungssprache nur lokale, voneinander unabhängige Objekte kennt, die über ihre Schnittstellen zusammengeschaltet werden. Notwendig ist deshalb eine objektorientierte Umformulierung des funktionalen MKS-Algorithmus. Erstaunlicherweise ist das auf einfache Weise möglich und führt zu einer verständlichen Formulierung der $O(n)$ Methode.

Die wesentliche Idee besteht darin, *alle* Schnittkräfte und Schnittmomente $\hat{\mathbf{f}}$ im MKS als *lineare* Funktionen der am Schnitt auftretenden Beschleunigung und Winkelbeschleunigung $\hat{\mathbf{a}}$ angegeben zu können, d.h.

$$\hat{\mathbf{f}} = \mathbf{I}\hat{\mathbf{a}} + \mathbf{b} \;\; ; \qquad \mathbf{I} = \mathbf{I}^T \; . \tag{4.36}$$

Dabei wird vorausgesetzt, daß weder $\mathbf{I}$ noch $\mathbf{b}$ selbst Funktionen der Beschleunigungen sind. Anstatt nun wie bisher $\hat{\mathbf{f}}$ im Cut eines Objekts aufzuführen, können stattdessen $\mathbf{I}$ und $\mathbf{b}$ benutzt werden, da mit der gleichzeitig im Cut übertragenen Beschleunigung und Winkelbeschleunigung $\hat{\mathbf{a}}$ die Schnittkraft und das Schnittmoment $\hat{\mathbf{f}}$ jederzeit berechnet werden können. Im folgenden wird bewiesen, daß die Beziehung (4.36) richtig ist.

Die Beziehung (4.36) trifft auf jeden Fall auf die Klassen *Sensor, Force* und *Body* zu. Die Schnittkräfte und Schnittmomente bei einem *Sensor*-Objekt sind Null, d.h. $\mathbf{I} = \mathbf{0}, \mathbf{b} = \mathbf{0}$. Die Gleichungen für Kraftelemente sind nur Funktionen von Positionen und Geschwindigkeiten, nicht aber von Beschleunigungen. Bei einem Objekt der Klasse *Force* ist also $\mathbf{I} = \mathbf{0}$ und $\mathbf{b}$ ist das Kraftgesetz. Die Gleichungen für ein Objekt der Klasse *Body* sind schon in der geforderten Form, siehe Gleichung (4.19) auf Seite 91. Es muß demnach nur noch gezeigt werden, daß (4.36) auch für Gelenke gültig ist.

Es wird vorerst angenommen, daß die Schnittkraft und das Schnittmoment am Cut b eines Gelenks in der Form (4.36) dargestellt werden können. Entsprechend zu einem Vorschlag von Wehage [Weha88] kann man dann die Gleichungen (4.5g, 4.9d, 4.15), die den Cut b eines Gelenks beschreiben, zusammen mit (4.36) in der folgenden linearen, symmetrischen Matrix-Gleichung zusammenfassen:

$$\begin{bmatrix} \mathbf{I}^b & -\mathbf{E} & \mathbf{0} & \mathbf{0} \\ -\mathbf{E} & \mathbf{0} & \boldsymbol{\Phi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Phi}^T & \mathbf{0} & \mathbf{L} \\ \mathbf{0} & \mathbf{0} & \mathbf{L} & \mathbf{E}-\mathbf{L} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{a}}^b \\ \hat{\mathbf{f}}^b \\ \ddot{\mathbf{q}} \\ \boldsymbol{\lambda}^l \end{bmatrix} = \begin{bmatrix} -\mathbf{b}^b \\ -\mathbf{C}\hat{\mathbf{a}}^a - \boldsymbol{\xi} - \boldsymbol{\zeta} \\ -\boldsymbol{\lambda}^e \\ \mathbf{0} \end{bmatrix} \; . \tag{4.37}$$

Gleichung (4.37) gibt an, daß die unbekannten Variablen des Cut-Frames b, d.h. $\hat{\mathbf{a}}^b$, $\hat{\mathbf{f}}^b$, $\ddot{\mathbf{q}}$, $\boldsymbol{\lambda}^l$, als lineare Funktionen der unbekannten Beschleunigung und Winkelbeschleunigung $\hat{\mathbf{a}}^a$ des Cut-Frames a angegeben werden können. Wird (4.37) explizit nach den unbekannten Variablen des Cut-Frames b aufgelöst, ergibt sich:

$$\boldsymbol{\lambda}^l = -\mathbf{L}^*\mathbf{h} \tag{4.38a}$$

$$\ddot{\mathbf{q}} = -\mathbf{M}^{-1}(\mathbf{E}-\mathbf{L}^*)\mathbf{h} \tag{4.38b}$$

$$\hat{\mathbf{a}}^b = \mathbf{C}\hat{\mathbf{a}}^a + \boldsymbol{\Phi}\ddot{\mathbf{q}} + \boldsymbol{\eta} \tag{4.38c}$$

$$\hat{\mathbf{f}}^b = \mathbf{I}^b\hat{\mathbf{a}}^b + \mathbf{b}^b \tag{4.38d}$$

mit

$$\begin{aligned}
\mathbf{h} &= \left(\boldsymbol{\Phi}^T \mathbf{I}^b \mathbf{C}\right) \hat{\mathbf{a}}^a + \left(\boldsymbol{\Phi}^T \mathbf{I}^b \mathbf{C}\boldsymbol{\eta} + \boldsymbol{\lambda}^e + \boldsymbol{\Phi}^T \mathbf{b}^b\right) && \text{(4.39a)}\\
\mathbf{M} &= \boldsymbol{\Phi}^T \mathbf{I}^b \boldsymbol{\Phi} && \text{(4.39b)}\\
\boldsymbol{\eta} &= \boldsymbol{\xi} + \boldsymbol{\zeta} && \text{(4.39c)}\\
\mathbf{L}^* &= \left(\mathbf{L}\mathbf{M}^{-1}\mathbf{L} + \mathbf{L} - \mathbf{E}\right)^{-1} \mathbf{L}\mathbf{M}^{-1} \quad = \quad \mathbf{L} \text{ if } dim(\mathbf{L}) = 1 \times 1 \ . && \text{(4.39d)}
\end{aligned}$$

Werden diese Gleichungen in Gleichung (4.5h), d.h. die Gleichung "$\mathbf{0} = \hat{\mathbf{f}}^a + \mathbf{C}^T \hat{\mathbf{f}}^b$", eingesetzt, ergibt sich der angenommene Zusammenhang (4.36), da $\hat{\mathbf{f}}^a$ linear von $\hat{\mathbf{f}}^b$, $\hat{\mathbf{f}}^b$ linear von $\hat{\mathbf{a}}^b$ wegen (4.38d) und $\hat{\mathbf{a}}^b$ linear von $\hat{\mathbf{a}}^a$ wegen (4.38c, 4.38b, 4.39a) abhängen. Nach einer kurzen Zwischenrechnung folgt:

$$\hat{\mathbf{f}}^a = \mathbf{I}^a \hat{\mathbf{a}}^a + \mathbf{b}^a \ ; \qquad \mathbf{I}^a = \mathbf{I}^{a^T} \tag{4.40}$$

mit

$$\begin{aligned}
\mathbf{I}^a &= -\mathbf{C}^T \mathbf{N} \mathbf{C} && \text{(4.41a)}\\
\mathbf{b}^a &= -\mathbf{C}^T \left(\mathbf{b}^b + \mathbf{N}\boldsymbol{\eta} - \mathbf{I}^b \boldsymbol{\Phi} \mathbf{M}^{-1} (\mathbf{E} - \mathbf{L}^*) \left(\boldsymbol{\lambda}^e + \boldsymbol{\Phi}^T \mathbf{b}^b\right)\right) && \text{(4.41b)}\\
\mathbf{N} &= \mathbf{I}^b - \mathbf{I}^b \boldsymbol{\Phi} \mathbf{M}^{-1} \left(\mathbf{E} - \mathbf{L}^*\right) \boldsymbol{\Phi}^T \mathbf{I}^b && \text{(4.41c)}\\
\mathbf{M} &= \boldsymbol{\Phi}^T \mathbf{I}^b \boldsymbol{\Phi} && \text{(4.41d)}\\
\mathbf{L}^* &= \left(\mathbf{L}\mathbf{M}^{-1}\mathbf{L} + \mathbf{L} - \mathbf{E}\right)^{-1} \mathbf{L}\mathbf{M}^{-1} \quad = \quad \mathbf{L} \text{ if } dim(\mathbf{L}) = 1 \times 1 \ . && \text{(4.41e)}
\end{aligned}$$

Wenn jetzt alle Objekte der Klassen *Sensor, Force* und *Body* gedanklich entfernt werden, so können die Schnittkräfte und Schnittmomente dieser Objekte am restlichen Teil des MKS in der Form (4.36) dargestellt werden. Der übrigbleibende Teil des MKS enthält nur noch Gelenk-Objekte. Die in den "Blättern" des MKS-Baums befindlichen Schnittkräfte und Schnittmomente am Cut b der Gelenke können jetzt auch in der Form (4.36) dargestellt werden, da die linearen Faktoren $\mathbf{I}$ und $\mathbf{b}$ der Schnittkräfte und Schnittmomente Through-Variablen sind und alle sonstigen Schnittkräfte und Schnittmomente an den Blättern schon in dieser Form sind. Mit (4.40) gilt dasselbe für den Cut a dieser Gelenk-Objekte. Jetzt werden die gerade eben betrachteten Blatt-Objekte entfernt. Dann liegt wieder dieselbe Situation vor, d.h. durch eine Rückwärtsrekursion von den "Blättern" zur "Wurzel" ergibt sich, daß (4.36) für alle Cuts zutreffend ist. Q.E.D.

In der neuen MKS-Bibliothek werden die Schnittkräfte und Schnittmomente in allen Cuts durch ihre linearen Faktoren ersetzt. Zusätzlich werden die betreffenden Gleichungen durch die oben angegebenen Gleichungen ersetzt. Der wesentliche Vorteil dieser Neu-Formulierung liegt darin, daß die Gleichungen eines MKS jetzt auf eine *explizit* lösbare Blockdreiecksstruktur transformiert werden können! Dies sieht man folgendermaßen ein:

Durch die Vorwärtskinematik können alle Positionen und Geschwindigkeiten ermittelt werden. Danach können die linearen Faktoren $\mathbf{I}$ und $\mathbf{b}$ bei allen Objekten der Klassen *Sensor, Force* und *Body* berechnet werden, da diese nur von Positionen und Geschwindigkeiten abhängen. In derselben Weise, wie beim obigen Beweis der Eigenschaft (4.36), ergeben sich über eine Rückwärtsrekursion die linearen Faktoren aller anderen Cuts. Hierbei müssen die Gelenkmatrizen $\mathbf{M}$ wegen (4.41d) invertiert werden. Wenn ein Gelenk nur einen Freiheitsgrad besitzt, ist das eine einfache Division. Die Rückwärtsrekursion ist beendet, wenn das Inertialsystem erreicht ist. Die Beschleunigung und Winkelbeschleunigung des Inertialsystems sind bekannt. Dann können mit (4.38) die noch verbleibenden Unbekannten aller

Objekte ermittelt werden, die direkt mit dem Inertialsystem verbunden sind. Jetzt sind die Beschleunigungen dieser Objekte an beiden Cuts bekannt, und mit (4.38) können die unbekannten Variablen der darauffolgenden Objekte bestimmt werden. Auf diese Weise ergibt sich eine Vorwärtsrekursion von der "Wurzel" zu den "Blättern", in deren Verlauf alle noch verbleibenden Unbekannten berechnet werden können.

Der Algorithmus ist $O(n)$, wobei n die Anzahl der MKS-Elemente ist, da die Gleichungen für jedes MKS-Element genau einmal ausgewertet werden. Das Verfahren ist äquivalent zum Algorithmus von Brandl, Johanni und Otter [Bran86], wurde hier jedoch auf eine objektorientierte Weise abgeleitet. Im Vergleich zu allen bisher veröffentlichten $O(n)$-Verfahren, gibt es noch eine entscheidende Verbesserung: Durch die Einbeziehung der Matrix $\mathbf{L}$ sind die Gleichungen auch für strukturvariable Systeme gültig, bei denen Freiheitsgrade eines Gelenks während einer Simulation "blockieren" können.

Erstaunlicherweise beeinflußt das "Blockieren" bzw. "Wiederfreigeben" eines Gelenkfreiheitsgrads die Effizienz praktisch überhaupt nicht. Dies sieht man am einfachsten an einem Gelenk mit einem Freiheitsgrad. In diesem Fall sind $\mathbf{M}$ und $\mathbf{L}$ skalare Variablen und es müssen nur die zusätzlichen Terme $\mathbf{M}^{-1}(\mathbf{E}-\mathbf{L})$ sowie $-\mathbf{Lh}$ berechnet werden. Dies sind 2 Multiplikationen und 1 Subtraktion pro Gelenk, also eine vernachlässigbare Effizienz-Einbuße[11]. Im Gegensatz hierzu ergibt sich bei der Transformation auf die traditionelle Form (4.22, 4.29) eine doppelt so große Massenmatrix, was zu einer deutlichen Erhöhung der Zahl an Operationen[12] führt.

Das Auftreten der Matrix $\mathbf{L}$ im $O(n)$ Algorithmus kann leicht interpretiert werden: Angenommen kein Freiheitsgrad eines Gelenks ist gesperrt, dann ist $\mathbf{L}=\mathbf{0}$ und damit auch $\mathbf{L}^*=\mathbf{0}$. In diesem Fall sind (4.41) die üblichen Rekursionsformeln eines Gelenks. Wenn jedoch alle Freiheitsgrade in einem Gelenk gesperrt sind, dann ist $\mathbf{L}=\mathbf{E}$ und damit auch $\mathbf{L}^*=\mathbf{E}$ und die Gleichungen (4.41) reduzieren sich auf die Rekursionsformeln für ein Gelenk mit Null Freiheitsgraden. Anschaulich bedeutet dies: Bei der Rekursion von einem Gelenk zum nächsten gehen nur die linearen Faktoren ein. *Wie* diese linearen Faktoren berechnet wurden ist unerheblich. Damit kann zu jedem Zeitpunkt ein Gelenktyp einfach durch einen anderen Gelenktyp ersetzt werden. Abgesehen von dieser lokalen Modifikation, wird die restliche Rekursionsvorschrift durch diese Änderung nicht beeinflußt.

4.5 Mehrkörpersysteme mit Reibung

Im letzten Abschnitt wurde gezeigt, wie *strukturvariable* Mehrkörpersysteme effizient behandelt werden können. Diese Eigenschaft wird nun benutzt, um Mehrkörpersysteme zu modellieren, bei denen Reibung in vielen Komponenten auftritt. Damit wird es möglich Reibung in Lagern und Getrieben, Reib-Kupplungen und Reib-Bremsen auf einfache Weise zu beschreiben. Aufgrund der Bedeutung, die Reibungseffekte in technischen Anwendungen haben, gibt es zu diesem Themenbereich eine große Zahl von Veröffentlichungen. Insbesondere gibt es dazu mit der Tribologie ein eigenes Fachgebiet, das sich vorwiegend mit detaillierten quasi-statischen Fragestellungen beschäftigt.

[11] In der MKS-Bibliothek werden diese Terme durch **if**-Ausdrücke ersetzt.

[12] Man kann die Inversion der "vergrößerten" Massenmatrix auf die Inversion einer Matrix mit einer Dimension $\leq f$ zurückführen. Dies erfordert jedoch eine spezielle Strategie mit Umkopieren von Daten und ist wesentlich komplizierter, als die einfache Hinzunahme der beiden Terme im $O(n)$ Algorithmus.

Im Zusammenhang mit Mehrkörpersystemen können zwei Problemstellungen unterschieden werden. Zum einen muß der Effekt der Reibung zwischen zwei aufeinander gleitenden Oberflächen verstanden und modelliert werden. Zum anderen muß die dynamische Kopplung berücksichtigt werden, die auftritt, wenn mehrere Reibelemente im selben System vorhanden sind. In der Literatur wird immer davon ausgegangen, daß Coulomb'sche Reibung mit Hilfe von 3 Schaltzuständen (gleiten vorwärts, gleiten rückwärts, haften) beschrieben werden kann, siehe z.B. [Cell79, Mitc90, Eich92, Gloc93]. Es wird hier gezeigt, daß dies nicht genügt, und daß 5 Schaltzustände für eine in jeder Situation korrekte Simulation benötigt werden. Weiterhin wird die dynamische Kopplung über eine Iterationsvorschrift behandelt, die eine neue Alternative darstellt zu den vor kurzem veröffentlichten Ansätzen [Gloc93, Gloc93a], in denen die dynamische Kopplung durch Transformation auf ein Komplementaritätsproblem behandelt wird.

Ein einfaches Reibmodell ist in Bild 4.10 zu sehen. Auf der Abszisse ist die Relativgeschwindigkeit v_{rel} zwischen den aufeinander gleitenden Oberflächen aufgetragen. Die Ordinate gibt den Wert der Reibkraft an, die tangential zu den Oberflächen wirkt. Wenn die Relativgeschwindigkeit nicht verschwindet, ist die Reibkraft eine Funktion der Geschwindigkeit, die von der Beschaffenheit der Oberflächen und dem benutzten Schmiermittel abhängt. Verschwindet die Relativgeschwindigkeit so tritt "Haften" ein, d.h. die beiden vorher aufeinandergleitenden Körper sind nun fest zueinander fixiert. Die zwischen den beiden Körpern auftretende Zwangskraft wird durch das Bewegungsverhalten des gesamten Systems bestimmt. Wenn der Betrag der Zwangskraft die maximale statische Haftreibkraft R_{max} übersteigt, so tritt wieder Gleiten ein. Hierbei vermindert sich die Reibkraft (näherungsweise) unstetig auf die kinetische Reibkraft R_v.

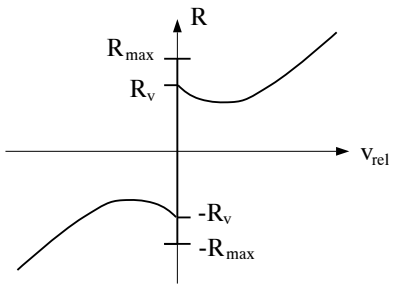


Bild 4.10: Einfaches Reibmodell.

Eine gute Einführung in reibungsbehaftete mechanische Systeme, zusammen mit einer umfangreichen Literaturliste, ist in [Arms91] zu finden. Dort werden auch detailliertere Reibmodelle behandelt. Ein neues, kompaktes Reibmodell, welches alle bekannten Effekte zu vereinen scheint, wurde vor kurzem in [Canu93] entwickelt. Drei Reibeffekte sind hier erwähnenswert:

- Die beiden Körper sind in der Haftphase nicht vollkommen zueinander fixiert. Stattdessen sind kleine Bewegungen möglich, die proportional zur Relativverschiebung sind und durch eine Feder mit steiler Kennlinie beschrieben werden können. Dieser sogenannte Dahl-Effekt wird in der Regel vernachlässigt, so wie Elastizitäten in ähnlichen Größenordnungen in den Gelenken oder Armen auch nicht berücksichtigt werden. Soll der Dahl-Effekt berücksichtigt werden, so führt dies aufgrund der steifen Feder auf ein steifes Differentialgleichungssystem und damit zu erhöhten Rechenzeiten.

- Die Größe der statischen Haftreibkraft R_{max} hängt von der Vorgeschichte, der sogenannten Dwell-Zeit, ab. Die Dwell-Zeit ist die Zeit zwischen dem Eintreten und dem Verlassen der Haftphase. Bei sehr kleiner Dwell-Zeit ist die statische Haftreibkraft gleich der kinetischen Reibkraft ($R_{max} = R_v$), d.h. es tritt dann keine unstetige Änderung der Reibkraft beim Übergang vom Haften zum Gleiten auf. Dieser Effekt kann relativ einfach in die folgende Reibmodellierung mit aufgenommen werden, indem R_{max} nicht als Konstante, sondern als zeitabhängige Größe behandelt wird. In [Arms91] wird die Formel

$$R_{max}(t_d) = R_{max_\infty} - (R_{max_\infty} - R_v)e^{-\gamma t_d^m} \tag{4.42}$$

  angegeben. Hierbei ist R_{max_∞} die Haftreibkraft bei sehr großer Dwellzeit, t_d ist die Dwellzeit und γ und m sind empirische Parameter.

- Neben der Relativgeschwindigkeit hängt die Reibkraft noch linear von der Normalkraft f_N ab, die senkrecht zur Gleitrichtung wirkt, d.h. $R = \mu(v_{rel})f_N$. Die Normalkraft kann in zwei Anteile aufgespaltet werden: Eine konstante Kraft, die aufgrund einer Vorspannung der beiden Körper zueinander, oder auf Grund anderer technischer Gegebenheiten, vorliegt (z.B. Anpreßkraft einer Reibkupplung oder einer Bremse), sowie eine dynamisch veränderliche Kraft, die sich aufgrund der Bewegung des mechanischen Systems ergibt. In Antriebssträngen, Kupplungen und Bremsen kann die dynamische Normalkraft oft gegenüber der Vorspannkraft vernachlässigt werden, siehe auch [Arms91]. Dies vereinfacht die Modellierung wesentlich und wird im folgenden vorausgesetzt. Es gibt aber eine Reihe von Problemen bei denen diese Annahme nicht zutrifft. Zum Beispiel wird die Normalkraft beim Einführen eines Bolzens in eine Bohrung allein durch die Bewegung des Mehrkörpersystems bestimmt und eine konstante statische Vorspannkraft ist nicht vorhanden. Solche Systeme können mit der unten besprochenen Vorgehensweise nicht direkt behandelt werden; siehe hierzu [Seyf92, Gloc93].

Das Hauptproblem bei der Behandlung von Reibung liegt in der variablen Struktur des Reibelements. Diese kann als das Blockieren und Wiederfreigeben von Gelenkfreiheitsgraden interpretiert werden. In der Gleitphase liegt ein Gelenk vor, das eine Relativbewegung in der Gleitrichtung erlaubt. In der Haftphase ist diese Relativbewegung gesperrt und der Bewegungsfreiheitsgrad des Gelenks verschwindet.

Eine oft benutzte Methode, um die Schwierigkeiten eines strukturvariablen Systems zu vermeiden, besteht darin, den senkrechten Teil der Reibkennlinie durch eine steile Kennlinie zu ersetzen. Durch eine solche Kennlinie werden aber Schwingungen erzeugt, die eigentlich im System nicht enthalten sind. Die Kennlinie wird deshalb so gewählt, daß diese Schwingungen möglichst gering sind. Durch die Steilheit der Kennlinie wird das Differentialgleichungssystem steif, was zu stark erhöhten Rechenzeiten führen kann. Dazu wurde in [Mitc90] am Beispiel eines Antriebsstrangs mit Reibkupplung gezeigt, daß sich die Rechenzeiten um mehr als den Faktor 10 erhöhen, wenn statt der nachfolgend beschriebenen Methode, die Kennlinien-Methode benutzt wird.

Es ist schon seit langem bekannt, wie *ein* Reibelement mit Hilfe von Zustandsereignissen numerisch zuverlässig und effizient behandelt werden kann [Cell79, Mitc90, Eich92]. Hierzu wird die Kennlinie von Bild 4.10 in drei Bereiche aufgeteilt: $v_{rel} < 0, v_{rel} = 0, v_{rel} > 0$.

Wenn sich das Reibelement in der Gleitphase befindet, wird die Relativgeschwindigkeit v_{rel} als Indikatorfunktion benutzt (siehe auch Kapitel 3.1 ab Seite 45). Ein Ereignis tritt ein, wenn die Relativgeschwindigkeit Null wird. Dieser Zeitpunkt muß vom Integrator genau ermittelt werden, um die Integration dann an dieser Stelle anzuhalten. Dort wird das Modell vom Gleitbereich in den Haftbereich umgeschaltet und die Integration wird neu gestartet. In der Haftphase werden die beiden Haftkraftreserven "$R_{max} - f_{reib}$" und "$-R_{max} - f_{reib}$" als Indikatorfunktionen benutzt. Wenn eine der beiden Funktionen das Vorzeichen wechselt, ist die maximale statische Haftreibkraft erreicht und die Integration wird wiederum angehalten. Je nachdem, welche der beiden Indikatorfunktionen das Vorzeichen wechselt, wird das Modell entweder in den Vorwärts-Gleitbereich oder in den Rückwärts-Gleitbereich umgeschaltet.

Diese Vorgehensweise ist schon wesentlich besser als die Kennlinien-Methode, da in keinem der Integrationsbereiche unstetige Funktionen auftreten und da auch keine künstlichen Steifigkeiten in das Modell eingeführt werden. Drei Problemfelder führen aber dazu, daß die Simulation von Systemen mit Reibung immer noch nicht befriedigend gelöst ist und deswegen keineswegs nur eine Standardaufgabenstellung ist:

1. Zum einen ist der Übergang vom Haften zum Gleiten kritisch, da nach dem Umschalten die Relativgeschwindigkeit immer noch Null ist. Im Gleitbereich wird aber der Null-Durchgang der Relativgeschwindigkeit als Indikator zum Schalten vom Gleiten zum Haften benutzt. Diese Indikatorfunktion ist aber zu Beginn der Integration schon Null. Wie auf Seite 48 erwähnt, führt z.B. der Integrator DASSLRT in diesem Fall zunächst einen kleinen Schritt aus, um ein eindeutiges Vorzeichen der Indikatorfunktion zu erhalten. Wenn die Indikatorfunktion Null bleibt, bricht der Integrator mit einer Fehlermeldung ab. Dieser Fall, daß die Indikatorfunktion Null bleibt, kann ohne weiteres auftreten. Der kleine Schritt könnte auch schon wieder zu groß sein, weil das System vom Gleit- in den Haftbereich zurückgeht. Eine weitere Schwierigkeit ergibt sich, wenn im Gleitbereich die Beschleunigung groß genug ist, damit ein Reibelement direkt vom Vorwärtsgleiten in das Rückwärtsgleiten schalten kann. Mit der obigen Strategie wird dagegen nur vom Vorwärtsgleiten in das Haften geschaltet. Nach dem Neustart der Integration ist die Haftkraftreserve dann sofort überschritten. Dies führt jedoch *nicht* zu einem Schalten in das Rückwärtsgleiten, da kein Nulldurchgang einer Indikatorfunktion auftritt. Das Reibelement ist dann in einem falschen Schaltzustand.

2. Generell gibt es Schwierigkeiten, wenn in einem Modell mehrere Ereignisse zum selben Zeitpunkt auftreten können. Dies ist z.B. der Fall, wenn mehrere Reibelemente vorhanden sind oder wenn durch eine digitale Steuerung Stellgrößen unstetig aufgebracht werden. Dann kommt es bei den Ereignisaktionen auf die Reihenfolge der Auswertung an. Weiterhin kann folgende Situation auftreten: Beim Übergang vom Gleiten zum Haften bei einem Reibelement 1 wird in der Regel die Relativbeschleunigung unstetig auf Null geändert. Durch diese Unstetigkeit ändern sich auch die Zwangskräfte und Zwangsmomente in allen anderen Gelenken unstetig. Dies kann dazu führen, daß die Haftkraftreserve eines anderen Reibelements 2 überschritten wird und dieses Reibelement in den Gleitzustand schalten kann, *bevor* die Integration wieder gestartet wird. Wenn dies nicht erfolgt, beginnt der Integrations-Neustart mit einer überschrittenen Haftkraftreserve. Dies führt jedoch wiederum *nicht* zu einem Schalten in den Gleitbereich, da kein Nulldurchgang einer Indikatorfunktion eintritt.

3. Um Reibelemente mit der obigen Methode behandeln zu können, muß das Blockieren und Freigeben von Gelenkfreiheitsgraden möglich sein. Bei kommerziellen Mehrkörperprogrammen wird so etwas zur Zeit nicht unterstützt. Wenn Reibelemente in einer allgemeinen Simulationsumgebung, wie ACSL, modelliert werden, so werden meist alle zu den Schaltstrukturen möglichen Modelle als unterschiedliche Modelle eingebracht, und es wird zwischen diesen Modellen umgeschaltet, siehe z.B. [Mitc90]. Dies ist aber nur bei sehr einfachen mechanischen Systemen praktikabel.

Im folgenden werden Lösungen für diese Problembereiche vorgeschlagen. Die wesentliche Grundidee, um die unter Punkt 1 aufgeführten Schwierigkeiten zu vermeiden, besteht darin, zwei neue Schaltbereiche hinzuzunehmen und damit 5 Schaltzustände zu betrachten: Hierbei werden die *Übergänge* zwischen Haften und Vorwärtsgleiten, sowie zwischen Haften und Rückwärtsgleiten, als neue Bereiche eingeführt. Diese Bereiche sind dadurch definiert, daß die Relativgeschwindigkeit Null ist, der im Haftbereich gesperrte Freiheitsgrad aber wieder freigegeben ist. Dies bedeutet, daß die Reibkraft den vorgegebenen Wert $\pm R_v$ besitzt und die Relativbeschleunigung, je nach Zustand, entweder größer oder kleiner Null ist. Den kompletten Schaltvorgang kann man jetzt, im Gegensatz zur obigen dreiwertigen Schaltlogik, als deterministischen endlichen Automaten (abgekürzt: DEA) behandeln, wie er auf Seite 62 besprochen wurde. Das Übergangsdiagramm dieses DEA ist in Bild 4.11 zu sehen. Ausgehend von einem Zustand des DEA, z.B. von *Forward*, geben die Relationen an den

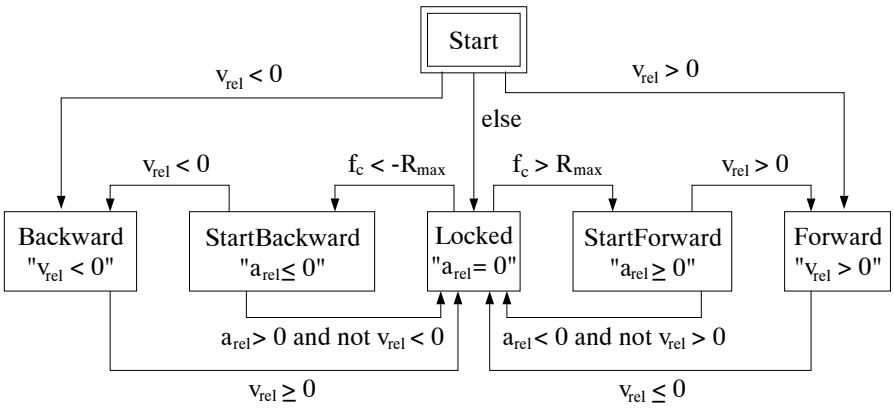


Bild 4.11: Übergangsdiagramm des Reibmodells.

Kanten eindeutig an, in welchen Zustand der DEA bei gegebenen Werten der beschreibenden Variablen zu schalten hat. Vor dem Integrationsstart, und bei einem Ereignis, werden die Zustände des DEA solange durchlaufen bis ein Zustand erreicht ist, von dem aus kein weiteres Schalten erfolgt. Erst dann wird die Integration gestartet, bzw. nach einem Ereignis neu gestartet. Hierdurch ist garantiert, daß keine der unter Punkt 1 aufgeführten Schwierigkeiten auftreten kann. Jedes Schalten im DEA bewirkt eine komplette Neuauswertung des Differentialgleichungssystems, da die Variablen nach dem Schalten neu berechnet werden müssen.

Zur Demonstration sollen einige Schaltzyklen diskutiert werden. Beim Start der Integration wird aufgrund der Relativgeschwindigkeit entschieden, bei welchem Zustand begonnen wird. Wenn die Geschwindigkeit Null ist, wird im Zustand *Locked* begonnen. Unter der Annahme eines gesperrten Gelenks wird das Modell jetzt ausgewertet. Wenn die Zwangskraft f_c größer als die statische Haftreibkraft R_{max} wird, so schaltet der DEA in den Zustand *StartForward*,

d.h. der entsprechende Gelenkfreiheitsgrad wird freigegeben. Da die Relativgeschwindigkeit Null ist, bleibt der DEA in diesem Zustand und die Integration wird gestartet. Als Indikatorfunktionen werden jetzt die Relativgeschwindigkeit v_{rel} und die Relativbeschleunigung a_{rel} benutzt. Mit der auf Seite 50 erläuterten Strategie wird ein sehr kleines Intervall um identisch verschwindende Indikatorfunktionen gelegt, hier für die Relativgeschwindigkeit. Wenn dieses Intervall verlassen wird, tritt ein Ereignis auf. Im Beispiel tritt damit ein Ereignis auf, wenn v_{rel} entweder positiv wird oder wenn die Relativbeschleunigung kleiner als Null wird (und v_{rel} nicht positiv ist). In der Regel wird also nach einem kleinen Zeitschritt ein Ereignis eintreten und der DEA wird nach *Forward* schalten. Wenn die Relativgeschwindigkeit wieder kleiner als Null wird, tritt wiederum ein Ereignis ein und der DEA schaltet direkt in den Zustand *Locked*. Dann wird das Modell noch einmal ausgewertet. Sollte jetzt $f_c < -R_{max}$ sein, schaltet der DEA weiter zu *StartBackward*.

Im DEA ist eine kleine Hysterese eingebaut, um ein sofortiges Zurückschalten von *StartForward* und *StartBackward* nach *Locked* zu vermeiden, wenn im vorigen Schaltzyklus von *Locked* in diese Zustände geschaltet wurde. Wie auf Seite 61 erläutert, kann es sein, daß aufgrund numerischer Fehler die Beschleunigung a_{rel} in *StartForward* bzw. *StartBackward* als Null berechnet wird, obwohl $f_c > R_{max}$ bzw. $f_c < -R_{max}$ ist. Aus diesem Grund wird erst zurückgeschaltet, wenn $a_{rel} < 0$ bzw. $a_{rel} > 0$ wird.

Das Problem strukturvariabler Modelle wurde schon im letzten Abschnitt gelöst. Dort wurde gezeigt, daß bei Mehrkörpersystemen in Baumstruktur durch das objektorientierte O(n) Verfahren das Blockieren und Freigeben von Gelenken praktisch ohne Effizienzverlust behandelt werden kann. Im Reibelement muß demnach nur festgelegt werden, ob ein Gelenk blockiert oder nicht. Der Mehrkörperalgorithmus ermittelt dann alle notwendigen Variablen, je nach Blockierzustand.

Mit der in Kapitel 4 vorgestellten Strategie wird das gleichzeitige Auftreten mehrerer Ereignisse zuverlässig behandelbar. Hierzu wird der obige DEA mit der Transformationsvorschrift von Seite 63 in eine Folge von (im wesentlichen Boole'schen) Gleichungen umgewandelt. Diese Gleichungen werden zusammen mit *allen* anderen Gleichungen des Modells auf Blockdreiecksform sortiert. Dadurch ist garantiert, daß die "Ereignisaktionen" in der richtigen Reihenfolge ausgewertet werden, auch wenn mehrere Ereignisse zum selben Zeitpunkt auftreten.

Wenn mehrere Reibelemente vorhanden sind, kann es vorkommen, daß sich die DEAs bei einem Ereignispunkt gegenseitig beeinflussen. Schaltet z.B. ein Reibelement 1 von *Forward* nach *Locked* und werden die Modellgleichungen neu ausgewertet, kann die Zwangskraft in einem anderen (sich im Haftzustand befindlichen) Reibelement 2 größer als die statische Haftreibkraft werden. Der DEA von Reibelement 2 schaltet dann nach *StartForward*. Durch den DEA-Mechanismus und das Sortieren wird garantiert, daß das Schalten auch beim Vorliegen dieser Verkopplungen korrekt erfolgt.

Beim obigen Vorgehen verbleibt nur eine Schwierigkeit: Es kann nicht garantiert werden, daß das Schalten der DEAs konvergiert. Es ist möglich, daß sich die DEAs eines Modells gegenseitig so beeinflussen, daß eine "unendliche Schaltschleife" auftritt. Dies ist eine prinzipielle Schwierigkeit bei dynamisch verkoppelten Reibelementen. Das Problem kann auch so formuliert werden: Wenn ein Ereignis auftritt, finde einen konsistenten Satz von Schaltzuständen und Variablen, sodaß die dynamischen Gleichungen und die Reibgesetze erfüllt sind. Die obige Methode löst dieses Problem durch sukzessives Schalten in den DEAs der

Reiblemente, was einer Fixpunktiteration entspricht. In [Loet81, Gloc93, Gloc93a] wird das Problem auf die Lösung eines *linearen Komplementaritätsproblems* zurückführt[13]. Dieses hat die folgende Form:

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b} \;\; ; \;\; \mathbf{y} \geq 0 \;\; ; \;\; \mathbf{x} \geq 0 \;\; ; \;\; \mathbf{y}^T\mathbf{x} = 0. \tag{4.43}$$

Hierbei sind die Unbekannten $\mathbf{x}$ und $\mathbf{y}$ gesucht. Es gibt eine reichhaltige Lösungstheorie für diese Art von Gleichungen, siehe [Murt88]. Zum Beispiel kann der Algorithmus von Lemke auf jedes Gleichungssystem (4.43) angewandt werden. Konvergenzaussagen gibt es jedoch nur für spezielle Strukturen der Matrix $\mathbf{A}$, wie Symmetrie und positive Definitheit. Leider hat $\mathbf{A}$ bei Reibproblemen eine Struktur, für die keine Konvergenzbeweise bekannt sind, siehe [Gloc93a]. Das sichere Auffinden eines konsistenten Satzes von Schaltzuständen bei Reibelementen ist damit noch nicht zufriedenstellend gelöst.

Reibelemente können mittels des oben erläuterten Verfahrens genauso einfach wie andere "normale" Kraftelemente verwendet werden. Dies soll an einem nicht-trivialen Beispiel demonstriert werden, welches weder mit Hand (aufgrund der Komplexität), noch mit kommerziell verfügbaren Mehrkörperprogrammen, gelöst werden kann. Hierzu wird eine neue Klasse *ExtFriction* eingeführt, die als externes Kraftgesetz bei einer Reihe von MKS-Elementen, z.B. Gelenken, benutzt werden kann. Die Klasse wird aus dem DEA von Bild 4.11 mit der auf Seite 63 angegebenen DEA-Transformation gebildet, und von der Klasse *ExtForceL* durch Vererbung abgeleitet (die Klasse *ExtForceL* ist auf Seite 87 beschrieben):

```
model class (ExtForceL) ExtFriction
    parameter Rmax = 0      {maximale statische Haftreibkraft}
    terminal  Rv            {(positive) Gleitreibkraft}
    terminal  R             {Reibkraft}

    local Forward  = false, StartForward  = false,
          Backward = false, StartBackward = false, Start = true

    fL = if Forward  or StartForward  then  Rv else
         if Backward or StartBackward then -Rv else 0
    R  = fL + fLc

    Locked                = not ( Forward  or StartForward  or
                                  Backward or StartBackward or Start )
    new(Start)            = false

    new(Forward)          = qd > 0 and (Forward  or StartForward  or Start)
    new(Backward)         = qd < 0 and (Backward or StartBackward or Start)

    new(StartForward)     = Locked and fLc > Rmax  or StartForward  and not
                            (qd > 0 or qd <= 0 and qdd < 0)
    new(StartBackward)    = Locked and fLc < -Rmax or StartBackward and not
                            (qd < 0 or qd >= 0 and qdd > 0)
end
```

Die maximale statische Haftreibkraft wird als Parameter *Rmax* definiert. Es wird angenommen, daß die Vorwärts- und Rückwärts-Gleitreibkraftfunktionen betragsmäßig gleich sind. Da hier zahlreiche Kennlinienvarianten auftreten können, wird die Gleitreibkraft für

[13] In den zitierten Arbeiten werden Reibmodelle betrachtet, bei denen auch die Normalkraftabhängigkeit berücksichtigt wird.

das Vorwärtsgleiten Rv nur als **terminal** Variable definiert, d.h. die Funktion $Rv = Rv(qd)$ muß außerhalb des Reibelements festgelegt werden. Über den **cut** $extL$, der von der Klasse *ExtForceL* geerbt wird, kann das Reibelement an den entsprechenden Kraftelement-Cut von Gelenken angeschlossen werden. Hierbei ist q die Relativkoordinate im Gelenk (z.B. der Drehwinkel bei einem Drehgelenk oder die Verschiebung bei einem Translationsgelenk) und qd, qdd sind deren erste und zweite Ableitung. Die Boole'sche Variable *Locked* wird im Reibelement gesetzt und bestimmt den Schaltzustand des Gelenks. Die Through-Variable fL ist die eingeprägte Kraft und die Through-Variable fLc ist die Zwangskraft im Gelenk. fL wird im Reibelement als Gleitreibkraft berechnet, während fLc im Gelenk bestimmt wird. Die Summe von fL und fLc ist die tatsächlich im Gelenk wirkende Reibkraft und wird über die Terminal-Variable R für Informationszwecke nach außen zur Verfügung gestellt. Die restlichen Anweisungen in der Klassenbeschreibung sind eine direkte Abbildung des DEA.

Als Demonstrationsbeispiel wird der schon auf Seite 96 eingeführte 6-freiheitsgradige Industrieroboter Manutec r3 benutzt, wobei das dort besprochene Robotermodell um Reibelemente erweitert wird. In *jedem* Gelenk des Roboters soll Coulomb'sche Reibung wirken, wobei die von Türk [Tuer90] gemessenen Reibkennlinien verwendet werden. Dieses Modell ist schon recht komplex, weil es durch das mögliche Haften in jedem Gelenk insgesamt $2^6 = 64$ unterschiedliche Modellstrukturen gibt[14] und weil aufgrund der 6 DEAs mit je 5 Zuständen insgesamt $5^6 = 15625$ verschiedene Schaltzustände möglich sind.

Das Robotermodell von Seite 96 wird nun einfach um 6 Reibmodelle erweitert:

```
@mbssim.lib        {verwende O(n) MKS-Bibliothek}
model robot
    submodel (Inertial)     i   (g = 9.81, ng3 = -1)

    submodel (RevoluteLS) r1 (n3 = 1)  , r2 (n1 = 1)   , r3 (n1 = 1)
    submodel (RevoluteLS) r4 (n3 = 1)  , r5 (n1 = 1)   , r6 (n3 = 1)
    submodel (Bar)        b3 (r3 = 0.5), b5 (r3 = 0.73), bL (r1 = rL1, r2 = rL2, r3 = rL3)

    submodel (Body) m1   (I33 = 1.16)
    submodel (Body) m2   (m = 56.5, r1 = 0.172, r3 = 0.205  , I11 = 2.58   , I22 = 0.73 ,
                                                              I33 = 0.64   , I31 = -0.46)
    submodel (Body) m3   (m = 26.4, r1 = 0.064, r3 = -0.034, I11 = 0.279  , I22 = 0.413,
                                                              I33 = 0.245  , I31 = -0.07)
    submodel (Body) m4   (m = 28.2            , r3 = 0.32   , I11 = 1.67   , I22 = 1.67,
                                                              I33 = 0.081)
    submodel (Body) m5   (m = 5.2             , r3 = 0.023  , I11 = 0.0125, I22 = 0.0153,
                                                              I33 = 0.0081)
    submodel (Body) Load (m = mL)

    submodel (ExtFriction) R1 (Rmax = 42)  , R2 (Rmax = 105) , R3 (Rmax = 42)
    submodel (ExtFriction) R4 (Rmax = 26.7), R5 (Rmax = 39.6), R6 (Rmax = 16.8)

    parameter mL = 5, rL1 = 0.1, rL2 = 0, rL3 = 0
    input     u(6)            {Antriebsmomente in den Gelenken}
    output    R(6)            {Reibkräfte in den Gelenken

    connect i to r1 to r2 to b3 to r3 to r4 to b5 to r5 to r6 to bL,
            m1 at r1:b,  R1 at r1:extL,  m4  at r4:b,  R4 at r4:extL,
            m2 at r2:b,  R2 at r2:extL,  m5  at r5:b,  R5 at r5:extL,
```

[14] Wenn z.B. alle 6 Gelenke im Haftzustand sind, hat der Roboter 0 Freiheitsgrade; wenn alle Gelenke im Gleitzustand sind, hat der Roboter 6 Freiheitsgrade.

$m3$ **at** $r3{:}b$, $R3$ **at** $r3{:}extL$, $load$ **at** $b6{:}b$, $R6$ **at** $r6{:}extL$

$u(1) = r1.f$
$\ldots$
$u(6) = r6.f$
$R(1) = R1.R$
$\ldots$
$R(6) = R6.R$

$R1.Rv = 42 + 8.96 * abs(r1.qd)$
$R2.Rv = 105 + (\ \textbf{if}\ abs(r2.qd) < 0.62\ \textbf{then}\ (21/0.62) * abs(r2.qd)$
$\textbf{else}\ \ 21 + 24.5 * (abs(r2.qd) - 0.62)\ )$
$R3.Rv = 42 + (\ \textbf{if}\ abs(r3.qd) < 2.2\ \textbf{then}\ (12/2.2) * abs(r3.qd)$
$\textbf{else}\ \ 12 + 3 * (abs(r3.qd) - 2.2)\ )$
$R4.Rv = 21.8 + 9.8 * abs(r4.qd)$
$R5.Rv = 30.1 + (\ \textbf{if}\ abs(r5.qd) < 1.26\ \textbf{then}\ (3.2/1.26) * abs(r5.qd)$
$\textbf{else}\ \ 3.2 + 0.5 * (abs(r5.qd) - 1.26)\ )$
$R6.Rv = 10.9 + 3.9 * abs(r6.qd)$

end

Bei den Drehgelenken werden Objekte der Klasse *RevoluteLS* benutzt, d.h. diese Drehgelenke können blockiert werden ("L") und der Drehwinkel und seine erste Ableitung werden als Zustandsgrößen benutzt ("S"). Im Vergleich zum vorherigen Robotermodell werden zusätzlich 6 Objekte der Klasse *ExtFriction* deklariert und mit der **connect** Anweisung an die entsprechenden Gelenke gebunden. Weiterhin werden die Gleitreibkraftfunktionen mit Hilfe von if-Anweisungen am Ende des Modells beschrieben. Man beachte, daß in den ersten 3 Gelenken $R_{max} = R_v$ ist, während in den letzten 3 Gelenken jeweils eine Unstetigkeit vorliegt. Auf diese sehr einfache Weise wird die Reibung in den 6 Gelenken definiert. Als Eingangsgrößen in das Modell werden die Antriebsmomente in den Gelenken benutzt. Als Ausgangsgrößen werden die in den Gelenken wirkenden Reibkräfte verwendet.

Für eine Simulation werden die Eingangsgrößen und die Anfangswerte der Zustandsgrößen auf Null gesetzt, mit Ausnahme von: $u_1(t) = -40\ Nm$, $u_6(t) = -24\ Nm$, $\dot{q}_1 = 1\ rad/s$. Durch diese Wahl der Eingangsgrößen und der Anfangswerte tritt der oben erwähnte Fall auf, daß mehrere Reibelemente gleichzeitig an einem Ereignispunkt schalten. Im folgenden Simulationsprotokoll sind die auftretenden Schaltvorgänge zusammengestellt:

```
Starting Time = 0.
     Iteration 1: Variable fr1.Forward       changed to True
                  Variable fr1.Start         changed to False
                  Variable fr2.Start         changed to False
                  Variable fr3.Start         changed to False
                  Variable fr4.Start         changed to False
                  Variable fr5.Start         changed to False
                  Variable fr6.Start         changed to False
     Iteration 2: Variable fr2.StartBackward changed to True

An event occured at Time = 2.85387872231191E-10
     Iteration 1: Variable fr2.Backward      changed to True
                  Variable fr2.StartBackward changed to False

An event occured at Time = 4.91991757360510E-02
     Iteration 1: Variable fr1.Forward       changed to False
     Iteration 2: Variable fr3.StartBackward changed to True
                  Variable fr6.StartBackward changed to True

An event occured at Time = 4.91991757388258E-02
```

```
        Iteration 1: Variable fr6.Backward      changed to True
                     Variable fr6.StartBackward changed to False

An event occured at Time = 4.91991757569666E-02
        Iteration 1: Variable fr3.Backward      changed to True
                     Variable fr3.StartBackward changed to False

An event occured at Time = 5.39751964428538E-02
        Iteration 1: Variable fr2.Backward      changed to False

An event occured at Time = 9.39294865468687E-02
        Iteration 1: Variable fr3.Backward      changed to False
```

Beim Start der Integration sind alle Gelenke blockiert, mit Ausnahme von Gelenk 1, da dessen Winkelgeschwindigkeit ungleich Null ist. Bei der zweiten Iteration ergibt sich, daß diese Annahme nicht vollkommen richtig war, so daß Gelenk 2 in den *StartBackward* Zustand schaltet. Nach einer kurzen Integrationszeit geht Gelenk 2 in reines Rückwärtsgleiten über. Nach 0.049 Sekunden tritt ein Ereignis ein, da Gelenk 1 in die Haftphase übergeht. Auf Grund der unstetigen Änderung der Winkelbeschleunigung von Gelenk 1 gibt es einen "Ruck" der dazu führt, daß die Gelenke 3 und 6 zu gleiten beginnen. Bei 0.054 Sekunden und bei 0.094 Sekunden gehen die Gelenke 2 und 3 jeweils wieder in das Haften über.

In Bild 4.12 sind Ergebnistrajektorien dieses Simulationslaufs zu sehen. Im linken Bild sind die Reibmomente der Gelenke 1, 3 und 6 aufgetragen. Man sieht deutlich die unstetige Änderung der Reibmomente zum Zeitpunkt t=0.049. Im rechten Bild sind die Winkelge-

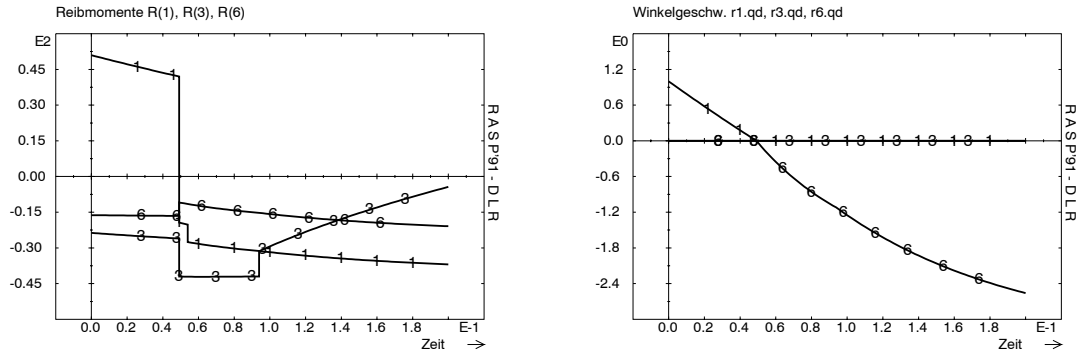


Bild 4.12: Simulationsergebnis des Roboters mit Reibung.

schwindigkeiten der Gelenke 1, 3 und 6 zu sehen. Zum Zeitpunkt t=0.049 verschwindet die Winkelgeschwindigkeit von Gelenk 1. Gleichzeitig werden die Winkelgeschwindigkeiten der Gelenke 3 und 6 ungleich Null. Da die Winkelgeschwindigkeit von Gelenk 3 recht klein bleibt, ist die Änderung dieser Winkelgeschwindigkeit im Bild kaum zu sehen.

4.6 Mehrkörpersysteme mit kinematischen Schleifen

Die bis jetzt besprochenen Algorithmen können nur bei Mehrkörpersystemen in Baumstruktur angewendet werden. In diesem Abschnitt wird gezeigt, wie Mehrkörpersysteme behandelt werden können, die auch kinematische Schleifen enthalten.

Eine übliche Vorgehensweise zur Behandlung kinematischer Schleifen besteht darin, ein Mehrkörpersystem durch Schnitte in ausgewählten Gelenken in ein MKS mit Baumstruktur

umzuwandeln (siehe z.B. [Robe88, Leis92]). Die Gleichungen des MKS in Baumstruktur werden aufgestellt und um die Zwangsgleichungen der Schnittgelenke erweitert. Im folgenden wird gezeigt, wie diese Gleichungen mit der objektorientierten Modellbeschreibung erzeugt und gelöst werden können. Eine interessante, alternative Methode wurde von Hiller, Kecskeméthy und Woernle entwickelt [Woer88, Hill93, Kecs93, Kecs93a]. Hierbei werden kinematische Schleifen nicht an einem Gelenk, sondern an mehreren Gelenken geschnitten. Für viele technisch wichtige Schleifentypen gelangt man damit zu einem explizit auflösbaren System von Gleichungen, was beim ersten Verfahren nicht der Fall ist. Es erscheint möglich, auch dieses Verfahren mit einer objektorientierten Modellierungssprache wie Dymola zu realisieren. Dies führt jedoch über den Rahmen der vorliegenden Arbeit hinaus und wurde nicht im Detail untersucht.

Beim ersten Verfahren muß der Anwender aufgrund der kinematischen Struktur eines Mehrkörpersystems entscheiden, durch welche Schnitte das MKS in eine Baumstruktur überführt wird. Das MKS in Baumstruktur wird mit den in den letzten Abschnitten erläuterten Klassen modelliert. Danach werden die Schnittgelenke mit Hilfe von Spezialklassen beschrieben und entsprechend der physikalischen Struktur mit dem schon modellierten MKS in Baumstruktur zusammengeschaltet.

Die Klassen für die Schnittgelenke ergeben sich aus folgenden Überlegungen: Auf Grund eines Schnittgelenks wird die Relativbewegung der beiden Cut-Frames des Schnittgelenks zueinander eingeschränkt. Wenn das Schnittgelenk f Freiheitsgrade besitzt, so gibt es jeweils $(6-f)$ Zwangsbedingungen auf Lage-, Geschwindigkeits- und -Beschleunigungsebene. Diese können entweder direkt für einen bestimmten Gelenktyp angegeben werden, oder durch Elimination der verallgemeinerten Koordinaten $\mathbf{q}$ des Gelenks aus den Gleichungen (4.9) von Seite 81 ermittelt werden. Weiterhin müssen $(6-f)$ voneinander unabhängige, unbekannte Schnittkräfte $\boldsymbol{\lambda}^c$ ausgewählt werden und das resultierende Schnittmoment und die resultierende Schnittkraft $\hat{\mathbf{f}}^b$ am Cut b des Gelenks muß als Funktion dieser Unbekannten ausgedrückt werden. Ein Schnittgelenk wird damit durch Angabe der folgenden Gleichungen definiert, die Funktionen der Relativgrößen des Gelenks sind:

$$\mathbf{0} = \mathbf{g}_p\left({}^b\mathbf{r}^{ab},\ {}^b\mathbf{T}^a\right) \tag{4.44a}$$

$$\mathbf{0} = \mathbf{g}_v\left({}^b\hat{\mathbf{v}}^{ab},\ {}^b\mathbf{r}^{ab},\ {}^b\mathbf{T}^a\right) \tag{4.44b}$$

$$\mathbf{0} = \mathbf{g}_a\left({}^b\hat{\mathbf{a}}^{ab},\ {}^b\hat{\mathbf{v}}^{ab},\ {}^b\mathbf{r}^{ab},\ {}^b\mathbf{T}^a\right) \tag{4.44c}$$

$$\hat{\mathbf{f}}^b = \boldsymbol{\Psi}_c\left({}^b\mathbf{r}^{ab},\ {}^b\mathbf{T}^a\right)\boldsymbol{\lambda}^c + \boldsymbol{\Psi}_e\left({}^b\mathbf{r}^{ab},\ {}^b\mathbf{T}^a\right)\boldsymbol{\lambda}^e \quad . \tag{4.44d}$$

Als Beispiel wird das auf Seite 86 besprochene, einfache Drehgelenk behandelt. Ein Drehgelenk wird durch folgende Gleichungen als Funktion seiner Minimalkoordinate, d.h. des Drehwinkels q, beschrieben:

$${}^b\mathbf{T}^a = \mathbf{n}\mathbf{n}^T + \left(\mathbf{E} - \mathbf{n}\mathbf{n}^T\right)\cdot\cos(q) - \mathbf{skew}(\mathbf{n})\cdot\sin(q) \tag{4.45a}$$

$${}^b\hat{\mathbf{v}}^{ab} = \begin{bmatrix} {}^b\boldsymbol{\omega}^{ab} \\ {}^b\mathbf{v}^{ab} \end{bmatrix} = \begin{bmatrix} \mathbf{n} \\ \mathbf{0} \end{bmatrix}\dot{q} \tag{4.45b}$$

$${}^b\hat{\mathbf{a}}^{ab} = \begin{bmatrix} {}^b\boldsymbol{\alpha}^{ab} \\ {}^b\mathbf{a}^{ab} \end{bmatrix} = \begin{bmatrix} \mathbf{n} \\ \mathbf{0} \end{bmatrix}\ddot{q} \tag{4.45c}$$

Um das Drehgelenk als Schnittgelenk verwenden zu können, muß neben der Drehachse $\mathbf{n}$ noch ein Vektor $\mathbf{u}$ im Cut-Frame b angegeben werden, der auf der Drehachse senkrecht steht. Mit diesen beiden gegebenen Vektoren kann ein dritter Vektor $\mathbf{w}$ berechnet werden, der auf $\mathbf{n}$ und $\mathbf{u}$ senkrecht steht. Da das Drehgelenk einen Freiheitsgrad besitzt, müssen auf Lage-, Geschwindigkeits- und Beschleunigungsebene je 5 Zwangsgleichungen angegeben werden. Diese können aus (4.45) erhalten werden. Da die translatorischen Anteile der Gleichungen (4.45) nicht von q abhängen, können diese direkt als Zwangsgleichungen verwendet werden. Aus den rotatorischen Anteilen kann q durch geeignete Projektionen mit $\mathbf{u}$ und $\mathbf{w}$ eliminiert werden. Dies führt auf

$$\mathbf{g}_p = \begin{bmatrix} \mathbf{u}^{T\,b}\mathbf{T}^a\,\mathbf{n} \\ \mathbf{w}^{T\,b}\mathbf{T}^a\,\mathbf{n} \\ {}^b\mathbf{r}^{ab} \end{bmatrix} = \mathbf{0} \tag{4.46a}$$

$$\mathbf{g}_v = \begin{bmatrix} \mathbf{u}^{T\,b}\boldsymbol{\omega}^{ab} \\ \mathbf{w}^{T\,b}\boldsymbol{\omega}^{ab} \\ {}^b\mathbf{v}^{ab} \end{bmatrix} = \mathbf{0} \tag{4.46b}$$

$$\mathbf{g}_a = \begin{bmatrix} \mathbf{u}^{T\,b}\boldsymbol{\alpha}^{ab} \\ \mathbf{w}^{T\,b}\boldsymbol{\alpha}^{ab} \\ {}^b\mathbf{a}^{ab} \end{bmatrix} = \mathbf{0} \tag{4.46c}$$

$$\hat{\mathbf{f}}^b = \begin{bmatrix} \mathbf{u} & \mathbf{w} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{u} & \mathbf{w} & \mathbf{n} \end{bmatrix} \begin{bmatrix} \lambda_1^c \\ \vdots \\ \lambda_5^c \end{bmatrix} + \begin{bmatrix} \mathbf{n} \\ \mathbf{0} \end{bmatrix} \lambda^e \ . \tag{4.46d}$$

Das Gesamt-Gleichungssystem eines Mehrkörpersystems setzt sich jetzt zusammen aus den MKS-Gleichungen des aufspannenden Baums, sowie aus den obigen Gleichungen der Schnittgelenke. Das Gleichungssystem ist groß und dünnbesetzt. Ähnlich wie für baumstrukturierte Systeme in Abschnitt 4.3.4 ab Seite 101, kann auch dieses Gleichungssystem mit Hilfe des Tearing-Verfahrens auf ein kleines, dichtbesetztes Gleichungssystem transformiert werden. Dabei können geeignete Tearing-Variablen und Tearing-Gleichungen durch folgende Fragestellung anschaulich ermittelt werden (siehe Gleichungssystem (2.11) auf Seite 29):

> Welche *unbekannten* Variablen $\mathbf{x}_2$ müssen *bekannt* und welche *bekannten* Variablen $\mathbf{b}_2$ müssen *unbekannt* sein, damit die anderen unbekannten Variablen $\mathbf{x}_1$ sowie $\mathbf{b}_2$ berechnet werden können?

Es zeigt sich, daß als Tearing-Variablen die zweiten Ableitungen der Gelenkkoordinaten $\ddot{\mathbf{q}}$ aller Gelenke des aufspannenden Baums und die Schnittkräfte $\boldsymbol{\lambda}^c$ aller Schnittgelenke verwendet werden können. Die Tearing-Gleichungen sind, wie bei baumstrukturierten Systemen, die Gleichungen (4.12) auf Seite 83 mit der die verallgemeinerten Gelenkkräfte $\boldsymbol{\lambda}^e$ der Gelenke im aufspannenden Baum berechnet werden, sowie die Zwangsgleichungen (4.44a, 4.44b, 4.44c) aller Schnittgelenke. Dies kann folgendermaßen eingesehen werden:

Wenn als "Zustandsgrößen" die Variablen $\mathbf{q}$ und $\dot{\mathbf{q}}$ verwendet werden und die Tearing-Variablen $\ddot{\mathbf{q}}$ und $\boldsymbol{\lambda}^c$ bekannt sind, dann können die verallgemeinerten Kräfte $\boldsymbol{\lambda}^e$ in den Gelenken des aufspannenden Baums berechnet werden, weil dies genau das inverse dynamische Problem ist. Da die Variablen $\boldsymbol{\lambda}^c$ als bekannt angesehen werden, wirken sich die Schnittgelenke dabei wie zusätzliche Kraftelemente aus, die durch die Gleichungen (4.44d)

beschrieben sind. Bei der Lösung des inversen dynamischen Problems können auch die Absolutgrößen jedes Cut-Frames berechnet werden, insbesondere der Cut-Frames der Schnittgelenke. Wenn die Absolutgrößen der Schnittgelenk Cut-Frames bekannt sind, können die relativen kinematischen Größen der Schnittgelenke bestimmt werden und damit können die Zwangsgleichungen (4.44a, 4.44b, 4.44c) der Schnittgelenke, also die noch verbleibenden Tearing-Gleichungen, ausgewertet werden. QED.

Das Tearing-Verfahren führt deswegen zu einem Gleichungssystem, wie es üblicherweise mit Hilfe von mechanischen Prinzipien abgeleitet wird (siehe z.B. [Robe88]):

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} = \mathbf{h}(\mathbf{q},\dot{\mathbf{q}}) + \boldsymbol{\lambda}^e + \mathbf{G}^T(\mathbf{q})\,\boldsymbol{\lambda}^c \tag{4.47a}$$

$$\mathbf{0} = \mathbf{g}_p = \mathbf{g}_p(\mathbf{q}) \qquad (\mathbf{G} = \frac{\partial \mathbf{g}_p}{\partial \mathbf{q}}) \tag{4.47b}$$

$$\mathbf{0} = \mathbf{g}_v = \mathbf{G}(\mathbf{q})\dot{\mathbf{q}} \tag{4.47c}$$

$$\mathbf{0} = \mathbf{g}_a = \mathbf{G}(\mathbf{q})\ddot{\mathbf{q}} + \frac{\partial \mathbf{G}(\mathbf{q})\dot{\mathbf{q}}}{\partial \mathbf{q}}\,\dot{\mathbf{q}}\ . \tag{4.47d}$$

Hierbei ist (4.47a) die Dynamikgleichung des aufspannenden Baums, (4.47b) beschreibt die Zwangsgleichungen aller Schnittgelenke auf Lageebene, (4.47c) beschreibt die Zwangsgleichungen aller Schnittgelenke auf Geschwindigkeitsebene und (4.47d) beschreibt die Zwangsgleichungen aller Schnittgelenke auf Beschleunigungsebene. Das Gleichungssystem (4.47) ist ein überbestimmtes differential-algebraisches Gleichungssystem in den unbekannten Gelenkkoordinaten $\mathbf{q}(t)$ aller Gelenke des aufspannenden Baums und in den Zwangskräften $\boldsymbol{\lambda}^c(t)$ aller Schnittgelenke.

Das Gleichungssystem kann auch auf folgende Weise interpretiert werden: Die Gleichungen (4.47a, 4.47b) beschreiben eine singuläre DAE mit dem Störungsindex 3. Durch zweimaliges Differenzieren von (4.47b) wird die überbestimmte DAE (2.19) von Seite 34 erhalten. Damit können alle in Kapitel 2.4 ab Seite 31 angegebenen Verfahren benutzt werden, um die DAE (4.47) zu lösen. Zwei Lösungsverfahren sollen kurz skizziert werden:

1. *Die Dummy Derivative Methode:*
   Hier muß der Anwender festlegen, welche Gelenkkoordinaten des aufspannenden Baums als Zustandsgrößen benutzt werden. Für die ausgewählten Gelenke müssen im Modell Gelenktypen verwendet werden, die die Gelenkkoordinaten als Zustandsgrößen definieren. Bei allen anderen Gelenken im aufspannenden Baum werden die verallgemeinerten Lage-, Geschwindigkeits- und Beschleunigungs-Koordinaten des jeweiligen Gelenks als voneinander unabhängige Unbekannte aufgefaßt. Mit diesem Vorgehen geht das obige Gleichungssystem (4.47) in eine Index-1 DAE über, die genauso viele Gleichungen wie Unbekannte besitzt. Die DAE besteht aus linearen und nichtlinearen Gleichungen in den unbekannten Variablen und kann z.B. mit DASSL gelöst werden.

2. *Transformation auf ein lösbares Index-2 System:*
   In Kapitel 2.4 ab Seite 40 wurde ein allgemeines Verfahren angegeben, um *jede* (strukturell) singuläre DAE auf eine spezielle Index-2 DAE zu transformieren, die mit leicht modifizierten Standardverfahren gelöst werden kann. Die Anwendung dieser Transformationsvorschrift auf (4.47) führt zu der folgenden Index-2 DAE in den unbekannten Variablen $\dot{\mathbf{p}}(t)$ $(= \dot{\mathbf{q}})$, $\dot{\mathbf{v}}(t)$ $(= \ddot{\mathbf{q}})$, $\mathbf{a}(t)$ $(= \ddot{\mathbf{q}})$, $\boldsymbol{\lambda}^c(t)$, $\boldsymbol{\mu}_p(t)$, $\boldsymbol{\mu}_v(t)$:

$$\dot{\mathbf{p}} = \mathbf{v} + \mathbf{G}^T(\mathbf{p})\,\boldsymbol{\mu}_p \tag{4.48a}$$

$$\dot{\mathbf{v}} = \mathbf{a} + \mathbf{G}^T(\mathbf{p})\,\boldsymbol{\mu}_v \tag{4.48b}$$

$$\mathbf{M}(\mathbf{p})\,\mathbf{a} = \mathbf{h}(\mathbf{p},\mathbf{v}) + \boldsymbol{\lambda}^e + \mathbf{G}^T(\mathbf{p})\,\boldsymbol{\lambda}^c \tag{4.48c}$$

$$\mathbf{0} = \mathbf{g}_p = \mathbf{g}_p(\mathbf{p}) \qquad (\mathbf{G} = \frac{\partial \mathbf{g}_p}{\partial \mathbf{p}}) \tag{4.48d}$$

$$\mathbf{0} = \mathbf{g}_v = \mathbf{G}(\mathbf{p})\,\mathbf{v} \tag{4.48e}$$

$$\mathbf{0} = \mathbf{g}_a = \mathbf{G}(\mathbf{p})\,\mathbf{a} + \frac{\partial \mathbf{G}(\mathbf{p})\mathbf{v}}{\partial \mathbf{p}}\,\mathbf{v}\ . \tag{4.48f}$$

In der Literatur gibt es Vorschläge, wie die $O(n)$ Verfahrensklasse auf MKS mit Schleifen verallgemeinert werden können. Entsprechend ist dies auch für die in Abschnitt 4.4 besprochene $O(n)$ Methode möglich. Hierzu muß das $O(n)$ Verfahren von Abschnitt 4.4 mit dem Tearing-Verfahren kombiniert werden.

4.7 Diskussion

In diesem Kapitel wurde gezeigt, wie starre Mehrkörpersysteme objektorientiert modelliert werden können. Es sind nur einige wenige Klassen notwendig, um die lokalen Eigenschaften von Gelenken, Kraftelementen, Beobachtungsgrößen und Trägheitseigenschaften von Starrkörpern zu beschreiben. MKS-Objekte werden entsprechend ihrer physikalischen Beziehungen zusammengeschaltet. Die Realisierung einer MKS-Bibliothek ist einfach, da nur die Gleichungen der lokalen Komponenten-Typen in die Bibliothek eingetragen werden müssen.

Je nachdem welche Größen als bekannt und welche Größen als unbekannt angesehen werden, können die Gleichungen eines MKS für verschiedene Aufgabenstellungen generiert werden. Aufgabenstellungen sind z.B. Vorwärtskinematik, inverses dynamisches Problem, direktes dynamisches Problem, inverse Kinematik, Stationärwertberechnung. In allen diesen Fällen können die entstehenden großen, dünnbesetzten Gleichungssysteme mit Hilfe des allgemeinen Tearing-Verfahrens *symbolisch* auf kleine, vollbesetzte Gleichungssysteme transformiert werden, die mit vorhandenen Standardverfahren gelöst werden können. Für baumstrukturierte Mehrkörpersysteme ist es möglich die attraktive $O(n)$ Verfahrensklasse anzuwenden, indem die Gleichungen der einzelnen Klassen etwas umformuliert werden. Weiterhin können damit, ohne Effizienzverluste, auch Mehrkörpersysteme behandelt werden die strukturvariabel sind, weil Gelenke während einer Simulation blockieren können. Diese Eigenschaft wird u.a. zur Behandlung von Coulomb'scher Reibung benötigt. In Kombination mit den, in Kapitel 3 ab Seite 45 besprochenen, Methoden zur Modellierung von ereignisabhängigen Modellen, können damit auf einfache Weise Mehrkörpersysteme behandelt werden, deren Modellstruktur sich aufgrund von Ereignissen ändert. Dies wurde am Beispiel eines Roboters gezeigt, in dessen 6 Gelenken Coulomb'sche Reibung vorhanden ist, was zu $2^6 = 64$ unterschiedlichen Modellstrukturen führt.

Noch unbefriedigend bei dem obigen Vorgehen ist, daß es noch nicht vollkommen "objektorientiert" ist, da Gelenkobjekte immer so eingebaut werden sollten, daß der Cut a des Gelenks "näher" am Inertialsystem ist. Bei Verwendung des $O(n)$ Verfahrens ist dies eine

notwendige Voraussetzung. Wird dieser Konvention beim Einsatz der Tearing-Methode nicht gefolgt, werden unnötigerweise kleine lineare Gleichungssysteme gelöst, die auch einfach analytisch invertiert werden könnten. Abhilfe erscheint möglich, indem z.B. explizite Sprachelemente angeboten werden, mit denen die Inverse einer Matrix analytisch vorgegeben werden kann. Das Tearing-Verfahren hat noch weitere, kleinere Nachteile. Zum einen wird nicht erkannt, daß die Matrix des entstehenden kleinen Gleichungssystems symmetrisch ist, was verhindert, daß effizientere und robustere Gleichungslöser angewendet werden können. Wahrscheinlich kann das Tearing-Verfahren aber entsprechend verallgemeinert werden, um eine solche Eigenschaft (automatisch) festzustellen. Zum anderen muß der Anwender die Tearing-Gleichungen und Tearing-Variablen vorgeben. Hier erscheint es denkbar, daß in der MKS-Bibliothek Tearing-Gleichungen und -Variablen vordefiniert werden können, die jeweils als Voreinstellung für die am häufigsten eingesetzte Aufgabenstellung verwendet werden.

Trotz der noch bestehenden kleineren Probleme, können zumindest Mehrkörpersysteme in Baumstruktur durch Verwendung des $O(n)$ Verfahrens zufriedenstellend und effizient behandelt werden. Die einfache und sichere Unterstützung bei der Formulierung von strukturvariablen Mehrkörpersystemen ist hier eine Eigenschaft, die bei vielen MKS-Programmen nicht gegeben ist.

Kapitel 5

Objektorientierte Modellierung von Antriebssträngen

Ein wichtiges Element mechanischer Systeme sind Antriebsstränge. Ein mechanischer Antriebsstrang besteht aus Wellen und Getrieben, wobei Elastizität, Dämpfung, Reibung und Lose vorhanden sein können. Da solche Antriebsstränge mechanische Bauteile sind, könnte man zur Modellierung die in Kapitel 4 diskutierten, allgemeinen Methoden der Mehrkörperdynamik einsetzen. Dies ist jedoch aus mehreren Gründen nicht zweckmäßig:

Zum einen erlaubt die rein 1-dimensionale Bewegungsrichtung von Antriebssträngen eine Reihe von Vereinfachungen für die Anwendersicht. Zum anderen können aufgrund der Rotationssymmetrie der bewegten Körper die dynamischen Gleichungen effizienter formuliert werden. Weiterhin bestehen Antriebsstränge immer aus einfachen kinematischen Schleifen, die speziell behandelt werden sollten, da dies viel effizienter möglich ist als bei allgemeinen 3-dimensionalen kinematischen Schleifen.

Die hier vorgestellte Modellierung von Antriebssträngen basiert auf einer Spezialisierung der Methoden aus Kapitel 4 auf 1-dimensionale mechanische Systeme. In einem ersten Schritt werden nur raumfeste Antriebsstränge betrachtet. Dies ergibt einfache und übersichtliche Gleichungen, an denen der O(n) Algorithmus für das direkte Problem, sowie die Behandlung strukturvariabler Systeme, besonders deutlich dargestellt werden können. Insbesondere zeigt sich, daß der O(n) Algorithmus im 1-dimensionalen Fall auf die aus der Maschinendynamik bekannten "reduzierten Trägheitsmomente" führt.

In einem zweiten Schritt werden Antriebsstränge betrachtet, die sich auf (3-dimensional) bewegten Körpern befinden und eventuell andere Starrkörper antreiben. Dies ist z.B. bei Antriebssträngen von Robotern oder bei Antriebssträngen auf Satelliten der Fall. Es wird gezeigt, daß durch geschickte Ausnutzung der Rotationssymmetrie nur einige wenige Terme zu den bestehenden dynamischen Gleichungen hinzugefügt werden müssen, um die dynamischen Effekte von 3-dimensional bewegten Antriebssträngen zu erfassen. Erfreulicherweise ändert sich dadurch die Anwendersicht überhaupt nicht. Die Definition eines Antriebsstrangs erfolgt genauso wie im rein 1-dimensionalen Fall. Es muß nur noch zusätzlich angegeben werden auf welchem Körper der Antriebsstrang befestigt ist und welcher andere Körper von ihm angetrieben wird. Die Effizienz des gesamten Mehrkörpersystem-Modells wird durch Antriebsstränge nicht wesentlich verringert. Bislang begnügt man sich häufig mit Näherungen (wie raumfesten Antriebssträngen), um die "komplizierte" 3-dimensionale Mo-

dellierung nicht durchführen zu müssen. Mit dem hier erläuterten Vorgehen erscheint dies nicht länger nötig, da die Berücksichtigung aller dynamischen Effekte von Antriebssträngen einfach möglich ist, vom Anwender keinen zusätzlichen Modellierungsaufwand erfordert, und nur zu einigen wenigen rechnerischen Zusatzoperationen führt.

5.1 Einführungsbeispiel

Anhand der Modellierung eines einfachen Antriebsstrangs soll zuerst die prinzipielle Modellierungssicht verdeutlicht werden. Im nächsten Abschnitt werden dann die verwendeten Klassen im Detail besprochen.

In Bild 5.1 ist schematisch ein einfacher Antriebsstrang zu sehen. Das von einem Elektromotor erzeugte Luftspaltmoment u treibt eine Welle 1, die über eine erste Getriebestufe $g1$ eine Welle 2 antreibt. Das Rotationsträgheitsmoment des "linken" Getrieberades wird zum

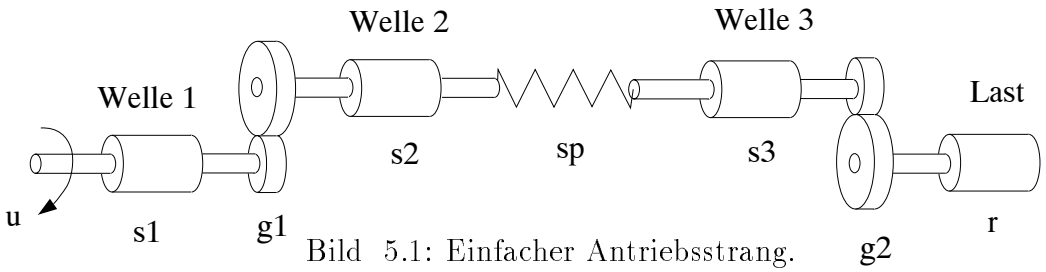


Bild 5.1: Einfacher Antriebsstrang.

Trägheitsmoment von Welle 1 und das Trägheitsmoment vom "rechten" Getrieberad wird zum Trägheitsmoment von Welle 2 geschlagen, so daß das Getriebe als ideales, trägheitsloses Übertragungselement behandelt werden kann. Die Welle 2 ist über ein Feder-Dämpferelement, das die Elastizitäten und Dämpfungen im Getriebe näherungsweise berücksichtigt, mit einer Welle 3 gekoppelt, die über eine zweite Getriebestufe $g2$ die Last antreibt. Dieses System kann direkt in das folgende Modell überführt werden:

```
model Drive
    submodel (DriveBase)    b
    submodel (Shaft)        s1 (J = 0.01), s2 (J = 0.1), s3 (J = 0.1)
    submodel (Rotor)        r  (J = 1)
    submodel (Gear)         g1 (i = −5), g2 (i = −5)
    submodel (DriveSpring)  sp (c = 1, d = 0.1)
    submodel (DriveVarS)    v1, v2

    input u

    connect b to v1 to s1 to g1 to s2 to sp to s3 to g2 to r, s2 to v2 to s3

    u = v1.f
end
```

Jeder Antriebsstrang muß genau ein Objekt der Klasse *DriveBase* enthalten. Dieses Objekt stellt die Basis bzw. die Umgebung dar, in der der Antriebsstrang gelagert ist. Trägheitsbehaftete Wellen werden mit der Klasse *Shaft* beschrieben, wobei der Parameter J das Trägheitsmoment um die Drehachse ist. Im obigen Modell werden die 3 Wellen $s1$, $s2$, $s3$ deklariert. Ein *Rotor* ist eine Welle, die nur einen Flansch besitzt und deswegen den Anfang oder Abschluß eines Antriebsstrangs darstellt. Ideale Getriebe werden mit der Klasse *Gear*

definiert und als trägheitslose Elemente aufgefaßt. Die Trägheitsmomente von Getriebestufen müssen durch Wellen extra modelliert werden. Der Parameter i eines Getriebeobjekts ist die Getriebeübersetzung. Eine negative Übersetzung bedeutet, daß An- und Abtrieb unterschiedliche Drehrichtungen besitzen. Lineare Elastizitäten und Dämpfungen im Antriebsstrang werden durch Objekte der Klasse *DriveSpring* beschrieben. Die Elemente eines Antriebsstrangs werden nun einfach wieder entsprechend der *physikalischen* Verbindungsstruktur mittels der **connect**-Anweisung zusammengeschaltet (oben: **connect** ..., $s1$ **to** $g1$ **to** ...r).

Wenn ein Antriebsstrang simuliert werden soll, muß festgelegt werden, welche Variablen als Zustandsgrößen benutzt werden. Bei Mehrkörpersystemen in Baumstruktur ist die Wahl von Relativkoordinaten in einem Gelenk fast immer die beste Wahl für die Zustandsgrößen. Aus diesem Grund wurden in Kapitel 4 bei allen Gelenken Klassen unterstützt, bei denen die Gelenkrelativkoordinaten automatisch als Zustandsgrößen definiert sind. Bei Antriebssträngen ist die Sachlage anders. Hier sind sowohl die Absolutwinkel und Absolutwinkelgeschwindigkeiten von Rotationskomponenten als auch die Relativwinkel und Relativwinkelgeschwindigkeiten zwischen Rotationskomponenten sinnvolle Zustandsgrößen. Deswegen werden bei Antriebssträngen die Zustandsgrößen explizit mit Hilfe der Klasse *DriveVarS* ausgewählt. Ein Objekt dieser Klasse wird zwischen zwei Cuts unterschiedlicher Komponenten angebracht. Der Relativwinkel und die Relativwinkelgeschwindigkeit dieser beiden Cuts zueinander, werden als Zustandsgrößen benutzt. Wenn einer der beiden Cuts mit der *DriveBase* verbunden ist, dann sind die Zustandsgrößen Absolutvariablen.

Um Zustandsgrößen bei Antriebssträngen auszuwählen, werden zuerst alle *starren* Übertragungselemente, die ohne Kraftelemente zusammengeschaltet sind, als eine Einheit aufgefaßt. Im obigen Modell bilden z.B. die Wellen 1 und 2, sowie die Getriebestufe 1, eine solche Einheit. Jede "Einheit" wird durch zwei Zustandsgrößen beschrieben. Dies sind ein Relativwinkel bezüglich einer anderen "Einheit" oder bezüglich der Antriebsbasis, sowie dessen erste Ableitung. Im obigen Modell werden als Zustandsgrößen der absolute Winkel und die absolute Winkelgeschwindigkeit der Welle 1 (**connect** b **to** $v1$ **to** $s1$), sowie der relative Winkel und die relative Winkelgeschwindigkeit zwischen Welle 2 und Welle 3, gewählt (**connect** $s2$ **to** $v2$ **to** $s3$).

Die obige Vorgehensweise zur Wahl der Zustandsgrößen ist unüblich. Zum Beispiel werden in der Bondgraph-Literatur (siehe z.B. [Karn90]) als Zustandsgrößen bei 1-dimensionalen mechanischen Systemen immer die Absolutwinkel und die Absolutwinkelgeschwindigkeiten *jedes* trägheitsbehafteten Bauteils benutzt. Meist führt das jedoch auf differential-algebraische Gleichungssysteme mit einem höheren Index, da trägheitsbehaftete Bauteile oft direkt, z.B. über ein Getriebe, miteinander gekoppelt sind. Damit reduziert sich die Zahl der Zustandsgrößen, weil durch die feste Verbindung eine Beziehung zwischen den Zustandsgrößen besteht. Dies ist zum einen lästig, da dann immer der Algorithmus von Pantelides (siehe Seite 36) angewandt werden muß um das System auf ein Index-1 System zu transformieren; zum anderen ist es bei strukturvariablen Elementen, wie z.B. bei einer Kupplung, oft günstiger, Relativwinkel bzw. Relativwinkelgeschwindigkeiten als Zustandsgrößen zu benutzen.

5.2 Aufgabeninvariante Grundelemente

In diesem Abschnitt werden die wenigen Modellklassen eingeführt, die zur objektorientierten Modellierung von Antriebssträngen benötigt werden. Wie in Kapitel 4 wird auch hier immer zusätzlich die entsprechende Bondgraph-Beschreibung angegeben, um den Leistungsfluß zu beschreiben.

Element-Verbindungen

Jedes Element eines Antriebsstrangs hat entweder einen oder zwei Flansche, womit das Element mechanisch starr mit den Flanschen anderer Elemente verbunden werden kann. Genau wie bei den elektrischen Komponenten mit der Oberklasse *TwoPin*, werden deswegen bei Antriebssträngen die beiden Oberklassen *DriveOneCut, DriveTwoCut* zur Beschreibung von Elementen mit denselben Verbindungseigenschaften eingeführt. Die beiden Klassen haben den folgenden Aufbau:

```
model class DriveOneCut
    main cut a ( ra, va, aa / ta)
    terminal Pa

    Pa = ta * va
end
```

```
model class DriveTwoCut
        cut   a   (ra, va, aa / ta)
        cut   b   (rb, vb, ab / tb)
    main cut   mc [a,b]
    main path  mp < a - b >

    terminal Pa, Pb

    Pa = ta * va
    Pb = tb * vb
end
```

Über den Flansch eines Elements, der hier als Cut bezeichnet wird, werden der absolute Winkel (ra, rb), die absolute Winkelgeschwindigkeit (va, vb), die absolute Winkelbeschleunigung (aa, ab) und das resultierende Schnittmoment (ta, tb) am Flansch übertragen. Zusätzlich wird noch über die Terminal-Variablen Pa, Pb der in einen Flansch *einfließende* Energiestrom berechnet.

Starre Übertragungselemente

In Bild 5.2 sind die Übertragungselemente von Antriebssträngen zusammen mit ihrer Bondgraph-Darstellung angegeben. Die physikalischen Gesetze der einzelnen Komponenten sind sehr einfach, wie den Klassenbeschreibungen zu entnehmen ist. Die Klasse *DriveBase* beschreibt die Umgebung in der der Antriebsstrang gelagert ist. Hier werden, entsprechend zur Klasse *Inertial* bei Mehrkörpersystemen, die kinematischen Größen auf Null gesetzt. In der Bondgraphdarstellung entspricht das einer Flow-Source SF, da die Flow-Variablen vorgegeben sind. Die Klasse *Rotor* definiert eine trägheitsbehaftete Welle mit *einem* Flansch und wird durch das Newton'sche Gesetz "$J\,\alpha_a = \tau_a$" beschrieben. Ganz entsprechend werden trägheitsbehaftete Wellen mit *zwei* Flanschen mit der Klasse *Shaft* und "$J\,\alpha_a = \tau_a + \tau_b$" beschrieben. Schließlich werden bei der Klasse *Gear* die abtriebsseitigen Größen aus den antriebsseitigen Größen mit Hilfe der Übersetzung i bestimmt.

model class (*DriveOneCut*) *DriveBase*
$ra = 0$
$va = 0$
$aa = 0$
end

model class (*DriveOneCut*) *Rotor*
parameter *J=0*
$J * aa = ta$
end

model class (*DriveTwoCut*) *Shaft*
parameter *J=0*
$ra = rb$
$va = vb$
$aa = ab$
$J * aa = ta + tb$
end

model class (*DriveTwoCut*) *Gear*
parameter *i=0*
$ra = i * rb$
$va = i * vb$
$aa = i * ab$
$0 = i * ta + tb$
end

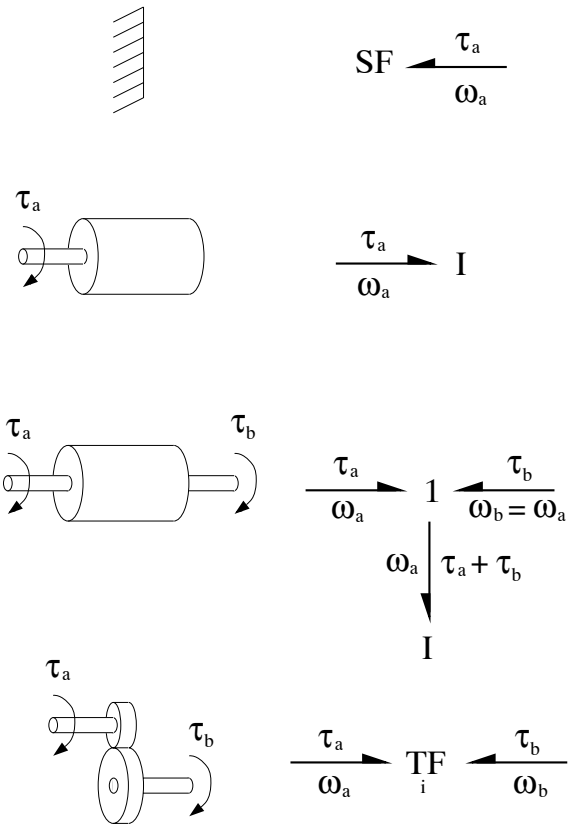


Bild 5.2: Starre Übertragungselemente von Antriebssträngen.

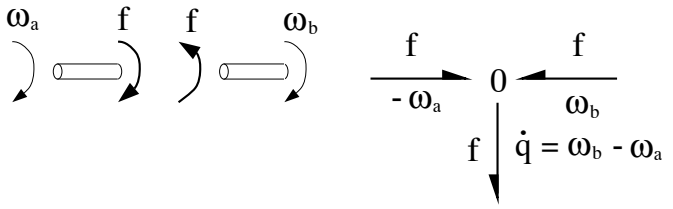
Kraftelemente

Kraftelemente "trennen" einen Antriebsstrang an einer Stelle auf und übertragen gemäß dem actio=reactio Prinzip auf die beiden entstehenden Wellenenden, jeweils ein gleichgroßes Moment f mit entgegengesetztem Vorzeichen. In Bild 5.3 sind die drei Basisklassen der Kraftelemente aufgeführt. Die erste Basisklasse, *DriveForce*, wird als Oberklasse aller "normalen" Kraftelemente benutzt, z.B. eines Feder-Kraftelements. In dieser Klasse werden der Relativwinkel q, die Relativwinkelgeschwindigkeit qd und die Relativwinkelbeschleunigung qdd des Cuts b bezüglich des Cuts a berechnet, da diese in der Regel im Kraftelement benötigt werden. Ein von der Klasse *DriveForce* abgeleitetes Element muß dann das vom Kraftelement erzeugte Drehmoment f als Funktion der Relativgrößen zur Verfügung stellen. Zum Beispiel hat die Klasse *DriveSpring* zur Beschreibung einer Drehfeder und eines dazu parallel geschalteten Drehdämpfers den folgenden Aufbau:

model class (*DriveTwoCut*) *DriveForce*
local *q*, *qd*, *qdd*, *f*

$q = rb - ra$
$qd = vb - va$
$qdd = ab - aa$
$ta = -f$
$tb = f$

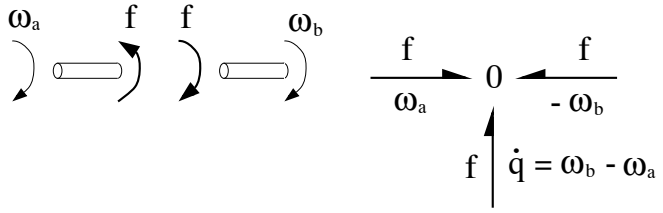
end



model class (*DriveTwoCut*) *DriveExt*
cut *ext* (*q*, *qd*, *qdd* / *f*)

$q = rb - ra$
$qd = vb - va$
$qdd = ab - aa$
$ta = f$
$tb = -f$

end



model class (*DriveOneCut*) *DriveExtBase*
cut *ext* (*q*, *qd*, *qdd* / *f*)

$q = ra$
$qd = va$
$qdd = aa$
$ta = -f$

end

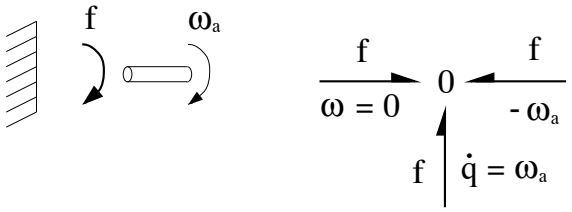


Bild 5.3: Oberklassen der Kraftelemente von Antriebssträngen.

model class (*DriveForce*) *DriveSpring*
parameter c=0, d=0, q0=0

$f = c * (q - q0) + d * qd$

end

Die zweite Basisklasse *DriveExt* von Bild 5.3 erlaubt über den zusätzlichen Cut *ext*, daß ein externes Kraft*gesetz* an ein Objekt der Klasse *DriveExt* angeschlossen werden kann. Hierbei werden die Vorzeichen der Drehmomente so gewählt, daß ein über diesen Cut einfließender Energiestrom positiv zählt. Externe Krafgesetze wurden schon auf Seite 80 besprochen. In ihnen wird nur eine Kraft, bzw. ein Moment, als Funktion kinematischer Relativgrößen berechnet. Mittels Hilfsklassen, wie *DriveExt*, wird die berechnete Kraft, bzw. das berechnete Moment, auf ein Mehrkörpersystem oder einen Antriebsstrang aufgeschaltet. Der Vorteil dieser Aufspaltung in "Angriffspunkt und Richtung" und in "Kraftgesetz" liegt darin, daß dasselbe Kraftgesetz z.B. in einem Antriebsstrang, in einem MKS-Gelenk oder in einem MKS-Kraftelement benutzt werden kann. Insbesondere kann als externes Kraftgesetz auch das auf Seite 120 im Detail besprochene Reibelement verwendet werden.

Die dritte Basisklasse, *DriveExtBase*, ist ein Spezialfall von *DriveExt*, wobei einer der beiden Antriebsstrang-Cuts die Basis des Antriebsstrangs bildet. Das heißt, dieses Kraftelement

überträgt ein Drehmoment zwischen einer Antriebsstrangkomponente und der Basis. Es ist manchmal zweckmäßig ein solches Kraftelement zur Verfügung zu haben, z.B. wenn ein Antriebsstrang hierarchisch aufgebaut wird, und wenn auf der aktuellen Modellebene die Basis, d.h. das Objekt der Klasse *DriveBase*, nicht zur Verfügung steht. Ansonsten fließt auch hier über den Cut *ext* ein Energiestrom eines externen Kraftgesetzobjekts ein.

Zustandsgrößen

Die bis jetzt besprochenen Klassen reichen aus, um Antriebsstränge zu modellieren. Es ist jedoch zweckmäßig noch weitere Klassen einzuführen, mit denen festgelegt wird, welche Größen als Zustandsvariablen benutzt werden. Im Zusammenhang mit der Spezialisierung des $O(n)$ Verfahrens von Mehrkörpersystemen werden diese Klassen vor allem auch deswegen benötigt, weil einige zusätzliche Gleichungen in diesen Klassen aufgeführt werden müssen. Wie auf Seite 131 schon erläutert, werden als Zustandsgrößen der Relativwinkel und die Relativwinkelgeschwindigkeit zwischen den beiden Cuts benutzt, die durch ein Objekt einer Zustandsgrößenklasse *DriveVarXX* verbunden werden. Aus den folgenden zwei Basisklassen werden die anderen Klassen durch Vererbung abgeleitet:

```
model class (DriveTwoCut) DriveVar
    cut ext (q, qd, qdd / f)

    q   = rb - ra
    qd  = vb - va
    qdd = ab - aa
    ta  = f
    tb  = -f
end
```

```
model class (DriveTwoCut) DriveVarL
    cut ext  (q, qd, qdd / f)
    cut extL (q, qd, qdd, Locked / fL, fLc)

    q   = rb - ra
    qd  = vb - va
    qdd = ab - aa
    ta  = f + fL + (if Locked then fLc else 0)
    tb  = -ta
    0   = if Locked then qdd else fLc
end
```

Die Klasse *DriveVar* berechnet den relativen Winkel q, die relative Winkelgeschwindigkeit qd und die relative Winkelbeschleunigung qdd. Über den zusätzlichen Cut *ext*, können externe Kraftgesetze angeschlossen werden. Die Klasse *DriveVarL* ist eine Erweiterung der Klasse *DriveVar*, um die relativen kinematischen Größen *blockieren* und *wiederfreigeben* zu können. Diese Eigenschaft wird zur Behandlung von Lager- oder Getriebe-Reibung, von Kupplungen und von Bremsen benötigt. Über den Cut *extL* können "blockierbare" externe Kraftgesetze, wie das Reibelement, angeschlossen werden. Wie üblich gibt die Boole'sche Variable *Locked* an, ob die Relativgrößen "blockiert" sein sollen oder nicht. Das Drehmoment fLc ist das Zwangsdrehmoment, wenn $Locked = True$ ist. Die Gleichung für das am Cut a wirkende Drehmoment ta besteht aus der Summe aller Drehmomente, die im Kraftelement erzeugt werden. Sehr wichtig ist dann noch die letzte Gleichung in dieser Klasse, in der angegeben wird, daß die Relativbeschleunigung verschwindet, wenn das Element "blockiert". Dies entspricht vollkommen der auf Seite 83 besprochenen Vorgehensweise bei der Behandlung blockierbarer Gelenke.

Basierend auf den obigen beiden Klassen, werden zuerst zwei einfache Unterklassen gebildet, in denen nur festgelegt wird, daß der Relativwinkel und die Relativwinkelgeschwindigkeit die Zustandsgrößen sind:

```
model class (DriveVar) DriveVarS
    qd  = der(q)
    qdd = der(qd)
end
```

```
model class (DriveVarL) DriveVarLS
    qd  = der(q)
    qdd = der(qd)
end
```

Es gibt zwei prinzipiell unterschiedliche Arten der Anwendung von Antriebssträngen:

Bei der einen Anwendungsart spielt die *Drehlage* des Antriebsstrangs eine wichtige Rolle, da diese z.B. die Position eines Drehgelenks bei einem Roboter oder die Position eines Führungsschlittens bei einer Werkzeugmaschine bestimmt.

Bei der anderen Anwendungsart kann der *Absolutwinkel* der Antriebsstrangkomponenten sehr groß werden und ist für die eigentliche Aufgabenstellung unerheblich. Dies ist z.B. beim Antriebsstrang eines Fahrzeugs oder beim Antrieb eines Bohrfutters einer Werkzeugmaschine der Fall. Hier ist es nicht nötig, daß der Absolutwinkel jeder Komponente immer berechnet wird. Überdies kann diese Berechnung numerisch ungünstig sein, da der Absolutwinkel mit der Zeit beliebig ansteigt und damit eine "instabile" Variable ist. Es ist einfach, die obigen Klassen auf diesen Fall zu spezialisieren. Da man sich nur für die Winkelgeschwindigkeiten interessiert, müssen nur die *Zustands*-Lagegrößen zu Null gesetzt werden. Damit sind automatisch die Absolutwinkel aller Drehteile Null. Wenn dieser Effekt gewünscht ist, müssen statt der obigen Klassen *DriveVarS*, bzw. *DriveVarLS*, die folgenden beiden Klassen benutzt werden:

```
model class (DriveVarS) DriveVarWS
    q   = 0
    qdd = der(qd)
end
```

```
model class (DriveVarLS) DriveVarLWS
    q   = 0
    qdd = der(qd)
end
```

Problemlösung

Mit den obigen Klassen können leicht Antriebsstränge modelliert werden. Damit können dann z.B. die Gleichungen für das Simulationsproblem oder für eine Stationärwertberechnung erzeugt werden. Aufgrund der einfachen Gleichungen reichen in der Regel die in Kapitel 2 erläuterten Methoden aus, wie: Transformation auf Blockdreiecksform, Ermittlung kleinster algebraischer Schleifen, symbolisches oder numerisches Lösen algebraischer Schleifen. Für die häufigste Aufgabenstellung – das Simulationsproblem – soll im folgenden die Lösungsstruktur weiter erläutert und an einem Beispiel veranschaulicht werden.

Wie im Einführungsbeispiel auf Seite 131 erläutert, werden alle starr miteinander verbundenen Übertragungselemente (d.h. Wellen und Getriebe) als eine Einheit aufgefaßt. Die Lage einer solchen Einheit wird eindeutig durch *einen* Winkel einer Komponente einer solchen Einheit bezüglich der Basis, oder bezüglich einer anderen Einheit, gekennzeichnet. Alle Einheiten sind über Kraftelemente miteinander gekoppelt. Angenommen, die Lage der Einheiten eines Antriebsstrangs werde jeweils durch einen Absolutwinkel, und die Winkelgeschwindigkeit werde durch eine absolute Winkelgeschwindigkeit charakterisiert. Diese bekannten Zustandsgrößen können dann benutzt werden, um den Absolutwinkel und die Absolutwinkelgeschwindigkeit von *jedem* anderen Element des Antriebsstrangs zu berechnen, da dies eine einfache Vorwärtsrekursion ist. Wenn alle Absolutgrößen bekannt sind,

können sofort auch alle gewünschten Relativgrößen berechnet werden. Damit können wiederum die eingeprägten Drehmomente eines jeden Kraftelements bestimmt werden. Jetzt können die Antriebsstrang-Einheiten voneinander entkoppelt gesehen werden, weil die nun bekannten Drehmomente der Kraftelemente die einzige Kopplung zwischen diesen Einheiten darstellen. Das heißt, jede Einheit kann nun für sich behandelt werden und für jede Einheit verbleibt ein lineares Gleichungssystem zur Bestimmung der Ableitungen ihrer beiden Zustandsgrößen.

Zusammengefaßt bedeutet dies, daß die Transformation auf Blockdreiecksform zu einem Gleichungssystem führt, in dem auf der Diagonalen für jede Antriebsstrang-Einheit genau ein nichttrivialer Block vorhanden ist, der zu einem linearen Gleichungssystem gehört und dessen Größe von der Zahl der Übertragungselemente der jeweiligen Einheit abhängt. Diese Diagonalblöcke sind voneinander entkoppelt. Man kann sich leicht überlegen, daß diese Struktur auch beibehalten wird, wenn als Zustandsvariablen statt absoluter Größen relative Größen, oder ein Gemisch absoluter und relativer Größen, benutzt werden. Zum Beispiel ergibt sich für das Einführungsbeispiel von Bild 5.1 auf Seite 130, das Absolut- und Relativgrößen als Zustandsvariablen benutzt, die folgende Blockdreiecksform

$$
\begin{array}{lll}
g1. & v1.q & = g1.i * [g1.rb] \\
v2. & v2.q & = [v2.rb] - g1.rb \\
sp. & [sp.q] & = v2.q \\
g1. & v1.qd & = g1.i * [g1.vb] \\
v2. & v2.qd & = [v2.vb] - g1.vb \\
sp. & [sp.f] & = sp.c * v2.q + sp.d * v2.qd \\
 & & \\
| \ g1. & v1.qdd & = g1.i * [g1.ab] \\
| \ s2. & s2.J * g1.ab & = [s2.ta] + sp.f \\
| \ g1. & s2.ta & = g1.i * [g1.ta] \\
| \ s1. & s1.J * [v1.qdd] & = u - g1.ta \\
 & & \\
| \ g2. & s3.aa & = g2.i * [g2.ab] \\
| \ r. & r.J * g2.ab & = -[g2.tb] \\
| \ g2. & 0 & = g2.tb + g2.i * [g2.ta] \\
| \ s3. & s3.J * [s3.aa] & = -sp.f + g2.ta \\
 & & \\
v2. & [v2.qdd] & = s3.aa - g1.ab
\end{array}
$$

In der linken Spalte ist jeweils angegeben, aus welcher Klasse die rechts stehende Gleichung stammt. Die Gleichungen sind hier schon in der richtigen Reihenfolge sortiert. In einem nächsten Schritt werden die Gleichungen nach den in eckigen Klammern stehenden Variablen aufgelöst. Ein senkrechter Strich "|" bei einer Gleichung zeigt an, daß diese Gleichung zu einem algebraischen Gleichungssystem gehört. Wie man sieht, gibt es in der Blockdreiecksform zwei voneinander entkoppelte lineare Gleichungssysteme mit jeweils 4 Gleichungen. Dies entspricht den beiden "Einheiten" bestehend aus {Welle 1, Getriebe 1, Welle 2}, sowie {Welle 3, Getriebe 2, Rotor}. In die beiden Gleichungssysteme gehen jeweils zwei unbekannte Beschleunigungen und zwei unbekannte Schnittkräfte ein. Beide Gleichungssysteme können problemlos symbolisch oder numerisch gelöst werden.

In Kapitel 4.3.4 ab Seite 101 wurde gezeigt, wie mit Hilfe des *Tearing*-Verfahrens die Gleichungen eines Mehrkörpersystems auf ein (lineares) algebraisches Gleichungssystem mit f Gleichungen überführt werden können (f = Zahl der Freiheitsgrade des Systems). Es ist naheliegend dieses Verfahren auch auf Antriebsstränge anzuwenden. Da bei Antriebssträngen schon entkoppelte Gleichungssysteme vorhanden sind, ist anzunehmen, daß das Tearing-

Verfahren zu f entkoppelten Gleichungen führt, wobei f die Zahl der Freiheitsgrade, bzw. die Zahl der Antriebsstrang-Einheiten, ist. Dies ist tatsächlich der Fall:

Als Tearing-Variable für eine Antriebsstrang-Einheit wird immer die (unbekannte) Ableitung der Winkelgeschwindigkeits-Zustandsgröße (= qdd) benutzt. Die Tearing-Gleichung ermittelt man wieder anschaulich aus der Eigenschaft, daß bei bekannter Tearing-Variable alle anderen Gleichungen aufgelöst werden können, wobei die Tearing-Gleichung als letzte übrig bleiben muß. Hier ist dies die dynamische Gleichung derjenigen Komponente, an der das Zustandsvariablen-Objekt der entsprechenden Antriebsstrang-Einheit angebracht ist. Zum Beispiel sind die Tearing-Variablen in Bild 5.1 auf Seite 130 die Größen $v1.qdd, v2.qdd$, und die Tearing-Gleichungen sind die dynamischen Gleichungen von Welle 1 (s1) und Welle 3 (s3).

Diese Wahl der Tearing-Variablen und -Gleichungen liegt folgende Überlegung zugrunde: Wenn die Winkelbeschleunigung qdd bekannt ist, können die Winkelbeschleunigungen aller anderen Komponenten der entsprechenden Antriebsstrang-Einheit ermittelt werden. Im ungünstigsten Fall hat eine solche Einheit zwei Enden, an denen bekannte, eingeprägte Drehmomente angreifen. Die dynamischen Gleichungen der Komponenten an den beiden Enden können über die dynamische Gleichung "$J\,\alpha = \tau_a + \tau_b$" nach dem inneren Zwangsmoment (z.B. τ_a) aufgelöst werden wobei α und τ_b bekannt sind und τ_a berechnet wird. Danach werden die beiden äußeren Komponenten entfernt und durch ihr (jetzt bekanntes) Zwangsmoment ersetzt. Dann liegt wieder dieselbe Situation vor wie am Anfang, da das "innere" Zwangsmoment der äußeren Komponente wieder über die dynamische Gleichung bestimmt werden kann. Diese Rückwärtsrekursion endet an derjenigen Komponente, an der das Zustandsvariablen-Objekt angebracht ist.

Wenn das Modell von Bild 5.1 auf Seite 130 um die folgenden Gleichungen erweitert wird

variable tear $v1.qdd\,[s1.ta]$
variable tear $v2.qdd\,[s3.ta]$,

um damit die Tearing-Variablen und -Gleichungen festzulegen, können die entstehenden algebraischen Gleichungssysteme vollkommen aufgelöst werden. Dies führt zu dem folgenden Ergebnis:

$$
\begin{array}{lll}
sp. & sp.f & = sp.c * v2.q + sp.d * v2.qd \\
 & & \\
| & Q101 & = 1/g1.i \\
| & Q102 & = s2.J * Q101 \\
| & Q103 & = Q102/g1.i \\
| & Q108 & = s1.J + Q103 \\
| & Q107 & = Q101 * sp.f \\
| & Q109 & = u + Q107 \\
| & v1.qdd & = Q109/Q108 \\
 & & \\
g1. & g1.ab & = v1.qdd/g1.i \\
| & Q110 & = 1/g2.i \\
| & Q111 & = r.J * Q110 \\
| & Q112 & = Q111/g2.i \\
| & Q116 & = s3.J + Q112 \\
| & v2.ab & = -sp.f/Q116 \\
 & & \\
sp. & v2.qdd & = v2.ab - g1.ab
\end{array}
$$

Ein senkrechter Strich am linken Rand zeigt die Gleichungen an, die beim Auflösen der

linearen Gleichungssysteme mittels Tearing entstanden sind. Die Variablen $Qxxx$ sind (automatisch) neu eingeführte Hilfsvariablen. Eine genauere Analyse der Auflösung zeigt, daß beim Tearing die Trägheitsmomente der Komponenten auf dasjenige Trägheitsmoment "reduziert" werden, an dem das Zustandsvariablen-Objekt (hier: $v1, v2$) befestigt ist. Zum Beispiel ergibt eine Zusammenfassung der Gleichungen des ersten Gleichungssystems die Gleichung

$$(s1.J + s2.J/(g1.i \cdot g1.i)) \cdot v1.qdd = u + sp.f/g1.i$$

Das Tearing-Verfahren führt also bei Antriebssträngen automatisch zu den aus der Maschinendynamik bekannten, reduzierten Trägheitsmomenten.

5.3 Effiziente Bewegungsgleichungen

In Kapitel 4.4 ab Seite 111 wurde ein effizientes $O(n)$ Verfahren für das Simulationsproblem bei Mehrkörpersystemen erläutert. Dieses kann einfach auf Antriebsstränge spezialisiert werden. Die wesentliche Idee des $O(n)$ Algorithmus besteht darin, die Schnittkräfte und Schnittmomente als lineare Funktionen der Beschleunigung und Winkelbeschleunigung am Schnitt anzugeben. Bei Antriebssträngen führt dies zu der Gleichung:

$$\tau = I\alpha + b \ . \tag{5.1}$$

Hierbei ist τ das Schnittmoment an einem Cut, α ist die absolute Winkelbeschleunigung an diesem Cut, und I, b sind die (noch unbekannten) linearen Faktoren. Wenn die linearen Faktoren I, b an einem Cut der starren Übertragungselemente (= Wellen, Getriebe) bekannt sind, können die entsprechenden linearen Faktoren am anderen Cut berechnet werden. Mit den Gleichungen von Bild 5.2 auf Seite 133 und den Beziehungen $\tau^a = I^a\,\alpha^a + b^a$, $\tau^b = I^b\,\alpha^b + b^b$ ergibt sich:

$$\begin{array}{lllll}
\textit{Rotor}: & J\,\alpha_a = \tau_a & \Longrightarrow & I_a = J \ ; & b_a = 0 \\
\textit{Shaft}: & J\,\alpha_a = \tau_a + \tau_b & \Longrightarrow & J = I_a + I_b \ ; & 0 = b_a + b_b \\
\textit{Gear}: & 0 = i\,\tau_a + \tau_b & \Longrightarrow & 0 = i\,i\,I_a + I_b; & 0 = i\,b_a + b_b \quad .
\end{array}$$

Ein Kraftelement berechnet aus bekannten Winkeln und Winkelgeschwindigkeiten ein eingeprägtes Drehmoment f, das an zwei Cuts mit unterschiedlichem Vorzeichen angreift. Damit kann dieses Drehmoment ebenfalls in der geforderten Form (5.1) angegeben werden: $I = 0, b = f$.

Die Rekursionsvorschrift für einen Antriebsstrang ergibt sich jetzt ähnlich wie beim Tearing-Verfahren. Im ungünstigsten Fall hat eine Antriebsstrang-Einheit zwei Enden, wobei die linearen Faktoren an den beiden äußersten Cuts aus bekannten Größen berechnet werden können, weil die Elemente an den Enden entweder Kraftelemente oder Rotoren sind. Liegen Rotoren vor, werden diese entfernt. Dann sind die beiden neuen Enden des Antriebsstrangs auf jeden Fall entweder Wellen oder Getriebe, für die die linearen Faktoren der äußersten Cuts bekannt sind. Da die linearen Faktoren am einen Cut bekannt sind, können sie am jeweils anderen Cut berechnet werden. Wenn dies geschehen ist, werden die beiden Komponenten entfernt und es liegt wieder der vorhergehende Fall vor. Auf diese Weise wird eine Rekursion aufgebaut, die an derjenigen Komponente endet, an der das Zustandsvariablen-Objekt befestigt ist.

Beim Zustandsvariablen-Objekt selbst muß jetzt eine Annahme getroffen werden, welcher Cut des Objekts an der gerade betrachteten Einheit befestigt ist. Hier wird angenommen, daß dies immer der Cut b ist. Aufgrund der Rekursion wird die Schnittkraft an diesem Cut durch bekannte lineare Faktoren angegeben. Andererseits ist die Schnittkraft f an diesem Cut aber auch ein bekanntes, eingeprägtes Drehmoment (z.B. Null). D.h. in der Gleichung $f = I^b\,\alpha^b + b^b$ sind alle Größen bekannt, mit Ausnahme von α^b. Deswegen kann die absolute Winkelbeschleunigung α^b am Cut b aus dieser Gleichung berechnet werden zu: $\alpha^b = (f - b^b)/I^b$. Wenn das Zustandsvariablen-Objekt mit dem anderen Cut a an der Basis befestigt ist, so ist die gesuchte Ableitung der Zustandsgröße qdd gleich der berechneten Winkelbeschleunigung $qdd = \alpha^b$. Wenn Cut a an einer anderen Antriebsstrang-Einheit befestigt ist, so muß erst abgewartet werden, bis die Rekursion dieser Einheit beendet ist. Sobald dies der Fall ist, können über eine Vorwärtsrekursion alle Winkelbeschleunigungen dieser anderen Einheit berechnet werden. D.h. aber auch, daß α^a für das Zustandsvariablen-Objekt berechnet wird. Die noch unbekannte Ableitung der Zustandsgröße der als erstes untersuchten Einheit ergibt sich schließlich zu: $qdd = \alpha^b - \alpha^a$.

Man muß also nur die aufgestellten Rekursionsgleichungen in die jeweilige Klasse einbauen. Aufgrund der obigen Überlegungen ist dann sichergestellt, daß auf eine Blockdreiecksform transformiert werden kann, die explizit lösbar ist. Auch der Fall blockierbarer Komponenten kann einfach behandelt werden. Wenn ein Zustandsvariablen-Objekt der Klasse *DriveVarL* im blockierten Zustand ist ($Locked = True$), dann ist $\tau^a = -\tau^b$, und die linearen Faktoren von τ^b werden an τ^a weitergegeben. Dies führt zu folgender Rekursionsvorschrift:

$$
\begin{aligned}
I^a &= \textbf{if } Locked \textbf{ then } -I^b \textbf{ else } 0 && \text{(5.2a)}\\
b^a &= \textbf{if } Locked \textbf{ then } -b^b \textbf{ else } f + f_L && \text{(5.2b)}\\
qdd &= \textbf{if } Locked \textbf{ then } 0 \textbf{ else } -aux/I^b && \text{(5.2c)}\\
f_{Lc} &= \textbf{if } Locked \textbf{ then } -aux \textbf{ else } 0 && \text{(5.2d)}\\
aux &= f + f_L + b^b + I^b\,\alpha^a && \text{(5.2e)}\\
\alpha^b &= \alpha^a + qdd \quad . && \text{(5.2f)}
\end{aligned}
$$

Bei der obigen Ableitung wurde vorausgesetzt, daß am Cut b der Zustandsvariablen-Objekte die Rekursion für eine Antriebsstrang-Einheit endet. Dies bedeutet, daß Zustandsvariablen-Objekte *immer* so angeordnet werden müssen, daß Cut a entweder mit der Basis, oder mit einer anderen Einheit verbunden ist, die direkt oder indirekt über ein Zustandsvariablen-Objekt mit der Basis verbunden ist.

Da die Gleichungen jeder Komponente eines Antriebsstrangs nur genau einmal ausgewertet werden, berechnet der Algorithmus die unbekannten Größen in $O(n)$ Operationen, wobei n die Zahl der Antriebsstrang-Elemente ist. Wird dieses $O(n)$ Verfahren auf das Beispiel von Bild 5.1 aus Seite 130 angewandt, ergibt sich der folgende Code:

$$
\begin{array}{lll}
sp. & sp.f & = sp.c * v2.q + sp.d * v2.qd\\
g1. & g1.ba & = -sp.f/g1.i\\
g1. & g1.Ia & = s2.J/(g1.i * g1.i)\\
s1. & s1.Ia & = s1.J + g1.Ia\\
v1. & v1.qdd & = -(g1.ba - u)/s1.Ia\\
g2. & g2.Ia & = r.J/(g2.i * g2.i)\\
s3. & s3.Ia & = s3.J + g2.Ia\\
g1. & g1.ab & = v1.qdd/g1.i\\
v2. & v2.qdd & = -(sp.f/s3.Ia + g1.ab) \quad .
\end{array}
$$

Ein Vergleich mit den beiden vorher benutzten Methoden zeigt, daß der mit dem $O(n)$ Verfahren generierte Code schon bei diesem einfachen Antriebsstrang deutlich kompakter und effizienter ist. Das Ergebnis des Tearing-Verfahrens ist sehr ähnlich zum $O(n)$ Verfahren, da beim $O(n)$ Verfahren auch wiederum "reduzierte Trägheitsmomente" (z.B. "$s1.J + s2.J/(g1.i \cdot g1.i)$") berechnet werden. Der Code des Tearing-Verfahrens hat nur etwas andere Zwischengrößen.

5.4 Strukturvariable Antriebsstränge

Unter strukturvariablen Antriebssträngen sollen hier solche verstanden werden, bei denen sich die Zahl der Freiheitsgrade während einer Simulation, z.B. wegen einer Kupplung oder wegen einer Reibbremse, ändert.

Eine Problem besteht bisher darin, das "Blockieren" und "Wiederfreigeben" von Freiheitsgraden effizient zu behandeln. Wie im letzten Abschnitt gezeigt wurde, gibt es keinerlei Effizienzeinbußen, wenn das $O(n)$ Verfahren benutzt wird. Man ist dann allerdings in der Wahl der Zustandsvariablen nicht mehr frei, da die blockierbaren Relativgrößen als Zustände benutzt werden müssen. Dies ist jedoch meist keine wesentliche Einschränkung. Ein weiteres Problem besteht bisher in der Beschreibung und Behandlung der Schaltvorgänge. Mit den in Kapitel 3 erläuterten Methoden ist dieses Problem jedoch gelöst.

Das prinzipielle Vorgehen soll am Antriebsstrang von Bild 5.4 verdeutlicht werden. Dieses Beispiel wurde [Mitc90] entnommen und um eine Bremse erweitert. Der Antriebsstrang besteht aus einem Rotor $s3$, der von einem Motor über das Drehmoment u angetrieben wird. Um den Motor vor Lastüberhöhungen zu schützen, ist eine Rutschkupplung *clutch* am Rotor angebracht. Das abtriebsseitige Trägheitsmoment der Kupplung wird mit der

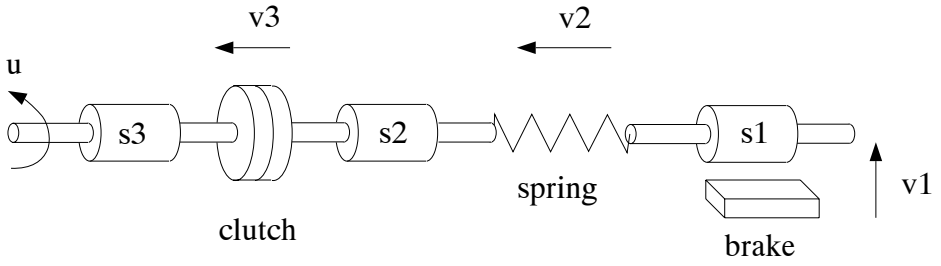


Bild 5.4: Strukturvariabler Antriebsstrang mit Kupplung und Bremse.

Welle $s2$, und die Elastizität im Antriebsstrang wird durch die Drehfeder *spring*, modelliert. Die Drehfeder treibt die Last $s1$ an. Diese kann über eine Reibbremse *brake* abgebremst werden. Durch die Rutschkupplung sind $s3$ und $s2$ im Normalfall starr gekoppelt. Wenn eine Lastüberhöhung eintritt, "rutscht" die Kupplung, so daß sich $s3$ relativ zu $s2$ bewegt. Im Normalfall bewegt sich die Last $s1$ relativ zur Umgebung. Wenn gebremst wird, so wird ein eingeprägtes Moment auf $s1$ aufgebracht, bis sich die Last in Ruhe befindet. Solange das dann wirkende Zwangsmoment kleiner als das maximale statische Haftreibmoment ist, bleibt $s1$ fest mit der Umgebung verbunden.

Dieses Beispiel ist schon hinreichend kompliziert, so daß sowohl eine Modellierung mit Hand,

als auch eine Modellierung mit einer allgemeinen Simulationssprache wie ACSL, recht aufwendig wird. Mit einem üblichen Mehrkörperprogramm kann das Beispiel nicht behandelt werden, da solche Programme in der Regel die Modellierung von Schaltvorgängen nicht unterstützen. Mit den vorher erläuterten Klassen für Antriebsstränge kann dieses System aber in einfacher Weise modelliert werden:

model *DriveClutch*
  submodel (*DriveBase*) b
  submodel (*Rotor*) $s3$ ($J = 1$)
  submodel (*Shaft*) $s1$ ($J = 1$), $s2$ ($J = 0.05$)
  submodel (*DriveExt*) u
  submodel (*ExtSpring*) *spring* ($c = 160,\ d = 1$)
  submodel (*ExtFriction*) *clutch* ($Rmax = Rmaxf$)
  submodel (*ExtBrake*) *brake* ($Rmax = Rmaxb$)
  submodel (*DriveVarLS*) $v1,\ v3$
  submodel (*DriveVarS*) $v2$

  parameter $Rmaxb = 80,\ Rmaxf = 80,\ Tbrake = 0.5$
  output $w1,\ w2,\ w3,\ Rc,\ Rb$

  connect b **to** $v1$ **to** $s1$ **to** $v2$ **to** $s2$ **to** $v3$ **to** $s3$, u **at** $(b, s3)$,
    spring **at** $v2$:*ext*, *clutch* **at** $v3$:*extL*, *brake* **at** $v1$:*extL*

  $brake.Brake = Time > tbrake$
  $u.f = \textbf{if}\ brake.Brake\ \textbf{then}\ 0\ \textbf{else}\ 200 * \sin(100 * Time)$
  $brake.Rv = Rmaxb$
  $clutch.Rv = Rmaxf$

  $w1 = s1.va$
  $w2 = s2.va$
  $w3 = s3.va$
  $Rc = v3.tb$
  $Rb = v1.tb$
end

Mit den **submodel**-Anweisungen werden alle benötigten Objekte deklariert. Die Rutschkupplung kann mit dem allgemeinen Reibelement der Klasse *ExtFriction* von Kapitel 4.5 auf Seite 120 beschrieben werden. Für die Bremse wird die Klasse *ExtBrake* benutzt, die weiter unten erläutert wird. Aufgrund der beiden blockierbaren Komponenten müssen die Relativgrößen $v1$ und $v3$ bei der Bremse und bei der Kupplung als (blockierbare) Zustandsvariablen verwendet werden. Weiterhin werden die Relativgrößen der Drehfeder auch als Zustandsvariablen benutzt. Um die Ergebnisse einer Simulation mit den Ergebnissen von [Mitc90] vergleichen zu können, werden als Ausgangsvariablen die drei absoluten Winkelgeschwindigkeiten von Welle 1, Welle 2 und Rotor ($w1, w2, w3$), sowie die Reibmomente in der Kupplung und in der Bremse (Rc, Rb), benutzt. Mit der **connect** Anweisung werden alle Objekte zusammengeschaltet. Man beachte, daß die Drehfeder, die Kupplung und die Bremse als externe Kraftgesetze definiert sind und deswegen über den Cut *ext* bzw. *extL* an bestehende Objekte angeschlossen werden. Die Anweisung "$brake.Brake = Time > tbrake$" gibt an, daß nach *tbrake* Sekunden gebremst wird. Solange nicht gebremst wird, wird die in [Mitc90] benutzte Funktion für das Antriebsdrehmoment u benutzt. Wenn gebremst wird, wird das Antriebsdrehmoment abgeschaltet. Schließlich werden mit den restlichen Gleichungen die Gleitreibmomente von Kupplung und Bremse, sowie die Ausgangsvariablen, berechnet.

Eine Reibbremse wird auch durch ein Reibmodell beschrieben. Im Gegensatz zur "reinen"

Reibung, ist dieses Reibmodell jedoch nur aktiv, solange gebremst wird. Diese Aktion wird durch die Boole'sche Variable *Brake* angezeigt. Man kann deswegen das Reibmodell *ExtFriction* von Seite 120 verwenden, muß die Reibung jedoch aufgrund der Variable *Brake* an- bzw. abschalten können. Die folgende Klasse *ExtBrake* wurde durch geringfügige Modifikationen aus der Klasse *ExtFriction* gewonnen:

```
model class (ExtForceL) ExtBrake
    parameter Rmax = 0      {maximale statische Haftreibkraft}
    terminal  Rv            {(positive) Gleitreibkraft}
    terminal  R             {Reibkraft}
    terminal  Brake         {true/false: bremsen/nicht bremsen}

    local Forward  = false, StartForward  = false,
          Backward = false, StartBackward = false, Start = true

    fL = if Forward  or StartForward  then  Rv else
         if Backward or StartBackward then -Rv else 0
    R  = fL + fLc

    Locked                = Brake and not ( Forward  or StartForward  or
                                            Backward or StartBackward or Start )
    new(Start)            = not Brake

    new(Forward)          = Brake and qd > 0 and (Forward  or StartForward  or Start)
    new(Backward)         = Brake and qd < 0 and (Backward or StartBackward or Start)

    new(StartForward)     = Brake and (Locked and fLc > Rmax  or StartForward
                                       and not (qd > 0 or qd <= 0 and qdd < 0))
    new(StartBackward)    = Brake and (Locked and fLc < -Rmax or StartBackward
                                       and not (qd < 0 or qd >= 0 and qdd > 0))
end
```

Bei allen Schaltzuständen des Reibmodells wurde die Boole'sche Variable *Brake* mit aufgenommen, so daß bei $Brake = false$ die Schaltzustände des Reibmodells definiert auf $false$ gesetzt sind. In diesem Fall wirkt keine Gleitreibkraft ($fL = 0$) und kein Freiheitsgrad ist gesperrt ($Locked = false$), so daß das Reibmodell keine Wirkung nach außen hat. Durch die Anweisung "**new**$(Start) =$ **not** $Brake$" wird der $Start$zustand des Reibmodells beim Beginn des Bremsens korrekt gesetzt. Wenn das Bremsen z.B. beendet wird, ist $Brake = false$ und $Start = true$. Bei einem erneuten Bremsen beginnt $Start$ damit beim Wert $true$, schaltet aber nach der ersten Iteration aufgrund von "**new**$(Start) =$ **not**$Brake$" sofort nach $false$. Dies entspricht dem Schaltverlauf beim Reibmodell $ExtFriction$.

Für ein typisches Simulationsexperiment werden die Anfangsbedingungen von [Mitc90] benutzt, d.h. alle Zustandsgrößen sind Null, mit Ausnahme von: $v1.qd = 90\ [rad/sec]$, $v3.qd = 10\ [rad/sec]$. In [Mitc90] ist die Endzeit 0.5 Sekunden. Um die Ergebnisse vergleichen zu können, wird eine Endzeit von 1 Sekunde gewählt, wobei nach 0.5 Sekunden die Bremse einfällt. In Bild 5.5 sind ausgewählte Trajektorien der Simulation zu sehen.

Aus Skalierungsgründen sind in den beiden linken Bildern die Verläufe bis 0.5 Sekunden aufgetragen und in den beiden rechten Bildern über die gesamte Simulationszeit. In den oberen Bildern sind die absoluten Winkelgeschwindigkeiten der drei Wellen zu sehen. Zu Beginn der Simulation rutscht die Kupplung. Nach einer kurzen Zeitspanne wird die Relativgeschwindigkeit Null und der Rotor und Welle 2 sind zueinander fixiert. Im Bild sind

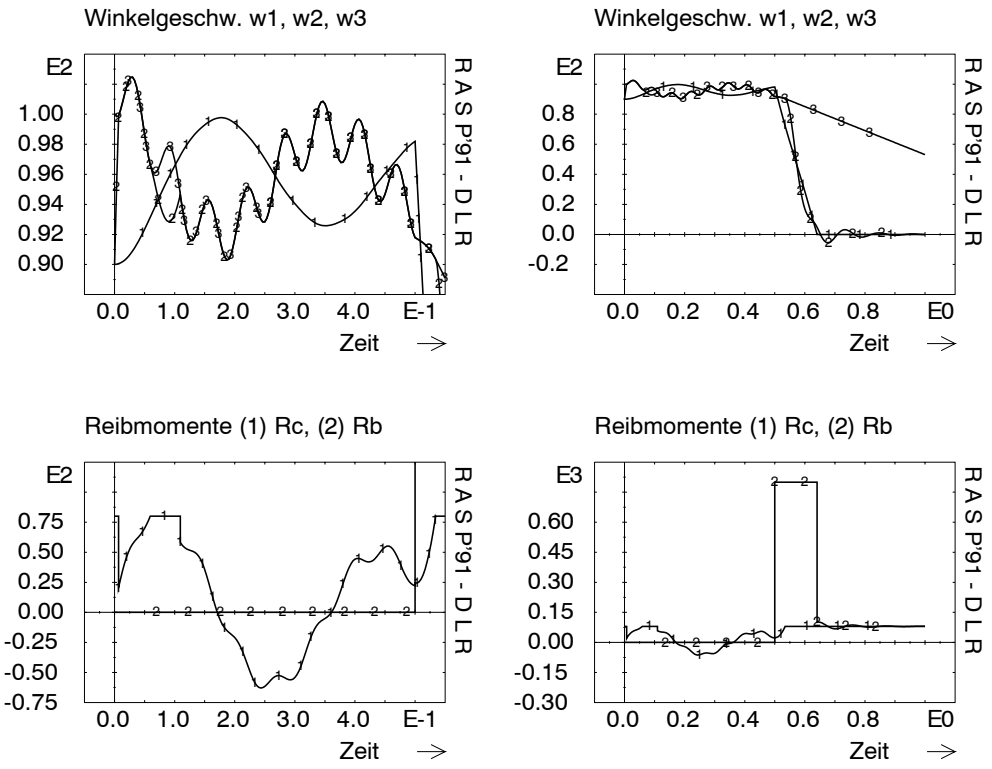


Bild 5.5: Simulationsergebnis des Antriebsstrangs mit Kupplung und Bremse.

beide Trajektorien deswegen die meiste Zeit über deckungsgleich. Nach 0.5 Sekunden wird gebremst. Dies führt zu einem steilen Abfall der Winkelgeschwindigkeiten von Welle 1 und Welle 2. In dieser Phase ist die Kupplung wieder im Rutschzustand.

5.5 Antriebsstränge in Mehrkörpersystemen

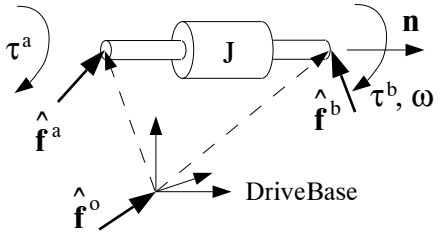
In diesem Abschnitt wird gezeigt, wie Antriebsstränge behandelt werden können, die sich auf einem 3-dimensional bewegten Körper befinden. Da ein Antriebsstrang aus Starrkörpern und Drehgelenken aufgebaut ist, könnte man diese Elemente mit den in Kapitel 4 erläuterten Methoden der Mehrkörperdynamik behandeln. Alle trägheitsbehafteten Elemente eines Antriebsstrangs sind jedoch rotationssymmetrisch und es ist schon seit langem bekannt, daß diese Eigenschaft die dynamischen Gleichungen vereinfacht, siehe z.B. [Witt77][1]. Weiterhin bestehen Antriebsstränge immer aus kleinen kinematischen Schleifen die speziell behandelt werden sollten, um einen effizienten Code zu erhalten.

Basisgleichungen

Es wird angenommen, daß ein Antriebsstrang auf einem Starrkörper gelagert ist, der sich räumlich bewegt. An einer beliebigen Stelle des Starrkörpers wird ein Koordinatensystem

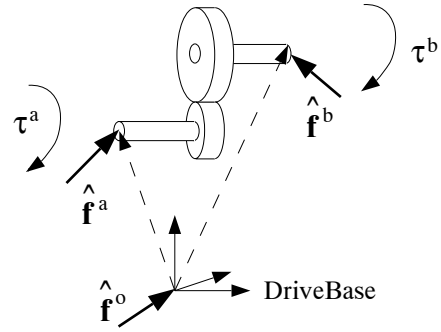
[1]Die allgemeine Berücksichtigung beliebiger Antriebsstränge auf einem bewegten Körper, sowie eine entsprechende Verallgemeinerung des $O(n)$ Verfahrens, sind neu.

angebracht. Dieses Koordinatensystem stellt die *Basis* des Antriebsstrangs dar, wie sie in den vorherigen Abschnitten mit der Klasse *DriveBase* beschrieben wurde. Alle Vektoren des Antriebsstrangs werden in dieser Basis dargestellt. Die einzelnen Komponenten eines Antriebsstrangs werden jetzt vollständig freigeschnitten, wie in Bild 5.6 für eine Welle (Klasse *Shaft*) und ein Getriebe (Klasse *Gear*) gezeigt. Eine Welle mit einem Flansch (Klasse *Rotor*) ist als Speziallfall enthalten. Im Bild ist ein Stirnradgetriebe gezeigt. Die angegebenen Beziehungen gelten jedoch für einen beliebigen Getriebetyp (z.B. Kegelradgetriebe).



$$\hat{\mathbf{f}}^o = \begin{bmatrix} \mathbf{n}J\alpha + \boldsymbol{\omega}^o \times \mathbf{n}J\omega \\ \mathbf{0} \end{bmatrix} - \mathbf{C}_a^T\hat{\mathbf{f}}^a - \mathbf{C}_b^T\hat{\mathbf{f}}^b \quad (5.3)$$

$$J\alpha = \tau^a + \tau^b - J(\mathbf{n}^T\boldsymbol{\alpha}^o) \quad (5.4)$$



$$\hat{\mathbf{f}}^o = -\mathbf{C}_a^T\hat{\mathbf{f}}^a - \mathbf{C}_b^T\hat{\mathbf{f}}^b \quad (5.5)$$

$$\omega^a = i\omega^b \quad (5.6)$$

$$0 = \tau^b + i\tau^a \quad (5.7)$$

Bild 5.6: Grundgleichungen von Welle und Getriebe

Sowohl bei der Welle als auch beim Getriebe sind 3 Schnittebenen vorhanden: Die beiden Flansche, mit denen die Komponente mit anderen Elementen verbunden werden kann, sowie die Lager, in denen die Komponente im Starrkörper gelagert ist. Aufgrund des Schnittprinzips werden an den 3 Schnittebenen resultierende Schnittkräfte und Schnittmomente angetragen. Hierbei ist $\hat{\mathbf{f}}^a$ das Schnittmoment und die Schnittkraft am Flansch a, $\hat{\mathbf{f}}^b$ ist das Schnittmoment und die Schnittkraft am Flansch b und $\hat{\mathbf{f}}^o$ ist das resultierende Schnittmoment und die resultierende Schnittkraft aller Lager der Komponente bezüglich der Antriebsstrangbasis.

Im Anhang A.5 ab Seite 191 ist die Ableitung der dynamischen Gleichungen für einen rotationssymmetrischen Starrkörper 1 angegeben, der in einem anderen Starrkörper 2 gelagert ist. Ausgehend von Impuls- und Drallsatz für beide Körper, können die Terme der dynamischen Gleichungen auf andere Weise zusammengefaßt werden. Hierzu werden beide Starrkörper zuerst als *ein gemeinsamer* Starrkörper aufgefaßt und die gemeinsame Masse, der gemeinsame Schwerpunkt und der gemeinsame Trägheitstensor gebildet. Dies ist möglich, da aufgrund der Rotationssymmetrie von Körper 1, der gemeinsame Schwerpunkt und der gemeinsame Trägheitstensor *nicht* von der aktuellen Winkellage von Körper 1 bezüglich Körper 2 abhängen. Im Anhang A.5 ab Seite 191 ist gezeigt, daß der Impuls- und Drallsatz, angeschrieben für beide Körper, äquivalent zum Impuls- und Drallsatz, angeschrieben für den als *ein* Körper betrachteten Vereinigungskörper, ist, mit einem Zusatzterm, der die relative Bewegung von Körper 1 bezüglich Körper 2 berücksichtigt.

Auf Grund dieser Eigenschaft werden bei der Modellierung eines Antriebsstrangs die Masse, der Schwerpunkt und der Trägheitstensor des Antriebsstrangs bei dem Basisstarrkörper berücksichtigt, in dem der Antriebsstrang gelagert ist. Das heißt, bei einer Vermessung wird ein Antriebsstrang zusammen mit dem entsprechenden Basisstarrkörper als *ein* Körper betrachtet. Die dann noch in den dynamischen Gleichungen zu berücksichtigenden Restterme sind in Bild 5.6 aufgeführt und im Anhang abgeleitet.

Gleichung (5.3) ist der Restterm des Impuls- und Drallsatzes für eine Welle bezüglich der Antriebsstrangbasis. Das vom Basisstarrkörper auf die Welle wirkende resultierende Lagermoment, bzw. die resultierende Lagerkraft, $\hat{\mathbf{f}}^o$ setzt sich zusammen aus den Schnittmomenten und Schnittkräften $(\hat{\mathbf{f}}^a, \hat{\mathbf{f}}^b)$ der beiden Flansche, sowie einem dynamischen Zusatzterm. Die Matrizen $\mathbf{C}_a, \mathbf{C}_b$ transformieren die Schnittmomente und Schnittkräfte der Flansche in die Basis und sind durch Gleichung (4.6b) auf Seite 75 definiert.

Der dynamische Zusatzterm setzt sich aus dem Relativdrall um die Drehachse $(\mathbf{n}J\alpha)$, sowie einem weiteren Term $(\boldsymbol{\omega}^o \times \mathbf{n}J\omega)$, zusammen. Hierbei ist $\mathbf{n}$ ein Einheitsvektor in Richtung der Drehachse, J ist das Trägheitsmoment der Welle um diese Achse, ω ist die (skalare) relative Winkelgeschwindigkeit bezüglich der Antriebsstrang-Basis und α ist die erste zeitliche Ableitung von ω. Schließlich ist $\boldsymbol{\omega}^o$ die absolute Winkelgeschwindigkeit und $\boldsymbol{\alpha}^o$ ist die absolute Winkelbeschleunigung der Basis bezüglich des Inertialsystems. Die Projektion der Schnittkräfte und Schnittmomente auf die freie Bewegungsrichtung nach Gleichung (4.12) von Seite 83 unter Berücksichtigung von (5.3), führt auf Gleichung (5.4). Hierbei sind τ^a, τ^b die Projektionen der Schnittmomente auf die Drehachse der Welle $(\tau^a = [\mathbf{n}^T\, \mathbf{0}^T]\, \hat{\mathbf{f}}^a)$. Gleichung (5.4) entspricht dem in den letzten Abschnitten benutzten 1-dimensionalen dynamischen Gesetz der Welle, wobei durch den Zusatzterm "$J(\mathbf{n}^T\boldsymbol{\alpha}^o)$" die Winkelbeschleunigung des Basis-Starrkörpers mitberücksichtigt wird. Der gesamte dynamische "Effekt", den eine Welle besitzt, wird durch (5.3, 5.4) vollständig beschrieben.

Die Gleichungen eines Getriebes ergeben sich direkt aus Bild 5.6 und sind in den Gleichungen (5.5, 5.6, 5.7) zusammengefaßt, da Getriebe als trägheitslos angenommen werden. Gleichung (5.5) ist eine Kräfte- und Momentenbilanz bezüglich der Antriebsstrangbasis. Gleichungen (5.6, 5.7) geben die bekannten Beziehungen zwischen den Antriebs- und Abtriebsdrehzahlen und den auf die Drechachsen projizierten Schnittmomenten (τ^a, τ^b) an.

Antriebsstränge auf bewegten Körpern

Der einfachste Fall eines 3-dimensional bewegten Antriebsstrangs liegt vor, wenn dieser auf einem Starrkörper befestigt ist und eine rotationssymmetrische Last antreibt, die Teil des Antriebsstrangs ist. Ein Antriebsstrang setzt sich aus Wellen, Getrieben und Kraftelementen zusammen. Wenn jedes starre Übertragungselement j nach Bild 5.6 freigeschnitten wird, und die Gleichungen (5.3, 5.5) für jedes Element angeschrieben werden, so ergibt sich das insgesamt vom Basiskörper auf die Lager des Antriebsstrangs wirkende Schnittmoment und die wirkende Schnittkraft $\hat{\mathbf{f}}^o_{ges}$ als Summe der Lagerreaktionen der Einzelkomponenten, d.h.

$$\hat{\mathbf{f}}^o_{ges} = \sum_j \hat{\mathbf{f}}^o_j \quad . \tag{5.8}$$

Auf Grund der Summenbildung kürzen sich alle Schnittkräfte und Schnittmomente $\hat{\mathbf{f}}^a_j$, $\hat{\mathbf{f}}^b_j$ der Flansche heraus, da z.B. $\hat{\mathbf{f}}^a_j$ mit positiven Vorzeichen an einem Element j wirkt und mit

negativem Vorzeichen an der anschließenden Komponente. Die Lagerreaktionen setzen sich demnach ausschließlich aus den dynamischen Zusatztermen der Gleichung (5.3) sowie eventueller eingeprägter Momente zusammen, die zwischen den Lagern und dem Antriebsstrang wirken. Die noch verbleibenden Gleichungen (5.4, 5.6, 5.7) legen die Beziehungen zwischen den relativen kinematischen Größen zwischen den Antriebsstrangkomponenten und dem Basisstarrkörper fest. Mit Ausnahme des Zusatzterms "$J(\mathbf{n}^T\boldsymbol{\alpha}^o)$", sind diese Gleichungen mit denjenigen eines raumfesten Antriebsstrangs identisch (siehe Bild 5.2 auf Seite 133).

Man beachte, daß aufgrund des "Herauskürzens" der Schnittkräfte und Schnittmomente im Antriebsstrang, diese auch nicht berechnet werden können. Das ist anschaulich darin begründet, daß jeder Antriebsstrang "vorspannbar" ist. Wenn z.B. ein Lager senkrecht zur Drehrichtung etwas versetzt eingebaut wird, so hat dies auf die Bewegung des Antriebsstrangs keinen Einfluß, wohl aber auf die Belastungen im Antriebsstrang. Diese können nur bei Berücksichtigung von Verformungen, nicht aber mit einem reinen Starrkörpermodell, ermittelt werden.

Es ist wünschenswert, daß ein Antriebsstrang immer auf dieselbe Weise definiert wird, gleichgültig ob er raumfest ist oder sich auf einem bewegten Körper befindet. Aus diesem Grund werden die bisherigen Antriebsstrang-Klassen nicht um einen zusätzlichen Cut erweitert, der die Lagerreaktionen überträgt. Stattdessen wird die akkumulierte Summe aller Lagerreaktionen über die beiden Cuts a und b einer Komponente weitergegeben und gegebenenfalls um den aktuellen dynamischen Term des Elements vergrößert. Die Klasse *DriveOneCut* zur Beschreibung des Flansches einer Komponente wird deswegen folgendermaßen modifiziert:

```
model class DriveOneCut
    main cut a ( ra, va, aa, v0a(3), a0a(3) / ta, t0a(3))
    terminal Pa

    Pa = fa * va
end
```

In Cut a werden die absolute Winkelgeschwindigkeit $v0a$ und die absolute Winkelbeschleunigung $a0a$ der Antriebsstrangbasis, sowie die akkumulierten Lagerreaktionen $t0a$ mit aufgenommen. Die Größen $v0a$ und $a0a$ werden in den Elementklassen nicht modifiziert, sondern nur von einem Element an das andere weitergereicht. *DriveTwoCut* wird entsprechend modifiziert. Die anderen Antriebsstrangklassen werden nun nur noch um die dynamischen Zusatzterme erweitert. Zum Beispiel hat die Klasse *Shaft* dann den folgenden Aufbau (vergleiche auch mit der ersten Definition in Bild 5.2 auf Seite 133):

```
model class (DriveTwoCut) Shaft
    parameter J = 0                          {Trägheitsmoment um Drehachse}
    parameter n1 = 0, n2 = 0, n3 = 0         {Drehachse bezüglich DriveBase}
    local    absn, n(3)

    absn = sqrt(n1*n1 + n2*n2 + n3*n3)
    n(1) = n1/absn
    n(2) = n2/absn
    n(3) = n3/absn

    ra      = rb
    va      = vb
    aa      = ab
    ta + tb = J*aa + J*n'*a0
    t0a + t0b = n*J*aa + skew(v0)*(n*J*va)
end
```

In der obigen Klasse sind die neuen Parameter $n1, n2, n3$ vorhanden, die die Koordinaten eines Vektors festlegen, der in Richtung der Drehachse der Welle zeigt. Die Größen $t0a$ und $t0b$ sind die akkumulierten Lagerreaktionen am Cut a und am Cut b der Welle. Wenn ein Antriebsstrang raumfest ist, wird, wie bisher auch, die Klasse *DriveBase* benutzt, um die Antriebsstrangbasis zu definieren. In dieser Klasse müssen nur die absolute Winkelgeschwindigkeit und Winkelbeschleunigung der Basis zu Null gesetzt werden, da sich diese nicht bewegt. Wenn sich ein Antriebsstrang auf einem bewegten Körper befindet, so wird stattdessen die spezielle Klasse *DriveMbsBase* zur Verfügung gestellt. Diese hat den folgenden Aufbau:

```
model class DriveMbsBase
            cut   a ( Ta(3,3), r0a(3), va(6), aa(6) / fa(6) )
            cut   b ( rb, vb, ab, v0b(3), a0b(3) / tb, t0b(3))
    main cut   mc [a,b]
    main path mp < a - b >

    rb   = 0
    vb   = 0
    ab   = 0
    v0b  = va(1:3)
    a0b  = aa(1:3)
    0    = fa(1:3) + t0b
    0    = fa(4:6)
end
```

Da ein Objekt dieser Klasse mit dem Cut a an einem Cut eines Mehrkörpersystemobjekts und am Cut b einer Antriebsstrangkomponente befestigt wird, entspricht Cut a dem Cut eines MKS und Cut b entspricht dem Cut eines Antriebsstrangs. In der Klasse wird die absolute Winkelgeschwindigkeit und Winkelbeschleunigung des MKS-Cuts an den Antriebsstrang weitergegeben und als Winkelgeschwindigkeit und Winkelbeschleunigung der Antriebsstrangbasis angesehen. Weiterhin stehen die (akkumulierten) Lagerreaktionen $t0b$ des Antriebsstrangs mit den Schnittmomenten $fa(1:3)$ des MKS-Cuts im Gleichgewicht.

Die Modellierung eines Antriebsstrangs auf einem bewegten Körper ist jetzt denkbar einfach: Zuerst wird der Antriebsstrang wie im 1-dimensionalen Fall beschrieben. Nur die Richtungen der Drehachsen müssen zusätzlich als Parameter bei den trägheitsbehafteten Elementen mit

angegeben werden. Danach muß nur noch das Basisobjekt der Klasse *DriveBase* durch ein Objekt der Klasse *DriveMbsBase* ersetzt werden und Cut a des *DriveMbsBase* Objekts an der gewünschten Stelle im Mehrkörpermodell angebracht werden!

Es kommt häufig vor, daß ein Antriebsstrang auf einem bewegten Starrkörper 1 gelagert ist und einen Starrkörper 2, der mit einem Gelenk am Starrkörper 1 befestigt ist, antreibt. Dies ist z.B. bei Robotern der Fall. Aus Vereinfachungsgründen wird (ohne Beschränkung der Allgemeinheit) angenommen, daß der Cut a des Gelenks die Basis des Antriebsstrangs darstellt und der Antriebsstrang einen (Relativ-) Freiheitsgrad des Gelenks antreibt. In der entsprechenden Gelenkklasse werden die beiden zusätzlichen Cuts *DriveBase* und *DriveVar* (bzw. *DriveVarL* für blockierbare Gelenke) eingeführt. Der Cut *DriveBase* entspricht dem Cut b der Klasse *DriveMbsBase*, d.h. hier wird der Antriebsstrang gelagert. Genauso wie bei der Klasse *DriveMbsBase* werden die Lagerreaktionen $t0b$ zu den Schnittmomenten des Gelenk-Cuts a hinzuaddiert. Der Cut *DriveVar* entspricht dem Cut b eines Objekts der Klasse *DriveVar* mit der Zustandsgrößen definiert werden, wobei der Cut a dieser Klasse mit der Antriebsstrangbasis verbunden ist. Eine Komponente des Antriebsstrangs muß immer mit dem Cut *DriveVar* verbunden werden. Damit werden die das Gelenk beschreibenden Relativgrößen immer als Zustandsgrößen ausgewählt. Aufgrund von Gleichung (4.12) auf Seite 83 muß die Projektion des Schnittmoments und der Schnittkraft am Cut b des Gelenks auf die freie Bewegungsrichtung des Gelenks gleich der dort angreifenden Kraft, bzw. des dort angreifenden Moments, $\boldsymbol{\lambda}^e$ sein. Im Falle eines Antriebsstrangs ist dies gerade das in der Drehrichung wirkende Schnittmoment τ. Dieses über den Cut *DriveVar* übertragene Schnittmoment muß im Gelenk entsprechend auf $\boldsymbol{\lambda}^e$ geschaltet werden.

Die Modellierung eines Antriebsstrangs auf einem bewegten Körper, der einen anderen Körper antreibt, ist damit wiederum einfach möglich: Zuerst wird der Antriebsstrang wie im 1-dimensionalen Fall beschrieben. Hierbei muß der Winkel und die Winkelgeschwindigkeit zwischen Basis und "Last" mittels eines Objekts der Klasse *DriveVarS* (oder *DriveVarLS*) als Zustandsgröße ausgewählt werden. Danach werden die "Last", das Objekt der Klasse *DriveVarS* und die Basis der Klasse *DriveBase* entfernt und der restliche Teil des Antriebsstrangs wird am Cut *DriveVar* des gewünschten Gelenks angebracht!

O(n)-Verfahren für Antriebsstränge auf bewegten Körpern

Antriebsstränge sollten auch bei Mehrkörpersystemen eingesetzt werden können, bei denen das in Kapitel 4.4 ab Seite 111 erläuterte O(n) Verfahren erwünscht ist. Hierzu müssen die entsprechenden O(n) Gleichungen von Antriebssträngen (siehe Kapitel 5.3 ab Seite 139) um die noch fehlenden Terme erweitert werden, die von dem bewegten Körper herrühren, auf dem der Antriebsstrang gelagert ist.

Bei solchen Antriebssträngen gibt es zweierlei Arten von Schnittmomenten: Das in Drehrichtung einer Welle wirkende Schnittmoment τ, sowie die akkumulierten Lagerreaktionen $\boldsymbol{\tau}^o$ an dieser Stelle. Es kann gezeigt werden, daß beide Schnittmomente immer als lineare Funktionen der hier auftretenden (relativen) Winkelbeschleunigung der Welle α, sowie der absoluten Winkelbeschleunigung des Basiskörpers $\boldsymbol{\alpha}^o$ angegeben werden können:

$$\tau = I\,\alpha + b + \mathbf{n}^{o\,T}\boldsymbol{\alpha}^o \tag{5.9a}$$

$$\boldsymbol{\tau}^o = \mathbf{I}^o\,\boldsymbol{\alpha}^o + \mathbf{b}^o + \mathbf{n}^o\,\alpha \quad . \tag{5.9b}$$

Im folgenden wird dies bewiesen. Anstatt die Schnittgrößen $\tau, \boldsymbol{\tau}^o$ bei den jeweiligen Cuts zu übergeben, werden stattdessen die linearen Faktoren $I, b, \mathbf{I}^o, \mathbf{b}^o, \mathbf{n}^o$ zur Charakterisierung der Schnittmomente benutzt und in den Cuts verwendet.

Zuerst wird wiederum der einfachere Fall betrachtet, daß der Antriebsstrang auf einem bewegten Starrkörper angebracht ist und eine rotationssymmetrische Last antreibt, die zum Antriebsstrang gerechnet wird. Die dynamischen Gleichungen für die starren Übertragungselemente von Antriebssträngen sind in Tabelle 5.1 zusammengestellt. Sie basieren auf den Gleichungen (5.3–5.7) von Seite 145. Die gesuchten linearen Faktoren für die Übertragungs-

Rotor	Welle	Getriebe
$\tau^a = J\alpha^a + J(\mathbf{n}^T\boldsymbol{\alpha}^o)$	$\tau^a + \tau^b = J\alpha^a + J(\mathbf{n}^T\boldsymbol{\alpha}^o)$	$\tau^b + i\,\tau^a = 0$
$\boldsymbol{\tau}_a^o = \mathbf{n}J\alpha^a + \boldsymbol{\omega}^o \times \mathbf{n}J\omega^a$	$\boldsymbol{\tau}_a^o + \boldsymbol{\tau}_b^o = \mathbf{n}J\alpha^a + \boldsymbol{\omega}^o \times \mathbf{n}J\omega^a$	$\boldsymbol{\tau}_a^o + \boldsymbol{\tau}_b^o = 0$

Tabelle 5.1: Dynamische Gleichungen von Übertragungsgliedern von Antriebssträngen.

elemente können aus einem Koeffizientenvergleich von Tabelle 5.1 und den Gleichungen (5.9) gewonnen werden. Das Ergebnis ist in Tabelle 5.2 zusammengestellt. Man sieht, daß

Rotor	Welle	Getriebe
$I^a = J$	$I^a + I^b = J$	$I^b + i\,i\,I^a = 0$
$b^a = 0$	$b^a + b^b = 0$	$b^b + i\,b^a = 0$
$\mathbf{I}_a^o = \mathbf{0}$	$\mathbf{I}_a^o + \mathbf{I}_b^o = \mathbf{0}$	$\mathbf{I}_a^o + \mathbf{I}_b^o = \mathbf{0}$
$\mathbf{b}_a^o = \boldsymbol{\omega}^o \times \mathbf{n}J\omega^a$	$\mathbf{b}_a^o + \mathbf{b}_b^o = \boldsymbol{\omega}^o \times \mathbf{n}J\omega^a$	$\mathbf{b}_a^o + \mathbf{b}_b^o = \mathbf{0}$
$\mathbf{n}_a^o = J\mathbf{n}$	$\mathbf{n}_a^o + \mathbf{n}_b^o = J\mathbf{n}$	$\mathbf{n}_b^o + i\,\mathbf{n}_a^o = \mathbf{0}$

Tabelle 5.2: Lineare Faktoren bei Übertragungsgliedern von Antriebssträngen.

die linearen Faktoren bei Rotoren immer aus bekannten Größen berechnet werden können. Bei Wellen und Getrieben können die linearen Faktoren an einem Cut berechnet werden, wenn die linearen Faktoren am anderen Cut bekannt sind. Die Behandlung von Kraftelementen ist einfach, da Kraftgesetze nicht von Beschleunigungen abhängen. Das heißt, die linearen Faktoren bei Kraftelementen sind alle Null, mit Ausnahme von b, das gleich dem eingeprägten Moment f ist.

Die Rekursionsvorschrift für einen Antriebsstrang ergibt sich jetzt ähnlich wie in Kapitel 5.3 ab Seite 139 für den $O(n)$ Algorithmus bei raumfesten Antriebssträngen. Im ungünstigsten Fall hat eine Antriebsstrang-Einheit zwei Enden, für die die linearen Faktoren an den beiden äußersten Cuts aus bekannten Größen berechnet werden können, da die Elemente an den Enden entweder Kraftelemente oder Rotoren sind. Liegen dort Rotoren vor, so werden diese entfernt. Dann sind die beiden neuen Enden des Antriebsstrangs auf jeden Fall entweder Wellen oder Getriebe, für die die linearen Faktoren der äußersten Cuts bekannt sind. Mit den Gleichungen der Tabelle 5.2 können dann die linearen Faktoren der beiden

inneren Cuts berechnet werden. Wenn die beiden jetzt betrachteten Elemente, Wellen oder Getriebe, entfernt werden, liegt wieder der vorher betrachtete Fall vor und es wird eine Rekursion aufgebaut, die an derjenigen Komponente endet, an der das Zustandsvariablen-Objekt befestigt ist.

Beim Zustandsvariablen-Objekt muß wieder eine Festlegung getroffen werden, welcher Cut des Objekts an der gerade betrachteten Einheit befestigt ist. Hier wird angenommen, daß dies immer der Cut b ist. Auf Grund der Rekursion werden die beiden Schnittmomente $\tau^b, \boldsymbol{\tau}_b^o$ an diesem Cut durch *bekannte* lineare Faktoren beschrieben. Andererseits ist das Schnittmoment "$\tau^a = -\tau^b = f$", wobei f ein bekanntes, eingeprägtes Drehmoment ist, das z.B. auch Null sein kann. Am Cut b sind damit in Gleichung (5.9a) alle Größen bekannt mit Ausnahme von α^b und $\boldsymbol{\alpha}^o$. Damit kann α^b als Funktion von $\boldsymbol{\alpha}^o$ ausgedrückt werden:

$$\alpha^b = -(f + b^b + \mathbf{n}_b^{o\,T} \boldsymbol{\alpha}^o)/I^b \quad . \tag{5.10}$$

Da am Cut a nur ein eingeprägtes Moment f wirkt, folgt wegen (5.9a)

$$I^a = 0, \quad b^a = f, \quad \mathbf{n}_a^o = 0 \quad . \tag{5.11}$$

Die akkumulierten Lagerreaktionen $\boldsymbol{\tau}^o$ werden nur vom Cut b an den Cut a weitergegeben, sodaß gilt: $\mathbf{0} = \boldsymbol{\tau}_a^o + \boldsymbol{\tau}_b^o$. Durch Einsetzen von (5.9b, 5.10) in diese Beziehung wird α^b eliminiert und es folgt

$$\mathbf{0} = \left(\mathbf{I}_a^o + \mathbf{I}_b^o - \frac{\mathbf{n}_b^o \mathbf{n}_b^{o\,T}}{I^b} \right) \boldsymbol{\alpha}^0 + \left(\mathbf{b}_a^o + \mathbf{b}_b^o - \mathbf{n}_b^o \frac{f + b^b}{I^b} \right) \quad .$$

Da diese Gleichung für beliebiges $\boldsymbol{\alpha}^o$ gelten muß, müssen die beiden Klammerausdrücke für sich genommen verschwinden und man erhält die folgende Beziehung:

$$\mathbf{I}_a^o + \mathbf{I}_b^o = \frac{\mathbf{n}_b^o \mathbf{n}_b^{o\,T}}{I^b} \tag{5.12a}$$

$$\mathbf{b}_a^o + \mathbf{b}_b^o = \mathbf{n}_b^o \frac{f + b^b}{I^b} \quad . \tag{5.12b}$$

Mit den Gleichungen (5.12, 5.11) sind alle linearen Faktoren am Cut a bekannt und die Rekursion kann fortgesetzt werden. Mit einer (unten näher erläuterten) Vorwärtsrekursion, werden die Winkelbeschleunigungen am Cut a, d.h. $\alpha^a, \boldsymbol{\alpha}^o$, berechnet. Mit Gleichung (5.10) kann dann die Winkelbeschleunigung α^b am Cut b bestimmt werden und die gesuchte Ableitung der Zustandsgröße $qdd = \ddot{q}$, ergibt sich schließlich als Differenz der beiden Winkelbeschleunigungen: $\ddot{q} = \alpha^b - \alpha^a$.

Auch der Fall blockierbarer Komponenten kann behandelt werden. Wenn ein Zustandsvariablen-Objekt der Klasse *DriveVarL* im blockierten Zustand ist ($Locked = True$), dann werden alle linearen Faktoren von Cut b einfach an Cut a weitergegeben, da die beiden Cuts fest miteinander verbunden sind. Eine Zusammenfassung der Beziehungen im freigegebenen und blockierten Zustand ergibt:

$$I^a = \textbf{if } Locked \textbf{ then } -I^b \textbf{ else } 0 \tag{5.13a}$$

$$b^a = \textbf{if } Locked \textbf{ then } -b^b \textbf{ else } f + f_L \tag{5.13b}$$

$$\mathbf{n}_a^o = \textbf{if } Locked \textbf{ then } -\mathbf{n}_b^o \textbf{ else } \mathbf{0} \tag{5.13c}$$

$$\mathbf{I}_a^o + \mathbf{I}_b^o = \textbf{if } Locked \textbf{ then } \mathbf{0} \textbf{ else } \mathbf{n}_b^o \mathbf{n}_b^{o\,T} / I^b \tag{5.13d}$$

$$\mathbf{b}_a^o + \mathbf{b}_b^o = \textbf{if } Locked \textbf{ then } \mathbf{0} \textbf{ else } (f + b^b)/I^b \tag{5.13e}$$

$$\ddot{q} = \textbf{if } Locked \textbf{ then } 0 \textbf{ else } -aux/I^b \tag{5.13f}$$

$$f_{Lc} = \textbf{if } Locked \textbf{ then } -aux \textbf{ else } 0 \tag{5.13g}$$

$$aux = f + f_L + b^b + \mathbf{n}_b^{o\,T} \boldsymbol{\alpha}^o + I^b \alpha^a \tag{5.13h}$$

$$\alpha^b = \alpha^a + \ddot{q} \quad . \tag{5.13i}$$

Die Rekursion *im Antriebsstrang* endet, wenn die Basis des Antriebsstrangs, d.h. ein Objekt der Klasse *DriveMbsBase*, erreicht ist. In diesem Objekt müssen die linearen Faktoren des Antriebsstrangs auf die linearen Faktoren **I, b** des Mehrkörpersystems abgebildet werden. Diese werden mit der Definitionsgleichung (4.36) auf Seite 112, d.h.

$$\begin{bmatrix} \boldsymbol{\tau}^{base} \\ \mathbf{f}^{base} \end{bmatrix} = \mathbf{I} \begin{bmatrix} \boldsymbol{\alpha}^o \\ \mathbf{a}^o \end{bmatrix} + \mathbf{b} \quad , \tag{5.14}$$

bestimmt. An der Basis eines Antriebsstrangs ist $\alpha = 0$ und "$\boldsymbol{\tau}^{base} + \boldsymbol{\tau}^o = 0$". Mit (5.14, 5.9b) ergeben sich deswegen die linearen Faktoren des Mehrkörpersystems zu:

$$\mathbf{I} = -\begin{bmatrix} \mathbf{I}^o & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{5.15a}$$

$$\mathbf{b} = -\begin{bmatrix} \mathbf{b}^o \\ \mathbf{0} \end{bmatrix} \quad . \tag{5.15b}$$

Ab diesem Zeitpunkt wird die "normale" Rekursion *im Mehrkörpersystem* fortgesetzt. Bei der letzten Vorwärtsrekursion des O(n) Algorithmus werden alle kinematischen Größen im Mehrkörpersystem, insbesondere $\boldsymbol{\alpha}^o$, berechnet. Da $\boldsymbol{\alpha}^o$ jetzt bekannt ist, kann danach mit (5.13h, 5.13f, 5.13i) eine *Vorwärtsrekursion im Antriebsstrang* ablaufen, mit der alle Relativbeschleunigungen im Antriebsstrang berechnet werden.

Trotz der etwas länglichen Herleitung ist die Implementierung des obigen Algorithmus einfach: In die Klassen des raumfesten Antriebsstrangs müssen nur die Rekursionsvorschriften der Tabelle 5.2 und die Gleichungen (5.13) aufgenommen werden, und die neue Klasse *DriveMbsBase* muß hinzugefügt werden. Aufgrund der obigen Überlegungen ist dann sichergestellt, daß ein Mehrkörpersystem (in Baumstruktur), mit einem oder mit mehreren Antriebssträngen, immer auf eine explizit lösbare Blockdreiecksstruktur transformiert werden kann.

Auch die Anwendersicht ist wiederum einfach: Zuerst wird der Antriebsstrang wie im 1-dimensionalen Fall beschrieben. Nur die Richtungen der Drehachsen bei den trägheitsbehafteten Elementen müssen zusätzlich als Parameter mit angegeben werden. Danach muß nur noch das Basisobjekt der Klasse *DriveBase* durch ein Objekt der Klasse *DriveMbsBase* ersetzt werden und es muß der Cut a des *DriveMbsBase* Objekts an der gewünschten Stelle im Mehrkörpermodell angebracht werden!

Wenn ein Antriebsstrang auf einem bewegten Starrkörper 1 gelagert ist und einen Starrkörper 2 antreibt, der mit einem Gelenk am Starrkörper 1 befestigt ist, sind der Antriebsstrang, das Gelenk und die beiden Starrkörper enger gekoppelt, sodaß die Rekursion *im Mehrkörpersystem* etwas modifiziert werden muß. Da ein Antriebsstrang vorhanden ist, ändern sich die beiden Ausgangsgleichungen (4.5h) auf Seite 75 und (4.15a) auf Seite 85

zu:

$$\mathbf{0} = \hat{\mathbf{f}}^a + \mathbf{C}^T \hat{\mathbf{f}}^b + \begin{bmatrix} \boldsymbol{\tau}_t^o \\ \mathbf{0} \end{bmatrix} \tag{5.16a}$$

$$0 = \lambda^e + L\lambda^l + \boldsymbol{\Phi}^T \hat{\mathbf{f}}^b + \tau_t \quad . \tag{5.16b}$$

Hierbei ist $\hat{\mathbf{f}}^a$ das am Cut a und $\hat{\mathbf{f}}^b$ das am Cut b des Gelenks wirkende Schnittmoment bzw. die Schnittkraft. $\boldsymbol{\tau}_t^o$ ist die Lagerreaktion des Antriebsstrangs auf die Basis des Antriebsstrangs, d.h. auf den Cut a des Gelenks, und τ_t ist das Schnittmoment des Antriebsstrangs an der Achse des Gelenks. Der untere Index t soll im folgenden die Antriebsstranggrößen kennzeichnen. Da der Antriebsstrang nur einen "Freiheitsgrad" des Gelenks antreibt, werden hier nur Gelenke mit einem Freiheitsgrad betrachtet. Durch die Größe λ^e werden noch zusätzlich im Gelenk wirkende verallgemeinerte Kräfte berücksichtigt. Der Freiheitsgrad des Gelenks kann optional mittels der Matrix L "blockiert" werden. Es wird angenommen, daß die Schnittmomente des Antriebsstrangs durch ihre linearen Faktoren beschrieben werden:

$$\tau_t = I_t \ddot{q} + b_t + \mathbf{n}_t^{o\,T} \boldsymbol{\alpha}^a \tag{5.17a}$$

$$\boldsymbol{\tau}_t = \mathbf{I}_t^o \, \boldsymbol{\alpha}^a + \mathbf{b}_t^o + \mathbf{n}_t^o \ddot{q} \quad . \tag{5.17b}$$

Hierbei wurde berücksichtigt, daß die Winkelbeschleunigung der Antriebsstrangbasis gleich der Winkelbeschleunigung des Cuts a des Gelenks ($\boldsymbol{\alpha}^o = \boldsymbol{\alpha}^a$) ist, und daß die relative Winkelbeschleunigung des Antriebsstrangs an der Achse des Gelenks gleich der zweiten Ableitung der verallgemeinerten Gelenkkoordinate ($\alpha = \ddot{q}$) ist.

Die Rekursionsvorschrift für ein Mehrkörpersystem muß aufgrund der modifizierten Ausgangsgleichungen noch einmal hergeleitet werden. Die Ableitung läuft vollkommen analog zu derjenigen auf Seite 112, wobei die obigen Zusatzterme mit zu berücksichtigen sind. Zuerst werden wiederum alle den Cut b eines Gelenks beschreibenden Gleichungen in der folgenden linearen, symmetrischen Matrix-Gleichung zusammengefaßt:

$$\begin{bmatrix} \mathbf{I}^b & -\mathbf{E} & \mathbf{0} & \mathbf{0} \\ -\mathbf{E} & \mathbf{0} & \boldsymbol{\Phi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Phi}^T & I_t & L \\ \mathbf{0} & \mathbf{0} & L & 1-L \end{bmatrix} \begin{bmatrix} \hat{\mathbf{a}}^b \\ \hat{\mathbf{f}}^b \\ \ddot{q} \\ \lambda^l \end{bmatrix} = \begin{bmatrix} -\mathbf{b}^b \\ -\mathbf{C}\hat{\mathbf{a}}^a - \boldsymbol{\eta} \\ -\lambda^e - b_t - \mathbf{n}_t^{o\,T} \boldsymbol{\alpha}^a \\ 0 \end{bmatrix} \quad . \tag{5.18}$$

Gleichung (5.18) gibt an, daß die unbekannten Variablen des Cut-Frames b, d.h. $\hat{\mathbf{a}}^b$, $\hat{\mathbf{f}}^b$, $\ddot{q}, \lambda^l$, als lineare Funktionen der unbekannten Beschleunigung und Winkelbeschleunigung $\hat{\mathbf{a}}^a$ des Cut-Frames a angegeben werden können. Wird (5.18) explizit nach den unbekannten Variablen des Cut-Frames b aufgelöst, ergibt sich:

$$\lambda^l = -L\,h \tag{5.19a}$$

$$\ddot{q} = -(1-L)h/M \tag{5.19b}$$

$$\hat{\mathbf{a}}^b = \mathbf{C}\hat{\mathbf{a}}^a + \boldsymbol{\Phi}\ddot{q} + \boldsymbol{\eta} \tag{5.19c}$$

$$\hat{\mathbf{f}}^b = \mathbf{I}^b \hat{\mathbf{a}}^b + \mathbf{b}^b \tag{5.19d}$$

mit

$$h = \left(\boldsymbol{\Phi}^T \mathbf{I}^b \mathbf{C} + \begin{bmatrix} \mathbf{n}_t^o \\ \mathbf{0} \end{bmatrix}^T \right) \hat{\mathbf{a}}^a + \left(\boldsymbol{\Phi}^T \mathbf{I}^b \boldsymbol{\eta} + \boldsymbol{\lambda}^e + \boldsymbol{\Phi}^T \mathbf{b}^b + b_t \right) \tag{5.20a}$$

$$M = \boldsymbol{\Phi}^T \mathbf{I}^b \boldsymbol{\Phi} + I_t \tag{5.20b}$$

$$L = \textbf{if } \textit{Locked} \textbf{ then } 1 \textbf{ else } 0 \quad . \tag{5.20c}$$

Werden diese Gleichungen in (5.16a) eingesetzt, ergibt sich, unter Berücksichtigung von (5.17) und einer Zwischenrechnung, die analog zur vorherigen Herleitung läuft:

$$\hat{\mathbf{f}}^a = \mathbf{I}^a \hat{\mathbf{a}}^a + \mathbf{b}^a \;\; ; \qquad \mathbf{I}^a = \mathbf{I}^{a^T} \tag{5.21}$$

mit

$$\mathbf{I}^a = - \begin{bmatrix} \mathbf{I}_t^o & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} - \mathbf{C}^T \mathbf{N} \mathbf{C} \tag{5.22a}$$

$$\mathbf{b}^a = - \begin{bmatrix} \mathbf{b}_t^o \\ \mathbf{0} \end{bmatrix} - \mathbf{C}^T \left(\mathbf{b}^b + \mathbf{N} \boldsymbol{\eta} - \boldsymbol{\gamma} (1 - L)(\lambda^e + \boldsymbol{\Phi}^T \mathbf{b}^b + b_t)/M \right) \tag{5.22b}$$

$$\mathbf{N} = \mathbf{I}^b - \boldsymbol{\gamma}\, \boldsymbol{\gamma}^T (1 - L)/M \tag{5.22c}$$

$$\mathbf{M} = \boldsymbol{\Phi}^T \mathbf{I}^b \boldsymbol{\Phi} + I_t \tag{5.22d}$$

$$\boldsymbol{\gamma} = \mathbf{I}^b \boldsymbol{\Phi} + \begin{bmatrix} {}^b\mathbf{T}^a \, \mathbf{n}_t^o \\ \mathbf{0} \end{bmatrix} \tag{5.22e}$$

$$L = \textbf{if } \textit{Locked} \textbf{ then } 1 \textbf{ else } 0 \quad . \tag{5.22f}$$

Ein Vergleich mit (4.41) auf Seite 113 zeigt, daß die Rekursionsvorschriften praktisch gleich geblieben sind und nur an einigen Stellen die Zusatzterme des Antriebsstrangs mit auftreten. Der Einfluß des Antriebsstrangs drückt sich vor allem im "reduzierten Trägheitsmoment" I_t aus, welches nur in Gleichung (5.22d) eingeht. Hier wird das "reduzierte Trägheitsmoment" I_t vom Antriebsstrang auf das projizierte "reduzierte Trägheitsmoment" $\mathbf{I}^b$ der "außen" liegenden Körper aufaddiert.

5.6 Diskussion

In diesem Kapitel wurde gezeigt, wie Antriebsstränge recht einfach modelliert werden können, selbst wenn sich ein Antriebsstrang auf einem bewegten Körper befindet oder wenn strukturvariable Elemente wie Lagerreibung, Rutschkupplungen, Bremsen vorhanden sind. Ein Anwender muß nur angegeben, welche Elemente eines Antriebsstrangs benutzt werden sollen und wie diese zusammengeschaltet sind. Durch einfaches Vertauschen des die Basis charakterisierenden Objekts, kann ein kompletter Antriebsstrang auf einem beliebig bewegten Körper plaziert werden. Hierbei werden alle auftretenden dynamischen Effekte berücksichtigt. Dies hat zur Folge, daß die Rekursionsformel des O(n) Algorithmus bei Mehrkörpersystemen geringfügig modifiziert werden muß. Die Spezialisierung des O(n) Algorithmus auf 1-dimensionale mechanische Systeme führt zu den in der Maschinendynamik bekannten "reduzierten Trägheitsmomenten".

Das mit Dymola zur Verfügung stehende Modellierungswerkzeug erleichtert die Implementierung von Antriebsstrang-Modellen erheblich. Die einfachen Gleichungen der einzelnen

Komponenten müssen nur in einer Klassenbeschreibung zur Verfügung gestellt werden. Der automatische Sortieralgorithmus, der die Modellgleichungen auf Blockdreiecksform transformiert, übernimmt den Hauptteil der Arbeit. Auch beim $O(n)$-Verfahren muß man sich nur einmal anschaulich klar machen, daß auf eine explizit lösbare Blockdreiecksform transformiert werden kann.

In diesem Kapitel wurden nur die wesentlichen Basiskomponenten von Antriebssträngen behandelt. In [Lasc88] werden detailliert noch eine große Anzahl praktisch wichtiger, zusätzlicher Modellkomponenten besprochen, z.B. Elektromotoren, Pumpen, Kompressoren, elastische Kupplungen, hydrodynamische Getriebe, Differentialgetriebe. Das Einbringen solcher Modellklassen ist einfach, da dies vor allem über spezielle Kraftgesetze erfolgen kann. Verzweigungsgetriebe, wie Differentialgetriebe, haben im Gegensatz zu den hier behandelten Getriebetypen mehr als 2 Flansche. Auch für Verzweigungsgetriebe können Rekursionsformeln angegeben werden. Dies hat keinen Einfluß auf die bestehenden Klassen.

Kapitel 6

Objektorientierte Modellierung regelungstechnischer Systeme

Mechatronische Systeme enthalten praktisch immer auch regelungstechnische Komponenten. Diese werden normalerweise durch eine Verschaltung von Ein/Ausgangsblöcken beschrieben und grafisch als Blockschaltbilder dargestellt. Regelungstechnische Simulationsumgebungen, wie SIMULINK, basieren auf dieser Art der Modellierung. In diesem Kapitel wird gezeigt, wie Ein/Ausgangsblöcke mit der hier verwendeten objektorientierten Technik modelliert werden können. Damit können physikalische Komponenten, in denen der Energiefluß eine Rolle spielt, leicht mit regelungstechnischen Wirkkomponenten verschaltet werden, in denen der Informationsfluß wichtig ist.

Regelungssysteme werden üblicherweise direkt als Verschaltung von rückwirkungsfreien Wirkkomponenten entworfen, so daß eine Modellierung als Blockschaltbild die "natürlichste" Form ist. Die Behandlung von Blockschaltbildern mit Ein/Ausgangsblöcken bereitet keine Schwierigkeiten, da die hierfür benötigten Algorithmen, z.B. das Sortieren von Blöcken, als einfacher Spezialfall in den in Kapitel 2 ab Seite 17 besprochenen Verfahren enthalten sind. Durch die (funktionell) mächtigeren Werkzeuge können auch spezielle Probleme gelöst werden, die bei allgemeinen Simulationsumgebungen oft zu Schwierigkeiten führen. Zum Beispiel werden lineare algebraische Schleifen in Blockschaltbildern hier automatisch entweder symbolisch, oder numerisch mit einem Gaußverfahren, gelöst. Nichtlineare algebraische Schleifen können entweder numerisch mit einem Newton/Raphson-Verfahren iterativ behandelt werden oder das gesamte Blockschaltbild wird direkt in ein differential-algebraisches Gleichungssystem (DAE) überführt und mit einem DAE-Integrator gelöst.

6.1 Einführungsbeispiel

Anhand der Modellierung eines einfachen regelungstechnischen Systems soll zuerst die prinzipielle Modellierungssicht dargestellt werden. Im nächsten Abschnitt werden dann die verwendeten Klassen im Detail besprochen.

In Bild 6.1 ist ein einfacher Regelkreis aus [Mitc91] zu sehen. Dieser besteht aus einer Strecke 2-ter Ordnung, einem Meßfilter erster Ordnung im Rückführkreis, sowie aus einem Lead-Lag Regler. Die einzelnen Komponenten werden durch Übertragungsfunktionen beschrieben.

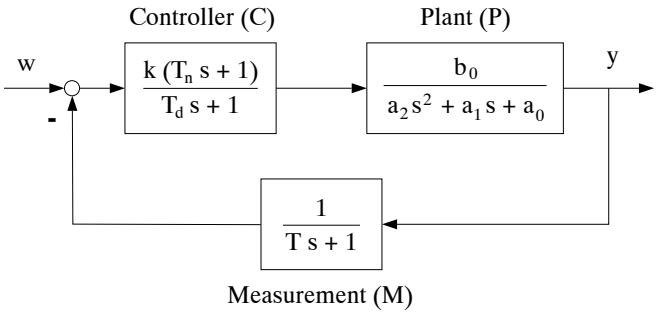


Bild 6.1: Einfacher Regelkreis.

Hierbei ist "s" die komplexe Variable der Laplace-Transformation. Dieses System kann folgendermaßen modelliert werden:

```
model ControlLoop
   submodel (LeadLag) C    (k = 100, Tn = 0.02, Td = 0.005)
   submodel (Coeff2)  P    (b0 = 0.5, a2 = 0.012, a1 = 0.2, a0 = 1)
   submodel (PT1)     M    (T = 0.002)
   submodel (Sum2)    Sum  (k2 = -1)

   input  w
   output y

   connect Sum to C to P to M to Sum:in2

   w = Sum.u1
   y = P.y
end
```

Mit den **submodel** Anweisungen werden die benötigten Blöcke definiert. Hierbei beschreibt die Klasse *LeadLag* ein System mit einem Pol und einer Nullstelle, *Coeff2* ein System mit einem Zähler- und einem Nennerpolynom zweiter Ordnung und *PT1* ein Verzögerungsglied erster Ordnung. Der Summierer ist ebenfalls ein eigenes Objekt, wobei die Klasse *Sum2* einen Summierer mit 2 Eingängen definiert. Alle Eingänge eines Summierers werden mit je einem Verstärkungsfaktor multipliziert, dessen Voreinstellung 1.0 ist. Durch die Parameterdefinition $k2 = -1$ wird die Verstärkung vom Eingang 2 auf -1 gesetzt, um das negative Vorzeichen beim Summierer zu berücksichtigen. Die Objekte werden entsprechend zum Blockschaltbild verschaltet. Schließlich werden mit den beiden letzten Gleichungen die Zuordnungen der Eingangs- und Ausgangsgröße festgelegt.

In Dymola können in einer Verschaltung nur Cuts, aber keine Variablen, referenziert werden. Zum Beispiel kennzeichnet $Sum{:}in2$ den Cut $in2$, d.h. den zweiten Eingang des Summierers. Die Variable dieses Cuts ist als $u2$ deklariert. Entsprechendes gilt für den ersten Eingang. Andererseits können in Gleichungen nur Variablen angesprochen werden. Aus diesem Grund wird durch die Gleichung $w = Sum.u1$ die erste Eingangsvariable des Summierers charakterisiert. Das obige Modell kann von Dymola problemlos in eine Zustandsform überführt werden.

6.2 Aufgabeninvariante Grundelemente

In diesem Abschnitt werden die Modellklassen besprochen, die zur Behandlung von Blockschaltbildern erstellt wurden. Diese entsprechen den üblichen Standardblöcken, wie sie in der Regel in Blockschaltbild-Editoren zur Verfügung stehen. Im Unterschied zu allgemeinen Simulationsumgebungen sind diese Blöcke aber nicht "hart codiert" in einem Programmsystem, sondern stehen in einer Textdatei und können einfach *von jedem Anwender* um neue Blocktypen erweitert werden.

Blöcke haben die Eigenschaft, daß Signale eindeutig als Eingangs- oder als Ausgangsgrößen definiert sind. Es scheint deswegen sinnvoll zu sein, die Interfacevariablen eines entsprechenden Objekts mit dem Attribut **input** oder **output** zu versehen, um nur diese Kausalität zu erlauben. Auch bei Blockdiagrammen gibt es jedoch unterschiedliche Aufgabenstellungen. Zum Beispiel könnten bei einer Stationärwertberechnung die *Ausgänge* bestimmter Blöcke vorgegeben werden, wobei die hierzu gehörenden *Eingänge* gesucht sind, siehe z.B. Kapitel 4.3.6 ab Seite 107. Um solche speziellen Aufgabenstellungen auch behandeln zu können, wird den Terminal-Variablen von Objekten keine Richtung zugeordnet. Dies hat auf die Erstellung eines Zustandsraummodells keinen Einfluß, da bei der Transformation auf Blockdreiecksform die Verschaltungsrichtungen automatisch ermittelt werden.

Lineare kontinuierliche Blöcke

Alle zur Verfügung gestellten linearen, kontinuierlichen Blöcke besitzen einen Eingang und einen Ausgang. Diese gemeinsame Eigenschaft wird in der Oberklasse *SISO* (= *S*ingle *I*nput *S*ingle *O*utput) einmal definiert:

```
model class SISO
            cut   in  (u)
            cut   out (y)
      main cut    mc  [a,b]
      main path   mp  < a - b >
end
```

Die Eingangsvariable wird mit u und die Ausgangsvariable wird mit y gekennzeichnet. Auf Grund der **main path** Deklaration sind die im obigen Beispiel gezeigten **connect** Anweisungen mit "**to**" möglich. Die **main cut** Deklaration erlaubt Anweisungen der Form "**connect** *block* **at** $(n1, n2)$" um einen Block *block* zwischen dem Ausgang $n1$ und dem Eingang $n2$ zweier anderer Blöcke zu plazieren.

Lineare, kontinuierliche Systeme werden in Form von Übertragungsfunktionen definiert. Hierbei werden drei Darstellungsformen unterstützt: technisch häufig auftretende Formen[1], sowie Koeffizienten oder Wurzeln der Zähler- und Nennerpolynome einer allgemeinen Übertragungsfunktion.

Die Klasse je eines Vertreters dieser Darstellungsformen ist in Tabelle 6.1 zu sehen. Dymola unterstützt zur Zeit noch keine Matrizen. Es ist deswegen nicht möglich z.B. alle Übertragungsfunktionen in Koeffizientendarstellung mit nur *einer* Klasse zu beschreiben (hierbei

[1] Die folgenden Blöcke werden zur Verfügung gestellt: P, PT1, PT2, DT1, I, I2, PI, PID, Lead-Lag.

Name	Funktion	Klasse
PID	$y = k\,(1 + \frac{1}{T_i s} + T_d s)\frac{1}{\frac{T_d}{N}s + 1}\,u$	**model class** (*SISO*) *PID* **parameter** $k = 1,\ N = 10,\ Ti,\ Td$ **local** $x1 = 0,\ x2 = 0$ **der**$(x1) = x2$ **der**$(x2) = (u/Ti - x2) * N/Td$ $y = k * (x1 + (1 - N) * Ti * x2 + N * u)$ **end**
Coeff2	$y = \frac{b_2 s^2 + b_1 s + b_0}{a_2 s^2 + a_1 s + a_0}\,u$	**model class** (*SISO*) *Coeff2* **parameter** $a2 = 1,\ a1 = 0,\ a0 = 0$ $b2 = 0,\ b1 = 0,\ b0 = 0$ **local** $x1 = 0,\ x2 = 0$ **der**$(x1) = x2$ **der**$(x2) = (u - (a0 * x1 + a1 * x2))/a2$ $y = (b0 - a0 * b2/a2) * x1 +$ $(b1 - a1 * b2/a2) * x2 + (b2/a2) * u$ **end**
PoleZero	$y = k\,\frac{s - z}{s - p}\,u$	**model class** (*SISO*) *PoleZero* **parameter** $k = 1,\ z,\ p$ **local** $x = 0$ **der**$(x) = p * x + u$ $y = k * (\ (p - z) * x + u\ )$ **end**

Tabelle 6.1: Klassen zur Beschreibung von linearen, kontinuierlichen Blöcken.

würden die Zähler- und Nennerkoeffizienten mit je einem Vektor definiert werden). Aus diesem Grund werden für unterschiedliche Ordnungen der Übertragungsfunktionen je eine Klasse zur Verfügung gestellt. Zum Beispiel gibt es die 5 Klassen *Coeff1, Coeff2, ..., Coeff5* um Übertragungsfunktionen bis zur Ordnung 5 in der Koeffizientendarstellung angeben zu können. In den Klassen werden die Übertragungsfunktionen durch entsprechende Zustandsraummodelle beschrieben. Hierzu wird eine Koeffizientendarstellung der Übertragungsfunktionen verwendet, die als Regelungsnormalform realisiert ist.

Lineare zeitdiskrete Blöcke

Hier werden digitale Übertragungsglieder betrachtet, bei denen das Eingangssignal zu periodischen Abtastzeitpunkten ermittelt wird und das Ausgangssignal ohne Verzögerung berechnet und über ein Halteglied 0-ter Ordnung ausgegeben wird. Die zur Verfügung gestellten Blöcke enthalten jeweils einen idealen Abtaster (= Analog/Digital Wandler), sowie ein ideales Halteglied (= Digital/Analog Wandler). Werden diskrete Blöcke, die dieselbe Abtastzeit besitzen, zusammengeschaltet, so verhalten sich die beiden zusammengeschalteten Blöcke genauso, wie wenn ein A/D-Wandler, zwei rein diskrete Blöcke und ein Halteglied verschaltet würden. Der Grund liegt darin, daß die zu den Abtastzeitpunkten auftretenden Ereignisse der Blöcke immer zum selben Zeitpunkt eintreten, weil die Ausdrücke zur Berechnung des nächsten Ereignisses in den Blöcken identisch sind. Dann werden die Modellgleichungen der Blöcke aber auch immer an denselben Zeitpunkten ausgewertet.

Da alle zur Verfügung gestellten diskreten Blöcke einen idealen A/D- und D/A-Wandler besitzen, wird in der Oberklasse *Discrete* diese gemeinsame Eigenschaft einmal definiert:

```
model class Discrete
    parameter Tsample                    {Abtastzeit}
    local     Tnext = 0, Sample

    Sample = Time >= Tnext
    when Sample then
        new(Tnext) = Tnext + Tsample
    endwhen
end
```

Mit dem Parameter $Tsample$ wird die Abtastzeit für periodische Abtastung definiert. $Tnext$ gibt den Zeitpunkt der nächsten Abtastung an. Wenn die logische Variable *Sample* wahr *wird*, tritt ein Ereignis ein und mit Hilfe der in Kapitel 3.2 ab Seite 51 erläuterten **when**-Anweisung wird der nächste Abtastzeitpunkt festgelegt. Im diskreten Block muß jetzt nur noch auf die Variable *Sample* abgefragt werden. Wenn diese Variable wahr wird, sind die Gleichungen des Blocks auszuwerten. Da der Wert einer Variablen, die in einer **when**-Anweisung gesetzt wird, ihren Wert beibehält bis die Variable explizit wieder einen neuen Wert erhält, ist das Halteglied 0-ter Ordnung automatisch enthalten. Man beachte, daß die Gleichung zur Ermittlung von *Sample* nur die Zeit und eine Konstante enthält, so daß bei der Codeerzeugung ein *Zeitereignis* und kein Zustandsereignis generiert wird.

Die meisten diskreten Blöcke haben einen Eingang und einen Ausgang, so daß diese gemeinsame Eigenschaft, ähnlich wie bei den kontinuierlichen Blöcken, in der Oberklasse *dSISO* (= *d*iscrete SISO) definiert wird, die von *Discrete* durch Vererbung abgeleitet wird:

```
model class (Discrete) dSISO
            cut   in  (u)
            cut   out (y)
      main cut    mc [a,b]
      main path   mp < a - b >
end
```

Die meisten linearen, zeitdiskreten Blöcke werden als z-Übertragungsfunktion definiert, wobei in der Klassenbeschreibung zur Realisierung die diskrete Zustandsdarstellung in der Regelungsnormalform verwendet wird. Zum Beispiel wird die Funktion "$y = (b_2z^2 + b_1z + b_0)/(a_2z^2 + a_1z + a_0)\,u$" mit der folgenden Klasse beschrieben:

```
model class (dSISO) dCoeff2
    parameter a2 = 1, a1 = 0, a0 = 0, b2 = 0, b1 = 0, b0 = 0
    local     x1 = 0, x2 = 0

    when Sample then
        new(x1) = x2
        new(x2) = (u - (a0 * x1 + a1 * x2))/a2
        y       = (b0 - a0 * b2/a2) * x1 + (b1 - a1 * b2/a2) * x2 + (b2/a2) * u
    endwhen
end
```

Die Gleichungen innerhalb der **when** Anweisung werden nur an Abtastzeitpunkten ausgeführt. Hierbei ist z.B. **new**($x1$) der Wert, den $x1$ nach dem Ereignis, also beim Neustart der Integration, besitzen wird.

Begrenzungselemente

Es gibt einige einfache Blöcke um ein Signal zu begrenzen. Diese Blöcke sind in der hier benutzten Modellierungsumgebung fast nicht notwendig, da mit der **if**-Anweisung solche Begrenzungen gleich direkt angeschrieben werden können. Wenn ein Signal u vorgegebene Begrenzungen u_{min}, u_{max} nicht überschreiten darf, kann das einfach durch

$$y = \textbf{if } u > umax \textbf{ then } umax \textbf{ else if } u < umin \textbf{ then } umin \textbf{ else } u$$

definiert werden. Im Gegensatz zu fast allen allgemeinen, blockorientierten Simulationsumgebungen, wie z.B. SIMULINK, wird die obige Begrenzungsfunktion aber numerisch korrekt durch Zustandsereignisse behandelt (siehe auch Kapitel 3.1 ab Seite 45). Da PID-Regler mit begrenztem Ausgang und Antiwindup-Kompensation (siehe [Astr90]) häufig eingesetzt werden, wird dieses Begrenzungselement ebenfalls (als kontinuierlicher und diskreter Block) zur Verfügung gestellt. Zum Beispiel hat die Klassenbeschreibung des kontinuierlichen Blocks den folgenden Aufbau:

```
model class (SISO) PIDbound
    parameter k = 1, N = 10, Tw = 0, Ti, Td, ymax
    local       x1 = 0, x2 = 0, ydes, w

    der(x1) = x2
    der(x2) = (u/Ti - x2) * N/Td + Tw * (y - ydes)
    ydes    = k * (x1 + (1 - N) * Ti * x2 + N * u)
    y       = if ydes > ymax then ymax else
              if ydes < ymin then ymin else ydes
end
```

Der Ausgang y dieses Reglers ist begrenzt. Wenn sich der Regler in der Begrenzung befindet, wird die Antiwindup-Kompensation "$y - y_{des}$" über die Verzögerungszeit Tw aktiv und sorgt dafür, daß der Regler in diesem Grenzzustand ein stabiles System ist, dessen Zustände nach Null gehen.

Verbindungselemente

Summierer und Multiplizierer von Signalen werden als eigene Klassen zur Verfügung gestellt. Da keine Matrizen unterstützt werden, gibt es z.B. 3 Summierer mit je 2, 3 und 4 Eingängen. Um bei einem negativen Eingang eines Summierers nicht ein zusätzliches Verstärkungsobjekt einführen zu müssen, werden an allen Eingängen gleich Verstärkungen vorgesehen, deren Voreinstellung 1.0 ist. Damit hat z.B. ein Summierer mit zwei Eingängen die folgende Klassenbeschreibung:

```
model class Sum2
    parameter k1 = 1, k2 = 1
        cut in1 (u1)
        cut in2 (u2)
    main cut out (y)

    y = k1 * u1 + k2 * u2
end
```

Signalgeneratoren

Häufig werden spezielle periodische Signale zur Ansteuerung von Blöcken benötigt. Um die Erstellung geeigneter Klassen zu vereinfachen, wird eine Oberklasse *PeriodicSignal* eingeführt, in der die gemeinsamen Eigenschaften der periodischen Signale definiert sind. Hierzu wird angenommen, daß in den Unterklassen nur die Signalfunktionen für *eine* Periode als Funktion der Zeit t_p ($0 \leq t_p \leq$ Periodendauer) angegeben werden sollen, und daß die Ereignissteuerung von einer Periode zur nächsten in der Oberklasse durchzuführen ist. Die Klasse *PeriodicSignal* hat den folgenden Aufbau:

```
model class PeriodicSignal
    main cut out (y)          {Signalausgang}
    parameter T0 = 0          {Signal Startzeit}
    parameter f               {Signalfrequenz}

    local yp, tp, T, Tbegin = 0, Start = true

    T = 1/f
    when Time >= Tbegin + T or Start then
        new(Start)  = false
        new(Tbegin) = if not Start then Tbegin + T else
                      T0 + max(0, T * int((Time - T0)/T))
    endwhen

    tp = Time - Tbegin
    y  = if Time < T0 then 0 else yp
end
```

Beim Beginn einer Simulation wird mit dem **else** Zweig von "**new**$(Tbegin)$ ..." der Beginn der ersten Simulationsperiode des Signals ermittelt. Nach einer Signalperiode tritt ein Ereignis auf, bei dem der Beginn der nächsten Periode wieder entsprechend gesetzt wird. Da der Beginn $Tbegin$ der aktuellen Periode immer bekannt ist, kann die *in* einer Periode laufende Zeit t_p leicht berechnet werden. Mit Hilfe der obigen Klasse ist es jetzt einfach möglich, spezielle periodische Signale zu definieren. Hierzu muß nur y_p als Funktion von t_p angegeben werden. In Bild 6.2 sind 3 typische Signalklassen aufgeführt.

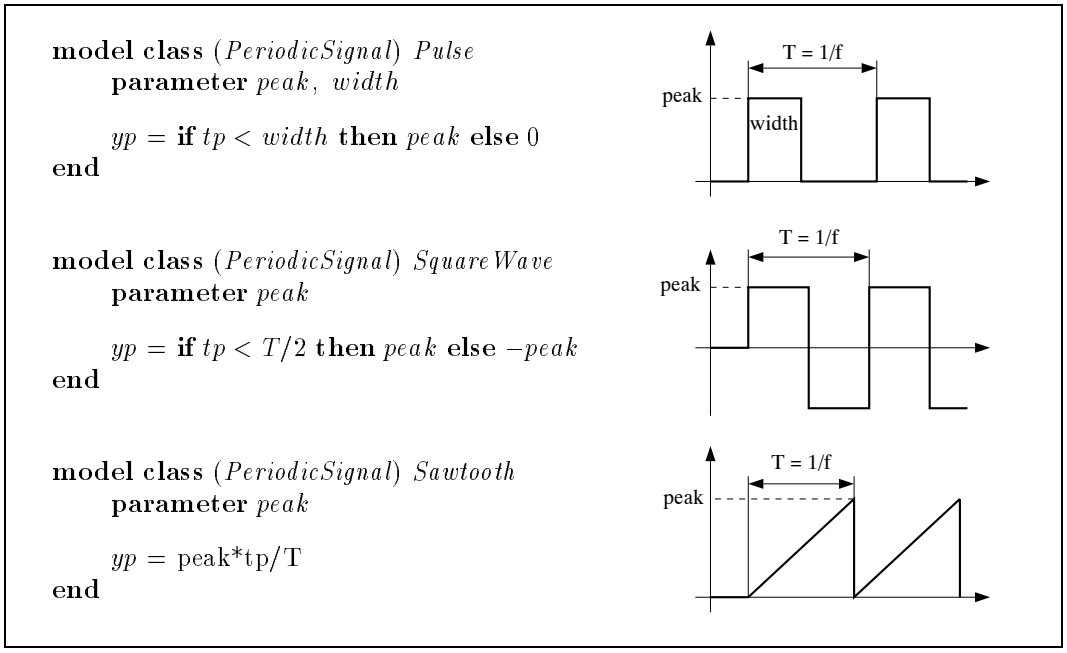


Bild 6.2: Klassen für periodische Signale.

6.3 Diskussion

Es wurde gezeigt, wie regelungstechnische Systeme, die in einer Ein/Ausgangsbeschreibung vorliegen, mit einer objektorientierten Modellierungssprache beschrieben werden können. Die zur Verfügung gestellten Blocktypen zeigen, daß es einfach ist, neue Blöcke einzuführen oder vorhandene zu erweitern. Mit der in Kapitel 3 ab Seite 45 besprochenen Modellierungstechnik für ereignisabhängige Systeme bereitet es auch keine Probleme, getaktete bzw. diskrete Blöcke so zu modellieren, daß diese numerisch robust und effizient durch einen Integrator bearbeitet werden können.

Die Modellierung von Blöcken ist zur Zeit etwas eingeschränkt, da in Dymola keine Matrizen unterstützt werden. Wenn ein Regler z.B. direkt durch eine Zustandsdarstellung beschrieben ist, so muß dieser als explizites Differentialgleichungssystem eingegeben werden. Ansonsten ist diese objektorientierte Modellierungsumgebung funktionell mächtiger als die meisten Simulationsumgebungen, da z.B. *automatisch* lineare und nichtlineare algebraische Schleifen behandelt werden.

Kapitel 7

Gesamtmodell des Industrieroboters Manutec r3

Die erläuterten Konzepte und Modellklassen werden in diesem Kapitel auf ein komplexes mechatronisches System angewandt, um die Leistungsfähigkeit der objektorientierten Modellierung zu demonstrieren. Untersucht wird der Industrieroboter Manutec r3, siehe Bild 7.1, der bei der DLR detailliert vermessen wurde. Hierbei wurden auch ausgewählte Bahnen des Roboters verfahren, so daß entsprechende Simulationen mit Messungen verglichen werden können.

Alle wichtigen physikalischen Effekte werden modelliert: die Mehrkörperdynamik, die Antriebsstränge inklusive Reibung, Elastizität, Dämpfung und Lose, die Dynamik der Motoren inklusive Stromregler, die Tachofilter und die Kaskadenregler. Hierfür werden die Bibliotheken für Mehrkörpersysteme, für Antriebsstränge, für elektrische Schaltkreise, für regelungstechnische Komponenten und für spezielle Kraftgesetze (Reibung) verwendet. Obwohl alle Modellierungsdetails angegeben werden, bleibt das Modell überschaubar und verständlich.

7.1 Ermittlung der Modellparameter durch Einzelmessungen

Die Parameter des Roboters wurden größtenteils von S. Türk durch Einzelmessungen am Roboter bestimmt, der dazu zum Teil zerlegt wurde.

Zur Ermittlung der mechanischen Kenndaten wurden die 6 Arme des Roboters auseinandergebaut. Die Massen wurden durch Wiegen bestimmt, die Schwerpunkte wurden durch Vermessung der Lotlinien der an einem Punkt aufgehängten Arme bestimmt, und die Trägheitstensoren wurden durch Schwingungsmessungen mit Torsionsdrähten ermittelt. Die Meßverfahren, die Fehlerrechnung inklusive Ausgleichsrechnung, und die ermittelten Modellparameter sind in [Eich86] dokumentiert. Das Ergebnis dieser Messungen wurde in [Tuer87, Otte88a] veröffentlicht.

Die Ermittlung der übrigen Modellparameter fand am zusammengebauten Roboter statt, mit dem ausgewählte Bahnen gefahren wurden. Hierbei wurde immer nur ein Gelenk bewegt, wobei die Bremsen der übrigen Gelenke eingeschaltet waren. Die analogen elektrischen

Bild 7.1: Bild des Industrieroboters Manutec r3.

Komponenten (Motor, Stromregler, Geschwindigkeitsregler) wurden durch Frequenzgangsmessungen bestimmt. Bei dem zur Verfügung stehenden Exemplar der Baureihe r3 wurden die Meßsignale durch die Analogauswertung verzerrt, so daß keine genügend genaue Proportionalität zur Drehzahl der Gelenke vorlag. Aus diesem Grunde hat Türk den vorhandenen Tiefpass durch ein Tachofilter dritter Ordnung ersetzt. Der Antriebsstrang (insbesondere Elastizität und Dämpfung) wurde ebenfalls durch Frequenzgangsmessungen identifiziert. Die Verstärkungsfaktoren der Lageregler konnten nicht gemessen werden, da diese digital realisiert sind. Sie wurden von Detzner [Detz86] aufgrund einfacher Entwurfsüberlegungen geschätzt. Die verwendeten Meßverfahren und die ermittelten Modellparameter sind in [Tuer90] beschrieben. Die kompletten Modelldaten sind noch einmal kompakt und übersichtlich in [Fran93] zusammengestellt.

Um die Modellvorstellungen zu verifizieren, wurden die Messungen mit begleitenden Simulationen verglichen. Hierbei handelte es sich um Modelle einer Achse des Roboters, sowie um ein Modell mit 3 Freiheitsgraden (die letzten 3 Gelenke des Roboters sind hierbei gesperrt) inklusive Antriebsstrang, Motor und Regler. Das benötigte Mehrkörpermodell wurde mit dem speziellen symbolischen MKS-Programm MyRobot [Otte88] generiert. Die restlichen Modellteile wurden mit der allgemeinen Simulationssprache ACSL [Mitc91] modelliert und

damals zusammen mit dem MKS-Teil innerhalb der ACSL Laufzeitumgebung simuliert.

Um mit dem Robotermodell nicht nur Simulationen, sondern auch Trajektorienoptimierungen (für die Bahnplanung) und Reglerentwürfe, durchführen zu können, wurde ein komplettes Modell des Roboters von Franke [Fran92] neu als Fortran-Unterprogramm im DSblock-Format [Otte92] realisiert. Der Mehrkörperteil wurde wiederum mit dem MKS-Programm MyRobot erzeugt. Die übrigen Modellteile wurden von Hand codiert. Dadurch war es möglich, das Modell innerhalb der ANDECS-Umgebung [Grue93] einzusetzen und die dort vorhandenen vielfältigen Analyse- und Entwurfsmethoden anzuwenden.

Bei dieser Realisierung des Modells zeigte es sich ganz deutlich, daß mit dem Gesamtmodell des Roboters r3 die Grenzen einer Modellerstellung, die direkt auf Fortran basiert, erreicht sind. Das implementierte Unterprogramm ist, insbesondere durch die Modellierung der vielen unstetigen Elemente mit Zustandsereignissen, unübersichtlich und wohl nur noch vom Programmautor wartbar und an spezifische Bedürfnisse anpaßbar. In etwas geringerem Umfang gilt diese Aussage auch für das zuerst erstellte Teilmodell in der Simulationssprache ACSL.

Bei der Erstellung dieser beiden Simulationsmodelle war noch nicht klar, wie die durch die Coulomb'sche Reibung eingeführten strukturvariablen Elemente zu modellieren sind, und insbesondere wie gleichzeitig auftretende Ereignisse, bzw. identisch verschwindende Indikatorfunktionen behandelt werden können. Aus diesem Grund arbeiten die beiden Programme nicht in allen Situationen korrekt. Es gibt zum Beispiel Schwierigkeiten, wenn zwei Reibelemente zum gleichen Zeitpunkt schalten.

7.2 Gesamtmodell als oberste Modell-Hierarchie

Mit den in den letzten Kapiteln erarbeiteten Modellklassen wird das Gesamtmodell des r3 Roboters hierarchisch aufgebaut. In Bild 7.2 ist die Struktur des Gesamtmodells zu sehen. Der Roboter wird als Mehrkörpersystem modelliert, das aus 6 starren Körpern und 6 einachsigen Drehgelenken besteht. Auf jedem Körper ist ein Antriebsstrang befestigt, der das Gelenk des nächsten Körpers antreibt. Der Block *"Antriebsstrang"* enthält Antriebswellen und Getriebe, wobei Elastizitäten, Dämpfung, Lose und Reibung mit modelliert werden. Jeder Antriebsstrang wird von einem Drehmoment angetrieben, das vom elektromagnetischen Feld des Motors erzeugt wird. Der Block *"Motor"* enthält den elektrischen Teil eines Elektromotors inklusive des Stromreglers. Der Sollwert eines Stromreglers wird von einer Kaskadenregelung im Block *"Regler"* vorgegeben. Sollwerte des Reglers sind die gewünschten Winkel und Winkelgeschwindigkeiten der Drehgelenke. Jeder Block in Bild 7.2 wird durch eine eigene Modellklasse beschrieben und in den nächsten Abschnitten besprochen. Die oberste Hierarchiestufe wird durch das folgende Modell beschrieben:

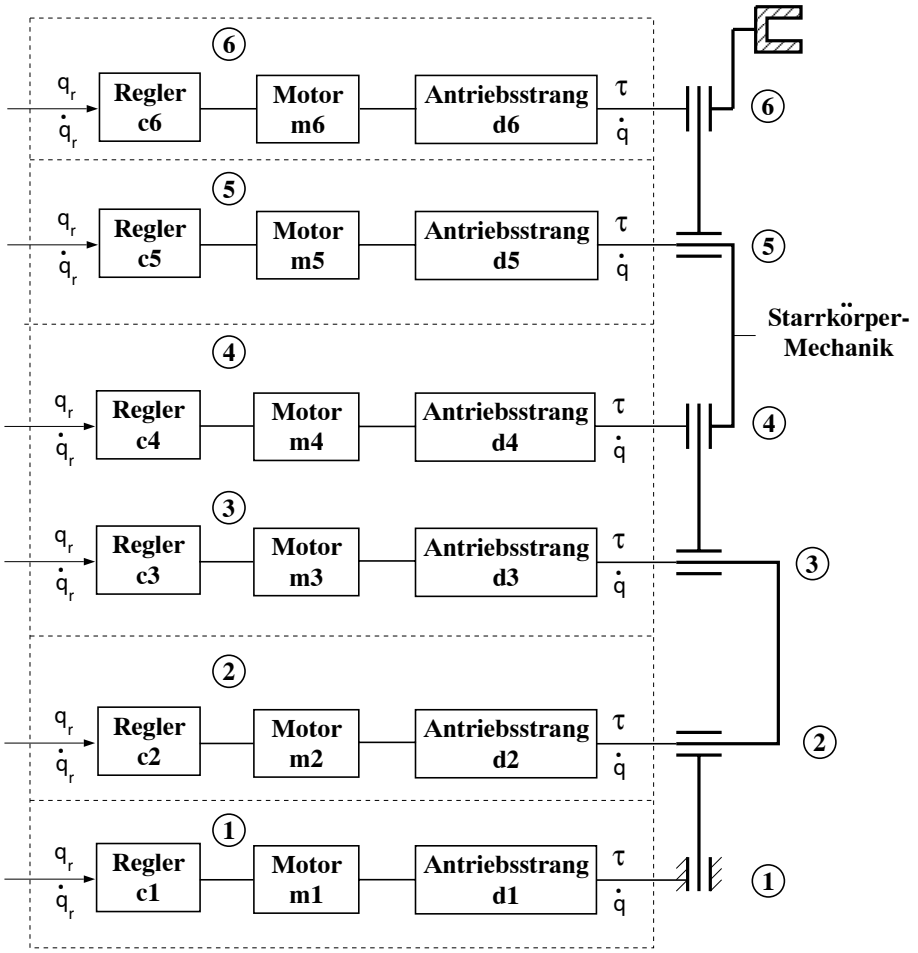


Bild 7.2: Struktur des Gesamtmodells des Roboters Manutec r3.

@*mbssim.lib* {Bibliothek für Mehrkörpersysteme (Simulationsproblem)}
@*block.lib* {Bibliothek für Blockdiagramme}
@*el.lib* {Bibliothek für elektrische Bauteile}
@*r3.lib* {Bibliothek für r3 Bauteile}

model *r3s6* {Gesamtmodell vom Roboter Manutec r3}
{*Regelungssysteme*}
submodel *(R3control) c1, c2, c3, c4, c5, c6*

{*elektrischer Teil der Motoren*}
submodel *(R3motor) m1 (k=1.1616, w=4590, D=0.6 , z=0.094 , max=9.0)*
submodel *(R3motor) m2 (k=1.1616, w=5500, D=0.6 , z=0.094 , max=9.0)*
submodel *(R3motor) m3 (k=1.1616, w=5500, D=0.6 , z=0.094 , max=9.0)*
submodel *(R3motor) m4 (k=0.2365, w=6250, D=0.55, z=0.022 , max=1.9)*
submodel *(R3motor) m5 (k=0.2608, w=6250, D=0.55, z=0.096 , max=2.0)*
submodel *(R3motor) m6 (k=0.0842, w=7400, D=0.27, z=0.044 , max=0.65)*

{*Antriebsstränge*}
submodel *(R3drive1) d1 (i=i1, n3=1, Rmax=0.4, c=43, d=0.005, e=0.01)*
submodel *(R3drive1) d2 (i=i2, n1=1, Rmax=0.5, c=8 , d=0.01 , e=0.06)*
submodel *(R3drive1) d3 (i=i3, n1=1, Rmax=0.7, c=58, d=0.04 , e=0)*
submodel *(R3drive2) d4 (i=i4, n3=1, J=1.6E-4)*
submodel *(R3drive2) d5 (i=i5, n3=1, J=1.8E-4)*
submodel *(R3drive2) d6 (i=i6, n3=1, J=4.3E-5)*

```
{Mehrkörpersystem}
    submodel (R3mbs6)  robot  (mL=mL, rL1=rL1, rL2=rL2, rL3=rL3)

{Getriebeübersetzungen als Konstanten, da diese an 2 Stellen benötigt werden}
    constant  i1=-105, i2=210, i3=60, i4=-99, i5=79.2, i6=-99

{Masse und Position der Last als Parameter}
    parameter  mL=0, rL1=0, rL2=0, rL3=0

{geforderte Gelenkwinkel rq in [rad] and Winkelgeschwindigkeiten rqd in [rad/s];
 Regelfehler e, ed und Antriebsmomente t der Motoren in [Nm]}
    input   rq(6), rqd(6)
    output  e(6) , ed(6) , t(6)

{Verbindungsstruktur der Roboterkomponenten}
    connect  c1 to m1 to d1 to robot:j1,
             c2 to m2 to d2 to robot:j2,
             c3 to m3 to d3 to robot:j3,
             c4 to m4 to d4 to robot:j4,
             c5 to m5 to d5 to robot:j5,
             c6 to m6 to d6 to robot:j6

{Gleitreibungsgesetze für die ersten drei Gelenke}
    d1.Rv  =  0.4 + (0.13/160) * abs(d1.qd)
    d2.Rv  =  0.5 + ( if abs(d2.qd) < 130 then (0.1/130) * abs(d2.qd)
                                          else  0.1 + (0.1/230) * (abs(d2.qd) - 130) )
    d3.Rv  =  0.7 + ( if abs(d3.qd) < 130 then (0.2/130) * abs(d3.qd)
                                          else  0.2 + (0.1/230) * (abs(d3.qd) - 130) )

{Aufschaltung der Eingangsgrößen}
    c1.rq   =  i1 * rq(1)
    c2.rq   =  i2 * rq(2)
    c3.rq   =  i3 * rq(3)
    c4.rq   =  i4 * rq(4)
    c5.rq   =  i5 * rq(5)
    c6.rq   =  i6 * rq(6)
    c1.rqd  =  i1 * rq(1)
    c2.rqd  =  i2 * rq(2)
    c3.rqd  =  i3 * rq(3)
    c4.rqd  =  i4 * rq(4)
    c5.rqd  =  i5 * rq(5)
    c6.rqd  =  i6 * rq(6)

{Aufschaltung der Ausgangsgrößen}
    t(1)  =  m1.t
    t(2)  =  m2.t
    t(3)  =  m3.t
    t(4)  =  m4.t
    t(5)  =  m5.t
    t(6)  =  m6.t
    e     =  rq - robot.q
    ed    =  rqd - robot.qd
end
```

Mit den ersten Anweisungen werden die verwendeten Bibliotheken definiert. Die Bibliothek *mbssim.lib* lädt automatisch auch die Antriebsstrang-Bibliothek. Die Bibliothek *r3.lib* enthält die in den nächsten Abschnitten besprochenen Modellklassen. Mit den **submodel** Anweisungen werden alle benötigten Komponenten definiert. Da die Kaskadenregler in den

Gelenken identisch sind, werden 6 gleiche Regler deklariert. Andererseits hat der Roboter zwei strukturell unterschiedliche Typen von Antriebssträngen, sodaß die beiden Klassen *R3drive1*, *R3drive2* benötigt werden. Mit der **input** Anweisung werden die gewünschten Winkel und Winkelgeschwindigkeiten als Eingänge in die Kaskadenregler definiert. Mit der **output** Anweisung werden die von den Motoren erzeugten Drehmomente sowie die Regelfehler (= Differenz von gewünschtem und wirklichem Winkel bzw. Winkelgeschwindigkeit) deklariert. Mittels der **connect** Anweisung werden die Komponenten entsprechend zu Bild 7.2 zusammengeschaltet.

Alle Antriebsstränge haben in der Gleitphase unterschiedliche Reibkennlinien. Diese Kennlinien werden durch lineare Interpolation einiger weniger Meßpunkte approximiert. Im Modell wird dies mit **if**-Anweisungen erreicht, da noch keine komfortablere Definitionsart für Tabellen in Dymola zur Verfügung steht. Man beachte, daß die **if**-Anweisungen bei der Codegenerierung in Zustandsereignisse umgewandelt werden. Schließlich wird am Ende des Modells angegeben, welche Modellvariablen auf die Ein- und Ausgangsgrößen des Modells geschaltet werden. Damit ist das komplette Modell auf der obersten Hierarchiestufe definiert.

In den folgenden Abschnitten wird auf die Modellbildung der einzelnen Blöcke eingegangen.

7.3 Regelungssystem

Der r3 Roboter besitzt sechs identische Kakadenregler, die den Winkel und die Winkelgeschwindigkeit eines jeden Motorläufers regeln. Die Struktur eines Reglers ist in Bild 7.3 zu sehen. Der Regler hat zwei Rückkopplungs-Schleifen. Die innere Schleife ist analog realisiert und regelt die Winkelgeschwindigkeit des Motorläufers. Die äußere Schleife arbeitet digital mit einer Abtastzeit von 8 ms und regelt die Position der Motorwelle. Das Regelungssystem kann problemlos mit der in Kapitel 6 ab Seite 156 besprochenen Bibliothek für Blockschaltbilder modelliert werden. Das Modell hat den folgenden Aufbau:

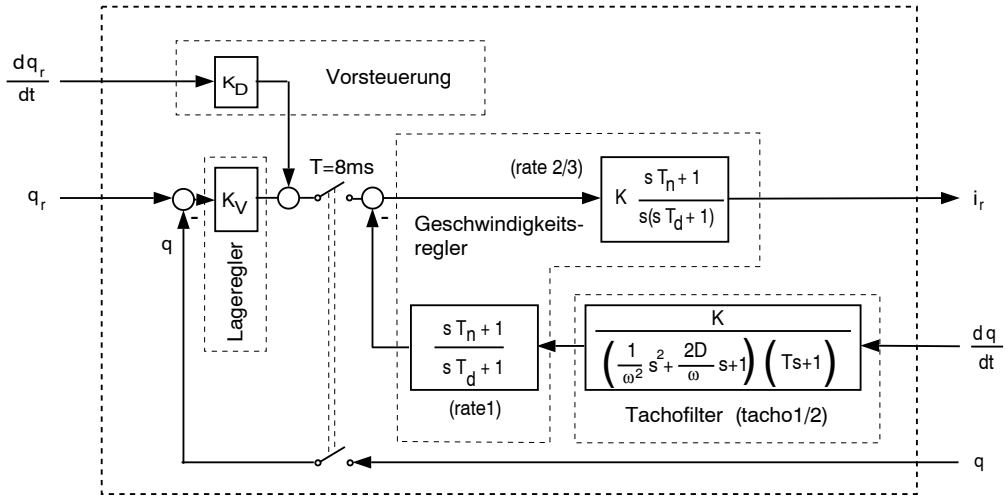


Bild 7.3: Modellstruktur eines Kaskadenreglers

```
model class R3control                    {Kaskadenregler für einen r3 Motor}
  {Basisblöcke von Tachofilter and Geschwindigkeitsregler}
      submodel (PT1)      tacho1 (T=8.475E-4)                    {Tachofilter 1}
      submodel (PT2)      tacho2 (k=0.03      , ω=2014, D=0.294) {Tachofilter 2}
      submodel (LeadLag)  rate1  (Tn=40E-3  , Td=20.2E-3)        {Geschw. Regler 1}
      submodel (LeadLag)  rate2  (Tn=9.95E-3, Td=0.56E-3)        {Geschw. Regler 2}
      submodel (I)        rate3  (k=340.8)                       {Geschw. Regler 3}
      submodel (Sum2)     sum    (k2=-1)
  {Verstärkungsfaktoren, Abtastzeit der Lageregler und lokale Variable}
      constant  Kd = 0.03, Kv = 0.3, SampleTime = 0.008
      local     NextTime = 0          {Anfangswert = 0}
  {Cut Definitionen}
            cut   in  (rq, rqd)       {geforderter Motorwinkel und Winkelgeschwindigkeit}
            cut   out (q, qd, ri)     {aktueller Motorwinkel und Winkelgeschwindigkeit,
                                       geforderter Motorstrom}
      main cut   cio [in,out]
      main path pio < in - out >

  {Verschaltungsstruktur von Tachofilter und Geschwindigkeitsregler}
      connect tacho1 to tacho2 to rate1 to sum:in2,  sum:out to rate2 to rate3

  {Aufschaltung der Eingangs- und Ausgangsgröße}
      qd = tacho1.u
      ri = rate3.y
  {Lageregler als Abtastsystem}
      when Time >= NextTime then
        new(NextTime) = Time + SampleTime
        sum.u1        = Kd * rqd + Kv * (rq - q)
      end when
end
```

Mit den **submodel** Anweisungen werden die benötigten Blöcke zur Modellierung des Tachofilters und des Geschwindigkeitsreglers definiert. Mit der **connect** Anweisung werden diese Blöcke entsprechend zu Bild 7.3 verschaltet. Auf Grund der einfachen Struktur des Positionsreglers, wird dieser gleich direkt mit Hilfe eines **when** Blocks beschrieben. Der Rumpf dieses Blocks wird nur zu den Abtastzeitpunkten ausgewertet.

7.4 Elektromotor

Die Motoren des r3 Roboters sind elektronisch kommutierte Synchronmaschinen. Durch die elektronische Kommutierung und die Stromregelung ist das dynamische Gesamtverhalten dem einer stromgeregelten Gleichstrommaschine ähnlich. Aus Vereinfachungsgründen wurde von Türk [Tuer90] nur das Ein/Ausgangsverhalten des "schnellen Subsystems" Motor vermessen und durch ein System 2ter Ordnung mit Totzone und Begrenzung approximiert. Um klar zu zeigen, daß die in mechatronischen Systemen häufig mit zu modellierenden elektrischen Bauteile problemlos in der hier benutzten Modellierungstechnik verwendet werden können, werden die Motoren hier als stromgeregelte Gleichstrommaschinen modelliert. Das Schaltbild ist in Bild 7.4 zu sehen. Die Parameter wurden so gewählt, daß das Ein/Ausgangsverhalten identisch zum approximierten Modell 2ter Ordnung von Türk ist.

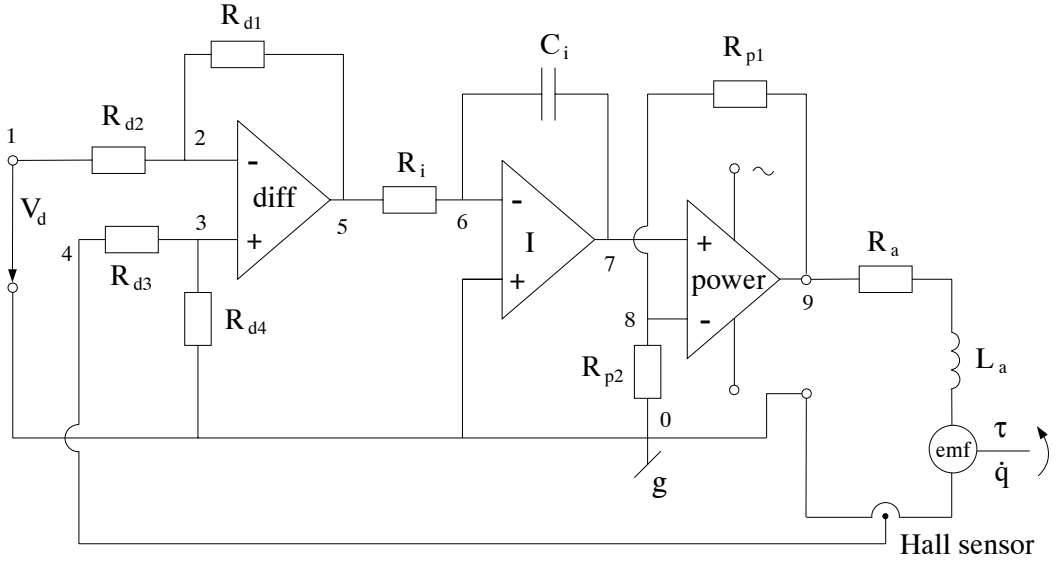


Bild 7.4: Schaltbild der r3 Motoren

Die zwei nichtlinearen Blöcke sind im Schaltbild nicht enthalten, werden aber in der folgenden Modellbeschreibung berücksichtigt:

model class *R3motor*

{*elektrische Komponenten*}

submodel (*Resistor*)	$Rd1(R = 100)$, $Rd2(R = 100)$, $Rd3$(R=100), $Rd4(R = 100)$	
submodel (*Resistor*)	Ri $(R = 10)$, $Rp1(R = 200)$, $Rp2$(R= 50) , Ra $(R = 250)$	
submodel (*CapacitorT*)	*Ci*	{Kapazität C ist Terminal Variable}
submodel (*InductorT*)	*La*	{Induktivität L ist Terminal Variable}
submodel (*Emf*)	*emf*(k=k)	{Elektro-Motorische-Kraft}
submodel (*Hall*)	*hall*	{Hall Sensor}
submodel (*OpAmpIdeal*)	*diff, I, power*	{ideale Operationsverstärker}
submodel (*Ground*)	*g*	
submodel (*DeadZone*)	$NL1$ $(zmax = z)$	{maximaler Wert der Totzone}
submodel (*Bound*)	$NL2$ $(max = max)$	{maximales Motormoment}

{*einstellbare Parameter*}

parameter *k, w, D, z, max=9*

{*Schnittstellen zum Regler (=control) und zum Rotor des Motors (=rotor)*}

cut *control (q, qd, ri)*

cut *rotor (q, qd, qdd / f)*

```
    main cut  mc    [control, rotor]
    main path mp    < control − rotor >

  {Verbindungsstruktur der Komponenten}
    node    n0, n1, n2, n3, n4, n5, n6, n7, n8, n9, m(q,qd,qdd/tmot)
    connect Rd1 at (n2,n5),     Rd2 at (n1,n2),     Rd3  at (n4,n3),
            Rd4 at (n3,n0),     Ri  at (n5,n6),     Rp1  at (n8,n9),
            Rp2 at (n8,n0),     Ci  at (n6,n7),     g    at n0,
            diff at (n3,n2,n5), I   at (n0,n6,n7),  power at (n7,n8,n9),
            n9 to Ra to La to emf to hall to n0, hall:m at n4, m at emf:mech

  {berechne Kapazität Ci und Induktivität La und verschalte NL1 und NL2}
    Ci.C    = 0.004 * D/w
    La.L    = Ra.R/(2 * D * w)
    ri      = NL1.u
    Rd2.Va  = NL1.y
    tmot    = NL2.u
    f       = NL2.y
end
```

Das elektrische Schaltbild 7.4 kann problemlos mit Hilfe der in Kapitel 2 ab Seite 17 besprochenen Klassen zur Modellierung elektrischer Bauteile beschrieben werden. Zuerst werden alle elektrischen Komponenten, wie Widerstände, Kapazitäten, Induktivitäten, Operationsverstärker, definiert. Der letzte Buchstabe "T" bei den Klassen *CapacitorT, InductorT* zeigt an, daß die Kapazität und die Induktivität nicht als Parameter, sondern als Terminal-Variablen gegeben sind. Dies ist notwendig, da diese beiden Konstanten aus anderen Konstanten berechnet werden und es in Dymola zur Zeit nicht möglich ist, Parameter durch Ausdrücke zu berechnen. Für die benutzten Operationsverstärker werden ideale Modelle verwendet. Die Klasse *OpAmp* hat deswegen den folgenden Aufbau:

```
model class OpAmp
    cut InP  (Vp / ip)          {nicht-invertierender Eingang }
    cut InM  (Vm / im)          {invertierender Eingang }
    cut Out  (Vo / io)          {Ausgang }
    main cut mc [InP, InM, Out]

    Vp = Vm
    ip = 0
    im = 0
end
```

Bei einem idealen Operationsverstärker wird die Potentialdifferenz zwischen den beiden Eingängen, sowie die Ströme in den beiden Eingängen, vernachlässigt.
Ein idealer Hall-Sensor ist eine stromgesteuerte Spannungsquelle. In der obigen Anwendung bedeutet dies, daß das Potential am Knoten $n4$ proportional zum Strom ist, der durch den Gleichstrommotor fließt.

Nachdem alle Komponenten deklariert sind, werden die Schnittstellen zum Regler (**cut** *control*) und zur Motorwelle (**cut** *rotor*) definiert und alle Bauteile entsprechend zu Bild 7.4 zusammengeschaltet.

7.5 Antriebsstrang

Das vom Motor erzeugte Drehmoment im Luftspalt treibt über ein Getriebe die Achse eines Gelenks an. In den Lagern des Getriebes und des Gelenks, sowie bei den im Eingriff befindlichen Zähnen der Getrieberäder, ist beträchtliche Reibung vorhanden. Darüberhinaus sind Elastizität, Dämpfung und Lose in den Antriebssträngen der ersten drei Gelenke zu berücksichtigen. Die äußeren drei Getriebe sind so steif, daß die Getriebe-Elastizität vernachlässigt werden kann.

Um das Modell zu vereinfachen, wird in [Tuer90] nur das vom Roboterhersteller angegebene Trägheitsmoment des Motorläufers berücksichtigt. Die Trägheitsmomente der Getriebe werden vernachlässigt. Weiterhin wird angenommen, daß Reibung nur am Motorläufer wirkt. Damit ergibt sich für die ersten drei Antriebsstränge die in Bild 7.5 gezeigte Struktur. Die Feder in Bild 7.5 soll die Elastizität, die Dämpfung und die Lose charakterisieren.

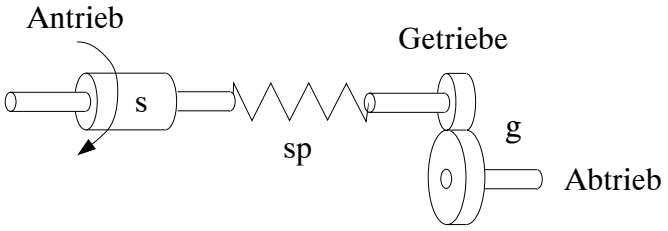


Bild 7.5: Struktur der Antriebsstränge der ersten 3 Gelenke.

Der Antriebsstrang kann problemlos mit der in Kapitel 5.5 ab Seite 149 erläuterten Bibliothek für Antriebsstränge auf bewegten Mehrkörpersystemen modelliert werden. Das Modell hat den folgenden Aufbau:

```
model class (DriveOneCut) R3drive1
   {Komponenten vom Antriebsstrang}
      submodel (DriveBaseJoint) b
      submodel (Shaft)          s   (J = 0.0013, n1 = n1, n2 = n2, n3 = n3)
      submodel (Gear)           g   (i = i)
      submodel (R3spring)       sp  (c = c, d = d, b = b)
      submodel (ExtFriction)    fr  (Rmax = Rmax)
      submodel (DriveVarLS)     v

   {Parameter vom Antriebsstrang}
      parameter i, c, d, b, Rmax, n1 = 0, n2 = 0, n3 = 0

   {Schnittstelle zum Motor (Antrieb) und zur Last (Abtrieb)}
      terminal  Rv                        {Gleitreibungsgesetz}
      cut       motor (q, qd, qdd / f)
      main path mp < motor - a >

   {Verbindungsstrukur}
      connect b to v to s to sp to g to a,
              motor at v:ext, fr at v:extL, b:joint at a

      Rv = fr.Rv
end
```

Auf Grund der am Motorläufer wirkenden Reibung, muß die Welle s bezüglich der Antriebsstrangbasis sperrbar sein, d.h. der Winkel und die Winkelgeschwindigkeit von s bezüglich der Basis müssen als Zustandsvariablen benutzt werden. Wie in Kapitel 5.5 ab Seite 149 erläutert, müssen die verallgemeinerten Koordinaten des Gelenks, das von einem Antriebsstrang angetrieben wird, ebenfalls als Zustandsvariablen verwendet werden. Damit liegen alle Zustandsgrößen für den Antriebsstrang fest.

Mit den **submodel** Anweisungen werden die benötigten Bauteile definiert. Mit der Klasse *DriveVarLS* werden der Winkel und die Winkelgeschwindigkeit zwischen Welle s und der Antriebsstrangbasis als "sperrbare" Zustandsgrößen eingeführt. Die Klasse *ExtFriction* wurde in Kapitel 4.5 ab Seite 114 ausführlich erläutert und beschreibt die am Motorläufer wirkende Reibung. Die **connect** Anweisung legt dann die Verschaltungsstruktur fest. Das speziell für diesen Antriebsstrang benötigte Kraftgesetz um Elastizität, Dämpfung und Lose zu beschreiben, wird mit der folgenden Klasse zur Verfügung gestellt:

model class *(DriveForce) R3spring*
  parameter $c = 0, d = 0, b = 0, q0 = 0$

  $f = d * qd + ($ **if** $q - q0 > b$ **then** $c * (q - q0 - b)$ **else**
  if $q - q0 < -b$ **then** $c * (q - q0 + b)$ **else** $0)$
end

Die drei äußeren Antriebsstränge sind eine Spezialisierung der drei inneren Antriebsstränge, wobei eine starre Kopplung zwischen Motorläufer und Getriebe angenommen wird. Dann ist es gleichgültig ob die Reibung am Motorläufer oder (bei entsprechender Umrechnung) in der Gelenkachse wirkt, wie im folgenden angenommen. Da Reibelemente jetzt direkt in einem Gelenk wirken, müssen beim Mehrkörpermodell des Roboters die letzten 3 Gelenke sperrbar sein.

7.6 Mehrkörpersystem

Der mechanische Teil des r3 Roboters wird als Mehrkörpersystem modelliert. Dieses besteht aus 6 Armen und 6 Drehgelenken. Wegen der vernachlässigten Getriebeelastizitäten in den letzten 3 Gelenken, können diese Gelenke im Laufe einer Simulation aufgrund von Reibung gesperrt sein. Das MKS wird folgendermaßen modelliert:

```
model class R3mbs6
  {Ineertialsystem}
    submodel (Inertial)  i (ng3 = −1, g = 9.81)

  {Gelenke und Verbindungsstangen}
    submodel (RevoluteL)   r1 (n3 = 1)  , r2 (n1 = 1)   , r3  (n1 = 1)
    submodel (RevoluteLS)  r4 (n3 = 1)  , r5 (n1 = 1)   , r6  (n3 = 1)
    submodel (Bar)         b3 (r3 = 0.5), b5 (r3 = 0.73), bL (r1 = rL1,  r2 = rL2,  r3 = rL3)

  {Körperdaten}
    submodel (Body) m1   (I33 = 1.16)
    submodel (Body) m2   (m = 56.5, r1 = 0.172, r3 = 0.205  , I11 = 2.58   , I22 = 0.73   ,
                                                              I33 = 0.64   , I31 = −0.46)
    submodel (Body) m3   (m = 26.4, r1 = 0.064, r3 = −0.034, I11 = 0.279  , I22 = 0.413  ,
                                                              I33 = 0.245  , I31 = −0.07)
    submodel (Body) m4   (m = 28.2              , r3 = 0.32   , I11 = 1.67   , I22 = 1.67   ,
                                                              I33 = 0.081)
    submodel (Body) m5   (m = 5.2               , r3 = 0.023  , I11 = 0.0125, I22 = 0.0153,
                                                              I33 = 0.0081)
    submodel (Body) Load (m = mL)

  {Reibelemente in den letzten 3 Gelenken}
    submodel (ExtFriction) R4 (Rmax = 26.7), R5 (Rmax = 39.6), R6 (Rmax = 16.8)

  {Masse und Position der Last als Parameter}
    parameter  mL = 0, rL1 = 0, rL2 = 0, rL3 = 0
    local      vaux

  {Achsen der Gelenke als Schnittstelle nach außen (für die Antriebsstränge)}
    cut  j1, j2, j3, j4, j5, j6

  {Verbindungsstruktur der Komponenten}
    connect  i     to r1 to r2 to b3 to r3 to r4 to b5 to r5 to r6 to bL,
             m1    at r1:b,   j1 at r1:Drive,
             m2    at r2:b,   j2 at r2:Drive,
             m3    at r3:b,   j3 at r3:Drive,
             m4    at r4:b,   j4 at r4:Drive,   R4 at r4:extL,
             m5    at r5:b,   j5 at r5:Drive,   R5 at r5:extL,
             load  at bL:b,   j6 at r6:Drive,   R6 at r6:extL

  {Gleitreibgesetze}
    vaux   = 79.2 * abs(r5.qd)
    R4.Rv = 21.78 + 9.801 * abs(r4.qd)
    R5.Rv = 79.2 * (0.38 + (if vaux < 100 then (0.04/100) * vaux
                                          else  0.04 + (0.08/200) * (vaux − 100) )
    R6.Rv = 10.89 + 3.9204 * abs(r6.qd)
  end
```

Das MKS kann problemlos mit Hilfe der in Kapitel 4 ab Seite 66 eingeführten Klassen beschrieben werden. Da auf der obersten Hierarchiebene die Bibliothek *mbssim.lib* angegeben wurde, wird der O(n) Algorithmus zur Aufstellung der Bewegungsgleichungen benutzt. Zuerst werden mit den **submodel** Anweisungen wiederum die benötigten Komponenten deklariert. Die ersten 3 Gelenke sind von der Klasse *RevoluteS*, weil diese Gelenke nicht blockiert werden und die Relativwinkel und Relativwinkelgeschwindigkeiten als Zustandsgrößen benutzt werden. Die 3 letzten Gelenke sind von der Klasse *RevoluteLS*, da aufgrund von Reibung im Gelenk ein Blockieren möglich ist. Die Stangen *b*3, *b*5, *bL* beschreiben den Versatz zwischen Gelenken, sowie den Angriffspunkt der Lastmasse. Die Starrkörper des Roboters werden mit der Klasse *Body* definiert. Der äußerste Roboterkörper ist sehr klein und gegenüber dem Rotor des 6-ten Motors vernachlässigbar. Aus diesem Grund sind nur

5 Arme, sowie die Last, aufgeführt. Die Schnittstellen nach außen sind die Achsen der 6 Drehgelenke, an denen Antriebsstränge befestigt werden können.

7.7 Ausgewählte Simulationen

Mit Hilfe des Dymola Compilers wird das in den vorigen Abschnitten erläuterte Modell des Roboters r3 auf Blockdreiecksform transformiert und als Fortran Unterprogramm im DSblock-Format auf eine Datei ausgegeben. Der gesamte Generierungsvorgang, inklusive Einlesen der Bibliotheken, Transformation auf Blockdreiecksform und Codeerzeugung, dauert ca. 1 Minute[1]. Das Modell ist in Zustandsform und besitzt 66 Zustände, 12 Eingänge, 18 Ausgänge und 138 Indikatorfunktionen. Der Code zur Auswertung einer rechten Seite enthält rund 950 Multiplikationen und Divisionen, sowie 760 Additionen und Subtraktionen. Der erzeugte DSblock-Code kann direkt in die ANDECS Simulationsumgebung DSSIM [Otte93b] eingebunden werden. Innerhalb von DSSIM stehen standardmäßig 10 unterschiedliche Integrationsverfahren mit variabler Schrittweite (und teilweise variabler Ordnung) zur Verfügung.

In einem ersten Schritt werden die Simulationen wiederholt, die zur Verifizierung der Messungen von Türk [Tuer90] durchgeführt worden sind. Ein typisches Simulationsergebnis ist in Bild 7.6 zu sehen. Dieses ist äquivalent zum Mess- und Simulationsexperiment aus [Tuer90], Seite 122/123. Hierzu wird der Roboter in seine Referenzposition gefahren. Die Anfangsbedingungen aller Zustandsgrößen sind Null. Die zwei äußeren Gelenke sind gebremst. Die Geschwindigkeits- und Positionsregler aller Gelenke sind "abgeklemmt" und es werden Sprünge auf die Stromsollwerte mit den folgenden Sprunghöhen gegeben: $ri_1 = -2.5$, $ri_2 = -2$, $ri_3 = -2.5$, $ri_4 = -3$. Gemessen werden die Spannungen am Ausgang der Tachofilter (in Volt). Zur Durchführung entsprechender Simulationen müssen die Kaskadenregler aus dem Modell entfernt und die letzten beiden Gelenke gesperrt werden. Aufgrund der übersichtlichen, objektorientierten Modellstruktur, und da Gelenke durch Setzen der logischen Variablen *Locked* gesperrt werden können, ist das einfach und schnell möglich.

Im Rahmen der Zeichengenauigkeit sind die Ergebnisse von Bild 7.6 mit den in Bild 43 aus [Tuer90] gezeigten Simulationen identisch. Die Messungen weichen nur geringfügig von den Simulationen ab. Zum quantitativen Vergleich sind in Tabelle 7.1 die Werte der Tachospannung am Ende der Simulation (0.2 s) angegeben, da dort die größte Differenz zwischen Simulation und Messung vorliegt. Die Ablesegenauigkeit von Bild 43 aus [Tuer90] beträgt ca. ± 0.25. Der Verstärkungsfaktor des Tachofilters ist 0.03 (siehe Klasse *R3control* auf Seite 170). Deswegen erreicht z.B. der Motor 4 bei diesem Experiment fast seine maximale Drehzahl von 3200 U/min ($n = 9.8 \cdot 60/(2 \cdot \pi \cdot 0.03)$).

Um das Verhalten des Gesamtmodells an einer anspruchsvollen Aufgabe zu demonstrieren, wurden von A. Lewald die Sollverläufe der Reglereingänge durch Trajektorien-Optimierung so bestimmt, daß der Roboter an seinen Leistungsgrenzen arbeitet. Die Solltrajektorien wurden durch die folgende Aufgabenstellung ermittelt:

[1] Auf einer Apollo Workstation mit Motorola 68040 Prozessor.

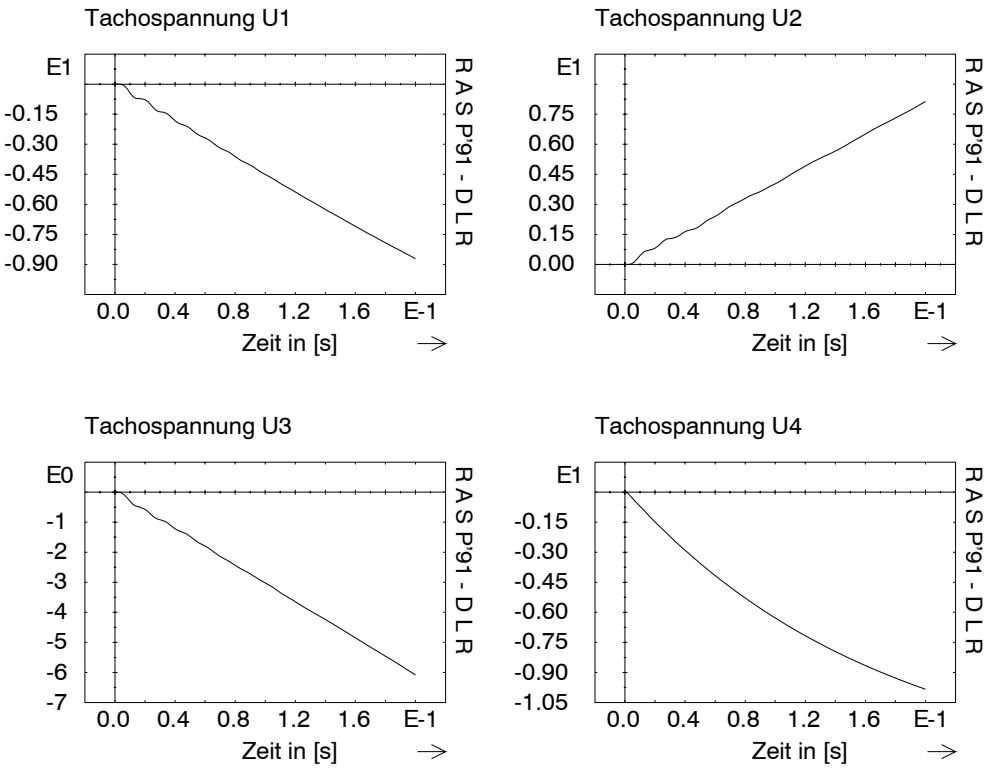


Bild 7.6: Simulationen zum Vergleich mit Meßdaten.

	Simulation (Bild 7.6)	Simulation (Bild 43 aus [Tuer90])	Messung (Bild 43 aus [Tuer90])
Tachospannung U1	-8.71	-8.5	-8.2
Tachospannung U2	8.07	8.0	8.5
Tachospannung U3	-6.09	-6.0	-6.5
Tachospannung U4	-9.84	-10.0	-9.5

Tabelle 7.1: Vergleich von Simulation und Messung (Werte der Tachospan. bei 0.2 s).

Der Roboter soll von der Anfangsstellung $\mathbf{q} = [-1,\ 2,\ 1,\ 0,\ -2,\ 0]$ in *minimaler Zeit* in die Endstellung $\mathbf{q} = [1,\ 0.5,\ -1,\ 0,\ 1.64,\ 0]$ gefahren werden. Hierbei sind $\mathbf{q}$ die Gelenkwinkel des Roboters in $[rad]$. Am Anfang und am Ende soll der Roboter in Ruhe sein, d.h., die Anfangswerte aller Geschwindigkeitsgrößen sind Null. Für diese Optimierung wird nur das Starrkörpermodell des Roboters, inklusive idealem Antriebsstrang (d.h. keine Reibung, Elastizität und Lose) benutzt. Die zum Abfahren der Trajektorie benötigten Motormomente sollen 70% der maximalen Motormomente ($= 0.7 \cdot [9,\ 9,\ 9,\ 1.9,\ 2,\ 0.65]\ Nm$) nicht überschreiten. Es ist nicht sinnvoll von den maximal möglichen Motormomenten auszugehen, da die Regler noch Stellreserven benötigen, und da die Reibung bei der Optimierung nicht berücksichtigt wird. Weiterhin darf die zu ermittelnde Trajektorie nicht die maximalen Winkelgeschwindigkeiten der Motorwellen überschreiten ($|\dot{\mathbf{q}}_{max}| = [3,\ 1.5,\ 5.2,\ 3.4,\ 4.3,\ 3.7]\ rad/s$).

Die Lösung dieser Aufgabenstellung wurde mit einem neuen, von A. Lewald entwickelten Trajektorien-Optimierungsverfahren durchgeführt [Lewa93]. Dieses Verfahren beruht auf den neuen Ergebnissen von Kapitel 4 über strukturvariable Mehrkörpersysteme. Die Grundidee besteht darin, die Zustandsbeschränkungen des Modells (hier: maximale Motorgeschwindigkeiten), nicht im Optimierer, sondern *im Modell* zu berücksichtigen. Das heißt, das Modell garantiert, daß die Zustandsbeschränkungen nicht verletzt werden. Wenn eine Zustandsbeschränkung erreicht wird, schaltet das Modell in eine entsprechend andere Modellstruktur um. Bei der obigen Aufgabenstellung entspricht das einem "Blockieren" von Gelenken. Wenn die maximale Geschwindigkeit eines Motors erreicht ist, so wird das entsprechende Gelenk "blockiert", d.h. die Gelenkwinkel-Beschleunigung wird auf Null bzw. die Gelenkwinkel-Geschwindigkeit wird auf den konstanten Wert der Maximalgeschwindigkeit gesetzt. Das vom Modell berechnete Zwangsmoment gibt dann an, welches Moment auf dieses Gelenk aufgebracht werden muß, damit das Gelenk genau mit seiner Maximalgeschwindigkeit verfahren wird. Lewald zeigt in [Lewa93], daß dieses neue Verfahren bei der Ermittlung von Randsteuerungen rund 2 Größenordnungen schneller ist, als bisher bekannte Methoden.

In Bild 7.7 ist ein Bildschirmabzug von einer 3D-Animation des Roboters zu sehen, bei der die durch Optimierung ermittelte Trajektorie abgefahren wird. Der Roboter fährt hierbei von "rechts oben" nach "links unten". Die Verfahrzeit der Trajektorie, und damit die minimale Endzeit, beträgt ca. 1.1 Sekunden. Die Regelfehler beim Verfahren dieser Trajektorie sind etwas zu groß. Aus diesem Grund wird die Sollvorgabe um 10% zeitskaliert, d.h., der Roboter erreicht nach rund 1.2 Sekunden die Endstellung. Die Regler des Manutec Roboter sind gut genug, um dieser so vorgebenen Sollvorgabe zu folgen. In Bild 7.8 sind die Ergebnisse einer Gesamtsystem-Simulation aufgetragen. Bei der Simulation wurde das in diesem Abschnitt besprochene Modell verwendet.

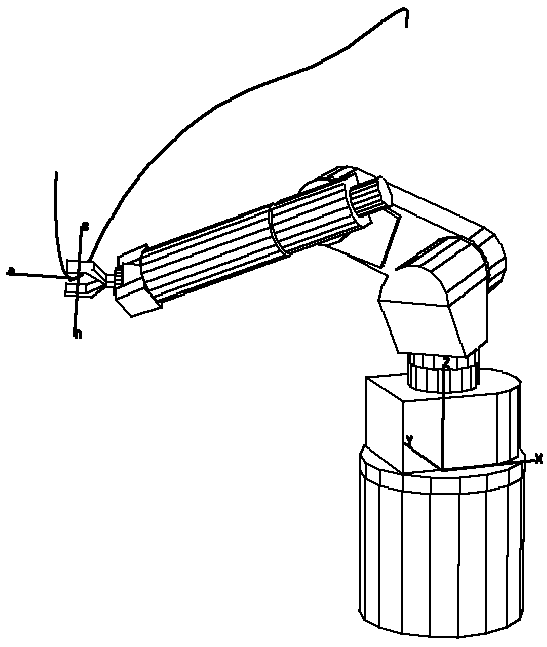
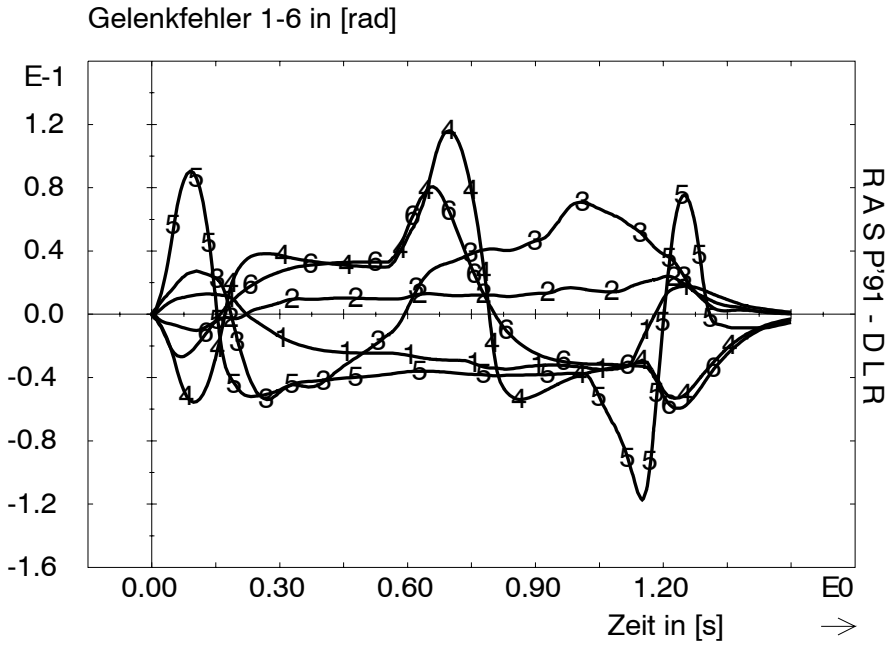
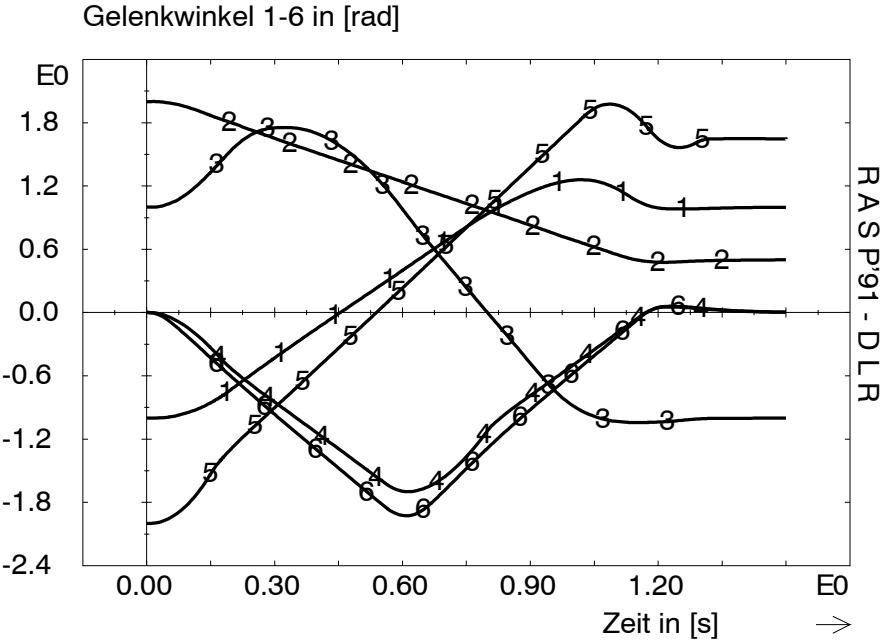
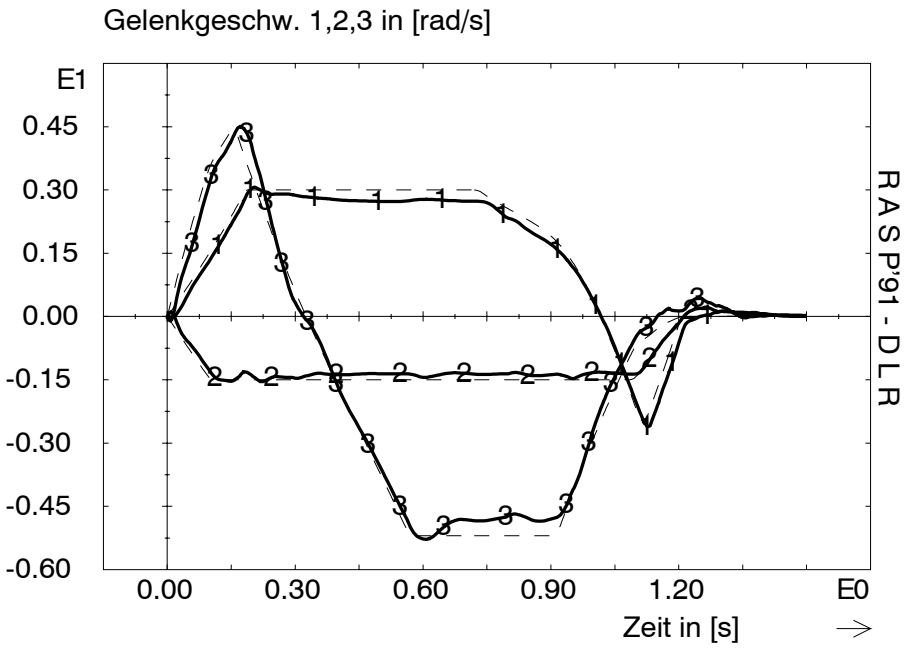


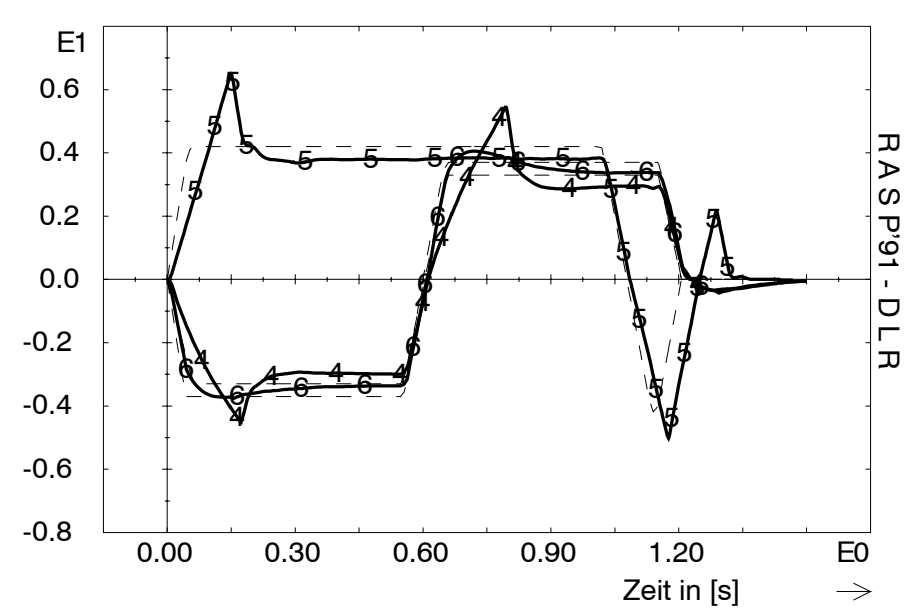
Bild 7.7: Animation der "zeitoptimalen" Sollvorgabe des Roboters Manutec r3.

Im ersten Bild sind die 6 Gelenkwinkelverläufe des Roboters aufgetragen. Man sieht z.B., daß der Verfahrweg von Gelenk 3 rund 2 $rad = 120^o$ beträgt. Im zweiten Bild sind die Regelfehler der Gelenkwinkel aufgetragen. Der größte Fehler beträgt ca. 0.1 $rad = 5.7^o$. Dies ist etwas hoch, für die extreme Sollvorgabe jedoch noch zufriedenstellend. In den beiden nächsten Bildern sind die Gelenkwinkel-Geschwindigkeiten zu sehen, sowie, als gestrichelten Linienzug, die Sollvorgaben für diese Geschwindigkeiten. Man sieht, daß die Sollvorgaben recht gut eingehalten werden. Die horizontalen Teile der gestrichelten Kurven bedeuten, daß das entsprechende Gelenk bei der Optimierung an einer Zustandsraumbeschränkung ist und hier mit seiner Maximalgeschwindigkeit fährt. Schließlich sind in den beiden letzten Bildern die aktuellen Motormomente, sowie, als gestrichelter Linienzug, die durch die Trajektorien-Optimierung berechneten (zeitskalierten) Motormomente zu sehen. Man sieht, daß der geregelte Roboter zu gewissen Zeiten mit Gelenk 3, 4 und 5 an der Momentengrenze des Motors fährt (horizontale Kurvenzüge).






Gelenkgeschw. 1,2,3 in [rad/s]
E1
0.45
0.30
0.15
0.00
-0.15
-0.30
-0.45
-0.60
0.00
0.30
0.60
0.90
1.20
E0
Zeit in [s]
RASP'91 - DLR


Gelenkgeschw. 4,5,6 in [rad/s]
E1
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
-0.8
0.00
0.30
0.60
0.90
1.20
E0
Zeit in [s]
RASP'91 - DLR

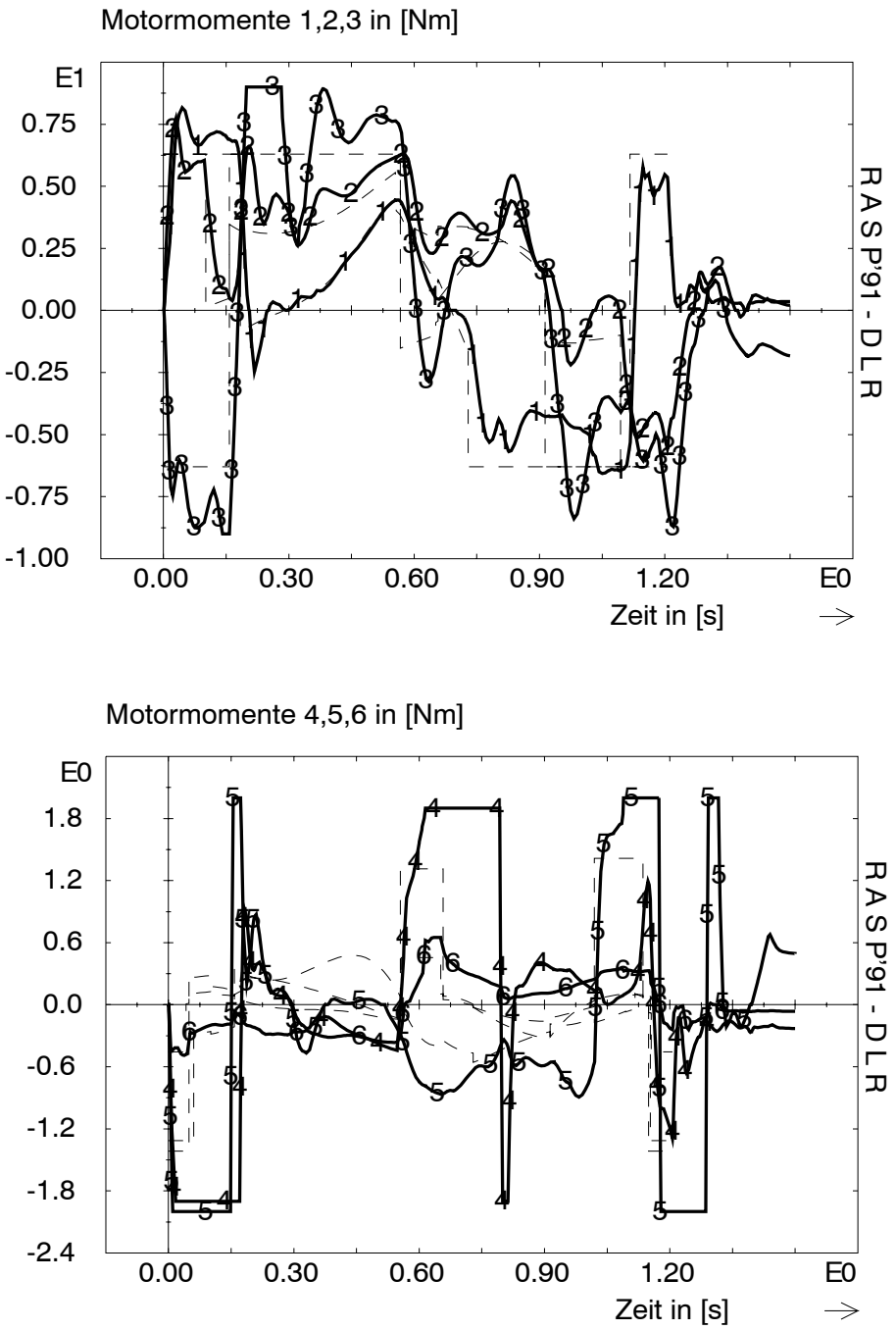


Bild 7.8: Simulationsergebnis für eine "zeitoptimale" Sollvorgabe des Roboters Manutec r3.

Kapitel 8

Zusammenfassende Diskussion und Ausblick

In dieser Arbeit wurde eine neue Methodik zur Modellierung von mechatronischen Systemen vorgestellt. Die Modellierung solcher Systeme ist nicht ganz einfach, weil die Einzelkomponenten aus unterschiedlichen Fachgebieten stammen. Für jedes Fachgebiet gibt es Modellbildungsumgebungen, um Systeme aus diesem Fachgebiet zufriedenstellend simulieren zu können. Es gab aber bisher keine Modellbildungsumgebung, mit der Gesamtmodelle von Systemen erstellt werden können, deren Einzelkomponenten aus unterschiedlichen Fachgebieten stammen. Zum Beispiel ist es nicht möglich, ein 3-dimensional bewegtes mechanisches System mit einem regelungstechnischen Blockschaltbild-Simulator wie SIMULINK, mit einer allgemeinen Simulationssprache wie ACSL, oder mit einem Elektronikprogramm wie SPICE zu modellieren. Umgekehrt können unstetige Elemente und schaltende Systeme nicht mit Mehrkörperprogrammen, Elektronikprogrammen oder den meisten Blockschaltbild-Simulatoren modelliert werden.

Zur fachgebietsübergreifenden Simulation wurden die allgemeinen Simulationssprachen entworfen. Diese setzen voraus, daß die Modellgleichungen von Komponenten gegeben sind und daß diese Gleichungen durch einfaches Sortieren in eine korrekte Abarbeitungsreihenfolge gebracht werden können. Bei fast allen physikalischen Systemen, z.B. in der Mechanik oder in der Elektrotechnik, besteht aber gerade die Hauptschwierigkeit darin, diese Gleichungen aus der Modelldefinition abzuleiten. Allgemeine Simulationssprachen oder Blockschaltbild-Editoren sind hier also kaum von Nutzen.

Die hier dargestellte Methodik basiert auf der grundlegenden Arbeit von Elmqvist [Elmq78] und weiterführender Arbeiten von Elmqvist und Cellier [Cell91, Cell93, Elmq93a]. Es ist schon lange bekannt, daß die Modellierung verschiedenartiger physikalischer Systeme auf den gleichen formalen Grundlagen beruhen (z.B. Across- und Through-Variablen). Elmqvist hat diese gemeinsamen Eigenschaften in seiner objektorientierten Modellierungssprache Dymola als Sprachelemente zur Verfügung gestellt und Algorithmen implementiert, um mit Dymola beschriebene Modelle in die für eine rechnerische Auswertung geeignete Zustandsform zu überführen.

Diese Vorgehensweise ist aus mehreren Gründen sehr vielversprechend: Zum einen können die spezifischen Modellklassen eines Fachgebiets auf einem recht hohen Sprachniveau in einer Bibliothek definiert werden. Wenn solche Bibliotheken vorliegen, kann *multidisziplinär*

modelliert werden, d.h. Modelle aus unterschiedlichen Fachgebieten können mit diesen Bibliotheken beschrieben und entsprechend der *physikalischen* Verschaltungsstruktur zusammengeschaltet werden. Aus ein und derselben Modellbeschreibung können die Gleichungen für unterschiedliche Aufgabenstellungen automatisch generiert werden, indem nachträglich festgelegt wird, welche Größen bekannt sind. Bis jetzt fehlten jedoch Modell-Bibliotheken, deren Leistungsumfang mit fachgebietsspezifischen Programmpaketen vergleichbar ist. Es gab nur kleine Experimental-Bibliotheken für einfache 1-dimensionale Systeme. Cellier hat einige davon in [Cell91] beschrieben.

In der vorliegenden Arbeit wird gezeigt, wie bekannte Modellierungsverfahren aus drei wichtigen Fachgebieten der Mechatronik (Mehrkörpersysteme, Antriebsstränge, regelungstechnische Systeme) in eine objektorientierte Darstellung umformuliert werden können. Weiterhin werden entsprechende Realisierungen in Form von Dymola-Bibliotheken vorgestellt. Dies ist insbesondere für Mehrkörpersysteme eine neue Vorgehensweise: Nur die *lokalen* Eigenschaften mechanischer Komponenten werden durch Gleichungen beschrieben. Die topologische Kopplung dieser Komponenten, die zugehörige physikalisch-mathematische Verschaltung, und die Transformation auf Zustandsform werden mit den allgemeinen Transformationsalgorithmen von Dymola durchgeführt, wobei Dymola selbst keinerlei Kenntnisse über mechanische Systeme besitzen. Traditionell beruhen Programmpakete in der Mechanik auf mechanischen Prinzipien bei denen die Gleichungen der Einzelkomponenten, die Verschaltung und die Transformation auf Zustandsform so eng miteinander gekoppelt sind, daß eine Trennung schwerfällt. Dieses Aufspalten der Modellierung in einen *fachgebietsspezifischen Teil*, in dem die Einzelkomponenten beschrieben werden, und in einen *fachgebietsunabhängigen Teil* um die Komponenten miteinander zu verschalten und in die Zustandsform zu überführen, ist eine Voraussetzung um *wirklich multidisziplinär* modellieren zu können.

Mit der objektorientierten Neuformulierung werden Modellierungsverfahren unter einem neuen Blickwinkel gesehen. Dies führt zu wichtigen, *neuen* Verallgemeinerungen bekannter Methoden, mit denen z.B. strukturvariable mechanische Systeme mit sehr vielen möglichen Konfigurationen effizient behandelt werden können. Dies erlaubt insbesondere auch, daß ein *symbolisches* Programm, wie Dymola, Code für ein Modell erzeugen kann, obwohl das Modell eine *große* Zahl von Konfigurationen mit einer jeweils unterschiedlicher Anzahl an Zustandsgrößen enthält. Demonstriert wurde das Verfahren an einem Roboter, der auf Grund von Haft/Gleitreibungszuständen 64 unterschiedliche Modellstrukturen besitzt. Mit den hier entwickelten, theoretischen Überlegungen kann auch die Leistungsfähigkeit reiner Mehrkörperprogramme (wie z.B. NEWEUL) erweitert werden.

Es genügt nicht, daß die variable Struktur eines Modells in der Simulation zufriedenstellend behandelbar ist. Es sind auch Mechanismen nötig, mit denen einfach definiert werden kann, wann von einer Modellstruktur auf die andere umgeschaltet werden muß. Hierzu wurde kürzlich von Elmqvist, Cellier und Otter [Elmq93b] eine grundlegend neue Methode entwickelt, die in Kapitel 3 im Detail beschrieben ist. Diese basiert auf "high-level" Modellierungs-Sprachelementen, die dann von einem Compiler in "low-level" Zeit- und Zustandsereignisse übersetzt werden. Dieses Vorgehen garantiert zum Beispiel, daß die Aktionen bei Ereignissen, die zufällig oder beabsichtigt zum selben Zeitpunkt eintreten, in der "richtigen" Reihenfolge abgearbeitet werden. Die Kombination der neuen, objektorientierten MKS-Bibliothek, die die Behandlung von Strukturänderungen erlaubt, mit einer einfachen und sicheren Definition von Schaltvorgängen, ermöglicht die Modellierung und Simulation technisch wichtiger mechanischer Systeme. Dies wurde am Beispiel eines Roboters mit Coulomb'scher Reibung

in allen 6 Gelenken demonstriert.

Alle in dieser Arbeit vorgestellten Klassen werden mit dem Dymola-Compiler standardmäßig mit ausgeliefert. Damit ist es für einen Dymola Anwender einfach möglich, die hier beschriebenen neuen Konzepte bei eigenen Anwendungen einzusetzen.

Die weitere Entwicklung von Dymola, und generell die Entwicklung objektorientierter Modellierungssprachen, wird vor allem in folgenden Richtungen vorangetrieben:

Basisalgorithmen:
Die Effizienz der eingesetzten, allgemeinen Algorithmen zur Transformation eines Modells in eine geeignete Form, z.B. die Zustandsform, muß noch gesteigert werden. Zum Beispiel kann mit dem Tearing-Verfahren ein MKS auf die Standardform transformiert werden, hierbei wird aber bisher nicht erkannt, daß die Massenmatrix symmetrisch ist. Weiterhin sollten zur Lösung linearer und nichtlinearer Gleichungssysteme numerisch arbeitende "Sparse Matrix" Verfahren verwendet werden. In elektrischen Schaltungsprogrammen wie SPICE, werden die Modelle mit dem Integrator aus Effizienzgründen sehr eng gekoppelt. Dies wird durch eine frühzeitige Diskretisierung der Einzelkomponenten erreicht. *Nach* der Diskretisierung werden die Bauteile zusammengeschaltet und auf eine geeignete Form transformiert (siehe z.B. [Chua87, McCa88]). Es wäre höchst interessant eine solche Vorgehensweise in ein allgemeines Werkzeug wie Dymola einzubauen, weil diese enge Kopplung von Modell und Integrator dann sofort in sehr vielen Fachgebieten eingesetzt werden kann. Von Elmqvist gibt es hierzu erste Vorschläge.

Bibliotheken:
Der funktionale Umfang der vorhandenen Bibliotheken muß noch erweitert werden und Bibliotheken anderer Fachgebiete sollten hinzukommen. (z.B. Hydraulik, Thermodynamik, Optik, chemische Prozeßtechnik). Obwohl die Erstellung von Dymola Bibliotheken relativ einfach ist, erfordert die Entwicklung einer leistungsfähigen Bibliothek den Einsatz von Spezialisten mit sehr gutem Grundlagenverständnis des jeweiligen Fachgebiets. In den einzelnen Fachgebieten ist soviel allgemeine und spezielle Modellierungserfahrung vorhanden, daß ein Nicht-Spezialist kaum eine Bibliothek erstellen kann, die mit der Leistungsfähigkeit vorhandener Spezial-Programmsysteme konkurrieren kann. *Wenn* eine Bibliothek aber einmal *verfügbar ist*, hat auch ein Nicht-Spezialist keine Schwierigkeiten, die Modellierungsverfahren des entsprechenden Fachgebietes zu verstehen. Dies ergibt sich aus der Eigenschaft, daß in einer Klassen-Bibliothek nur die "lokalen" Gleichungen von Basiskomponenten gespeichert sind. Die meist schwierig zu überblickende Art der Zusammenschaltung von Elementen mit nachträglicher Transformation auf Zustandsform, wird dann mit allgemeinen Werkzeugen (wie Dymola) automatisch durchgeführt.

Grafisch unterstützte Modellierung:
Ein Hauptvorteil fachgebietsspezifischer Programme liegt darin, daß heutzutage vielfach grafische Benutzungsoberflächen angeboten werden, die auf die Modellierung des jeweiligen Fachgebiets zugeschnitten sind. Zum Beispiel werden bei elektrischen Schaltungsentwurfsprogrammen Schaltungen so erstellt, daß elektrische Komponenten auf dem Bildschirm plaziert und mit Leitungen verbunden werden. Mehrkörpersysteme werden mit MKS-Programmen als 3-dimensionale Systeme eingegeben, die Ähnlichkeiten zu einem CAD-System haben (oder es werden selektierte Daten von einem CAD-System übernommen). Regelungstechnische Systeme werden grafisch durch Blockschaltbilder spezifiziert. Es ist unabdingbar, daß solche grafischen Ein- und Ausgabemöglichkeiten für ein multidisziplinäres

Modellierungssystem ebenfalls verfügbar werden.
Erste Ansätze und Erfahrungen gibt es dazu bei der objektorientierten Modellierungssprache Omola, für die ein experimenteller Objektdiagramm-Editor erstellt wurde. Basierend auf den Erfahrungen bei der Entwicklung des Omola-Editors, wurde von Dag Brück (einem der Entwickler des Omola-Editors) und von Hilding Elmqvist, mit finanzieller Unterstützung der DLR, ein kommerziell verfügbarer Objektdiagramm-Editor für Dymola erstellt, der zur Zeit als "Beta-Version" vorliegt. Ein typischer Bildschirmabzug von *Dymodraw* ist in Bild 8.1 zu sehen. Dieser stellt das in Kapitel 7 erläuterte objektorientierte Modell des Industrierobo-

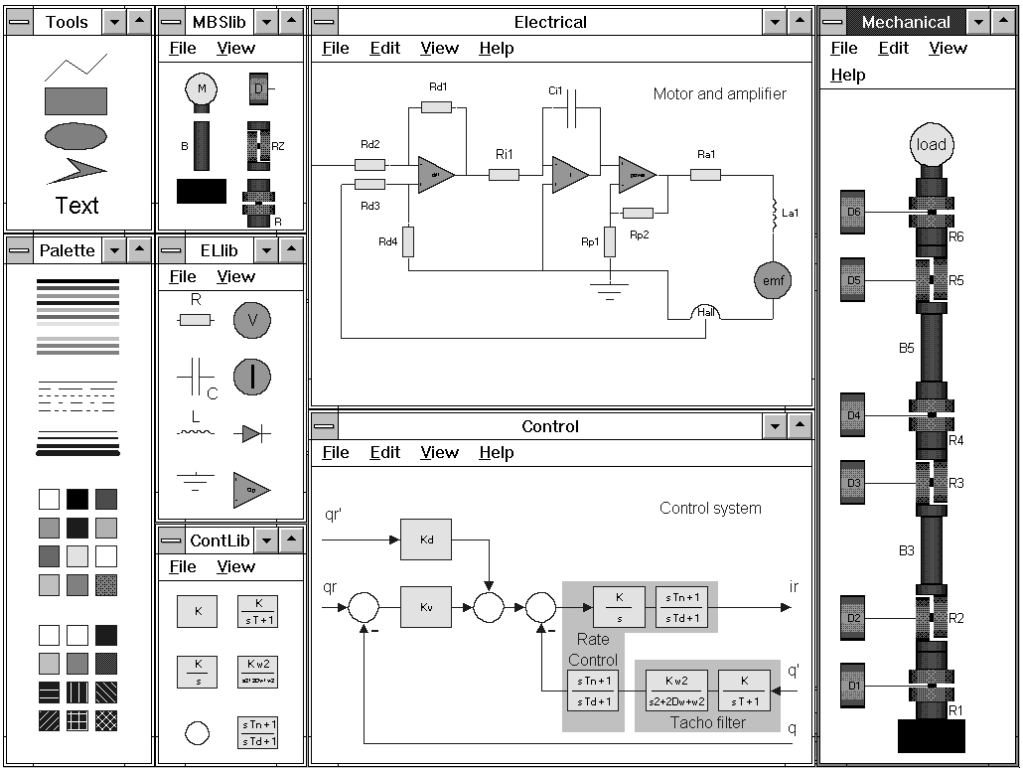


Bild 8.1: Dymodraw-Modell des r3 Roboters.

ters Manutec r3 grafisch dar. Die oberste Modell-Hierarchie ist im rechten Teil von Bild 8.1 mit der Überschrift "`Mechanical`" zu sehen, enthält hier auch schon die mechanischen Basisobjekte wie Drehgelenke und Starrkörper.

Die Antriebsstränge des Roboters inklusive Motor und Regler werden mit den Objekten `D1, D2, ..., D6` beschrieben, die bei den Dreh-Gelenken angebracht sind. Durch "Klicken" auf eines dieser Objekte wird in eine (nicht-dargestellte) Hierarchie verzweigt, in der die 3 Objekte *Regler, Motor* und *Getriebe* verschaltet sind. Das Objekt *Motor*, welches im oberen Teil von Bild 8.1 in der Mitte zu sehen ist, beschreibt den elektrischen Teil des Motors, inklusive Stromregelung, als elektrischen Schaltkreis. Im linken Teil des Bildes (`ELlib`) ist ein Teil der Klassenbibliothek für elektrische Bauteile zu sehen, die bei der Erstellung des Schaltkreises verwendet wurde.

Schließlich ist im unteren Teil von Bild 8.1 einer der Regler des Manutec Roboters in Blockschaltbild-Darstellung zu sehen. Wie beim Motor und der Mechanik des Roboters, werden auch hier die Basis-Bausteine einer Klassenbibliothek entnommen (`ContLib` in Bild 8.1 unten links) und entsprechend verschaltet.

Anhang A

Beweise und Herleitungen

A.1 Beweis zum differential-algebraischen Index

In diesem Abschnitt wird bewiesen, daß für jede DAE der durch die Gleichungen (2.19) auf Seite 34 eingeführte differential-algebraische Index, gleich dem Störungsindex (2.17) ist. Der neu eingeführte differential-algebraische Index wird im folgenden, abgekürzt, als "DA-Index" bezeichnet.

Aufgrund der Definition des DA-Index, stimmt dieser für Index-0 Systeme mit dem differentiellen Index und dem Störungsindex überein. Für den folgenden Beweis genügt es deswegen, Systeme mit DA-Index > 0 zu betrachten. In [Gear90] wird bewiesen, daß der differentielle Index und der Störungsindex für semi-explizite DAEs der Form

$$\dot{\mathbf{y}}_1(t) = \mathbf{f}_1\left(t, \mathbf{y}_1(t), \mathbf{y}_2(t)\right) \tag{A.1a}$$

$$\mathbf{0} = \mathbf{f}_2\left(t, \mathbf{y}_1(t), \mathbf{y}_2(t)\right) \tag{A.1b}$$

identisch ist[1]. Jede DAE (2.18) von Seite 33 kann durch die Transformation

$$\dot{\mathbf{x}}(t) = \mathbf{z}(t) \tag{A.2a}$$

$$\mathbf{0} = \mathbf{f}\left(t, \mathbf{x}(t), \mathbf{z}(t), \mathbf{w}(t)\right) \tag{A.2b}$$

in eine semi-explizite DAE (A.1) umgeformt werden. Es wird gezeigt, daß der DA-Index durch diese Umformung *nicht* geändert wird, und daß der DA-Index der semi-expliziten Gleichung (A.2) identisch zum differentiellen Index ist. Da der differentielle Index einer semi-expliziten Gleichung gleich dem Störungsindex ist, folgt damit, daß der DA-Index mit dem Störungsindex übereinstimmt.

Wenn die DAE "$\mathbf{f}(t, \mathbf{x}, \dot{\mathbf{x}}, \mathbf{w})$" den DA-Index j besitzt, so können, nach (j-1)-maligem Differenzieren der Funktion $\mathbf{f}$, die Variablen $\dot{\mathbf{x}}$ und $\mathbf{w}$ als Funktionen von $\mathbf{x}$ und t angegeben werden. Diese Eigenschaft bleibt erhalten, auch wenn $\dot{\mathbf{x}}$ in $\mathbf{f}$ in die Variable $\mathbf{z}$ umbenannt wird. Hierbei wird vorausgesetzt, daß die durch das Differenzieren neu auftretenden $\dot{\mathbf{x}}$-Variablen, immer sofort durch $\mathbf{z}$ ersetzt werden. Das heißt, nach (j-1)-maligem Differenzieren können $\mathbf{z}$ und $\mathbf{w}$ als Funktionen von $\mathbf{x}$ und t angegeben werden. Mit Gleichung (A.2a) ist dann aber

[1] Gear gibt eine allgemeinere Identität an. Semi-explizite DAEs sind als Spezialfall enthalten.

auch $\dot{\mathbf{x}}$ als Funktion von $\mathbf{x}$ und t bekannt, ohne daß neu differenziert zu werden braucht. Der DA-Index hat sich damit nicht geändert.

Zur Bestimmung des DA-Index der semi-expliziten Gleichung (A.2) muß nach Definition $(j-1)$ mal differenziert werden, um $\dot{\mathbf{x}}$, $\mathbf{w}$, $\mathbf{z}$ als Funktionen von $\mathbf{x}$ und t zu erhalten. Eine weitere Differentiation liefert $\dot{\mathbf{w}}$, $\dot{\mathbf{z}}$ als Funktion von $\mathbf{x}$ und t. Damit ist der differentielle Index gleich "$(j-1)+1=j$", d.h. differentieller Index und DA-Index sind bei dieser DAE identisch. QED.

A.2 Beweis zur allgemeinen DAE-Indexreduktion

In diesem Abschnitt wird die Gültigkeit der Transformation (2.35) von Seite 40 bewiesen, mit der jede *strukturell* singuläre DAE auf eine DAE mit Störungsindex 2 *automatisiert* zurückgeführt und dann z.B. mit einem (leicht modifizierten) Index-1 Integrator wie DASSL gelöst werden kann[2].

Mit dem Algorithmus von Pantelides [Pant88] wird die minimale Anzahl an Gleichungen (2.19) auf Seite 34 bestimmt, um den Störungsindex einer DAE zu ermitteln[3]. Das entstehende Gleichungssystem hat die folgende Struktur:

$$\mathbf{0} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \\ \vdots \\ \mathbf{f}_j \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \frac{d\,\mathbf{f}_{1a}}{dt} \\ \frac{d\,\mathbf{f}_{2a}}{dt} \\ \vdots \\ \frac{d\,\mathbf{f}_{(j-1)a}}{dt} \end{bmatrix} = \begin{bmatrix} \mathbf{f}\,(t,\mathbf{x},\mathbf{z}_1) \\ \mathbf{J}_2\,(t,\mathbf{x},\mathbf{z}_{1a}) \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}}_{1a} \end{bmatrix} + \mathbf{g}_2\,(t,\mathbf{x},\mathbf{z}_{1a}) \\ \mathbf{J}_3\,(t,\mathbf{x},\mathbf{z}_{2a}) \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}}_{2a} \end{bmatrix} + \mathbf{g}_3\,(t,\mathbf{x},\mathbf{z}_{2a}) \\ \vdots \\ \mathbf{J}_j\,(t,\mathbf{x},\mathbf{z}_{(j-1)a}) \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}}_{(j-1)a} \end{bmatrix} + \mathbf{g}_j\,(t,\mathbf{x},\mathbf{z}_{(j-1)a}) \end{bmatrix} \tag{A.3}$$

Hierbei besteht die Funktion $\mathbf{f}_{ia}$ aus einer Teilmenge der Gleichungen der Funktion $\mathbf{f}_i$. Die Jacobi-Matrizen $\mathbf{J}_i$ und die partiellen Zeitableitungen $\mathbf{g}_i$ sind Abkürzungen für:

$$\mathbf{J}_i = \begin{bmatrix} \frac{\partial \mathbf{f}_{(i-1)a}}{\partial \mathbf{x}} & \frac{\partial \mathbf{f}_{(i-1)a}}{\partial \mathbf{z}_{(i-1)a}} \end{bmatrix} \; ; \qquad \mathbf{g}_i = \frac{\partial \mathbf{f}_{(i-1)a}}{\partial t} \; .$$

Zwischen den Variablen $\mathbf{z}_{ia}$ bestehen die folgenden Beziehungen:

$$\begin{aligned} \mathbf{z}_1 &= \begin{bmatrix} \dot{\mathbf{x}} \\ \mathbf{w} \end{bmatrix} && ; \quad \mathbf{z}_{1a} \subseteq \mathbf{z}_1 \\ \mathbf{z}_2 &= \dot{\mathbf{x}} \cap \mathbf{z}_{1a} \cap \dot{\mathbf{z}}_{1a} && ; \quad \mathbf{z}_{2a} \subseteq \mathbf{z}_2 \\ &\cdots \\ \mathbf{z}_j &= \dot{\mathbf{x}} \cap \mathbf{z}_{(j-1)a} \cap \dot{\mathbf{z}}_{(j-1)a} \; . \end{aligned}$$

[2] Vergangene Werte der neu eingeführten Lagrange'schen Multiplikatoren müssen im Diskretisierungsschema den exakten Werte (= 0) erhalten.

[3] Der Algorithmus wurde entwickelt, um alle Gleichungen zu bestimmen, für die konsistente Anfangswerte der DAE erfüllt sein müssen. Dies ist äquivalent zur obigen Aufgabenstellung.

Dabei bedeutet z.B. "$\mathbf{z}_2 = \dot{\mathbf{x}} \cap \mathbf{z}_{1a} \cap \dot{\mathbf{z}}_{1a}$", daß sich $\mathbf{z}_2$ aus den Elementen der Vektoren $\dot{\mathbf{x}}, \mathbf{z}_{1a}, \dot{\mathbf{z}}_{1a}$ zusammensetzt. Elemente, die mehrmals auf der rechten Seite des Gleichheitszeichens auftreten, werden in $\mathbf{z}_2$ nur einmal aufgeführt[4]. Die Abkürzung "$\mathbf{z}_{2a} \subseteq \mathbf{z}_2$" besagt, daß die Variablen $\mathbf{z}_{2a}$ eine Untermenge der Variablen $\mathbf{z}_2$ sind. Mit den Abkürzungen

$$\begin{aligned} \mathbf{z} &= \mathbf{z}_1 \cap \mathbf{z}_2 \cap \ldots \cap \mathbf{z}_j \\ \mathbf{z}_a &= \mathbf{z}_{1a} \cap \mathbf{z}_{2a} \cap \ldots \cap \mathbf{z}_{(j-1)a} \qquad (\mathbf{z}_a \subseteq \mathbf{z}) \end{aligned}$$

kann das Gleichungssystem (A.3) kompakt geschrieben werden als

$$\dot{\mathbf{x}} = \mathbf{z}_0 \tag{A.4a}$$

$$\mathbf{0} = \mathbf{F}(t, \mathbf{x}, \mathbf{z}) = \begin{bmatrix} \mathbf{f}(t, \mathbf{x}, \mathbf{z}_1) \\ \mathbf{J}(t, \mathbf{x}, \mathbf{z}_a) \begin{bmatrix} \mathbf{z}_0 \\ \dot{\mathbf{z}}_a \end{bmatrix} + \mathbf{g}(t, \mathbf{x}, \mathbf{z}_a) \end{bmatrix} . \tag{A.4b}$$

Als Vorbereitung auf die nachfolgende Transformation, wurde $\dot{\mathbf{x}}$ überall durch $\mathbf{z}_0$ ersetzt und dafür die zusätzliche Identität (A.4a) eingeführt. Aufgrund der Konstruktion (A.3) ist sichergestellt, daß $\mathbf{J}$ in (A.4b) *zeilenregulär* ist. Sonst wäre das System nicht vom Index j, sondern von niedrigerem Index, da redundante Beziehungen zwischen den neu eingeführten Variablen $\mathbf{z}_i$ bestehen würden. Weiterhin ist durch den Pantelides Algorithmus garantiert, daß $\mathbf{z}$ aus $\mathbf{F}$ berechnet werden kann, d.h. daß $\partial\mathbf{F}/\partial\mathbf{z}$ spaltenregulär ist. Sonst hätte das System wiederum nicht den Index j.

(A.4b) ist ein überbestimmtes algebraisches Gleichungssystem in den unbekannten Variablen $\mathbf{z}$. Durch Einführen "dim($\mathbf{g}$)" zusätzlicher unbekannter Variablen $\boldsymbol{\mu}$ und "(dim($\mathbf{z}$) - dim($\mathbf{f}$))" zusätzlicher Gleichungen, kann die überbestimmte semi-explizite DAE (A.4) in die folgende DAE überführt werden:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_0 \\ \mathbf{0} \end{bmatrix} + \mathbf{J}^T(t, \mathbf{x}, \mathbf{z}_a)\,\boldsymbol{\mu} \tag{A.5a}$$

$$\mathbf{0} = \mathbf{F}(t, \mathbf{x}, \mathbf{z}) . \tag{A.5b}$$

Es wird jetzt der folgende, zentrale Satz bewiesen:

> *Jede Lösung der gegebenen DAE (2.18) von Seite 33 ist auch Lösung der Index-2 DAE (A.5) mit $\boldsymbol{\mu} = 0$. Umgekehrt führt jede Lösung von (A.5) auf $\boldsymbol{\mu} = 0$ und ist Lösung von (2.18).*

Der Satz wird auf ähnliche Weise bewiesen wie die Indexreduktionsmethode in [Gear88]. Der erste Teil des Satzes ist trivial. Wenn $\boldsymbol{\mu} = 0$ ist, liegt genau das Gleichungssystem vor, welches durch Konstruktion aus (2.18) gewonnen wurde. Damit erfüllt jede Lösung von (2.18) auch (A.5) mit $\boldsymbol{\mu} = 0$. Um den Index von (A.5) zu ermitteln, werden einige Gleichungen von (A.5b) (die genaue Gleichungsstruktur ist in (A.3) aufgeführt) *einmal*

[4] Wenn z.B. $\mathbf{z}_{1a}$ das Element $\dot{x}_1$ enthält, so tritt dieses auch in $\dot{\mathbf{x}}$ auf.

differenziert:

$$\mathbf{0} = \frac{d}{dt}\begin{bmatrix} \mathbf{f}_{1a} \\ \mathbf{f}_{2a} \\ \vdots \\ \mathbf{f}_{(j-1)a} \end{bmatrix}(t,\mathbf{x},\mathbf{z}) = \begin{bmatrix} \mathbf{J}_2\,(t,\mathbf{x},\mathbf{z}_{1a}) \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}}_{1a} \end{bmatrix} + \mathbf{g}_2\,(t,\mathbf{x},\mathbf{z}_{1a}) \\ \mathbf{J}_3\,(t,\mathbf{x},\mathbf{z}_{2a}) \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}}_{2a} \end{bmatrix} + \mathbf{g}_3\,(t,\mathbf{x},\mathbf{z}_{2a}) \\ \vdots \\ \mathbf{J}_j\,(t,\mathbf{x},\mathbf{z}_{(j-1)a}) \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}}_{(j-1)a} \end{bmatrix} + \mathbf{g}_j\,(t,\mathbf{x},\mathbf{z}_{(j-1)a}) \end{bmatrix} \tag{A.6}$$

oder kompakt geschrieben:

$$\mathbf{J}\,(t,\mathbf{x},\mathbf{z}_a) \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}}_a \end{bmatrix} + \mathbf{g}\,(t,\mathbf{x},\mathbf{z}_a) = \mathbf{0}\ . \tag{A.7}$$

Durch die Differentiation treten in Gleichung (A.7) die Variablen $\dot{\mathbf{x}}$ auf. Man beachte, daß in den entsprechenden Gleichungen von (A.5b) $\dot{\mathbf{x}}$ *nicht* auftritt, sondern $\mathbf{z}_0$! Die Ableitungen $\dot{\mathbf{x}}$ sind aber auch durch (A.5a) gegeben. Einsetzen von (A.5a) in (A.7) ergibt:

$$\mathbf{J}\,(t,\mathbf{x},\mathbf{z}_a) \begin{bmatrix} \mathbf{z}_0 \\ \dot{\mathbf{z}}_a \end{bmatrix} + \mathbf{g}\,(t,\mathbf{x},\mathbf{z}_a) + \mathbf{J}\,(t,\mathbf{x},\mathbf{z}_a)^T \mathbf{J}\,(t,\mathbf{x},\mathbf{z}_a)\,\boldsymbol{\mu} = \mathbf{0}\ .$$

Wegen (A.5b) verschwindet der erste Summenausdruck und es verbleibt die Gleichung

$$\mathbf{J}\,(t,\mathbf{x},\mathbf{z}_a)^T \mathbf{J}\,(t,\mathbf{x},\mathbf{z}_a)\,\boldsymbol{\mu} = \mathbf{0}\ .$$

Aufgrund seiner Konstruktion ist $\mathbf{J}$ zeilenregulär. Damit ist $\mathbf{J}\,\mathbf{J}^T$ regulär und $\boldsymbol{\mu} = 0$. Da $\partial\mathbf{F}/\partial\mathbf{z}$ spaltenregulär ist, kann $\mathbf{z}$, und damit auch $\mathbf{z}_0$, aus $\mathbf{F}$ ermittelt werden. Wegen (A.5a) und da $\boldsymbol{\mu} = 0$, kann dann auch noch die letzte Unbekannte $\dot{\mathbf{x}}$ bestimmt werden. Alle unbekannten Variablen können ermittelt werden, wenn Teile der DAE (A.5) *einmal* differenziert werden. Also hat die DAE (A.5) den Störungsindex 2. Nachdem die Lösung von (A.5) zu $\boldsymbol{\mu} = 0$ führt, erfüllt sie auch die gegebene DAE (2.18). QED.

A.3 Herleitung der Gleichungen für die Klasse Interact

In diesem Abschnitt werden die Beziehungen des Gleichungssystems (4.5) von Seite 75 hergeleitet. Die Herleitung kann sehr kompakt erfolgen, wenn die Gleichungen in koordinatensystem-unabhängiger Form dargestellt werden, und wenn z.B. die Euler'sche Differentiationsregel benutzt wird. Zur Demonstration werden die Gleichungen hier abgeleitet, indem nur die angegebenen Definitionsgleichungen benutzt werden.

Die Gleichungen (4.5a – 4.5e) ergeben sich direkt aus ihrer Definition. Die Gleichung (4.5f) kann in zwei Teile aufgeteilt werden, um ${}^b\boldsymbol{\omega}^{ab}$ und um ${}^b\mathbf{v}^{ab}$ zu bestimmen:

$$
\begin{aligned}
{}^b\boldsymbol{\omega}^{ab} &= {}^b\mathbf{T}^a\, {}^a\boldsymbol{\omega}^{ab} \\
&= {}^b\mathbf{T}^a\, \mathbf{vec}({}^b\dot{\mathbf{T}}^{a\,T}\, {}^b\mathbf{T}^a) \qquad \text{(wegen (4.3))} \\
&= {}^b\mathbf{T}^a\, \mathbf{vec}\left(\frac{d}{dt}\left({}^0\mathbf{T}^{b\,T}\, {}^0\mathbf{T}^a\right)^T\, {}^b\mathbf{T}^a\right) \qquad \text{(wegen (4.5a))} \\
&= {}^b\mathbf{T}^a\, \mathbf{vec}\left(\left({}^a\dot{\mathbf{T}}^0\, {}^0\mathbf{T}^b + {}^a\mathbf{T}^0\, {}^0\dot{\mathbf{T}}^b\right) {}^b\mathbf{T}^a\right) \\
&= {}^b\mathbf{T}^a\, \mathbf{vec}\left(\left({}^a\dot{\mathbf{T}}^0\, {}^0\mathbf{T}^a {}^0\mathbf{T}^{a\,T}\, {}^0\mathbf{T}^b + {}^a\mathbf{T}^0\, {}^0\mathbf{T}^b {}^0\mathbf{T}^{b\,T}\, {}^0\dot{\mathbf{T}}^b\right) {}^b\mathbf{T}^a\right) \\
&= {}^b\mathbf{T}^a\, \mathbf{vec}\left(\left(-\mathbf{skew}(\boldsymbol{\omega}^a)\, {}^a\mathbf{T}^b + {}^a\mathbf{T}^b\, \mathbf{skew}(\boldsymbol{\omega}^b)\right) {}^b\mathbf{T}^a\right) \qquad \text{(wegen (4.1))} \\
&= {}^b\mathbf{T}^a\, \mathbf{vec}\left(-\mathbf{skew}(\boldsymbol{\omega}^a) + \mathbf{skew}({}^a\boldsymbol{\omega}^b)\right) \\
&= {}^b\mathbf{T}^a\left(-\boldsymbol{\omega}^a + {}^a\boldsymbol{\omega}^b\right) \\
&= \boldsymbol{\omega}^b - {}^b\mathbf{T}^a\, \boldsymbol{\omega}^a
\end{aligned}
$$

$$
\begin{aligned}
{}^b\mathbf{v}^{ab} &= {}^b\mathbf{T}^a\, {}^a\mathbf{v}^{ab} \\
&= {}^b\mathbf{T}^a\, {}^a\dot{\mathbf{r}}^{ab} \qquad \text{(wegen (4.3))} \\
&= {}^b\mathbf{T}^a\, \frac{d}{dt}\left({}^0\mathbf{T}^{a\,T}\, \left({}^0\mathbf{r}^b - {}^0\mathbf{r}^a\right)\right) \qquad \text{(wegen (4.5b))} \\
&= {}^b\mathbf{T}^a\, \left({}^0\dot{\mathbf{T}}^{a\,T}\, {}^0\mathbf{T}^a\, {}^0\mathbf{T}^{a\,T}\, {}^0\mathbf{r}^{ab} + {}^0\mathbf{T}^{a\,T}\left({}^0\dot{\mathbf{r}}^b - {}^0\dot{\mathbf{r}}^a\right)\right) \\
&= {}^b\mathbf{T}^a\, {}^a\mathbf{T}^0\, {}^0\dot{\mathbf{r}}^b - {}^b\mathbf{T}^a\left(\mathbf{skew}(\boldsymbol{\omega}^a)\, {}^a\mathbf{r}^{ab} + \mathbf{v}^a\right) \qquad \text{(wegen (4.1))} \\
&= \mathbf{v}^b - {}^b\mathbf{T}^a\left(-\mathbf{skew}({}^a\mathbf{r}^{ab})\, \boldsymbol{\omega}^a + \mathbf{v}^a\right) \quad .
\end{aligned}
$$

Wenn die beiden Endgleichungen für ${}^b\boldsymbol{\omega}^{ab}$ und ${}^b\mathbf{v}^{ab}$ zusammengefaßt werden, ergibt sich (4.5f). Auf ganz entsprechende Weise kann (4.5g) hergeleitet werden. Gleichung (4.5h) folgt aus einer Kräfte- und Momenten-Bilanz unter der Annahme, daß ein Interact-Objekt masselos ist. Die Schnittkräfte und Schnittmomente an den beiden Cuts sind aufgeteilt in:

$$
\hat{\mathbf{f}}^a = \begin{bmatrix} \boldsymbol{\tau}^a \\ \mathbf{f}^a \end{bmatrix} , \qquad \hat{\mathbf{f}}^b = \begin{bmatrix} \boldsymbol{\tau}^b \\ \mathbf{f}^b \end{bmatrix} \quad .
$$

Wenn das Schnittmoment und die Schnittkraft des Cuts b in den Cut-Frame a transformiert werden und der Momentensatz um den Ursprung von Cut-Frame a angesetzt wird, sowie die Kräftesumme gebildet wird, ergibt sich:

$$
\mathbf{0} = \begin{bmatrix} \boldsymbol{\tau}^a \\ \mathbf{f}^a \end{bmatrix} + \begin{bmatrix} {}^b\mathbf{T}^{a\,T}\, \boldsymbol{\tau}^b + {}^a\mathbf{r}^{ab} \times \left({}^b\mathbf{T}^{a\,T}\, \mathbf{f}^b\right) \\ {}^b\mathbf{T}^{a\,T}\, \mathbf{f}^b \quad . \end{bmatrix}
$$

Mit der Abkürzung (4.6b) ergibt sich (4.5h).

A.4 Herleitung der Gleichungen für die Klasse LineForce

In diesem Abschnitt werden die Gleichungen (4.8) von Seite 79 hergeleitet.

Die Gleichungen (4.8a, 4.8b) sind die Definition für s und $\mathbf{n}$. s kann auch geschrieben werden als:

$$s \;=\; \left|{}^{b}\mathbf{r}^{ab}\right| \;=\; \left|{}^{a}\mathbf{r}^{ab}\right| \;=\; \sqrt{{}^{a}\mathbf{r}^{ab\,T}\,{}^{a}\mathbf{r}^{ab}} \;.$$

Die beiden zeitlichen Ableitungen von s ergeben sich dann zu:

$$\begin{aligned}
\dot{s} \;&=\; \frac{1}{2\sqrt{{}^{a}\mathbf{r}^{ab\,T}\,{}^{a}\mathbf{r}^{ab}}}\left({}^{a}\dot{\mathbf{r}}^{ab\,T}\,{}^{a}\mathbf{r}^{ab} \;+\; {}^{a}\mathbf{r}^{ab\,T}\,{}^{a}\dot{\mathbf{r}}^{ab}\right) \\
&=\; \frac{{}^{a}\dot{\mathbf{r}}^{ab\,T}\,{}^{a}\mathbf{r}^{ab}}{s} \\
&=\; {}^{a}\mathbf{n}^{T}\,{}^{a}\dot{\mathbf{r}}^{ab} \;=\; {}^{a}\mathbf{n}^{T}\,{}^{a}\mathbf{v}^{ab} \;=\; {}^{b}\mathbf{n}^{T}\,{}^{b}\mathbf{v}^{ab}
\end{aligned}$$

$$\begin{aligned}
\ddot{s} \;&=\; {}^{a}\mathbf{n}^{T}\,{}^{a}\dot{\mathbf{v}}^{ab} \;+\; {}^{a}\dot{\mathbf{n}}^{T}\,{}^{a}\mathbf{v}^{ab} \\
&=\; {}^{a}\mathbf{n}^{T}\,{}^{a}\mathbf{a}^{ab} \;+\; \frac{d}{dt}\left(\frac{{}^{a}\mathbf{r}^{ab}}{s}\right)^{T}{}^{a}\mathbf{v}^{ab} \\
&=\; {}^{a}\mathbf{n}^{T}\,{}^{a}\mathbf{a}^{ab} \;+\; \left(\frac{{}^{a}\dot{\mathbf{r}}^{ab}}{s} - \frac{{}^{a}\mathbf{r}^{ab}\,\dot{s}}{s^{2}}\right)^{T}{}^{a}\mathbf{v}^{ab} \\
&=\; {}^{a}\mathbf{n}^{T}\,{}^{a}\mathbf{a}^{ab} \;+\; \frac{1}{s}\cdot\left({}^{a}\mathbf{v}^{ab\,T}\,{}^{a}\mathbf{v}^{ab} \;-\; {}^{a}\mathbf{n}^{T}\,{}^{a}\mathbf{v}^{ab}\,\dot{s}\right) \\
&=\; {}^{b}\mathbf{n}^{T}\,{}^{b}\mathbf{a}^{ab} \;+\; \frac{1}{s}\cdot\left({}^{b}\mathbf{v}^{ab\,T}\,{}^{b}\mathbf{v}^{ab} \;-\; \dot{s}^{2}\right) \;.
\end{aligned}$$

A.5 Herleitung der Gleichungen für einen räumlich bewegten Antriebsstrang

In diesem Abschnitt werden die Gleichungen (5.3, 5.4) von Seite 145 hergeleitet.

Es werden zwei Starrkörper betrachtet. Der Starrkörper 1 ist rotationssymmetrisch und über ein Drehgelenk im Starrkörper 2 gelagert. Wenn beide Starrkörper freigeschnitten werden und der Impuls- und Drallsatz jeweils für den Massenmittelpunkt angeschrieben werden, so gilt mit (4.17) von Seite 90:

$$\hat{\mathbf{f}}^{1} \;=\; \begin{bmatrix} \boldsymbol{\tau}^{1} \\ \mathbf{f}^{1} \end{bmatrix} \;=\; \begin{bmatrix} \mathbf{I}^{1} & \mathbf{0} \\ \mathbf{0} & m^{1}\cdot\mathbf{E} \end{bmatrix}\begin{bmatrix} \boldsymbol{\alpha}^{1} \\ \mathbf{a}^{1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\omega}^{1}\times\mathbf{I}^{1}\boldsymbol{\omega}^{1} \\ \mathbf{0} \end{bmatrix} \tag{A.8a}$$

$$\hat{\mathbf{f}}^{2} \;=\; \begin{bmatrix} \boldsymbol{\tau}^{2} \\ \mathbf{f}^{2} \end{bmatrix} \;=\; \begin{bmatrix} \mathbf{I}^{2} & \mathbf{0} \\ \mathbf{0} & m^{2}\cdot\mathbf{E} \end{bmatrix}\begin{bmatrix} \boldsymbol{\alpha}^{2} \\ \mathbf{a}^{2} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\omega}^{2}\times\mathbf{I}^{2}\boldsymbol{\omega}^{2} \\ \mathbf{0} \end{bmatrix} \;. \tag{A.8b}$$

Hierbei ist $\mathbf{I}^1$ der Trägheitstensor des Körpers 1 bezüglich seines Massenmittelpunktes und $\mathbf{I}^2$ ist der Trägheitstensor des Körpers 2 bezüglich seines Massenmittelpunktes. Die kinematischen "1" Variablen sind die absolute Größen (z.B. ist $\boldsymbol{\alpha}^1$ die absolute Winkelbeschleunigung des Körpers 1). Gleiches gilt für die "2" Größen des Körpers 2. Alle Vektoren und Tensoren 2. Stufe werden in einem Koordinatensystem angeschrieben, welches fest mit Körper 2 verbunden ist. Der obige Ausdruck für den Impuls- und Drallsatz von Körper 1 ändert sich dadurch nicht, da der Körper rotationssymmetrisch ist, d.h. $\mathbf{T}^T\,\mathbf{I}^1\,\mathbf{T} = \mathbf{I}^1$. Wird jetzt die Kräfte- und Momentenbilanz für beide Starrkörper angeschrieben, so ergibt sich das resultierende Schnittmoment und die resultierende Schnittkraft $\hat{\mathbf{f}}^o$ zu:

$$\hat{\mathbf{f}}^o = \mathbf{C}_1^T\,\hat{\mathbf{f}}^1 \;-\; \mathbf{C}_a^T\,\hat{\mathbf{f}}^a \;-\; \mathbf{C}_b^T\,\hat{\mathbf{f}}^b \;+\; \mathbf{C}_2^T\,\hat{\mathbf{f}}^2 \quad . \tag{A.9}$$

Hierbei beschreiben die $\mathbf{C}_i$-Matrizen die Transformation zu einem gemeinsamen Punkt des Starrkörpers 2, nach (4.6b) von Seite 75. $\hat{\mathbf{f}}^a$ und $\hat{\mathbf{f}}^b$ sind die Schnittmomente und Schnittkräfte an den beiden Flanschen des Rotationskörpers 1 (siehe Bild 5.6 auf Seite 145).

Nach Gleichung (4.5f) und (4.5g) von Seite 75 kann die absolute Winkelbeschleunigung und die absolute Winkelgeschwindigkeit des Körpers 1 ausgedrückt werden durch die absoluten Größen des Körpers 2 und der Relativbewegung:

$$\boldsymbol{\alpha}^1 \;=\; \boldsymbol{\alpha}^2 \;+\; \mathbf{n}\cdot\ddot{q} \;+\; \boldsymbol{\omega}^2 \times \mathbf{n}\cdot\dot{q} \tag{A.10a}$$

$$\boldsymbol{\omega}^1 \;=\; \boldsymbol{\omega}^2 \;+\; \mathbf{n}\cdot\dot{q} \quad . \tag{A.10b}$$

Hierbei ist $\mathbf{n}$ ein Einheitsvektor, der in Richtung der Drehachse des Drehgelenks zeigt, mit dem der Körper 1 in Körper 2 gelagert ist und q ist der Drehwinkel des Gelenks. Einsetzen von (A.10) in (A.8a) und ausmultiplizieren ergibt:

$$\begin{aligned}\boldsymbol{\tau}^1 \;&=\; \mathbf{I}^1\,\left(\boldsymbol{\alpha}^2 + \mathbf{n}\cdot\ddot{q} + \boldsymbol{\omega}^2\times\mathbf{n}\cdot\dot{q}\right) \;+\; \left(\boldsymbol{\omega}^2 + \mathbf{n}\cdot\dot{q}\right)\times\mathbf{I}^1\,\left(\boldsymbol{\omega}^2 + \mathbf{n}\cdot\dot{q}\right)\\ &=\; \mathbf{I}^1\,\boldsymbol{\alpha}^2 \;+\; \mathbf{I}^1\,\mathbf{n}\cdot\ddot{q} \;+\; \mathbf{I}^1\,\left(\boldsymbol{\omega}^2 + \mathbf{n}\cdot\dot{q}\right) \;+\; \boldsymbol{\omega}^2\times\mathbf{I}^1\boldsymbol{\omega}^2 \;+\; \mathbf{n}\cdot\dot{q}\times\mathbf{I}^1\,\boldsymbol{\omega}^2\\ &\quad\; +\boldsymbol{\omega}^2\times\mathbf{I}^1\,\mathbf{n}\cdot\dot{q} \;+\; \mathbf{n}\cdot\dot{q}\times\mathbf{I}^1\,\mathbf{n}\cdot\dot{q} \quad .\end{aligned} \tag{A.11}$$

Einzelne Terme des Ausdrucks auf der rechten Seite können vereinfacht werden:

$$\mathbf{I}^1\,\mathbf{n} \;=\; J^1\cdot\mathbf{n} \tag{A.12}$$

$$\mathbf{n}\times\mathbf{I}^1\,\mathbf{n}\cdot\dot{q}^2 \;=\; \mathbf{n}\times\mathbf{n}\cdot J^1\cdot\dot{q}^2 \;=\; \mathbf{0} \tag{A.13}$$

$$\mathbf{I}^1\,\left(\boldsymbol{\omega}^2 + \mathbf{n}\cdot\dot{q}\right) \;+\; \mathbf{n}\cdot\dot{q}\times\mathbf{I}^1\,\boldsymbol{\omega}^2 \;=\; \mathbf{0} \quad . \tag{A.14}$$

Hierbei ist J^1 das Trägheitsmoment des Rotationskörpers 1 bezüglich seiner Drehachse $\mathbf{n}$. Der zweite Ausdruck (A.13) verschwindet, da das Kreuzprodukt von zwei gleichen Vektoren Null ist. Der dritte Ausdruck (A.14) ist auf Grund der Rotationssymmetrie von Körper 1 ebenfalls Null, wie durch explizites Ausrechnen in Koordinatenschreibweise verifiziert werden kann. Einsetzen dieser drei vereinfachten Ausdrücke in (A.11) und (A.11) in (A.8a) ergibt:

$$\hat{\mathbf{f}}^1 \;=\; \hat{\mathbf{f}}^{1a} \;+\; \hat{\mathbf{f}}^{1b} \tag{A.15a}$$

$$\hat{\mathbf{f}}^{1a} \;=\; \begin{bmatrix} \mathbf{I}^1 & \mathbf{0} \\ \mathbf{0} & m^1\cdot\mathbf{E} \end{bmatrix}\begin{bmatrix} \boldsymbol{\alpha}^2 \\ \mathbf{a}^1 \end{bmatrix} + \begin{bmatrix} \boldsymbol{\omega}^2\times\mathbf{I}^1\boldsymbol{\omega}^2 \\ \mathbf{0} \end{bmatrix} \tag{A.15b}$$

$$\hat{\mathbf{f}}^{1b} \;=\; \begin{bmatrix} \mathbf{n}\cdot J^1\ddot{q} \;+\; \boldsymbol{\omega}^2\times\mathbf{n}\cdot J^1\dot{q} \\ \mathbf{0} \end{bmatrix} \quad . \tag{A.15c}$$

Einsetzen von (A.15) in (A.9) führt zu:

$$\begin{aligned} \hat{\mathbf{f}}^o &= \mathbf{C}_1^T\,\hat{\mathbf{f}}^{1a} + \mathbf{C}_2^T\,\hat{\mathbf{f}}^2 + \mathbf{C}_1^T\,\hat{\mathbf{f}}^{1b} - \mathbf{C}_a^T\,\hat{\mathbf{f}}^a - \mathbf{C}_b^T\,\hat{\mathbf{f}}^b \\ &= \mathbf{C}_{12}^T\,\hat{\mathbf{f}}^{12} + \hat{\mathbf{f}}^{1b} - \mathbf{C}_a^T\,\hat{\mathbf{f}}^a - \mathbf{C}_b^T\,\hat{\mathbf{f}}^b \quad . \end{aligned} \tag{A.16}$$

Der Ausdruck $\mathbf{C}_{12}^T\,\hat{\mathbf{f}}^{12}$ ist der Impuls- und Drallsatz des Gesamtkörpers, der sich aus den Körpern 1 und 2 zusammensetzt. $\mathbf{C}_1^T\,\hat{\mathbf{f}}^{1b} = \hat{\mathbf{f}}^{1b}$, da $\hat{\mathbf{f}}^{1b}$ keine Translationskomponente hat. Die drei rechten Terme von (A.16) sind die gesuchten zusätzlichen dynamischen Anteile (5.3) des Rotationskörpers 1 zum Gesamtkörper, da sich Körper 1 relativ zu Körper 2 bewegt.

Gleichung (5.4) von Seite 145 ergibt sich bei Auswertung des d'Alembert'schen Prinzips (4.12) von Seite 83. Dieses besagt, daß am Drehgelenk die Projektion der Schnittkräfte und Schnittmomente auf die Rotationsachse $\mathbf{n}$ gleich dem im Gelenk wirkenden Drehmoment ist. Wenn kein eingeprägtes Drehmoment vorhanden ist, folgt bei Verwendung von (A.15):

$$\begin{aligned} \mathbf{0} &= \mathbf{n}^T\,\left(\boldsymbol{\tau}^1 - \boldsymbol{\tau}^a - \boldsymbol{\tau}^b\right) \\ &= \mathbf{n}^T\,\left(\mathbf{I}^1\,\boldsymbol{\alpha}^2 + \left(\boldsymbol{\omega}^2 \times \mathbf{I}^1\,\boldsymbol{\omega}^2\right) + \mathbf{n}\cdot J^1\ddot{q} + \boldsymbol{\omega}^2 \times \mathbf{n}\cdot J^1\dot{q} - \boldsymbol{\tau}^a - \boldsymbol{\tau}^b\right) \\ &= J^1\cdot\mathbf{n}^T\,\mathbf{I}^1 + \mathbf{n}^T\left(\boldsymbol{\omega}^2 \times \mathbf{I}^1\,\boldsymbol{\omega}^2\right) + \mathbf{n}^T\left(\boldsymbol{\omega}^2 \times \mathbf{n}\right)\cdot J^1\dot{q} - \tau^a - \tau^b \quad . \end{aligned} \tag{A.17}$$

Hierbei sind τ^a, τ^b die auf die Drehachse projizierten Schnittmomente an den Flanschen des Rotationskörpers 1. Der zweite und der vierte Ausdruck in (A.17) verschwinden, wie durch explizites Ausrechnen in Koordinantenschreibweise verifiziert werden kann. Damit folgt schließlich

$$J^1\ddot{q} = \tau^a + \tau^b - J^1\left(\mathbf{n}^T\,\boldsymbol{\alpha}^2\right) \quad ,$$

also Gleichung (5.4).

Anhang B

Dymola Syntax

In diesem Abschnitt wird die Syntax der objektorientierten Modellierungssprache Dymola, Version 1.9.5, in Form einer kontextfreien Grammatik (siehe z.B. [Aho88]) formal definiert und es wird kurz auf die Semantik der einzelnen Sprachelemente eingegangen. Dieser Abschnitt basiert auf [Elmq93].

Im folgenden kennzeichnen *kursiv* geschriebene Wörter nichtterminale Symbole, **fett** geschriebene Wörter kennzeichnen terminale Symbole, der senkrechte Strich "|" kennzeichnet Alternativen, eckige Klammern "[]" kennzeichnen eine optionale Produktion und geschweifte Klammern "{}" kennzeichnen Null oder beliebig viele Wiederholungen der geklammerten Produktion.

Lexikalische Grundelemente

ident → **letter** { **letter** | **digit** }
unsigned → **digit** { **digit** } ['.' { **digit** }] ['**E**' ['+' | '−'] **digit** { **digit** }]
number → ['+' | '−'] *unsigned*
Boolean → **True** | **true** | **False** | **false**

Ein "Identifier" *ident* besteht aus beliebig vielen Buchstaben und Ziffern. Es wird zwischen Groß- und Kleinbuchstaben unterschieden, d.h. die beiden Namen `Potential` bzw. `potential` kennzeichnen unterschiedliche Variablen. Die Syntax einer Zahl, *number*, folgt der in Programmiersprachen wie C oder Fortran üblichen Notation von Gleitpunktzahlen. Gültige Zahlen sind z.B. `88, 10.5, 1E6, -1.443E-11`. Dymola kennt zur Zeit als Datentyp nur Gleitpunktzahlen und Boole'sche Werte, jedoch z.B. keine ganzen Zahlen. Als Kommentar werden in Dymola alle Zeichenketten aufgefaßt, die sich in geschweiften Klammern "{}" befinden.

Hierarchische Modellstruktur

ModelSpecification → { *ModelDeclaration* }
ModelDeclaration → **model** [*InheritClause*] *model_ident* | (**type** | **class**)
[*InheritClause*] *class_ident*
Submodels
Declarations

```
                    { Equation | Connection }
                end
InheritClause   → '(' class_ident ')'
Submodels       → { ModelDeclaration | Invocations }
Invocations     → submodel [ '(' class_ident ')' ] Invocation { [,] Invocation }
Invocation      → model_ident [ '(' ParamItem { [,] ParamItem } ')' ]
ParamItem       → ( ident '=' ([−] unsigned) | variable ) | ([−] unsigned)
model_ident     → ident
class_ident     → ident
variable        → ident
```

Die Definition eines Modells besteht aus Definitionen von Modellklassen, gefolgt von der Definition *eines* Modells. Wenn eine Modell-Deklaration einen *InheritClause* enthält, wird der Modellrumpf von der entsprechenden Oberklasse in das Modell einkopiert. Ein Modellrumpf besteht aus *Submodel* Deklarationen, gefolgt von Deklarationen benötigter Variablen (*Declarations*). Danach folgt die Beschreibung der Verbindungsstruktur der Objekte (*Connection*). Abgeschlossen wird ein Modellrumpf mit den Gleichungen, die das Modell beschreiben. Mit einer *Submodel* Deklaration werden Objekte als Instanzen einer Klasse definiert. Beispiel für eine Modellstruktur:

```
model class mc1
end

model class mc2
end

model class (mc2) mc3                          {mc3 ist Subklasse von mc2}
    submodel (mc1) mm1, mm2(p1=5,p2=7)         {mm1 erhält die Voreinstellung der Parameter
                                                 mm2 erhält modifizierte Parameterwerte}
end

model m
    submodel mc1 (p2=4)                        {Objekt hat denselben Namen wie seine Klasse}
    submodel (mc3) a b                         {zwei Objekte der Klasse mc3}
    model mm                                   {kombinierte Definition und Instanz}
    end
end
```

Modelle werden über einen hierarchischen Namen angesprochen, der sich aus einer Aneinander-Reihung der entsprechenden Modellnamen der Hierarchie ergibt. Die Modellnamen werden durch "::" voneinander getrennt. Durch die obige Modellbeschreibung werden die folgenden 9 Instanzen erzeugt: m, m::mc1, m::a, m::a::mm1, m::a::mm2, m::b, m::b::mm1, m::b::mm2, m::mm.

Deklarationen

```
Declarations    → { VarDeclaration | CutDeclaration | NodeDeclaration | PathDeclaration }
VarDeclaration  → parameter | constant | local [ VarItem { [,] VarItem } ] |
                  terminal | input | output ) [ variable { [,] variable } ]
VarItem         → variable [ '=' number | Boolean ]
CutDeclaration  → [ main ] cut [ cut_ident [ Cut ] { [,] cut_ident [ Cut ] } ]
Cut             → '(' [ CutList ] ')' |
                  '[' { Cut | '.' } ']'
CutList         → CutItem { [,] CutItem } [ '/' CutItem { [,] CutItem } ]
CutItem         → ['−'] variable | '.'
```

NodeDeclaration → **node** [*node_ident* [*Cut*] { [,] *node_ident* [*Cut*] }]
PathDeclaration → [**main**] **path** [*PathItem* { [,] *PathItem* }]
PathItem → *path_ident* '<' (*Cut* | '.') '–' (*Cut* | '.') '>'
variable → *ident*
node_ident → *ident*
path_ident → *ident*
cut_ident → *ident*

Alle in einem Modell verwendeten Variablen müssen deklariert werden. Konstante Größen werden als **parameter** oder als **constant** Variablen deklariert. Der Unterschied zwischen den beiden Typen besteht im wesentlichen darin, daß **parameter** Variablen normalerweise zur Laufzeit vor einem Integrationsstart noch geändert werden können, im Gegensatz zu **constant** Größen. Lokale Modellvariablen werden mit **local** deklariert. Variable, auf die von außerhalb eines Modells zugegriffen werden soll, müssen den Typ **terminal**, **input** oder **output** erhalten. Beim Typ **terminal** ist nicht festgelegt, ob die Größe eine Ein- oder eine Ausgangsvariable ist. Es gibt die vordefinierte lokale Variable *Time*, die den aktuellen Zeitpunkt charakterisiert. Diese Variable darf nicht deklariert werden und wird in den Gleichungen genauso benutzt wie jede andere Variable auch.

Deklarierte **terminal**, **input** oder **output** Variablen können zu verschieden Gruppen zusammengefaßt werden. Eine Gruppe wird als **cut** bezeichnet. Eine Variable kann in mehreren Cuts benutzt werden. Die Variablen eines Cuts können von außerhalb des Modells mit dem Cutnamen *cut_ident* gemeinsam angesprochen werden (z.B. in einer Verschaltung). Ein Cut kann hierarchisch aufgebaut sein, also aus schon definierten Cuts bestehen. In diesem Fall werden "eckige" Klammern, statt "runder" Klammern für die Cut-Definition benutzt. Genau *ein* Cut kann als **main cut** deklariert werden. Wenn ein Cut-Name bei einer Verschaltung nicht explizit angegeben werden, so wird der **main cut** verwendet. Abkürzend kann die **terminal** Vereinbarung für eine Variable auch weggelassen werden, wenn diese Variable in einer **cut**-Deklaration benutzt wird. Alle Variablen links vom optionalen Schrägstrich in einer Cut-Deklaration sind "Across"-Variablen. Alle Variablen rechts vom Schrägstrich sind "Through"-Variablen.

Mit **node** Deklarationen können "Verschaltungspunkte" eingeführt werden, die einen Namen besitzen. Hierdurch kann die Verschaltung von Objekten vereinfacht bzw. übersichtlicher gemacht werden. Die **path** Deklaration legt einen "Pfad" von einem Cut A zu einem Cut B fest. Auch hierdurch kann die Verschaltungsdefinition vereinfacht werden (siehe unten). Genau *ein* Pfad kann als **main path** deklariert werden. Wenn der Name eines Pfads in einer Verschaltung nicht explizit verwendet wird, so wird der **main path** benutzt.

Beispiele für gültige Deklarationen:

parameter p1 p2=5, p3=5E-3
local x=1, y z
cut c1 (v1 . v2 / v3, -v4), c2(u)
main cut c2 [c1, (v5,f6) [(v7 v8) (v1/v3)]]
path p1 < c1 – c2 >
main path < (v9,v10) – [(v11/v12) (v13)] >

Verschaltung von Objekten

Connection → **connect** ['<' *ident* '>'] [*ConnectionExpr* { [,] *ConnectionExpr* }]
ConnectionExpr → *ConnectionSecondary* { (**at** | '=' | **to** | '−' | **from** | **par** | '//' | **loop**) *ConnectionSecondary* }
ConnectionSecondary → [**reversed** | '\'] *ConnectionPrimary*
ConnectionPrimary → *ConnectionOperand* | '(' { *ConnectionExpr* | '.' } ')'
ConnectionOperand → *cut_ident* | *model_ident* ':' *cut_ident* | *node_ident* | *path_ident* | *model_ident* '..' *path_ident*

Mit der **connect** Anweisung werden Objekte an ihren Cuts miteinander verbunden. Im wesentlichen muß angegeben werden, welcher Cut von welchem Objekt mit den Cuts von anderen Objekten verbunden wird. Hierzu gibt es eine Anzahl von Möglichkeiten, die in den folgenden Beispielen verdeutlicht werden:

connect m1:cut1 **at** m2:cut2
connect m1 **at** m2 {m1, m2 müssen einen **main cut** besitzen}
connect m1 **at** (n1 n2, m2:cut2)
connect m1..path1 **at** m2..path2
connect m1 **to** m2 **to** m3 {m1, m2 müssen einen **main path** besitzen}
connect common − (V0 // (C − R1) // (L − R2))

Ein Cut *cut1* in einem Modell *m1* wird durch *m1:cut1* charakterisiert. Die Operatoren "−" und **to** kennzeichnen eine Reihenschaltung von Objekten, entlang der definierten "Pfade". Die Operatoren "//" und **par** kennzeichnen eine Parallelschaltung von Objekten, ebenfalls entlang der definierten "Pfade". Die Operatoren "\" und **reversed** vertauschen die beiden Cuts eines Pfades.

Modellgleichungen

Equation → *Expression* '=' *Expression* |
when *Expression* **then**
{ *Expression* '=' *Expression* }
endwhen
Expression → *SimpleExpr* | **if** *Expression* **then** *SimpleExpr* **else** *Expression*
SimpleExpr → *LogicalTerm* { **or** *LogicalTerm* }
LogicalTerm → *LogicalFactor* { **and** *LogicalFactor* }
LogicalFactor → [**not**] *Relation*
Relation → *ArithmeticExpr* [('<' | '>') *ArithmeticExpr*]
ArithmeticExpr → ['+' | '−'] *term* { ('+' | '−') *term* }
term → *factor* { ('*' | '/') *factor* }
factor → *primary* ['**' *primary*]
primary → '(' *Expression* ')' | *unsigned* | *variable* | *model_ident*'.'*variable* |
der '(' *variable* ')' | **new** '(' *variable* ')' | **init** '(' *variable* ')' |
function_ident '(' *Expression* { ',' *Expression* } ')'

Eine Gleichung wird aus den üblichen logischen und arithmetischen Ausdrücken aufgebaut. Ein Ausdruck kann auch eine **if**-Anweisung sein. Standardmäßig wird ein **if**-Ausdruck in eine entsprechende **if**-Anweisung der Zielsprache übersetzt. Wenn die Option "*set Events on*" gesetzt ist, übersetzt Dymola einen **if**-Ausdruck in Zustands-Ereignisse, d.h. nur an einem Ereignispunkt wird von einem Zweig des **if**-Ausdrucks in den anderen geschaltet. Die Gleichungen in einer **when**-Anweisung werden nur an Ereignispunkten ausgeführt, wenn

die *Expression* im **when**-Kopf wahr *wird*. Der Operator **der** kennzeichnet die Zeitableitung einer Variablen, der Operator **new** kennzeichnet den nächsten Werte einer diskreten oder Boole'schen Variablen und der Operator **init** kennzeichnet den Anfangswert einer kontinuierlichen Zustandsvariablen beim Start nach einem Ereignis bzw. beim Start der Integration.

Dymola kennt die Funktionen `sqrt`, `exp`, `ln`, `sin`, `cos`, `tan`, `arcsin`, `arccos`, `arctan`, `arctan2`, `sinh`, `cosh`, `tanh` und wandelt die Namen dieser Funktionen bei der Codeerzeugung in die entsprechenden Funktionsnamen der Zielsprache um. Zum Beispiel wird der Funktionsname `arctan2` bei den Zielsprachen ACSL oder Fortran-DSBlock in den Fortran Intrinsic Funktionsnamen `ATAN2` umgewandelt. Alle anderen Funktionsnamen werden ohne Veränderung in die Zielsprache kopiert. Hierbei wird angenommen, daß der Benutzer eine entsprechende Funktion in der Zielsprache bereitstellt.

Beispiele für zulässige Gleichungen:

```
C*der(u) = i1 + i2
t1       = (f1x*sin(phi) - f1y*cos(phi))*L1
u        = if Time > 1 and Time < 2 then 1 else 2*Time
when Time > NextTime then
   new(NextTime) = NextTime + SampleTime
   new(x) = a*x + b*u
endwhen
```

Anhang C

Klassenhierarchien

In diesem Abschnitt sind die Klassenhierarchien der in dieser Arbeit erläuterten Klassen angegeben, zusammen mit einer Referenz auf die Definition der Klasse. Die Klassenhierarchien werden als "Baum" dargestellt. Z.B. ist die Klasse *Resistor* eine Unterklasse der Klasse *TwoPin* (siehe unten).

Elektrische Schaltungselemente

Klassenhierarchie	**Seite**	**Kurzbeschreibung**
TwoPin	18	Element mit zwei elektrischen Pins
Capacitor	18	ideale Kapazität
OpAmp	172	idealer Operationsverstärker
Resistor	18	idealer Ohm'scher Widerstand
Switch	54,55,60	idealer elektrischer Schalter
Diode	58,60,63	ideale Diode

Mehrkörpersysteme

Klassenhierarchie	**Seite**	**Kurzbeschreibung**
MbsOneCut	73	Element mit einer mechanischen Schnittstelle
Body	92	Starrkörper (Trägheitseigenschaften)
Inertial	94	Inertialsystem
MbsTwoCut	74	Element mit zwei mechanischen Schnittstellen
Interact	74–76,189	Verbindungselement (masselos)
Force	78	Kraftelement
DirForce	78	Richtungskraft
LineForce	79,191	Linienkraft
Spring	79	Feder
Joint	80–85	Gelenk
Bar	86	(masselose) Stange
CylinderS	89	Zylindergelenk
PrismaticBase	88	Translationsgelenk (Basisklasse)
RevoluteBase	86	Drehgelenk (Basisklasse)
RevoluteBase2	87	Drehgelenk (Basisklasse 2)
Revolute	87	Drehgelenk (rheonom)
RevoluteS	87	Drehgelenk mit Zuständen

RevoluteL	87	blockierbares Drehgelenk (rheonom)
RevoluteLS	87	blockierbares Drehgelenk mit Zuständen
SphereCut	89	Kugel-Schnittgelenk
Sensor	92	Beobachtungsgröße
LineSensor	92	Linien-Sensor
ExtForce	80	externes Kraftelement
ExtSpring	80	Feder als externes Kraftelement
ExtForceL	88	externes Kraftelement das "Blockieren" kann
ExtBrake	143	Reibbremse
ExtFriction	120	Reibung

Antriebsstränge

Klassenhierarchie	Seite	Kurzbeschreibung
DriveMbsBase	148	Antriebsstrangbasis auf MKS
DriveOneCut	132,147	Element mit einem Flansch
DriveBase	133	Antriebsstrangbasis
DriveExtBase	134	externe Antriebsstrangbasis
DriveTwoCut	132	Element mit zwei Flanschen
DriveExt	134	Kopplung eines externen Kraftgesetzes
DriveForce	134	Kraftelement
DriveSpring	134	Feder
Gear	134	Getriebe (trägheitslos)
Rotor	133	Rotor (trägheitsbehaftet)
Shaft	133,148	Welle (trägheitsbehaftet)
DriveVar	135	Relativgrößen
DriveVarS	136	Relativgrößen als Zustände
DriveVarWS	136	nur Winkelgeschwindigkeit als Zustand
DriveVarL	135	Relativgröße (blockierbar)
DriveVarLS	136	Relativgrößen als Zustände (blockierbar)
DriveVarLWS	136	nur Winkelgeschwindigkeit als Zustand (blockierbar)

Regelungstechnische Blockschaltbilder

Klassenhierarchie	Seite	Kurzbeschreibung
SISO	158	Single-Input-Single-Output Block
Coeff_x	159	Übertragungsfunktion in Koeffizientendarstellung
PID	159	PID-Regler
PIDbound	161	PID-Regler mit Antiwindup-Kompensation
PoleZero	159	Übertragungsfunktion in Pol/Nullstellen-Darstellung
Sum_x	162	Summierer
Discrete	160	Abtastsystem
dSISO	160	diskreter Single-Input-Single-Output Block
dCoeff_x	160	z-Übertragungsfunktion in Koeffizientendarstellung
PeriodicSignal	162	periodisches Zeitsignal
Pulse	163	Puls-Signal
Sawtooth	163	Sägezahn-Signal
SquareWave	163	Rechteck-Signal

Index

Literaturverzeichnis

[Aho88] Aho A.V., Sethi R. und Ullman J.D.: Compilerbau, Teil 1. Addison-Wesley, 1988.

[Anan93] Anantharaman M.: Flexible Multibody Dynamics – An Object-Oriented Approach. Proceedings of the NATO ASI on Computer Aided Analysis of Rigid and Flexible Mechanical Systems, Vol. II, S. 383–402, Tróia, Portugal, 27. Juni – 9. Juli, 1993.

[Ande90] Andersson M.: Omola – An Object-Oriented Language for Model Representation. Licenciate thesis TFRT-3208, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1990.

[Ande92] Andersson M.: Discrete Event Modelling and Simulation in Omola. Proceedings of IEEE Symposium on Computer-Aided Control System Design, S. 262–268, Napa, California, 17. – 19. März, 1992

[Andr90] Andrzejewski T., Eich E., Führer C., Leister G. und Otter. M.: Entwurf von Schnittstellen zur numerischen Integration von Mehrkörpersystemen. Technical Report TR R30–90, DLR, Institut für Robotik und Systemdynamik, Postfach 11 16, D–82234 Wessling, Okt. 1990.

[Arms79] Armstrong W.: Recursive Solution to the Equations of Motion of an n-link Manipulator. In Proc. of the Fifth World Congress on Theory of Machines and Mechanisms, 1979.

[Arms91] Armstrong-Hélouvry B.: Control of Machines with Friction. Kluwer Academic Publishers, 1991.

[Astr90] Åström K.J. und Wittenmark B.: Computer-Controlled Systems. Theory and Design. Prentice-Hall, Second Edition, 1990.

[Bae87] Bae D.S. und Haug J.: A recursive formulation for constrained mechanical system dynamics: Part I. Open loop systems. Mech. Struct. Mach., Vol. 15, S. 359–382, 1987.

[Bree84] Breedveld P.C.: Physical Systems Theory in Terms of Bond Graphs. Dissertation., ISBN 90-9000599-4, Enschede, Holland, 1984.

[Easy88] Boeing Computer Services: EASY5/W – User's Manual. Engineering Technology Applications (ETA) Division, Seattle, Wash, 1988.

[Bos86] Bos A.M.: Modelling Multibody Systems in Terms of Multibond Graphs with application to a motorcycle. Dissertation, Alphen a/d Rijn, Holland, ISBN 90-9001442-X, 1986.

[Bran86] Brandl H., Johanni R., und Otter M.: A very efficient algorithm for the simulation of robots and similar multibody–systems without inversion of the mass–matrix. In: "Theory of Robots. Selected Papers from the IFAC/IFIP/IMACS Symposium, Vienna/Austria, 3.–5. Dec. 1986", edited by P. Kopacek, I. Troch, K. Desoyer, pp. 95–100, Pergamon Press, 1988.

[Bren89] Brenan K.E., Campbell S.L. und Petzold L.R.: Numerical Solution of Initial–Value Problems in Differential–Algebraic Equations. Elsevier Science Publishers, 1989.

[Cann67] Cannon R.H.: Dynamics of Physical Systems. McGraw-Hill, New York, 1967.

[Canu93] Canudas de Wit C., Olsson H., Åström K.J. und Lischinsky P.: A New Model for Control of Systems with Friction. International Conference on Control Theory and its Application, Kibbutz Maab Hachamisha, Israel, Okt. 1993.

[Cell79] Cellier F.E.: Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools. Ph.D. dissertation, Diss ETH No 6483, ETH Zürich, CH–8092 Zürich, Schweiz, 1979.

[Cell91] Cellier F.E.: Continuous System Modeling. Springer-Verlag, New York, 1991.

[Cell92] Cellier F.E. und Elmqvist H.: The Need for Automated Formula Manipulation in Object-Oriented Continuous-System Modeling. IEEE Symposium on Computer-Aided Control System Design, CACSD'92, Napa, California, März 1992.

[Cell93] Cellier F.E. und Elmqvist H.: Automated Formula Manipulation Supports Object-Oriented Continuous-System Modeling. IEEE Control Systems Magazine, Vol. 13, S. 28-38, 1993.

[Cell93a] Cellier F.E., Elmqvist H., Otter M. und Taylor J.H.: Guidelines for Modeling and Simulation of Hybrid Systems. Proceedings of IFAC World Congress, Vol. 8, S. 391-397, Sydney, Australia, Juli 1993.

[Chua87] Chua L.O., Desoer C.A. und Kuh E.S.: Linear and Nonlinear Circuits. McGraw-Hill, 1987.

[Detz86] Detzner P.: Entwurf und Simulation der Bahnregelung des Roboters manutec r3. Diplomarbeit, Lehrstuhl für elektrische Antriebstechnik, Technische Universität München, Dez. 1986, und Technical Report IB 515-87/1, DLR, Institut für Robotik und Systemdynamik, Postfach 11 16, D–82234 Wessling, 1987.

[Dong79] Dongarra J.J., Moler C.B., Bunch J.R. und Stewart G.W.: LINPACK, User's Guide. SIAM, Philadelphia, 1979.

[Duff86] Duff I.S., Erismann A.M. und Reid J.K.: Direct Methods for Sparse Matrices. Oxford Science Publications, 1986.

[Eich90] Eich E., Führer C., Leimkuhler B. und Reich S.: Stabilization and Projection Methods for Multibody Dynamics. Research Report A281, Institut of Mathematics, Helsinki University of Technology, Otakaari 1, SF–02150 Espoo, Finland, August 1990.

[Eich92] Eich E.: Projizierende Mehrschrittverfahren zur numerischen Lösung von Bewegungsgleichungen technischer Mehrkörpersysteme mit Zwangsbedingungen und Unstetigkeiten. Dissertation, VDI-Fortschritt-Berichte, Reihe 18, Nr. 109, VDI-Verlag, Düsseldorf, 1992.

[Eich86] Eichberger A., Jaschinski A., Otter M. und Türk S.: Körperdaten des Roboters mantec r3. Technical Report IB 515-86/3, DLR, Institut für Robotik und Systemdynamik, Postfach 11 16, D–82234 Wessling, Feb. 1986.

[Elmq75] Elmqvist, H.: Simnon – An Interactive Simulation Program for Nonlinear Systems – User's Manual. M.S. thesis, Report CODEN:LUTFD2/(TFRT–7502), Department of Automatic Control, Lund Institute of Technology, Lund Sweden, 1975.

[Elmq78] Elmqvist H.: A Structured Model Language for Large Continuous Systems. Dissertation. Report CODEN:LUTFD2/(TFRT–1015), Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1978.

[Elmq90] Elmqvist H., Åström K.J., Schönthal T. und Wittenmark B.: *Simnon — User's Guide for MS-DOS Computers.* SSPA Systems, Gothenburg, Sweden, 1990.

[Elmq92] Elmqvist H.: An Object and Data-Flow based Visual Language for Process Control. ISA/92-Canada Conference & Exhibit, Toronto, Canada, April 1992.

[Elmq92a] Elmqvist H.: Dymola Application Note – Mechanical Systems. Draft, Dynasim AB, Research Park Ideon, Lund, Sweden, Mai 1993.

[Elmq92b] Elmqvist H.: Proposal – Discrete Event Models in Dymola. Electronic Mail an F.E. Cellier und M. Otter, 24. Okt. 1992.

[Elmq93] Elmqvist H.: Dymola — User's Manual. Dynasim AB, Research Park Ideon, Lund, Sweden, 1993.

[Elmq93a] Elmqvist H.: Object-Oriented Modeling and Automatic Formula Manipulation in Dymola. Scandinavian Simulation Society SIMS'93, Kongsberg, Norway, Juni 1993.

[Elmq93b] Elmqvist H., Cellier F.E. und Otter M.: Object–Oriented Modeling of Hybrid Systems. Proceedings ESS'93, European Simulation Symposium, S. xxxi-xli, Delft, Niederlande, Okt. 1993.

[Fahr91] Fahrenthold E.P. und Wargo J.D.: Vector Bond Graph Analysis of Mechanical Systems. ASME – Journal of Dynamic Systems, Measurement and Control, S. 344–353. Sept. 1991.

[Falk75] Falk G. und Ruppel W.: Mechanik, Relativität, Gravitation. Springer-Verlag, 1975, 2. Auflage.

[Feat83] Featherstone R.: The Calculation of Robot Dynamics Using Articulated-Body Inertias. The International Journal of Robotics Research, Vol. 2, S. 13–30, 1983.

[Fran92] Franke J.: Die DLR-Robotermodelle 3 und 4 des geregelten Manutec r3. Technical Report TR R76–92, DLR, Institut für Robotik und Systemdynamik, Postfach 11 16, D–82234 Wessling, 1992.

[Fran93] Franke J. und Otter M.: The Manutec r3 Benchmark Models for the Dynamic Simulation of Robots. Technical Report TR R101-93, DLR, Institut für Robotik und Systemdynamik, Postfach 11 16, D–82234 Wessling, März 1993.

[Fueh88] Führer C.: Differential-algebraische Gleichungssysteme in mechanischen Mehrkörpersystemen. Dissertation, Mathematisches Institut, Technische Universität München, 1988.

[Fueh91] Führer C. und Leimkuhler B.J.: Numerical solution of differential-algebraic equations for constrained mechanical motion. Numerische Mathematik, No. 59, S. 55-69, 1991.

[Gant59] Gantmacher F.R.: Matrizenrechnung, Teil 2. VEB Deutscher Verlag der Wissenschaften, Berlin, 1959. Sowie: Matrizentheorie, Nachdruck der 2. Auflage, Springer-Verlag, 1986.

[Gaus91] Gaus N. und Otter M.: Dynamic Simulation in Concurrent Control Engineering. IFAC Symposium on Computer Aided Design in Control Systems, Swansea, UK, Preprints S. 123-126, 15-17 July, 1991.

[Gear86] Gear C.W.: Maintaining solution invariants in the numerical solution of ODEs. SIAM Journal of Scientific and Statistical Computing, Vol. 7, S. 734–743, Juli 1986.

[Gear88] Gear C.W.: Differential-Algebraic Equation Index Transformations. SIAM Journal on Scientific and Statistical Computing, Vol. 9, S. 39–47, Jan. 1988.

[Gear90] Gear C.W.: Differential Algebraic Equations, Indices and Integral Algebraic Equations. SIAM Journal on Numerical Analysis, Vol, 27. S. 1527–1534, Dez. 1990.

[Gloc93] Glocker C. und Pfeiffer F.: Complementary problems in multibody systems with planar friction. Accepted for publication in "Applied Mechanics", erscheint demnächst.

[Gloc93a] Glocker C. und Pfeiffer F.: Stick-Slip Phenomena and Application. Proceedings of: Nonlinearity & Chaos in Engineering Dynamics, IUTAM Symposium, London, Juli 1993.

[Grue91] Grübel G. und Joos H.-D.: RASP and RSYST – Two Complementary Program Libraries for Concurrent Control Engineering. IFAC Symposium on Computer Aided Design in Control Systems, Swansea, UK, Preprints S. 101-106, 15-17 July, 1991.

[Grue93] Grübel G., Joos H.-D., Otter M. und Finsterwalder R.: The ANDECS Design Environment for Control Engineering. IFAC World Congress, Sydney, Australien, Juli 1993.

[Haug86] Haug E.J., Wu S.C. und Yang S.M.: Dynamics of Mechanical Systems with Coulomb Friction, Stiction, Impact and Constraint Addition–Deletion – Part I, Theory. Mechanism and Machine Theory, Vol. 21, S. 401–506, 1986.

[Hair89] Hairer E., Lubich C. und Roche M.: The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods. Lecture Notes in Mathematics, No. 1409, Springer Verlag, 1989.

[Hill92] Hiller M., Kecskeméthy A. und Stelzle W.: Ein Vergleich rekursiver Verfahren. Interner Bericht, Universität Duisburg, Fachgebiet Mechatronik & Mechanik. Eine Kurzfassung dieses Berichts erschien in: Stelzle W.: Ein Vergleich rekursiver Verfahren zur Untersuchung der Dynamik baumstrukturierter Mehrkörpersysteme ZAMM 73, S. 107 – 109, 1993.

[Hill93] Hiller M.: Dynamics of Multibody Systems with Minimal Coordinates. Proceedings of the NATO ASI on Computer Aided Analysis of Rigid and Flexible Mechanical Systems, Vol. I, S. 119–164, Tróia, Portugal, 27. Juni – 9. Juli, 1993.

[Hind83] Hindmarsh A.C.: ODEPACK, a systematized collection of ODE solvers. Scientific Computing, edited by R.S. Stepleman et. al., North-Holland, Amsterdam, 1983.

[Hock93] Hocke M., Rühle R. und Otter M.: A Software Environment for Analysis and Design of Multibody Systems. Advanced Multibody System Dynamics, edited by W. Schiehlen, S. 67–86, Kluwer Academic Publishers, 1993.

[Hoef85] Hoefer E.E.E. und Nielinger H.: SPICE – Analyseprogramm für elektronische Schaltungen. Springer-Verlag, 1985.

[Karn90] Karnopp D.C., Margolis D.L. und Rosenberg R.C.: System Dynamics: A Unified Approach. John Wiley, 2^{nd} edition, 1990.

[Kecs88] Kecskeméthy A.: Object-Oriented Modeling of a Spatial Mechanism. Mobile Input-File. Universität Duisburg, Fachgebiet Mechatronik, 1988.

[Kecs93] Kecskeméthy A.: Objektorientierte Modellierung der Dynamik von Mehrkörpersystemen mit Hilfe von Übertragungselementen. Dissertation, VDI Fortschritt-Berichte, Reihe 20, Nr. 88, 1993.

[Kecs93a] Kecskeméthy A. und Hillller M.: An Object-Oriented Approach for an Effective Formulation of Multibody Dynamics. Second U.S. National Congress on Computational Mechanics, Washington, U.S.A., 16.–18. August, 1993.

[Kecs93b] Kecskeméthy A.: Mobile – An Object-Oriented Tool-Set for the Efficient Modeling of Mechatronic Systems. Proceedings of the Second Conference on Mechatronics and Robotics, S. 447-462, Duisburg/Moers, 27.–29. Sept., 1993,

[Korn89] Korn G.A.: Interactive Dynamic-System Simulation. McGraw-Hill, 1989.

[Kreu90] Kreuzer E. and Schiehlen W.: NEWEUL – Software for the Generation of Symbolical Equations of Motion. Multibody Systems Handbook, edited by W. Schiehlen, Springer-Verlag, 1990.

[Kron63] Kron G.: Diakoptics – The piecewise Solution of Large-Scale Systems. MacDonald & Co., London, 1962.

[Lasc88] Laschet A.: Simulation von Antriebssystemen. Springer-Verlag, 1988.

[Leis92] Leister G.: Beschreibung und Simulation von Mehrkörpersystemen mit geschlossenen kinematischen Schleifen. Dissertation, VDI Fortschritt-Berichte, Reihe 11, Nr. 167, 1992.

[Lewa93] Lewald S.: Ein neuartiges Verfahren zur numerischen Berechnung zeitoptimaler Robotersteuerungen. Dissertation eingereicht bei der Fakultät für Maschinenbau der Ruhr-Universität Bochum, Okt. 1993.

[Loet81] Lötstedt P.: Coulomb Friction in Two-Dimensional Rigid Body Systems. Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM), Vol. 61, S. 605–615, 1981.

[Lubi91] Lubich C.: Extrapolation integrators for constrained multibody systems. IMPACT Comp. Sci. Eng., No. 3, S. 213-234, 1991.

[Luh80] Luh J.Y.S., Walker M.W. und Paul R.P.C.: On-line computational scheme for mechanical manipulators. Trans. ASME Journal of Dynamic Systems Meas. and Control, Vol. 102, S. 69-76, 1980.

[Mah90] Mah R.S.H.: Chemical Process Structures and Information Flows. Butterworths Verlag, 1990.

[Mant92] Mantooth H.A. und Vlach M.: Beyond SPICE with Saber and MAST. Proceedings from the IEEE International Symposium on Circuits and Systems, S. 77-80, San Diego, Mai 1992.

[Math92] Mathworks Inc.: SIMULINK – User's Manual. South Natick, Mass., 1992.

[Matt92] Mattsson S.E. und Söderlind G.: A New Technique for Solving High-Index Differential-Algebraic Equations Using Dummy Derivatives. IEEE Symposium on Computer-Aided Control System Design, CACSD'92, Napa, California, März 1992.

[Matt93] Mattsson S.E. und Söderlind G.: Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives. SIAM Journal on Scientific Computing. Vol. 14, S. 677-692, Mai 1992.

[McCa88] McCalla W.J.: Fundamentals of Computer-Aided Circuit Simulation. Kluwer Academic Publishers, 1988.

[Mitc90] Mitchell E.E.L. und Gauthier J.S.: Simulation of Frictional Surfaces and Clutch Mechanisms. Mitchell & Gauthier application sheet, Juni, 1990.

[Mitc91] Mitchell E.E.L., und Gauthier J.S.: ACSL: Advanced Continuous Simulation Language – Reference Manual. Edition 10.0, MGS, Concord., Mass., 1991.

[Murt88] Murty K.G.: Linear Complementarity, Linear and Nonlinear Programming. Heldermann Verlag, Berlin, 1988.

[Nage75] Nagel L.W.: SPICE2: A computer program to simulate semiconductor circuits. Berkeley, University of California, Electronic Research Laboratory, ERL – M 520, 1975.

[Nils89] Nilsson B.: Structured Modelling of Chemical Processes – An Object-Oriented Approach. Licenciate thesis TFRT-3203, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, Sept. 1989.

[Otte88] Otter M. und Schlegel C.: Symbolic generation of efficient simulation codes for robots. In Proc. 2nd European Simulation Multi-Conference, S. 407–411, Nice, Frankreich, 1988.

[Otte88a] Otter M. und Türk S.: The DFVLR Models 1 and 2 of the Manutec r3 Robot. Technical Report DFVLR–Mitt.88–13, DLR, Institut für Robotik und Systemdynamik, Postfach 11 16, D–82234 Wessling, Mai 1988.

[Otte90] Otter M., Hocke M., Daberkow A. und Leister G.: Ein objektorientiertes Datenmodell zur Beschreibung von Mehrkörpersystemen unter Verwendung von RSYST. Institut B für Mechanik der Universität Stuttgart, ISBN 3-927618-04-7, März 1990.

[Otte91] Otter M. und Gaus N: Modular Dynamic Simulation with Database Integration. DSSIM User's Guide, Version 2.1. Technical Report TR R50–91, DLR, Institut für Robotik und Systemdynamik, Postfach 11 16, D–82234 Wessling, Juni 1991.

[Otte92] Otter M.: DSblock: A neutral description of dynamic systems. Version 3.2. Technical Report TR R81-92, DLR, Institut für Robotik und Systemdynamik, Postfach 11 16, D–82234 Wessling, May 1992. Auch erhältlich via anonymous FTP von "mailbase.ac.uk": Die Dateien "dsblock*.txt" im Directory /pub/lists/engineering-cace/files.

[Otte92a] Otter M.: Multidisciplinary Simulation. Concurrent Engineering Tools and Technologies for Mechanical System Design, E.J. Haug (editor), NATO ASI Series F, S. 203–216, Springer Verlag, 1993.

[Otte93a] Otter M., Hocke M., Daberkow A. und Leister G.: An Object-Oriented Data Model for Multibody Systems. Advanced Multibody System Dynamics, edited by W. Schiehlen, S. 19–58, Kluwer Academic Publishers, 1993.

[Otte93b] Otter M.: The ANDECS Simulation Environment. The International Handbook on Robot Simulation Systems, edited by D. Wloka, Volume II, erscheint demnächst.

[Pant88] Pantelides C.C: The consistent initialization of differential-algebraic systems. SIAM Journal of Scientific and Statistical Computing, No. 9, S. 213-231, 1988.

[Payn61] Paynter H.M.: Analysis and Design of Engineering Systems. MIT Press, Cambridge, Mass., 1961.

[Petz82] Petzold L.R.: A description of DASSL: A differential/algebraic system solver. Proc. 10th IMACS World Congress, Montreal, Aug. 1982.

[Pfei88] Pfeiffer F.: Über unstetige, insbesondere stoßerregte Schwingungen. Zeitschrift für Flugwissenschaft und Weltraumforschung, Vol. 12, S. 358–367, 1988.

[Robe88] Roberson R.E. und Schwertassek R.: Dynamics of Multibody Systems. Springer-Verlag, 1988.

[Rose83] Rosenberg R.C. und Karnopp D.C.: Introduction to Physical System Dynamics. McGraw-Hill, 1983.

[Rose86] Rosenthal R. und Sherman M.A.: High Performance Multibody Simulations via Symbolic Equation Manipulation and Kane's method. The Journal of Astronautical Sciences, Vol. 34, No. 3, S. 223-239, 1986.

[Rose87] Rosenthal R.: Order N formulation for equations of motion of multibody systems. SDIO/NASA Workshop on Multibody Simulation, JPL, Arcadia, CA, 1987.

[Rubi62] Rubin D.I.: Generalized material balance. CEP Symp. Ser. 58, S 54–61, 1962.

[Rueh90] Rühle R.: RSYST – Ein Softwaresystem zur Integration von Daten und Programmen zur Simulation wissenschaftlich-technischer Systeme. Rechenzentrum der Universität Stuttgart, RUS-5, März 1990.

[Rueh93] Rühle R. et. al.: RSYST Unterprogramm- und Modul-Dokumentation, Version 3.5.7, Rechenzentrum der Universität Stuttgart, Okt. 1993.

[Rulk90] Rulka W.: SIMPACK – A Computer Program for Simulation of Large-motion Multibody Systems. Multibody Systems Handbook, edited by W. Schiehlen, Springer-Verlag, 1990.

[Schi86] Schiehlen W: Technische Dynamik. Teubner-Verlag, 1986.

[Schi90] Schiehlen W.: Multibody Systems Handbook, Springer-Verlag, 1990.

[Schi93] Schiehlen W.: Advanced Multibody System Dynamics. Simulation and Software Tools. Kluwer Academic Publishers, 1993.

[Seyf92] Seyfferth W. und Pfeiffer F.: Dynamics of Assembly Processes with a Manipulator. Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 2, S. 1303–1310, Raleigh, Juli 1992.

[Shah85] Shah S.C., Floyd M.A. and Lehman L.L: Matrix$_X$: Control Design And Model Building CAE Capability. Computer–Aided Control Systems Engineering, edited by M. Jamshidi and C.J. Herget, Elsevier Science Publishers, S. 181–207, 1985.

[Sham86] Shampine L.F.: Conservation laws and the numerical solution of ODEs. Comp. and Maths. with Appls., Part B, Vol. 12, S 1287–1296, 1986.

[Smit90] Smith R.C. and Haug E.J.: DADS – Dynamic Analysis and Design System. Multibody Systems Handbook, edited by W. Schiehlen, Springer-Verlag, 1990.

[Smit92] Smith R.J. und Dorf R.C.: Circuits, Devices and Systems. Fifth Edition, John Wiley & Sons, 1992.

[Stro91] Stroustrup B.: The C++ Programming Language. Second Edition, Addison-Wesley, 1991.

[Tarj72] Tarjan R.E.: Depth First Search and Linear Graph Algorithms. SIAM Journal of Comp., Vol. 1, S. 146–160, 1972.

[Tiet86] Tietze U. und Schenk C.: Halbleiter-Schaltungstechnik. Springer-Verlag, 8. Auflage, 1986.

[Tuer90] Türk S.: Zur Modellierung der Dynamik von Robotern mit rotatorischen Freiheitsgraden. Dissertation. Fortschritt-Berichte VDI, Reihe 8, Nr. 211, VDI–Verlag, Düsseldorf, 1990.

[Tuer87] Türk S. und Otter M.: Das DFVLR Modell Nr. 1 des Industrieroboters Manutec r3. Robotersysteme, Vol. 3, S. 101–106, Springer-Verlag, 1987.

[Vach93] Vachoux A. und Nolan K.: Analog and Mixed-Level Simulation with Implications to VHDL. NATO Advanced Study Institute: Fundamentals and Standards in Hardware Description Languages, Kluwer Academic Publishers, 1993.

[Vere74] Vereshchagin A.: Computer Simulation of the Dynamics of Complicated Mechanisms of Robot-Manipulators. Engineering Cybernetics, Vol. 6, S. 65–70, 1974.

[Walk82] Walker M. und Orin D.: Efficient Dynamic Computer Simulation of Robotic Mechanisms. Journal of Dynamic Systems, Measurement and Control, Vol. 104, S. 205–211., 1982

[Weha82] Wehage R.A. und Haug E.J.: Generalized coordinate partitioning for dimension reduction in analysis of constrained dynamic systems. J. Mech. Design, Vol. 134, S. 247–255, 1982.

[Weha88] Wehage R.A.: Application of matrix partitioning and recursive projection to order n solution of constrained equations of motion. Proceedings of the 20th Biennial ASME Mechanisms Conference, Orlando, Florida, 1988.

[Witt77] Wittenburg J.: Dynamics of Systems of Rigid Bodies. Teubner Verlag, 1977.

[Woer88] Woernle C.: Ein systematisches Verfahren zur Aufstellung der geometrischen Schließbedingungen in kinematischen Schleifen mit Anwendung bei der Rückwärtstransformation für Industrieroboter. Dissertation, VDI Fortschritt-Berichte, Reihe 18, Nr. 59, 1988.