Aufbau und Validierung eines elastischen Manipulatormodells

Construction and Validation of an Elastic Robotic Manipulator Model

von

Anthony Anjorin

Bachelorarbeit

im Studiengang
Computational Engineering mit Schwerpunkt Informatik
am Fachgebiet Simulation und Systemoptimierung
der Technischen Universität Darmstadt
bei Prof. Dr. Oskar von Stryk

Betreuer: Dipl. Ing. Sebastian Klug

Darmstadt, 2. Mai 2007

Inhaltsverzeichnis

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe.

Anthony Anjorin

Darmstadt, 2. Mai 2007

Kapitel 1

Kurzzusammenfassung

Elastizität, klassischerweise ein unerwünschter Aspekt, den es in der Robotik zu unterdrücken gilt, wird zunehmend als sehr vorteilhaft angesehen und ist aktuell ein intensiv erforschtes Gebiet.

Ziel dieser Bachelorarbeit war es, eine Bibliothek von modularen, leicht anpassbaren und wiederverwendbaren Bausteinen – *SimMechanics*[1] *Blöcken* – zur Verfügung zu stellen, die eine schnelle und einfache Modellierung eines elastischen Manipulators[2] ermöglicht. Ein solches aus Bausteinen der Bibliothek bestehendes Simulationsmodell sollte sich darüberhinaus für anschließende Untersuchungen (Optimierungsaufgaben, Regelung) eignen.

Als besonders wichtig galt die ausreichend genaue und möglichst realitätsnahe Modellierung der elastischen Aspekte des zu modellierenden Prototyps.

Als Validierung der Richtigkeit und Eignung zur Optimierung eines mit Hilfe dieser Bibliothek erstellten Simulationsmodells wurden anhand verschiedener Manipulatorenmodelle Verhalten und Ergebnisse bewertet, nachvollzogen und, wenn möglich, analytisch nachgeprüft.

[1] Eine Bibliothek in Simulink/MATLAB zur Erstellung von physikalischen Modellen.
[2] Ein Industrieroboter(arm).

Abstract

Elasticity, normally considered a disturbance, is gradually becoming a promising area of research and potential improvement in robotics.

The aim of this thesis was to develop a library of modular, reusable and easily configurable building blocks – *SimMechanics*[3] *blocks* – that can be used to expeditiously construct a model of an elastic robotic manipulator[4] for purposes of optimization and control investigation.

A central aim hereby was to model the elastic aspects of the given prototype as exactly and correctly as possible. A direct mapping from prototype to simulation model was to be striven for as much as possible.

To serve as validation of the correctness and suitability for optimization of a model built with this library, a series of different simulation models of elastic robotic manipulators were constructed and tested. Results and behaviour were discussed and analytically verified when possible.

[3]SimMechanics is a Simulink/MATLAB library used for building physical models.
[4]An industrial robot(arm)

Kapitel 2

Einleitung

2.1 Hintergrund und Motivation

Manipulatoren, die heute in der Industrie eingesetzt werden, sind im Großen und Ganzen starre Systeme. Die

Elastizität wird dabei meist als störender Nebeneffekt betrachtet und durch komplexe Kontrollalgorithmen und Strukturen so weit wie möglich unterdrückt.

Dafür gibt es natürlich Gründe:

- Elastizität führt schnell zu chaotischem, schwer zu kontrollierendem Verhalten.
- Die aufgrund der Elastizität entstehenden Schwingungen in den Griff zu bekommen erfordert eine zusätzliche auf einer Rückkopplung der Soll- und Istwerte basierende Regelung.
- Auch wenn es einem gelingt, das System zu regeln und unerwünschte Schwingungen zu unterdrücken, ist diese Regelung nur für eine bestimmte Einstellung der Modellparameter, einen bestimmten Bewegungsvorgang und in einem engen Traglastbereich optimal.
- Es wird mit zunehmender Anzahl elastischer Elemente schwerer, eine hohe Präzision mit Standardtechniken zu erreichen.

- Bei komplexeren Manipulatoren mit über drei Gliedern wird das Verhalten des Modells so chaotisch[1], dass die benötigte Regelung sehr schwer zu simulieren und optimal zu bestimmen wird.

Bei der Modellierung starrer Manipulatoren ist es dennoch nicht möglich, komplett auf die Elastizität zu verzichten, besonders bei sehr schnellen Bewegungen bzw. Richtungsänderungen sowie bei schweren Lasten.

Ein großer Nachteil des klassischen starren Industrieroboters ist, dass das Kosten-Nutzen-Verhältnis noch sehr hoch ist, besonders im Bereich kleiner und mittelständischer Unternehmen.

Einige Gründe dafür sind:

- Um einen sicheren Einsatz im direkten Umfeld des Menschen zu gewährleisten, sind teure zusätzliche Sensorik und Systeme erforderlich.
- Dadurch, dass man die Elastizität und Deformation zu minimieren versucht (Abb. 2.1), entsteht ein nicht optimales Traglast-Eigengewicht-Verhältnis, was besonders bei mobilen Servicerobotern zu einem erhöhten Energiebedarf führt.
- Bei schnellen Bewegungen, Richtungsänderungen oder gar Stößen müssen wegen der niedrigen Elastizität die Motoren kurzzeitig eine sehr hohe Leistung erbringen, was zu Spitzenwerten führt. Dies erhöht einerseits wieder den Energiebedarf und ist auch für die Motoren und Getriebe schädlich.

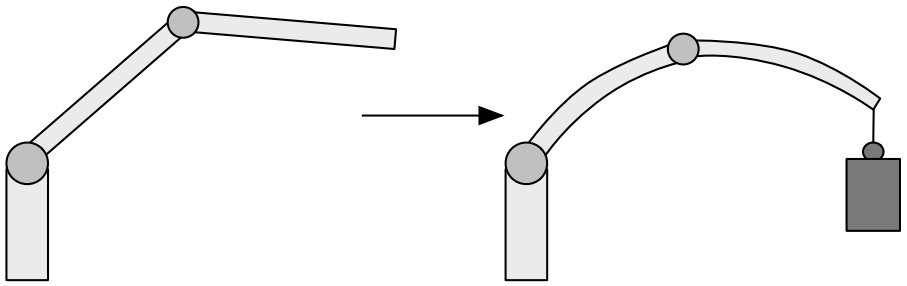
Angesichts dieser Probleme lohnt sich der Einsatz eines konventionellen Manipulators zunehmend nur bei der Automatisierung bestimmter Tätigkeiten, die unzureichend oder nur unter großer Gefahr von Menschen durchgeführt werden können.

Eine Lösung dieser Probleme bietet ein bionisch inspirierter, elastischer neuartiger Typ vom Roboterarm (Abb. 2.2). Im Projekt BioRob[2] wird diese Konstruktion entwickelt und dessen Vorteile werden erforscht und untersucht.

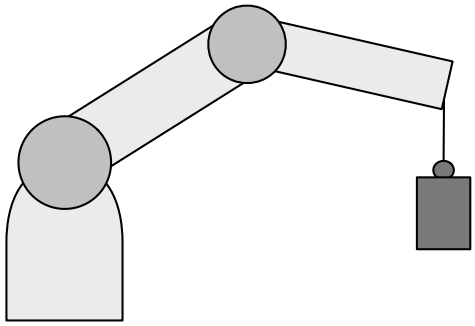
Erwartete Vorteile sind nach [1] und [3] unter anderem:

[1] Schon ein Doppelpendel kann chaotisches Verhalten aufweisen.

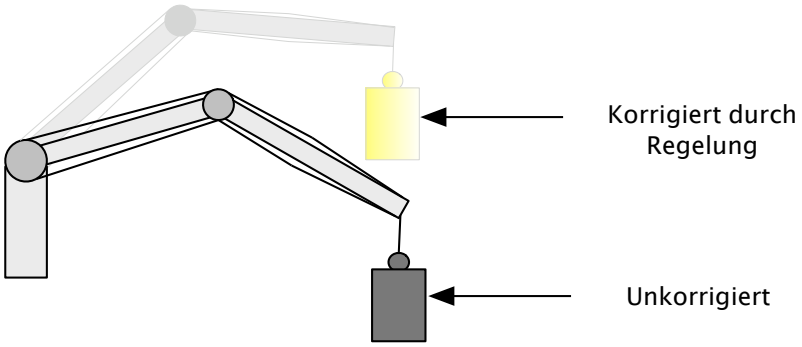
[2] Das Verbundprojekt BioRob [2] wird von der Technischen Universität Darmstadt koordiniert.



Das Problem: Elastische Deformation unter Last erschwert die Verhaltenskontrolle.



Die konventionelle Lösung: Motoren und Glieder ausbauen um Elastizität zu verringern, dies führt jedoch zu einem schlechten Traglast-Eigengewicht-Verhältnis.



Die Lösung mit elastischen Aktuatoren: Deformation tritt nur in Aktuatoren auf. Glieder sind von Biegung entlastet und Position kann durch Regelung korrigiert werden.

Abbildung 2.1: Nachteil des konventionellen starren Manipulators [1].

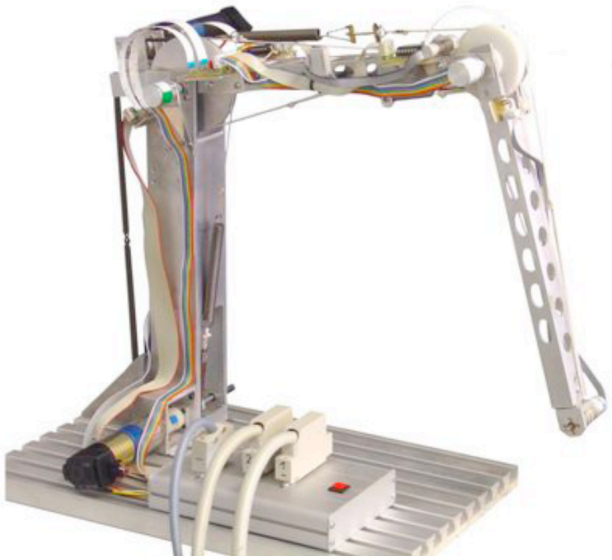


Abbildung 2.2: Labormodell eines bionisch inspirierten elastischen Manipulators [4].

- Durch eine elastische Antriebskonstruktion kann aufgrund der natürlichen Nachgiebigkeit eine hohe passive Sicherheit beim Einsatz im direkten Umfeld des Menschen ohne teure und komplizierte Sensorik erreicht werden.

- Da die Seilzüge und Federn (die zusammen die elastischen *Aktuatoren* bilden) jeden Zug abfangen, nehmen die Glieder der Roboter nur noch Druck auf, werden nicht mehr auf Biegung belastet und können viel gewichtsparender konstruiert werden. Dies führt zu einem guten Traglast-Eigengewicht-Verhältnis.

- Aus dem gleichen Grund treten Verformungen unter Belastung nur im elastischen Antrieb auf und nicht im Arm selber.

- Die Anforderungen, schwere Traglasten schnell bewegen zu können und eine hohe Positioniergenauigkeit aufzuweisen, sind schwer gleichzeitig zu erfüllen. Mit diesem Ansatz ist es jedoch möglich, Position- und Kraftregelung zu trennen und mit einem redundanten Kompositantrieb[3] optimal zu realisieren [1].

[3]Da der Kompositantrieb nur im statischen Fall hinzugeschaltet werden kann – also nicht bei der Ausführung einer Bahn – ist das Prinzip eher nur für Speziallösungen interessant.

- Da der Bedarf an komplexer Sicherheitssensorik entfällt, verspricht die Verwendung von Standardkomponenten eine kostengünstige Fertigung und Wartung.
- Mit einer geeigneten Regelung und Positions- und Geschwindigkeitsrückkopplung ist es möglich, Spitzenwerte in der Leistung der Motoren bei schnellen Bewegungen und Stößen zu vermeiden [5], um so die Langlebigkeit der Motoren zu gewährleisten.

2.2 Aufgabenstellung und Zielsetzung

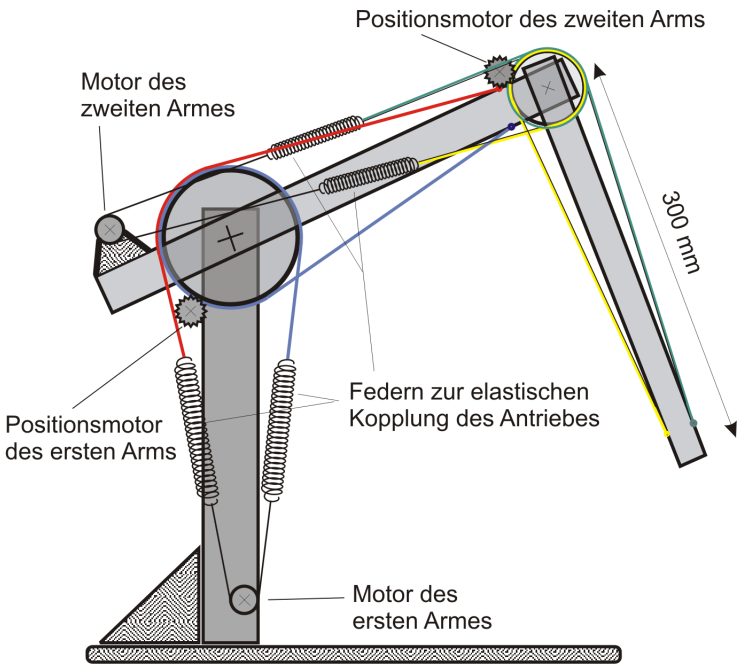


Abbildung 2.3: Schematische Abbildung eines Manipulators mit elastischen Aktuatoren [1].

Ziel des BioRob-Projekts ist es, Manipulatoren nach dem oben erwähnten Prinzip für verschiedene Wirtschaftszweige zu entwickeln. Dafür müssen Simulationsmodelle erstellt werden, um eine Reihe von Optimierungs- und Regelungsaufgaben, die aufgrund der erhöhten Elastizität entstehen, zu untersuchen und zu lösen.

Um dieses Ziel zu erreichen wird eine Bibliothek von Bausteinen benötigt, die eine schnelle und einfache Erstellung eines Simulationsmodells ermöglicht.

Anforderungen, die an eine solche Bibliothek gestellt werden, sind im Wesentlichen:

Die Bausteine sollten SimMechanics Blöcke sein: Da MATLAB bereits mit der Optimierungstoolbox[4], Systemregelungstoolbox und mit Simulink[5] eine für die Untersuchungen geeignete Umgebung zur Verfügung stellt, ist es wünschenswert und sinnvoll, die Bausteine der Bibliothek aus dem gleichen Guss – SimMechanics ist selber eine Bibliothek in Simulink – zu erstellen. Dies fördert außerdem die Interoperabilität[6] und nicht zuletzt die Benutzbarkeit und Erlernbarkeit, da SimMechanics den Aufbau und die Syntax von MATLAB/Simulink konsequent erweitert.

Die Elastizität sollte ausreichend und möglichst realitätsnah modelliert werden. Es ist für die Benutzbarkeit und Verständlichkeit unerlässlich, dass die Modellierung eine möglichst direkte Abbildung der Realität sein sollte. Insbesondere bei dem elastischen Antrieb, wie in Abb. 2.3, ist es wichtig, dass das Modell alle Möglichkeiten zur Parameteranpassung zulässt, was z.B. für die Optimierung eine wichtige Rolle spielen könnte.

Die Modelle sollten sich für Optimierung und Regelung eignen. Da Optimierungs- und Regelungsaufgaben mit den Simulationsmodellen erforscht und gelöst werden sollen, müssen die Modelle ein gewisses Maß an Stabilität und Effizienz aufweisen, um Berechnungen zu ermöglichen. Außerdem müssen die Schnittstellen der einzelnen Blöcke so konzipiert sein, dass benötigte Informationen z.B. über Position und Geschwindigkeit leicht herausgelesen und Impulse eingeschleust werden können, was bei einer Regelung mit rückgekoppeltem Positions- und Geschwindigkeitsfehler der Fall sein muss.

Zugleich sollte aber darauf geachtet werden, dass so viele Details wie möglich in einem Block gekapselt bleiben, um die Verständlichkeit zu erhöhen, die Fehlersuche zu erleichtern und die Modularität und Wiederverwendbarkeit der Blöcken zu unterstützen.

[4]Eine MATLAB-Toolbox ist eine Bibliothek von nützlichen, auf einem Gebiet spezialisierten Funktionen.

[5]Simulink ist eine Erweiterung von MATLAB, die die teilweise graphische Erstellung von mathematischen Modellen unterstützt.

[6]In diesem Kontext die Möglichkeit zur Anbindung an bestehende MATLAB bzw. Simulink Programme.

Darüberhinaus wurden nach Absprache mit den Endnutzern folgende Qualitätsmerkmale, die besonders beachtet werden sollte, festgelegt. Die Reihenfolge gibt die Priorisierung an:

Richtigkeit: Um zuverlässige Ergebnisse bei Untersuchungen zu bekommen, müssen die Modelle das zu simulierende Verhalten richtig wiedergeben. Dies bedeutet an erster Stelle, dass ein Modell keine Modellierungsfehler enthalten darf. Darüberhinaus muss die Modellierung auch fein genug sein, um alle nötigen Details zu berücksichtigen.

Modularität: Die Aufgabe war nicht die Modellierung eines *einzigen* konkreten Manipulators, sondern vielmehr eine Bibliothek von Modulen[7] zur Verfügung zu stellen, die das Erstellen mehrerer *verschiedener* Modelle ermöglicht. Folglich müssen die Module möglichst wiederverwendbar und anpassbar sein. Außerdem sollte es einfach sein, Teile eines einzelnen Moduls auszutauschen, um so eine *Plug-In*[8] ähnliche Möglichkeit anzubieten, ein Modul zu verändern. Dies und mehr wird insbesondere durch einen hohen Grad an Modularität erreicht.

Erweiterbarkeit: Es ist nicht möglich, jedes Modul in jeder erdenklichen Variation[9] in der Bibliothek anzubieten. Statt dessen ist es zweckmäßiger, Module mit einer zwar festen sinnvollen Voreinstellung zu versehen, aber ausreichende Möglichkeiten der Erweiterbarkeit zur Verfügung zu stellen.

Nahtlose Integration in SimMechanics/Simulink: Simulink besitzt eine Ansammlung von Bibliotheken, zu der unter anderem SimMechanics gehört. Es wurde als Ziel gesetzt, die zu entwickelnde Bibliothek so zu implementieren, dass keine Unterschiede zu anderen eingebauten Bibliotheken zu bemerken sind. Dies geht von dem richtigen *„Look and Feel“*[10] bis hin zu Masken[11] und einer automatischen Installation[12] beim Starten von Simulink.

[7] Fortan bezeichnet ein „Modul“ einen Baustein bzw. Block in der Bibliothek.

[8] In diesem Kontext eine leicht austauschbare Komponente.

[9] Orientierung der Drehachsen, Trägheitstensoren, relative Orientierung zu angrenzenden Modulen und Abmessung der Glieder.

[10] Der Begriff „Look and Feel“ (Aussehen und Handhabung/Anfühlen) bezeichnet meist durch Hersteller oder Konsortien standardisierte Design-Aspekte.

[11] Einstellungsdialoge für Subsysteme in Simulink.

[12] Erweiterung des *MATLAB-Pfads* um nötige Verzeichnisse, Einbinden und Laden der Bibliothek.

Verständlichkeit und Benutzbarkeit: Genau wie alle anderen Blöcke in SimMechanics sollen die Module der Bibliothek ausreichend dokumentiert sein, ganz im Stil von Simulink und SimMechanics. Außerdem wäre es wünschenswert und sinnvoll, eine Ansammlung von einfachen Simulationsmodellen bereitzustellen, die beispielhaft zeigen, wie die Module zu verändern, zu erweitern und mit einander zu verkoppeln sind.

Erweiterbare Dokumentation: Da eine ständige Erweiterung der Bibliothek vorgesehen und unterstützt werden sollte, ist es sinnvoll, den Dokumentationsprozess zu erleichtern. Es sollte also möglich sein, schnell und einfach eine einheitliche Dokumentation für ein verändertes Modul oder komplett neues Submodul[13] zu erstellen. Immer die gleiche HTML[14]-Struktur per *„Copy and Paste“*[15] zu verändern käme folglich nicht in Frage.

Effizienz: Um mit den Modellen auch Untersuchungen in vernünftiger Zeit durchführen zu können, muss ein gewisser Grad an Effizienz vorhanden sein. Dies sollte aber nicht auf Kosten der Verständlichkeit oder Modularität geschehen und deshalb wird eher eine Feineinstellung[16] in der Validierungsphase vorgesehen, um Engpässe zu finden und so nachträglich die Effizienz an den wichtigsten Stellen zu steigern. Selbstverständlich spielt die Effizienz dennoch eine wichtige Rolle und sollte ständig verbessert werden durch beispielsweise den Einsatz von eingebauten Blöcken statt Selbstimplementierungen, wenn möglich.

Um die Erfüllung der Anforderungen und das Sichern der Qualitätsmerkmale zu gewährleisten, sollte es nach der Erstellung der Bibliothek eine Validierungsphase geben, die sich in zwei Iterationen gliedern soll:

1. Validierung vor allem der Richtigkeit und der Erfüllung aller anderen Anforderungen und Qualitätsmerkmale.

2. Validierung der Eignung für Optimierung.

[13] Mit Submodul wird ein Modul bezeichnet, das von einem anderen übergeordneten Modul benutzt wird und als Plug-In dienen sollte.
[14] Hypertext Markup Language.
[15] In diesem Kontext lästiges Kopieren, Einfügen und Anpassen der HTML-Seiten.
[16] Fine tuning.

2.3 Übersicht

Nach dieser Einleitung wird in Kapitel 3 der Stand der Forschung erläutert, sowohl am Fachgebiet Simulation und Systemoptimierung der Technischen Universität Darmstadt als auch in Deutschland und weltweit. Außerdem wird das alte Simulationsmodell, das schon in Adams[17] implementiert wurde, kurz vorgestellt und mit der neu zu erstellenden Bibliothek verglichen.

Als nächstes wird in Kapitel 4 die im Rahmen dieser Arbeit entstandene Bibliothek vorgestellt. Kritische Entscheidungen werden in Einzelheiten diskutiert und anhand möglicher Alternativen in Bezug auf die priorisierten Qualitätsmerkmale bewertet.

In Kapitel 5 werden die Ergebnisse der Validierungsphase vorgestellt und jeweils in einer anschließenden Diskussion bewertet und nachvollzogen.

Schließlich wird in Kapitel 6 kurz zusammengefasst, was erreicht wurde und was nicht. Mögliche Verbesserungen und auf den Ergebnissen aufbauende bzw. weiterführende Fragestellungen, werden kurz angeschnitten.

Im Anhang ab Seite 61 werden noch detaillierte Fragestellungen beantwortet, unter anderem mit Hilfe einer Installationsanleitung, einer Erklärung der Verzeichnisstruktur der Bibliothek und einer Anleitung zur Erweiterbarkeit verschiedener Aspekte und zur Erstellung und Anpassung von Inhalt und Struktur der Dokumentation (Hilfe).

Im Glossar ab Seite 68 sind kurze Definitionen und Erklärungen wichtiger Begriffe und Notationen[18] zu finden.

[17] Adams ist ein Programm zur Mehrkörpersimulation und FEM-Berechnung.

[18] Wie z.B. MATLAB/Simulink/SimMechanics- und XML-spezifischen Abkürzungen bzw. Terminologien.

Kapitel 3

Stand der Forschung

Aktuell werden in Deutschland und weltweit die Vorteile und Einsatzmöglichkeiten der Elastizität in der Robotik untersucht und erprobt. Sowohl bei humanoiden Laufrobotern als auch bei Manipulatoren werden neue Modellierungsansätze, die ein hohes Maß an Elastizität vorsehen und ausnutzen, ausprobiert, optimiert und bewertet.

Am Fachgebiet Simulation und Systemoptimierung an der TUD werden Modelle mit redundanten hoch elastischen Aktuatoren untersucht, die die Biegeentlastung der Roboterglieder ermöglichen, was unter anderem zu einem sehr guten Verhältnis von Traglast zu Eigengewicht führt und eine hohe passive Sicherheit ohne teure Sensorik gewährleistet.

Zu diesem Zweck wurde schon ein Simulationsmodell in Adams erstellt und für die Optimierung und Bewertung dieses neuartigen Antriebsprinzips eingesetzt.

Probleme des Modells waren unter anderem:

- Die Anbindung an MATLAB oder andere externe Programme war schwer und kompliziert. Da häufig mit Hilfe der MATLAB-Optimierungstoolbox gearbeitet wird, in Simulink die Regelung implementiert wird und Motormodelle aufgestellt werden, wäre SimMechanics, unter diesem Gesichtspunkt, eine bessere Wahl der Simulationssprache.
- Bei chaotischen Systemen (z.B. bei hoher Elastizität und hochfrequenten Schwingungen) war die Stabilität des Modells nicht zufriedenstellend. Am Fachgebiet besteht die fundierte Hoffnung, dass SimMechanics zu stabileren Modellen führen sollte.

- Die Erstellung einer Bibliothek – einer Ansammlung von wiederverwendbaren und einstellbaren Modulen – ist nicht vorgesehen oder unterstützt in Adams. Da darüberhinaus ein neuer Entwurf des Prototyps vor kurzem erstellt wurde, müsste das Adams-Modell wesentlich verändert werden, was den Umstieg auf SimMechanics und eine in größerem Umfang erweiterbare Lösung noch attraktiver macht.

- Das Modell in Adams bildet die elastischen Aktuatoren als ein elastisches Gelenk ab. Dadurch wird das Simulationsmodell einfacher und effizienter (wegen eingebauter elastischer Gelenkmodule), aber die Möglichkeit, die Federkonstanten *unabhängig* von einander zu verändern, geht dabei verloren. Da dieses tun zu können unter anderem für die Optimierung und Regelung interessant sein könnte, sollte eine direktere realitätsnähere Lösung erarbeitet werden.

Kapitel 4

Simulationsmodell

Im folgenden wird nach einem *Top-Down-Ansatz*[1] die gewählte Dekomposition des Simulationmodells in Module motiviert und erläutert. Danach wird die leicht erweiterbare Möglichkeit zur Dokumentation und Einstellung, die die Bibliothek anbietet, vorgestellt. Anschließend wird anhand von Beispielen gezeigt, wie die erarbeiteten Module zusammengesetzt werden können, um Modelle konkreter Manipulatoren zu erstellen. Schließlich wird in der Diskussion 4.2 die Bibliothek als ganzes bewertet und es werden durch einen Vergleich mit Alternativen Vor- und Nachteile erläutert.

4.1 Ergebnisse

Wie in Abb. 4.1 zu sehen, wäre die erste sinnvolle Aufteilung eines Manipulators eine Dekomposition in ein *Basismodul*, das rotierbar oder fest sein kann, und in mehrere miteinander gekoppelte *Gelenkmodule*.

4.1.1 Module

4.1.1.1 Das Basismodul

Mit dem Basismodul wird die Position des Armes im Raum festgelegt – in SimMechanics-Terminologie – „geerdet“, da alle anderen Module ihre Positionen relativ dazu ableiten können. Der Aufbau des Basismoduls fällt relativ

[1] Vom Groben zum Feinen, in diesem Kontext zuerst die Hauptmodule, die sich aus anderen Submodulen zusammensetzen, die dann im Anschluss unter die Lupe genommen werden.

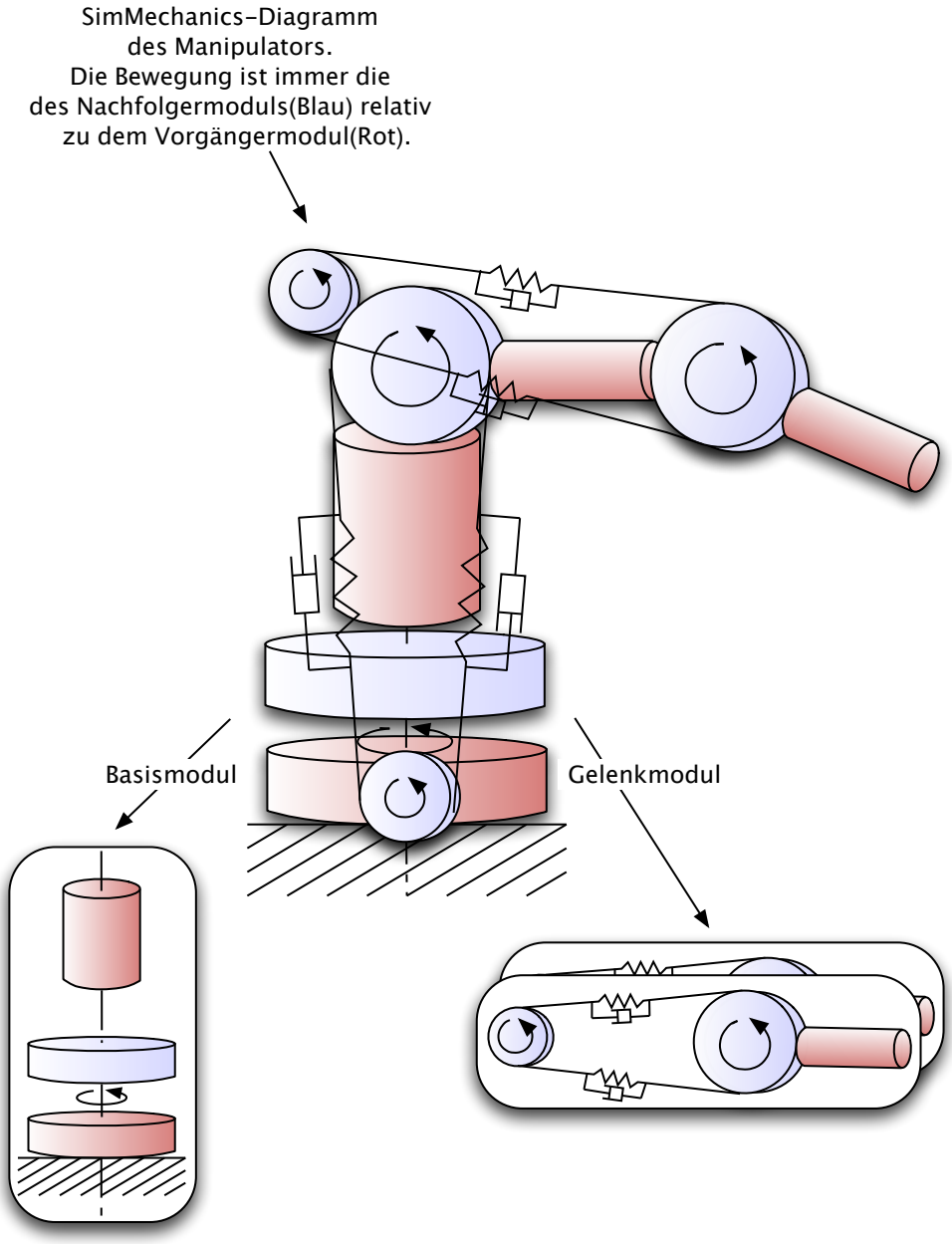


Abbildung 4.1: Dekomposition des Simulationsmodells in Module.

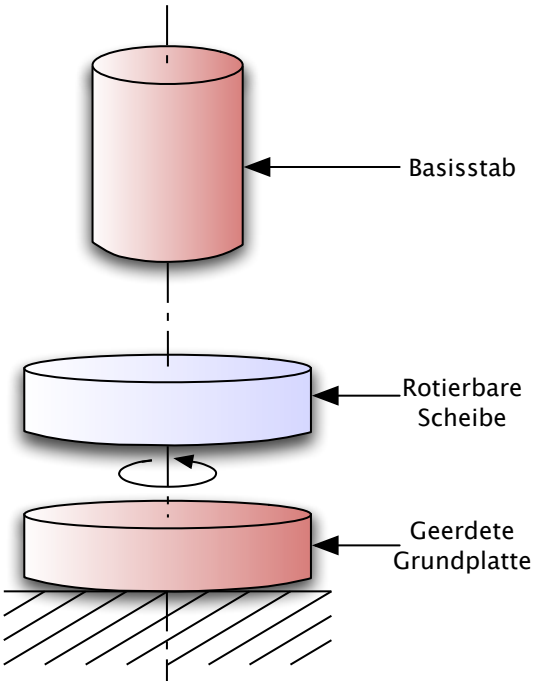


Abbildung 4.2: Aufbau des Basismoduls.

einfach aus: Wie in Abb. 4.2 zu sehen ist, bilden eine geerdete Scheibe und eine rotierbare, mit einem Stab[2] fest verbundene Scheibe, zusammen das Modul.

4.1.1.2 Das Gelenkmodul

Das Gelenkmodul (Abb. 4.3) ist dagegen das Herzstück der Bibliothek und setzt sich aus einem Antriebsrad und elastischen Aktuatoren, die dieses Rad mit einer größeren Scheibe verbinden, zusammen. Die Scheibe ist fest verbunden mit einem Stab und bietet die Möglichkeit, ein weiteres Gelenkmodul am Ende des Stabs anzuhängen.

4.1.1.3 Das Motormodul

Das Antriebsrad des Gelenkmoduls wird wiederum fest verbunden mit einer Scheibe, die sich in einem *Motormodul* befindet und angetrieben wird. Das

[2]Ein Stab ist im Vergleich zu seiner Länge sehr dünn und hat in der Modellvorstellung keine Ausdehnung.

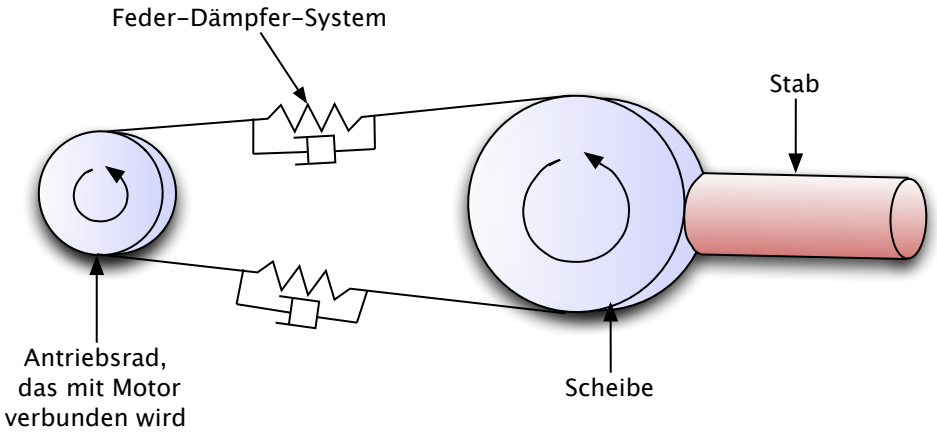


Abbildung 4.3: Aufbau des Gelenkmoduls.

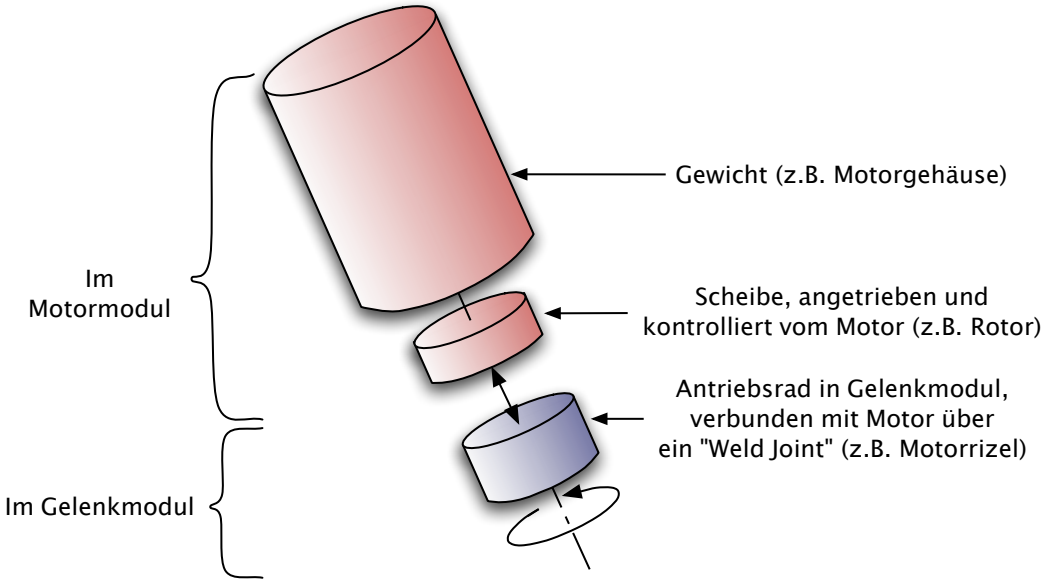


Abbildung 4.4: Aufbau und Einsatz des Motormoduls.

Motormodul kapselt erstens die Mechanik und Logik des Antriebs und zweitens die Regelung. Wird ein bestimmtes Bewegungsmuster vorgegeben, so ist das Motormodul dafür zuständig, durch eine Rückkopplung des Positions- und Geschwindigkeitsfehlers des Armes, die Aktuatoren durch Rotation des Antriebrades so zu spannen, dass der gewünschte Bewegungsvorgang möglichst optimal[3] ausgeführt wird. Abb. 4.4 zeigt, wie das Motormodul mit dem Gelenkmodul kombiniert werden sollte.

4.1.2 Submodule

Um die Modularität weiter zu erhöhen und dadurch die Wiederverwendbarkeit und Erweiterbarkeit zu verbessern, wird das relativ komplexe Gelenkmodul weiter in ein Feder-Dämpfer-System (Federmodul), das die Implementierung eines elastischen Aktuators kapselt, und in ein Sensormodul, das die Rolle eines *Adapters*[4] übernimmt, unterteilt.

Diese Module sind als Plug-Ins für das Gelenkmodul zu betrachten, denn durch Austausch des Federmoduls können verschiedene Implementierungen[5] der elastischen Aktuatoren problemlos eingesetzt und ausprobiert werden. Falls die vorgegebenen Schnittstellen nicht angemessen sind, kommt das Sensormodul als Adapter zwischen dem Antriebsrad bzw. der Scheibe und den Federmodulen ins Spiel: In dem Sensormodul können die Schnittstellen angepasst und unter anderem Umrechnungen oder Veränderungen unternommen werden, um – ohne die umliegende Struktur verändern zu müssen – eine große Vielfalt an verschiedenen Aktuatorenimplementierungen einbinden zu können.

4.1.2.1 Das Federmodul

In der Bibliothek stellt das Feder-Submodul eine relativ klare, einfache und intuitive Modellierung eines elastischen Aktuators zur Verfügung. Wie im Labormodell (Abb. 2.3) wird ein Aktuator als klassisches Feder-Dämpfer-System abgebildet. Je nach Drehung einer der beiden Räder, um die der Seilzug gebunden ist, wird die Feder mehr oder weniger ausgedehnt. Dabei

[3] Was als optimal betrachtet wird, ist variabel und hängt von den betrachteten Faktoren ab (z.B. Energie, Zeit, Stabilität).

[4] Der Adapter ist ein Entwurfsmuster und dient zur Übersetzung einer Schnittstelle in eine andere.

[5] Wie z.B. Muskelmodelle, die Ermüdung, Reflexe und Überlastung simulieren [6].

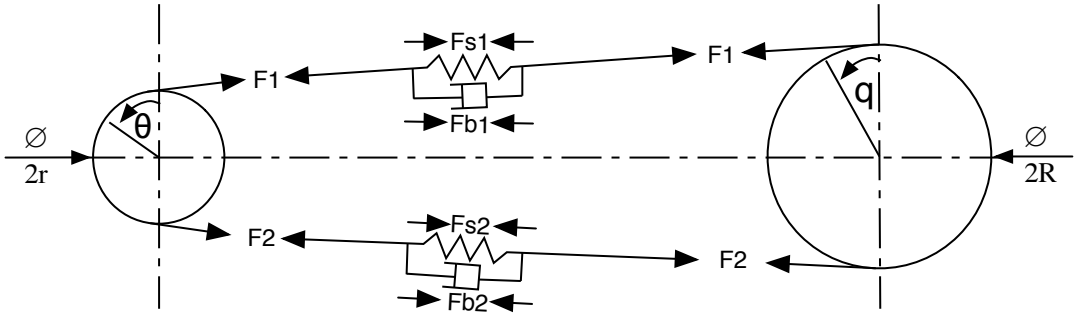


Abbildung 4.5: Funktionsweise des Federmoduls

ist zu beachten, dass die mit dem Seil gebundene Feder nicht zusammengestaucht werden kann. Sobald der Seilzug zusammensackt, wird die Federkraft nicht negativ sondern Null. Abb. 4.5 stellt diesen Ansatz graphisch dar.

Die folgenden Formeln lassen sich davon ableiten und werden im Modul mit Simulink-Blöcken ausgedrückt:

$$
\begin{aligned}
\theta &:= \text{Motorwinkel} \\
q &:= \text{Armwinkel} \\
K &:= \text{Federkonstante} \\
b &:= \text{Dämpfungskonstante} \\
\delta x &:= \text{Längenänderung der Feder} \\
\delta \dot{x} &:= \text{Relative Geschwindigkeit} \\
\delta_0 &:= \text{Anfangsausdehnung} \\
F &:= \text{Nötige Kraft, um Feder um } \delta x \text{ auszudehnen} \\
F_s &:= \text{Federkraft} \\
F_b &:= \text{Dämpfungskraft}
\end{aligned}
$$

$$
\begin{aligned}
F_1 &= F_{s1} + F_{b1} = \underbrace{K_1 \delta x_1 + F_{01}}_{F_{s1}} + \underbrace{b_1 \delta \dot{x_1}}_{F_{b1}} = K_1(\underbrace{\theta r - qR}_{\delta x_1} + \delta_{01}) + b_1(\underbrace{\dot{\theta} r - \dot{q} R}_{\delta \dot{x}_1}) \\
F_2 &= K_2(\underbrace{qR - \theta r}_{\delta x_2} + \delta_{02}) + b_2(\underbrace{\dot{q}R - \dot{\theta} r}_{\delta \dot{x}_2})
\end{aligned}
$$

Im Modul, positiver Winkel $= \phi_1$, negativer Winkel $= \phi_2$

$$
\rightsquigarrow \boxed{F = K(\phi_1 r_1 - \phi_2 r_2 + \delta_0) + b(\dot{\phi_1}\, r_1 - \dot{\phi_2}\, r_2)}
$$

4.1.2.2 Sensormodul

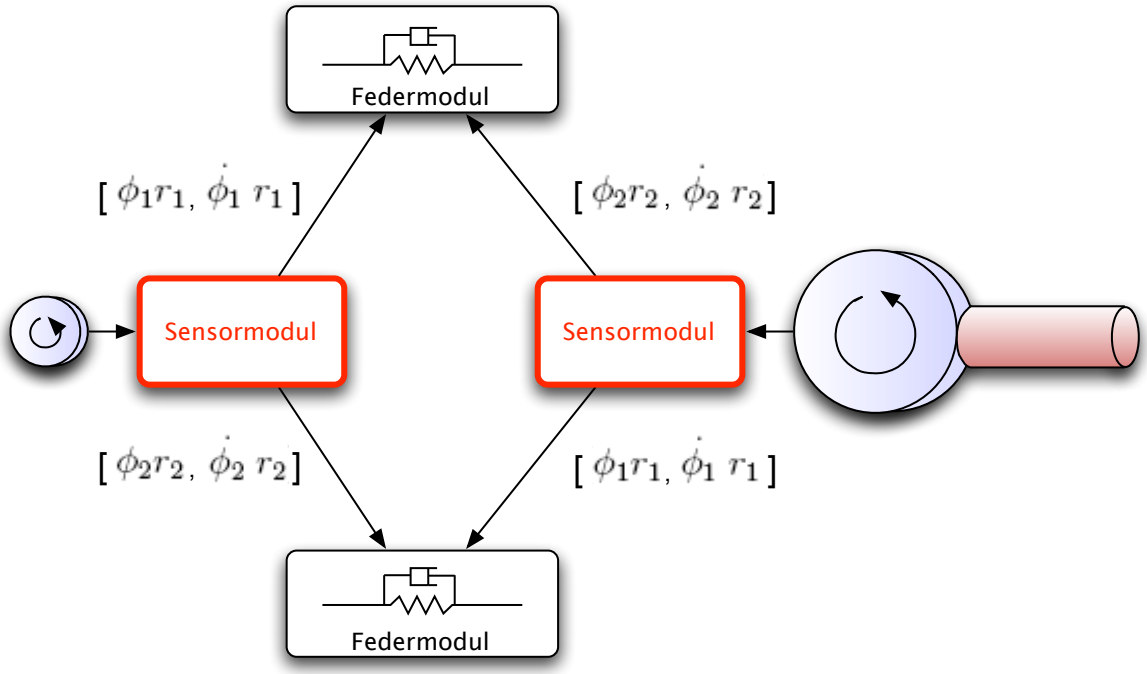


Abbildung 4.6: Einsatz des Sensormoduls.

Eine Eigenheit des eingebauten *Joint-Sensors* (Gelenksensors) in SimMechanics ist die Ausgabe des abgelesenen Winkels im Intervall $[-180°, 180°]$. Auch wenn dies für manche Anwendungen von Vorteil sein könnte, braucht das Federmodul einen kontinuierlichen Winkelverlauf, um die entsprechende Federkraft zu berechnen. Die nötige Übersetzung wird vom Sensormodul übernommen. Auch wenn diese Aufgabe nicht so komplex ist und problemlos direkt erledigt werden könnte, ist es unerwünscht, das ganze Gelenkmodul den Bedürfnissen des Federmoduls anzupassen. Es könnte gut sein, dass eine andere Implementierung des Federmoduls eine andere Übersetzung und somit einen anderen Adapter bräuchte. Abb. 4.6 zeigt schematisch wie das Sensormodul eingesetzt wird.

4.1.3 Erweiterbarkeit

4.1.3.1 Erweiterbare Dokumentation

Wie in der Einleitung erläutert, war es eine wichtige Anforderung, schnell und einfach veränderte bzw. angepasste Module einheitlich und ganz im Stil von MATLAB/Simulink/SimMechanics dokumentieren zu können.

Die gewählte Lösung war die Benutzung eines CSS[6] *Stylesheets*[7] , um das gleiche „Look and Feel“ wie in SimMechanics-Hilfeseiten zu erreichen. Lediglich eine HTML-Vorlage bereitzustellen war als Lösung nicht ausreichend, da diese durch „Copy and Paste“ für neue Seiten benutzt werden müsste. Was, wenn die Vorlage geändert werden sollte? – Ein Wartungsalptraum, da nämlich *alle* existierenden Seiten mitgeändert werden müssten! Außerdem ist der Aufwand, eine komplette HTML-Seite, die ziemlich viele Elemente enthält, wie alle anderen Hilfeseiten jedes Mal erstellen zu müssen, zu hoch und würde wahrscheinlich dazu führen, dass neue Module ohne Dokumentation vom Endbenutzer der Bibliothek hinzugefügt werden.

Aus vielen Lösungsmöglichkeiten wurde schließlich die Kombination von XML[8] mit XSLT[9] gewählt. Eine einfache, übersichtliche Metasprache wurde mit XML entwickelt, mit der es möglich ist, wenige Tags[10] für den sich ändernden Inhalt zu erstellen und dann mittels eines XSLT-Stylesheets, das die Tranformation XML → HTML definiert, automatisch den gleich bleibenden Inhalt hinzuzufügen und die entsprechenden HTML-Seiten zu erzeugen. Der ganze Prozess lässt sich mühelos als Projekt in Eclipse[11] durch die Ergänzung eines Ant-Skripts[12] ausführen. Durch dieses Dokumentationsframework ist Struktur und Style an einer einzigen Stelle festgelegt, was Änderungen und Erweiterungen erheblich erleichtert. Der Benutzer ist auch entlastet und kann mit den Vorlagen eine Hilfeseite zu einem neuen Modul schnell und

[6] CSS (Cascading Style Sheets) ist eine deklarative Stylesheet-Sprache für strukturierte Dokumente.

[7] Ein Stylesheet ist eine Formatvorlage, die die Trennung von Daten und Darstellung ermöglicht.

[8] XML, die eXtensible Markup Language (erweiterbare Auszeichnungssprache), ist ein Standard zur Modellierung von strukturierten Daten in Form einer Baumstruktur.

[9] XSLT, eXtensible Stylesheet Language Transformation, ist eine turing-vollständige Programmiersprache zur Transformation von XML-Dokumenten.

[10] In XML werden Tags verwendet, um Daten zu klassifizieren und strukturieren.

[11] Eclipse ist eine Entwicklungsumgebung (IDE).

[12] Ant ist ein in Java geschriebenes Werkzeug zum automatisierten Erzeugen von Programmen oder anderem aus Quelltext.

unkompliziert erstellen. Darüberhinaus besteht die Möglichkeit, neue Transformationen zu erstellen: XML → SVG? XML → LaTeX? Die Möglichkeiten sind ohne jegliche Veränderung der schon bestehenden XML-Hilfesourcen grenzenlos.

Details zur Benutzung des mitgelieferten Eclipse-Projekts, Möglichkeiten zur Erweiterung oder Anpassung der Struktur und des Aussehens der Hilfeseiten, und eine Erläuterung, wie die XML-Metasprache validiert[13] und um neue Tags ergänzt werden kann, befinden sich im Anhang ab Seite 61.

Abb. 4.7 zeigt beispielhaft eine vom Nutzer zu erstellende XML-Datei und die daraus automatisch erzeugte HTML-Hilfeseite.

4.1.3.2 Erweiterbare Einstellungsdialoge

Um kleinere Anpassungen der Module zu erleichtern, wie z.B. einen Orientierungswechsel, wurde zu jedem Block eine Initialisierungsfunktion erstellt. Wie üblich in Simulink gibt es zu jedem Block einen Einstellungsdialog, worüber eine Auswahl an Parametern vom Benutzer verändert werden können. Die restlichen Parameter (z.B. Rotationsachsen) im Modul sind übersichtlichkeitshalber entweder auf einen vernünftigen Wert festgelegt oder werden von den anderen Parameter auf eine bestimmte Weise abgeleitet bzw. berechnet. Diese Initialisierung geschieht in MATLAB-Funktionen, die für die Dialoge als sogenannte *„Callback Functions“*[14] dienen.

Um Anpassungen an einem Block vorzunehmen, müsste nur eine neue Initialisierungsfunktion erstellt werden und im Dialog bei den entsprechenden Parametern als „Callback Function“ eingestellt werden. Den Dialogen können natürlich auch neue Parameter und Werte (z.B. in Drop-Down-Menüs[15]) hinzugefügt werden.

Im nächsten Abschnitt 4.1.4 werden anhand von Beispielen gezeigt, wie einfach es ist, die Module zu verwenden und anzupassen. Weitere Details zur Erweiterbarkeit der Module sind dem Anhang ab Seite 61 zu entnehmen.

[13] Mit Hilfe einer DTD (Dokumenttyp-Definition): ein Satz an Regeln, der benutzt wird, um die Struktur von Dokumenten eines bestimmten Typs festzulegen.

[14] Eine Rückruffunktion bezeichnet eine Funktion, die als Parameter übergeben und unter gewissen Bedingungen aufgerufen wird.

[15] Drop-Down-Menüs sind mit der Maus anwählbare Menüs, die beim Anklicken eine Liste von Menüeinträgen preisgeben.

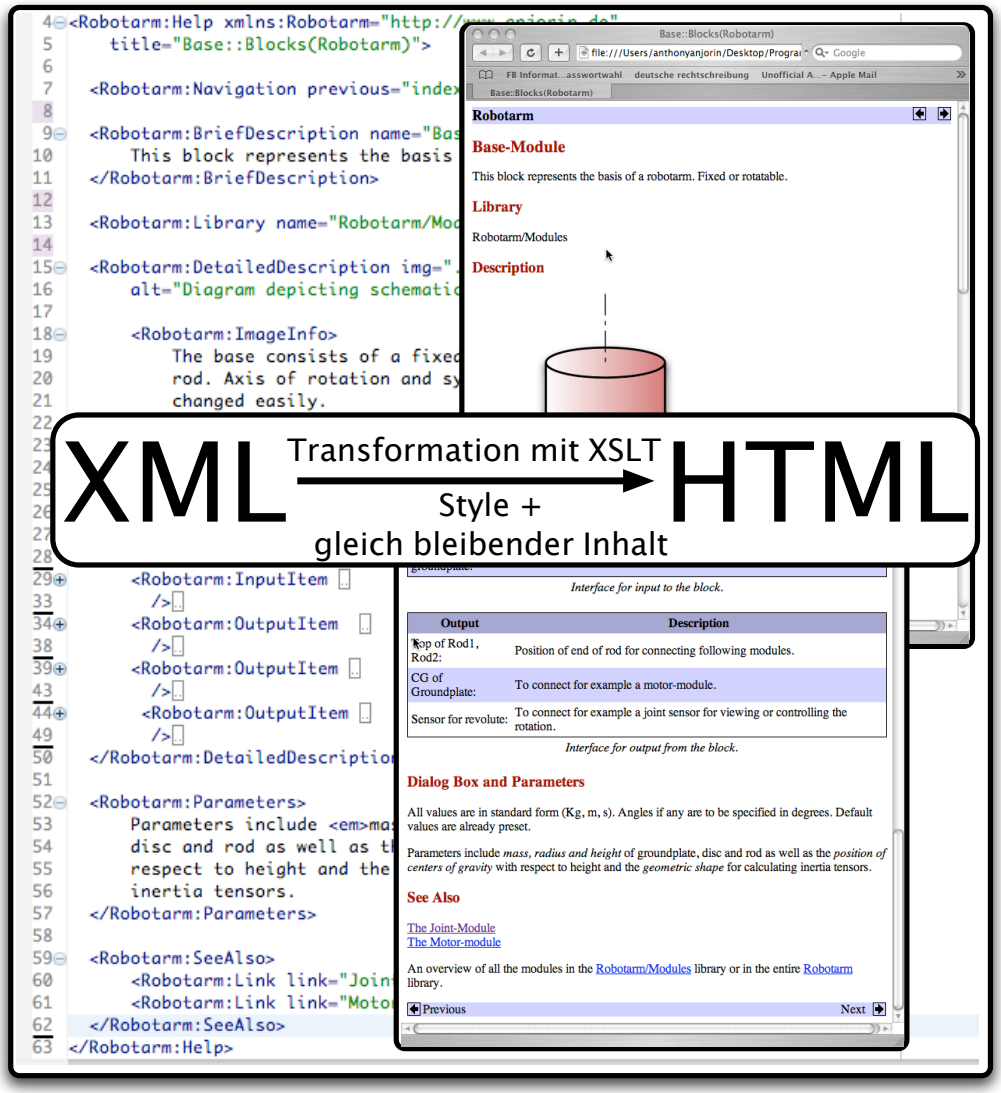


Abbildung 4.7: Erweiterbare Dokumentation durch XML und XSLT.

4.1.4 Beispiele

4.1.4.1 Modell eines zwei-gliedrigen Manipulators.

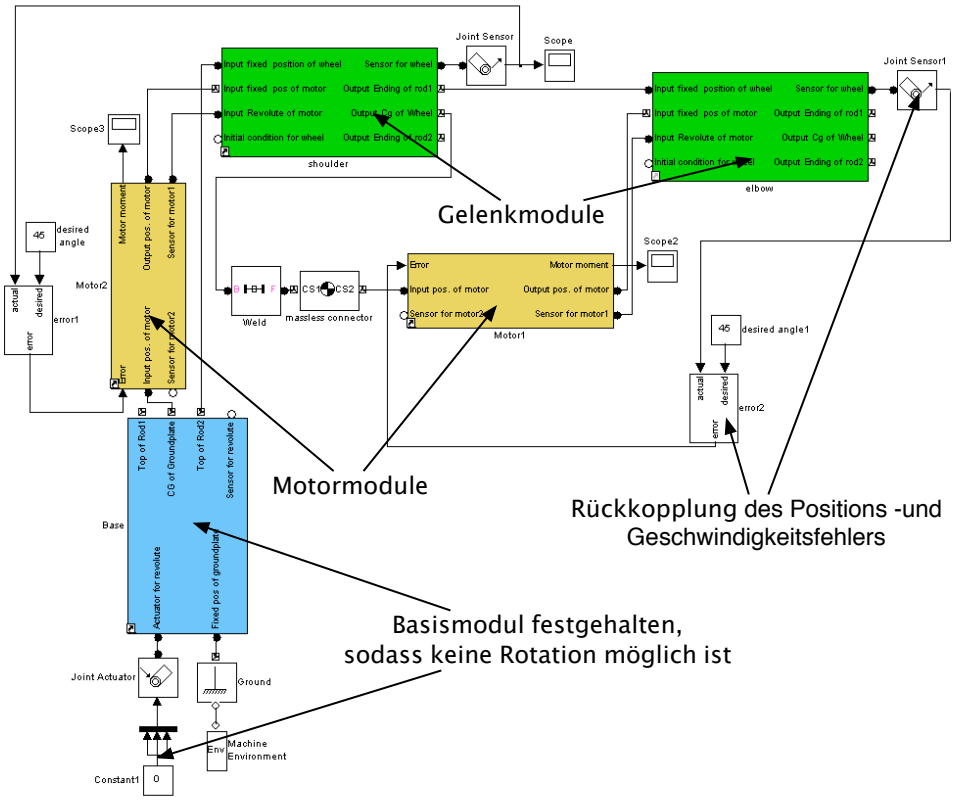


Abbildung 4.8: Blockdiagramm des zwei-gliedrigen Modells.

Wie in Abb. 4.9 zu sehen, sollte ein zwei-gliedriger Manipulator, dessen zweites Glied in der Anfangsposition vertikal nach unten zeigen sollte, mit Modulen aus der Bibliothek modelliert werden. Dies wurde erreicht, indem die Initialisierungsfunktion des zweiten Gelenkmoduls (Abb. 4.8) leicht verändert wurde. Weitere Details dazu sind dem Anhang ab Seite 61 zu entnehmen. Die Visualisierung des Modells setzt SimMechanics aus äquivalenten Ellipsen[16] zusammen.

Abb. 4.10 zeigt der Manipulator in seiner Endposition, nachdem der Bewegungsvorgang erfolgreich ausgeführt wurde (beide Glieder sollten auf 45°

[16] Aus den Schwerpunkten und Trägheitstensoren berechnete und folglich in Bezug auf diese zwei Eigenschaften äquivalente Ellipsen

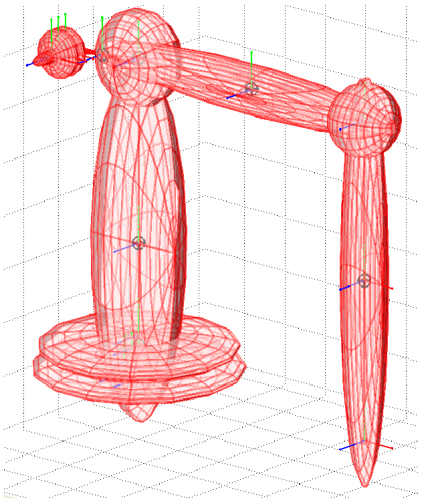


Abbildung 4.9: Visualisierung des zwei-gliedrigen Modells: Anfangsposition

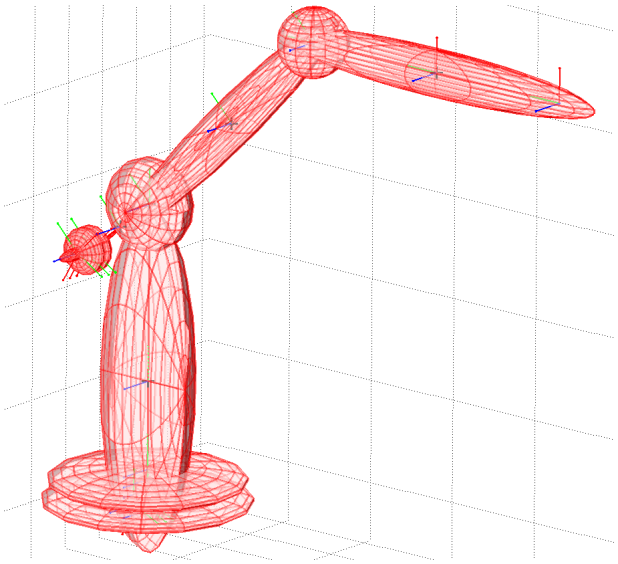


Abbildung 4.10: Visualisierung des zwei-gliedrigen Modells: Endposition

gefahren werden).

Mit Sensoren ist es möglich, verschiedene Modellparameter während des Bewegungsvorgangs abzulesen. Abb. 4.11 zeigt den Winkelverlauf der Schulter, des Ellbogens und der Motoren und darüberhinaus das Moment, das der jeweilige Motor aufbringen muss, um die Glieder auf die gewünschte Position zu bringen. Diese Verläufe hängen stark von den eingestellten Regelungsparametern, die die Motoren steuern, ab.

In diesem Beispiel ist die Regelung keinesfalls optimal und, da die Bestimmung der für ein bestimmtes Modell und einen bestimmten Bewegungsvorgang optimalen Regelung eine interessante Aufgabe und wichtige Einsatzmöglichkeit des Simulationsmodells darstellt, wird ein ähnliches, vereinfachtes Optimierungsproblem in Kapitel 5 betrachtet und gelöst.

4.1.4.2 Modell eines drei-gliedrigen Manipulators

In Abb. 4.12 ist der Aufbau eines Simulationsmodells eines drei-gliedrigen Manipulators zu sehen. In diesem Beispiel wurden keine Veränderungen an den Modulen vorgenommen, daher die gleiche Orientierung und Abmessung der drei Gelenkmodule in Abb. 4.13. Der Vergleich zwischen dem zwei-gliedrigen und diesem drei-gliedrigen Modell, sollte zeigen, wie ein *n-gliedriger* Manipulator sinngemäß, durch ein Zusammenhängen von n-Gelenkmodulen, problemlos aufgebaut werden könnte.

Details zu den Modulen und deren Schnittstellen sind ausführlich auf der jeweiligen Hilfeseite erläutert und, durch einen einfachen Rechtsclick, abrufbar.

4.1.5 Automatische Installation der Bibliothek

Um die Benutzbarkeit zu sichern, sollte die Installation und Verwendung der Bibliothek möglichst einfach und intuitiv sein. Zu diesem Zweck gibt es eine *Start-Up-Funktion*, die die Bibliothek zu dem unter Windows verfügbaren *Simulink-Library-Browser* hinzufügt und den Browser startet. So braucht der Benutzer lediglich die Start-Up-Funktion auszuführen und bekommt eine graphische Baumstruktur, die die Bibliothek wiederspiegelt und alle verfügbaren Module anzeigt. Außerdem wird für die *Session*[17] der MATLAB-Pfad[18] um das Verzeichnis[19] ergänzt, welches alle nötigen Initialisierungsfunktionen

[17] Sitzung: Zeitraum vom Starten bis zum Beenden von MATLAB.

[18] Der MATLAB-Pfad enthält alle Verzeichnisse, die von MATLAB durchsucht werden, um Funktionsnamen aufzulösen.

[19] Im Framework *„lib“* genannt.

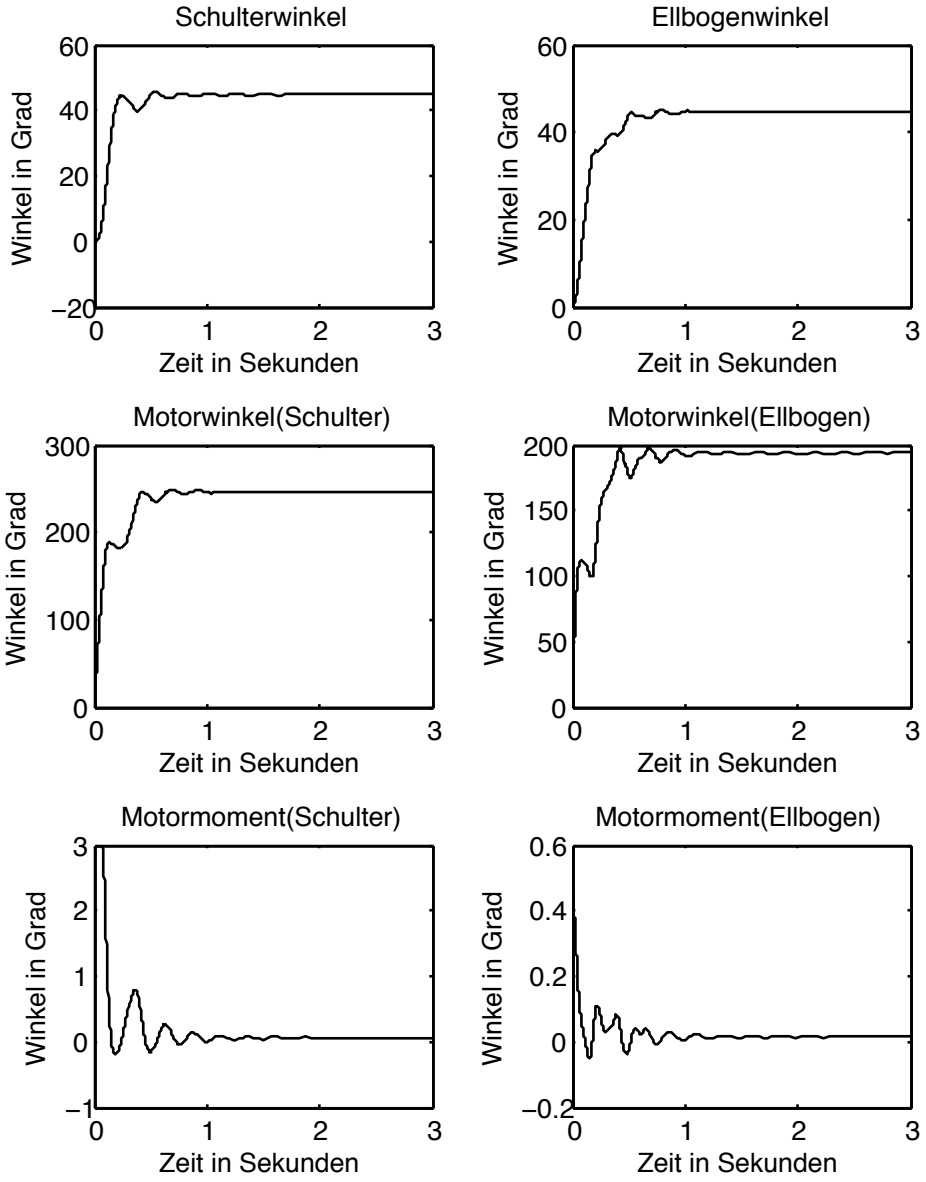


Abbildung 4.11: Verläufe verschiedener Modellparameter für den Bewegungsvorgang (des zwei-gliedrigen Modells).

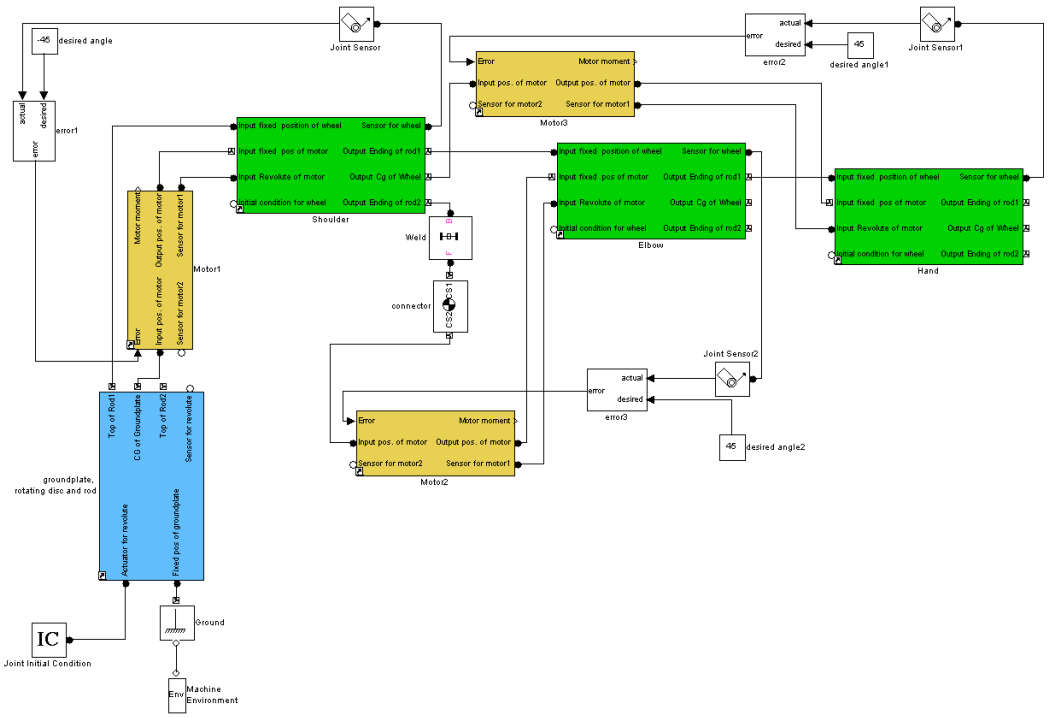


Abbildung 4.12: Blockdiagramm des drei-gliedrigen Modells.

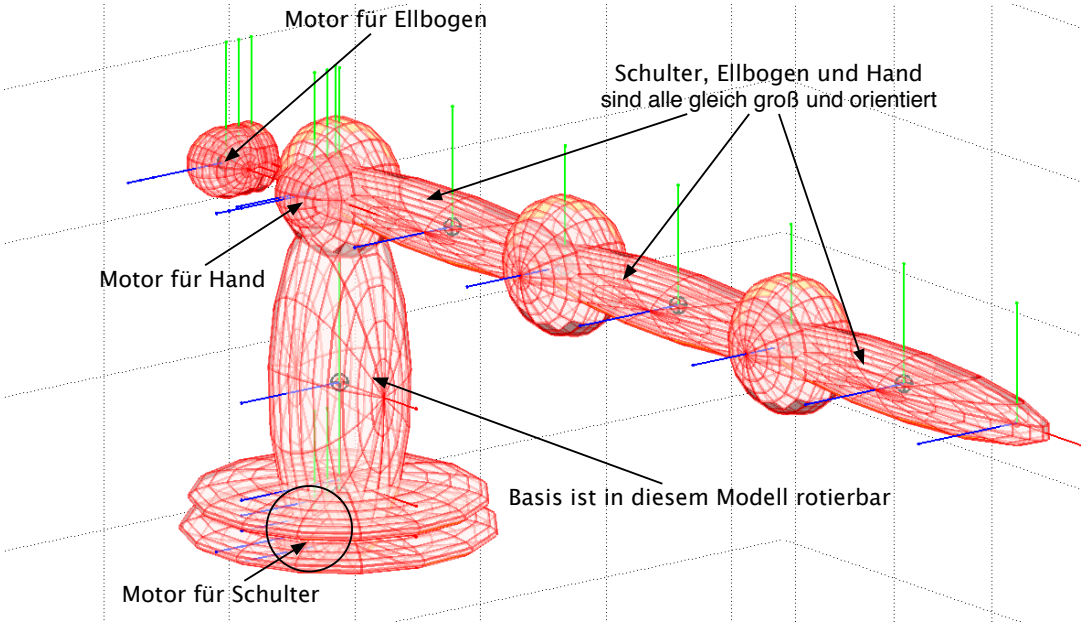


Abbildung 4.13: Visualisierung des drei-gliedrigen Modells.

enthält.

Somit ist es möglich, alle Module wie normale eingebaute Blocks zu verwenden, ohne sich Gedanken darüber machen zu müssen, ob die Implementierungen von MATLAB gefunden werden oder nicht[20]. Um die Beispiele jedoch auszuführen, müssen diese sich entweder im aktuellen Verzeichnis befinden oder manuell vom Benutzer dem MATLAB-Pfad hinzugefügt werden.

Weitere Details zum Verzeichnisaufbau und was in welchen Unterverzeichnissen eingeordnet sein sollte, befinden sich im Anhang ab Seite 61.

4.2 Diskussion

Die hier vorgestellte Bibliothek bietet eine modulare, erweiterbare, flexible und wiederverwendbare Lösung an. Bei jeder Entscheidung wurden alle Anforderungen und priorisierten Qualitätsmerkmale miteinbezogen. In den folgenden Abschnitten werden die wichtigsten kritischen Entscheidungen und die Alternativen, die in Betracht gezogen wurden, kurz erläutert.

4.2.1 Alternative Modellierungen

Wie in dem alten Modell in Adams ist es möglich, die elastischen Aktuatoren als ein elastisches Gelenk zu modellieren. Da elastische Gelenke bei der Modellierung starrer Manipulatoren sehr oft eingesetzt werden[21], gibt es schon einen entsprechenden Block in SimMechanics. Die Vorteile liegen auf der Hand: erstens müsste nichts Neues implementiert und – noch wichtiger – getestet werden, zweitens wäre die Berechnung mit einem solchen eingebauten Block effizienter. Leider ist es aber damit nicht möglich, die Steifigkeit der Aktuatoren *unabhängig* voneinander zu verändern.

Es wurde kurz in Erwägung gezogen, eine Lösung unter Benutzung *zweier* gekoppelter elastischer Gelenke zu erarbeiten, wurde aber wegen der erhöhten Komplexität und des fraglichen Effizienzgewinns verworfen.

Eine andere Lösung wäre, das Federmodul durch einen eingebauten Feder-Dämpfer-System-Block – mit den gleichen Vorteilen wie oben beschrieben –

[20] Was zu einem sogenannten *„bad-link“* führt.

[21] Um die wenige aber dennoch vorhandene Elastizität in diesen starren Systemen zu berücksichtigen.

zu ersetzen. Da aber dieser *„Body Spring & Damper Block“* die elastische Verbindung zweier *Körper* in *drei* Dimensionen vorsieht, wäre es, als schösse man mit einer Kanone auf Spatzen. Es müssten, um diesen Block einsetzen zu können, zwei Körper entsprechend der Rotation des Antriebsrades und der Scheibe bewegt werden, was umständlich und keine direkte Abbildung der Realität ist. Außerdem können die Aktuatoren keine negative Kraft aufbringen, was mit den eingebauten Blöcken nur schwer nachzubilden wäre.

Der gewählte Ansatz im Abschnitt 4.1.2.1 hatte den Nachteil, dass eine neue Implementierungs- und Testphase erforderlich war, war aber dadurch flexibel und führte letztendlich zu der einfachsten, intuitivsten Lösung.

4.2.2 Erweiterbarkeit vs. Komplexität

An vielen Stellen gab es die Möglichkeit, die Erweiterbarkeit der Module noch weiter zu unterstützen, indem durch eine Auswahl[22] unter anderem die Orientierung und Rotationsachsen automatisch anzupassen wären. Auch wenn dies verlockend klingt, war die Anzahl an Möglichkeiten überwältigend. Außerdem wird für jede unterstützte Veränderung die Implementierung der Initialisierungsfunktionen immer komplexer.

Aufgrunddessen wurde zu Gunsten der Verständlichkeit und niedriger Komplexität der Versuch, Erweiterungen automatisch zu unterstützen, verworfen und stattdessen wurden die Module möglichst einfach und für Erweiterungen offen gehalten.

[22] Als Ergänzung der Eingabemaske bzw. des Einstellungsdialogs des Moduls.

Kapitel 5

Validierung

In den folgenden Abschnitten werden, anhand von Ergebnissen gezielter Simulationen, wichtige Aspekte des Verhaltens eines mit Modulen der Bibliothek erstellten Simulationsmodells validiert.

Einfachheitshalber wird entweder mit einem zwei-gliedrigen oder mit einem ein-gliedrigen Manipulatormodell ohne Vorspannung[1] gearbeitet. Erzielte Ergebnisse sind dennoch auf ein n-gliedriges Modell mit beliebiger Vorspannung und auf andere Bewegungsvorgänge leicht übertragbar.

Für das Nachvollziehen und Nachrechnen der Ergebnisse – und auch später für die Optimierung – werden die folgenden Gleichungen benutzt, die sich auf Abb. 5.1 beziehen, die die auf den Manipulator wirkenden Kräfte schematisch darstellt. Für das einfachere, ein-gliedrige Modell können dieselben Gleichungen verwendet werden, indem entsprechende Teile gleich Null gesetzt werden.

Ohne Vorspannungen und für diesen bestimmten Bewegungsvorgang (erstes Glied wird auf 45° hochgefahren) ist die untere Feder am Ende der Simulation entspannt. Daher kann die entsprechende Federkraft aus den Gleichungen weggelassen werden, was die Übersicht verbessert, ohne die Allgemeinheit

[1] Eine Vorspannung entsteht, wenn die Federn schon vor dem Simulationslauf so vorgespannt sind, dass die jeweilige Federkraft zu Beginn der Simulation positiv wird.

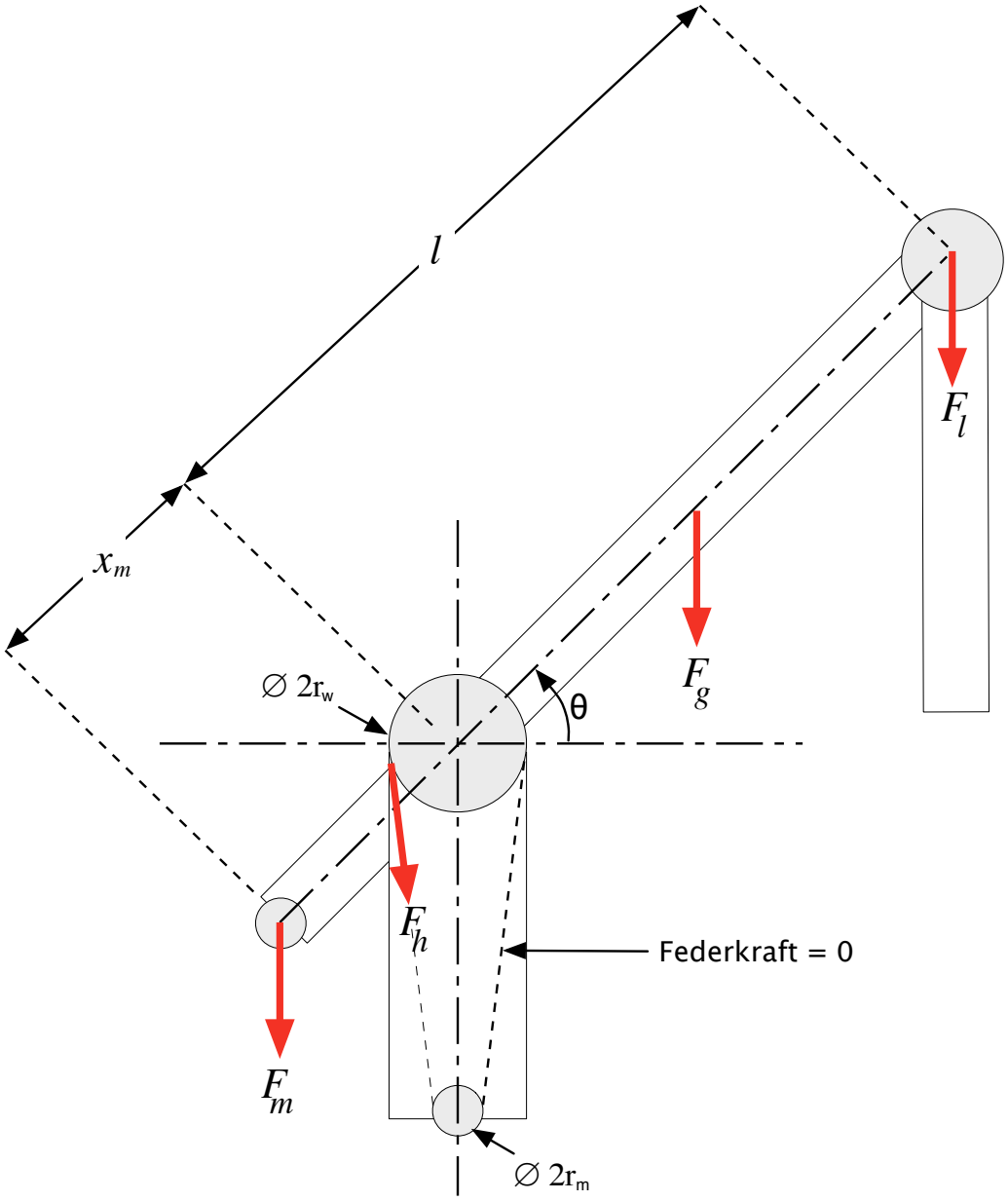


Abbildung 5.1: Schematische Darstellung der auf den Manipulator wirkenden Kräfte.

der Rechnungen zu beschränken.

$\theta :=$ Gelenkwinkel
$q :=$ Motorwinkel
$l :=$ Länge des ersten Gliedes
$r_w :=$ Radius der Scheibe
$r_m :=$ Radius des Motors für das erste Glied
$x_m :=$ Abstand des zweiten Motors vom Scheibenmittelpunkt
$F_h :=$ Haltekraft
$F_m :=$ Gewicht des Motors für das zweite Glied
$F_g :=$ Gewicht des ersten Gliedes (Ohne Scheibe)
$F_l :=$ Das gesamte Gewicht, das an dem ersten Glied hängt

Haltekraft $F_h(\theta)$ bei vorgegebenem Gelenkwinkel θ:

$$F_h r_w + F_m x_m cos\theta = F_g \frac{l}{2} cos\theta + F_l l cos\theta \tag{5.1}$$

$$F_h(\theta) = \frac{cos\theta}{r_w}(F_g \frac{l}{2} + F_l l - F_m x_m) \tag{5.2}$$

Haltekraft bzw. Federkraft $F_h(\delta x)$, die durch die Ausdehnung der Feder um δx zustande kommt:

$$F_h(\delta x) = \delta x K = \frac{\pi}{180^\circ}(q r_m - \theta r_w)K \tag{5.3}$$

Aus 5.2 und 5.3 lässt sich für einen gegebenen Gelenkwinkel θ der benötigte Motorwinkel q berechnen, der die notwendige Haltekraft aufbringt:

$$q = \frac{\frac{F_h(\theta)}{K}\frac{180^\circ}{\pi} + \theta r_w}{r_m} \tag{5.4}$$

Die Validierungsphase gliederte sich in zwei Iterationen. In der ersten wurde die *Richtigkeit* einiger Simulationsmodelle nachgeprüft – teils durch sinnvolle Vergleiche und Argumente, teils durch analytisches Nachrechnen. In der zweiten Iteration wurde ein einfaches Optimierungsproblem motiviert, aufgestellt und gelöst, um *die Eignung der Simulationsmodelle für Optimierung* zu demonstrieren.

5.1 Richtigkeit

Um die Richtigkeit des Verhaltens eines Simulationsmodells zu validieren, ist eine Aufteilung in zwei Teilaufgaben zweckmäßig. Als erstes wird das ungeregelte Verhalten untersucht: Dies bedeutet, dass die Motoren „ausgeschaltet“ bzw. festgehalten werden. Als nächstes wird das Verhalten des geregelten Simulationsmodells nachgeprüft: Hier sind alle Ergebnisse zwar von der gewählten Regelung und dem genauen Bewegungsvorgang abhängig, aber für einige Aspekte dennoch aussagekräftig.

5.1.1 Ungeregeltes Verhalten

In den nächsten Teilabschnitten 5.1.1.1 und 5.1.1.2 wird ein zwei-gliedriges Simulationsmodell aus seiner Ausgangslage, unter seinem Eigengewicht und der Wirkung der Schwerkraft, fallen gelassen. Dabei wird die Wirkung der Motoren deaktiviert - vorstellbar als ein Durchtrennen der Seilzüge. Schließlich werden in 5.1.1.3 die Motoren festgehalten und dieselbe Simulation mit einem ein-gliedrigen Modell, unter Dämpfung und unterschiedlichen Federkonstanten, ausgeführt.

5.1.1.1 Ungedämpfte Schwingung

Wenn bei der Simulation die Dämpfung[2] auch noch auf Null gesetzt wird, entspricht der Manipulator im Großen und Ganzen einem einfachen Doppelpendel. In SimMechanics gibt es als Beispiel ein vorgefertigtes Simulationsmodell eines physikalischen Doppelpendels. Dieses Modell wurde leicht verändert (Abmessungen, Trägheitstensoren, Gewicht und Schwerpunkte wurden so gut wie möglich dem Manipulatormodell angepasst) und dessen Verhalten als Vergleich genommen (Abb. 5.2).

5.1.1.2 Gedämpfte Schwingung

Wenn die gleiche Simulation unter Dämpfung ausgeführt wird, schwingt das System aus und kommt nach einigen Sekunden zur Ruhe. Abb. 5.3 zeigt die Winkelverläufe unter schwacher und etwas stärkerer Dämpfung.

[2]Sowohl in den Drehgelenken als auch in den Federn.

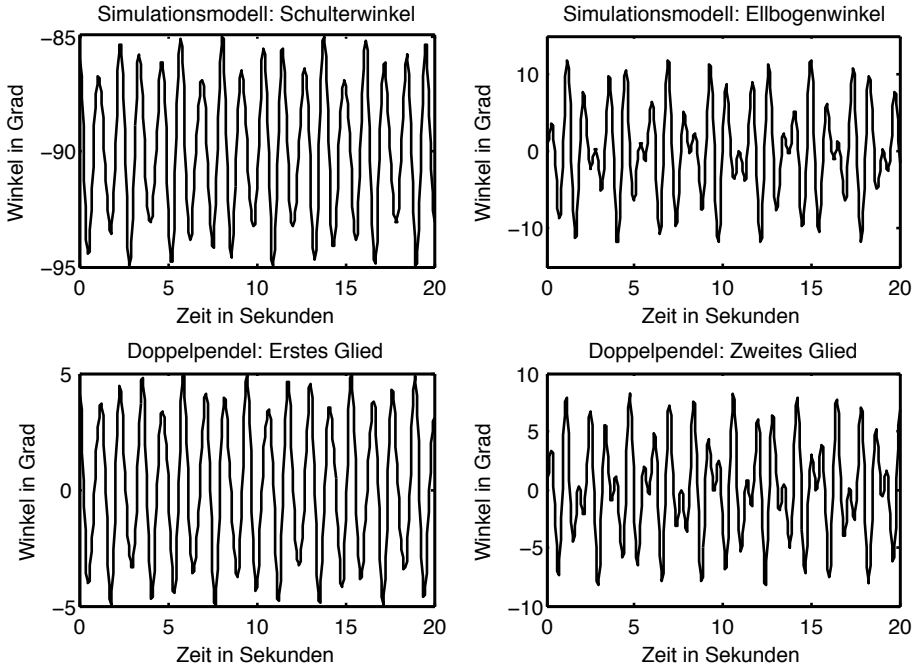


Abbildung 5.2: Vergleich des Modells mit einem Doppelpendel.

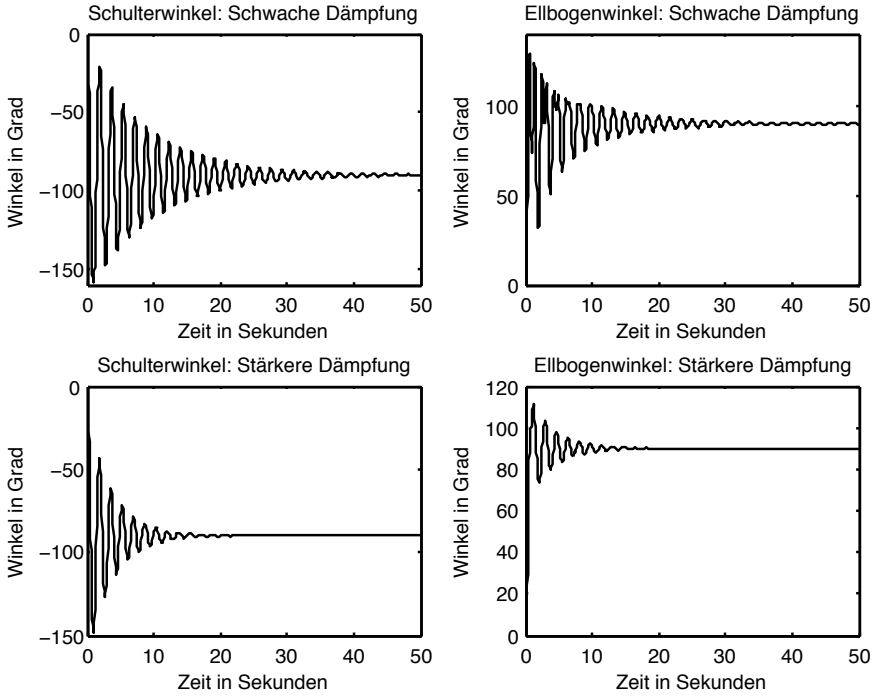


Abbildung 5.3: Verhalten des Modells unter Dämpfung.

5.1.1.3 Ausschwingen bei festgehaltenem Motor und unterschiedlichen Federkonstanten

Werden die Motoren festgehalten und der Manipulator wieder unter seinem Eigengewicht und Wirkung der Schwerkraft fallen gelassen, so schwingt das System, bis es eine Endposition erreicht, in der die Haltekraft F_h (Abb. 5.1) in der oberen Feder groß genug ist (5.2), aus. Übersichtlichkeitshalber wurde die Simulation mit einem ein-gliedrigen Modell, unter sehr starker Dämpfung, ausgeführt.

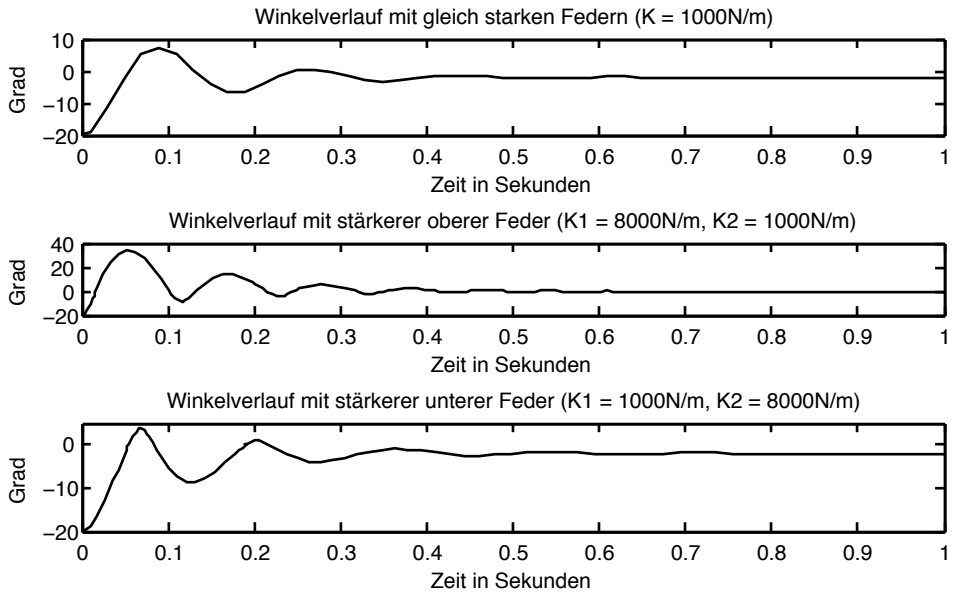


Abbildung 5.4: Endposition bei unterschiedlichen Federkonstanten.

Wenn die Federkonstanten variiert werden, ändert sich die Endposition entsprechend. Abb. 5.4 zeigt dies für einen Aufbau mit gleich starken Federn, einen mit stärkerer oberer Feder und einen mit stärkerer unterer Feder.

5.1.1.4 Diskussion

In Abb. 5.2 ist klar zu erkennen, dass das Verhalten des Modells, ohne Dämpfung und Regelung, dem eines Doppelpendels entspricht. Auch wenn die Plots nicht perfekt übereinstimmen, ist festzustellen, dass die harmonischen Funktionen, bis auf eine Phasenverschiebung und vielleicht einige Terme, identisch

sind. Die Unterschiede sind dadurch zu erklären, dass das Doppelpendel lediglich eine Vereinfachung des Modells ist und unter anderem die Zusammensetzung des Manipulators aus verschiedenen Körpern oder die Motoren, die ja noch mitschwingen, nicht berücksichtigt.

In Abb. 5.3 schwingt - wie bei einer Dämpfung zu erwarten - das System nach einigen Sekunden aus. Bei einer stärkeren Dämpfung ist die Amplitude der Schwingungen kleiner und die Auspendelzeit kürzer als bei einer schwachen Dämpfung.

In Abb. 5.4 ist das Ausschwingen des Modells bei unterschiedlichen Federkonstanten zu sehen. Es ist nachvollziehbar, dass bei einer stärkeren oberen bzw. unteren Feder die Auslenkungen nach unten bzw. oben kleiner sind.

Darüberhinaus, lässt sich die jeweilige Endposition analytisch nachprüfen:

$$\begin{aligned} q &= 0^\circ \text{ (Motoren werden festgehalten)} \\ r_w &= 0,04m \\ F_g &= 0,04 \cdot 9,81N \text{ (Masse des Gliedes)} \\ F_l &= 0N \text{ (Keine Last bzw. kein zweites Glied)} \\ F_m &= 0N \text{ (Kein zweiter Motor)} \\ l &= 0,28m \end{aligned}$$

Für gleich starke Federn ist die Endposition $\theta = -1,959^\circ$ und die Federkonstante $K = 1000N/m$. Da die Motoren festgehalten werden, lässt sich die Federkraft nach (5.3) mit δx aus der Simulation ausrechnen:

$$F_h(\delta x) = 1,3676N \tag{5.5}$$

Die nötige Haltekraft beträgt nach (5.2):

$F_h(\theta) := F_{soll} = 1,3726N$ in guter Übereinstimmung mit (5.5) aus der Simulation.

Analog ergibt sich für den Aufbau mit stärkerer oberer Feder:

$$\begin{aligned} \theta &= -0,2576^\circ \\ K_{oben} &= 8000N/m \\ K_{unten} &= 1000N/m \\ \rightsquigarrow F_{soll} &= 1,3734N \\ \rightsquigarrow F_h(\delta x) &= 1,4387N \text{ aus der Simulation.} \end{aligned}$$

Und für den Aufbau mit stärkerer unterer Feder:

$$\begin{aligned} \theta &= -1,974° \\ K_{oben} &= 1000N/m \\ K_{unten} &= 8000N/m \\ \rightsquigarrow F_{soll} &= 1,3726N \text{ (Wie zu erwarten genau wie in (5.5)).} \\ \rightsquigarrow F_h(\delta x) &= 1,3781N \text{ aus der Simulation.} \end{aligned}$$

5.1.2 Geregeltes Verhalten

In den folgenden Teilabschnitten 5.1.2.1 und 5.1.2.2 wird ein ein- bzw. zweigliedriger Manipulator auf 45° geregelt gebracht. Die Regelungskonstanten werden für jedes Szenario mit einem Optimierungsverfahren bestimmt (Abschnitt 5.2).

In der anschließenden Diskussion 5.1.2.3 werden für jeden Aufbau die Winkelverläufe des Motors und des Gliedes bzw. der Glieder, sowie der Verlauf des vom Motor aufgebrachten Moments nachvollzogen. Die Endposition des *Motors* wird dieses Mal analytisch nachgeprüft.

5.1.2.1 Unterschiedliche Federkonstanten

Abb. 5.5 zeigt die Ergebnisse eines Simulationslaufs mit schwachen Federn. Als Vergleich dazu wurde auch ein Simulationslauf mit starken Federn ausgeführt (Abb. 5.6).

Eine interessante Fragestellung ist noch, wie das Modell sich verhält, wenn unterschiedlich starke Federn kombiniert werden. Wie in Abb. 5.4 zu sehen, ist das Verhalten sicherlich anders. Die Frage ist jetzt, welches Szenario für diesen Bewegungvorgang „günstiger“ sei, denn auch wenn Ergebnisse von der Regelung abhängig sind, könnte es eventuell möglich sein - wenn für jeden Aufbau die optimale[3] Regelung genommen würde - dennoch eine diesbezügliche Aussage zu treffen.

In Abb. 5.7 sind Ergebnisse sowohl für einen Aufbau mit stärkerer oberer Feder als auch für einen mit stärkerer unterer Feder zu sehen.

[3] Wenigstens eine gute Approximation dessen konnte mit einem Optimierungsverfahren ermittelt werden.

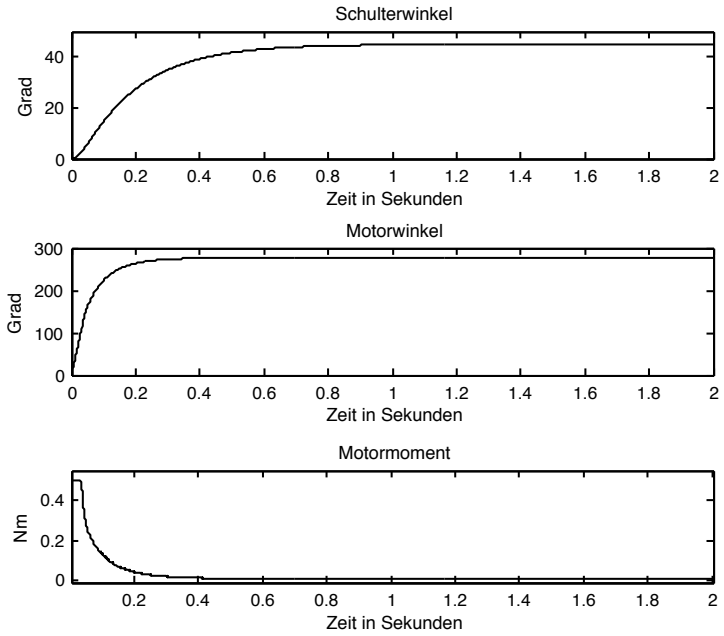


Abbildung 5.5: Verhalten für Aufbau mit schwachen Federn im Zeitinterval [0 2]. Sollwert für die Regelung ist 45° (für Schulterwinkel).

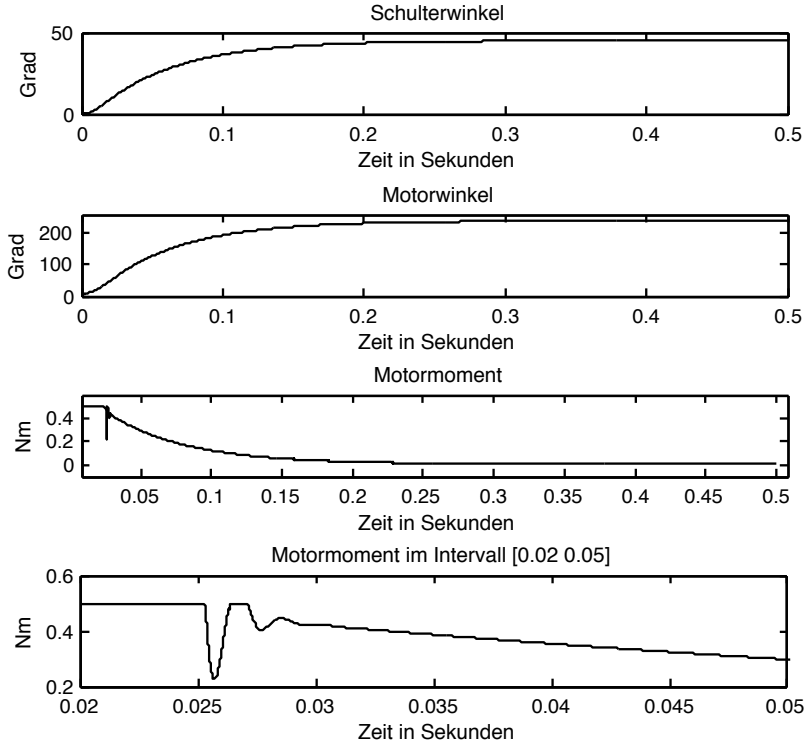


Abbildung 5.6: Verhalten mit starken Federn im Zeitinterval [0 0,5].

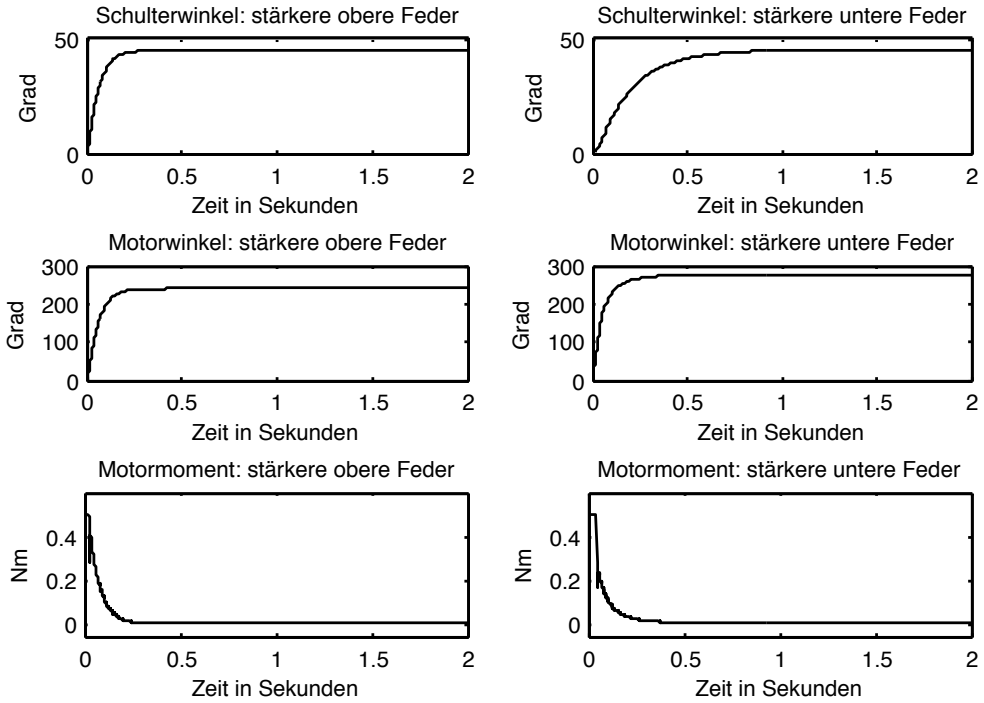


Abbildung 5.7: Vergleich für Kombinationen verschieden starker Federn.

5.1.2.2 Unterschiedlich große Lasten

Um auch ein zwei-gliedriges Modell zu validieren, wurde das erste Glied des Manipulators auf 45° gefahren, während das zweite Glied ohne Regelung am Ende des ersten Gliedes hing. In Abb. 5.1 ist genau dieser Aufbau schematisch dargestellt. Das zweite Glied ist also in diesem Szenario als leichte Last für den Manipulator aufzufassen. Abb. 5.8 zeigt die Resultate eines Simulationslaufs.

Abb. 5.9 zeigt das Verhalten des ein-gliedrigen Modells unter einer schweren Last, die am Ende des Gliedes „getragen“ wird.

5.1.2.3 Diskussion

Mit steigenden Federkonstanten wird die Eigenfrequenz des Systems immer höher. Dies bedeutet, dass die Regelung entsprechend komplizierter werden muss, um die hochfrequenten Schwingungen auszugleichen. Daher der kompliziertere Momentenverlauf in Abb. 5.6 im Vergleich zu dem glatten Momentenverlauf in Abb. 5.5. Dafür muss sich der Motor bei starken Federn

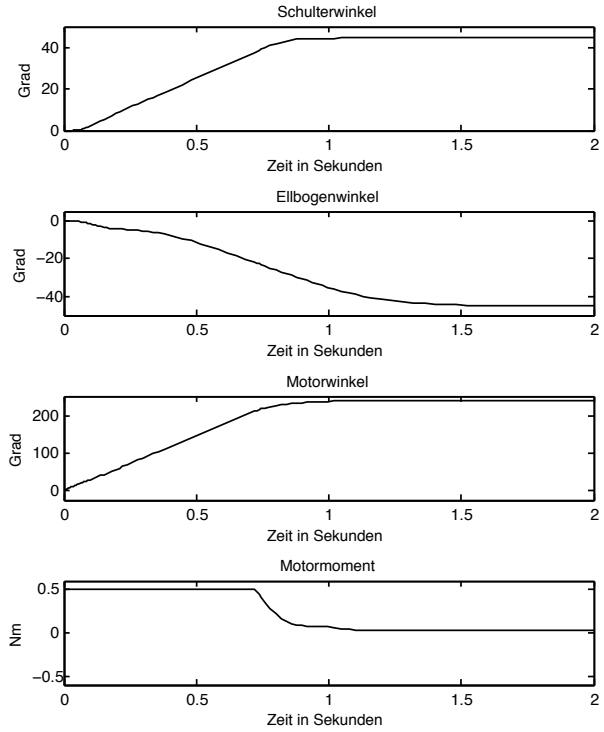


Abbildung 5.8: Verhalten eines zwei-gliedrigen Manipulators mit zweitem Glied als Last.

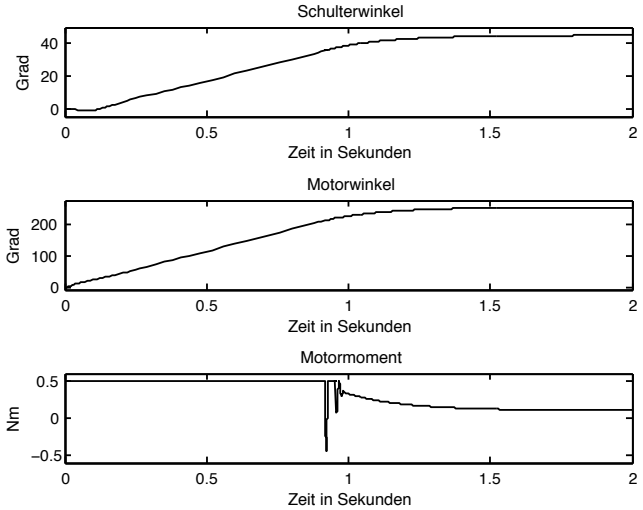


Abbildung 5.9: Verhalten eines ein-gliedrigen Manipulators unter schwerer Last.

nicht mehr so weit drehen, um die nötige Kraft in den Federn zu erzeugen. Außerdem können stärkere Federn, mit einer höheren Frequenz, dem Motor schneller nachschwingen. Daher beträgt die Anregelzeit[4] bei starken Federn nur etwa 0,3s, während bei schwachen Federn das System erst nach etwa 0,75s zur Ruhe kommt.

Dabei ist aber zu beachten, dass die Motoren des Simulationsmodells *sofort* auf die Regelung reagieren, was ein optimales Verhalten auch bei sehr starken Federn ermöglicht. Realistischere Motoren brauchen ein wenig Zeit, um auf Änderungen in der Regelung anzusprechen, und würden die Kontrolle bei einem hochfrequenten System entsprechend erschweren.

Mit schwachen Federn (K = 200N/m) und nach (5.2) und (5.4), ergibt sich als Endposition des Motors:

$$\begin{aligned} r_w &= 0,04m \\ F_g &= 0,04 \cdot 9,81N \\ F_l &= 0N \\ F_m &= 0N \\ l &= 0,28m \\ \rightsquigarrow q_{soll} &= 277,0932° \end{aligned}$$

In guter Übereinstimmung mit $q = 277,0017°$ aus der Simulation.

Analog ergibt sich für K = 50000N/m, $q_{soll} = 240,1484°$ und $q = 240,1005°$.

Bei einer Kombination verschieden starker Federn sieht man daran, dass das benötigte Motormoment, um das System auszuregeln, vor allem am Anfang Spitzen aufweist (Abb. 5.7), dass das System allgemein eine höhere Eigenfrequenz hat. Da die Bewegung nach oben verlief, spielt die obere Feder eine wichtigere Rolle. Folglich ist der Verlauf für eine stärkere obere Feder entsprechend komplizierter. Dafür lässt sich, aus den gleichen Gründen wie für den Aufbau mit starken Federn, das System mit stärkerer oberer Feder schneller auf die Endposition bringen. Da die untere Feder in der Endposition entspannt ist, stimmen die Endpositionen der Motoren mit den im letzten Abschnitt nachgerechneten Werten überein.

Auch wenn die idealen Motoren unrealistisch schnell auf die Regelung reagieren, ist dennoch zu erkennen, dass für stärkere Federn die Eigenfrequenz

[4]Die Anregelzeit ist die Zeitspanne, die beginnt, wenn der Wert der Regelgröße nach einem Sprung der Störgröße oder Führungsgröße einen vorgegebenen Toleranzbereich der Regelgröße verlässt, und die endet, wenn er den oberen Toleranzbereich erstmalig wieder eintritt [8].

des Systems höher wird, was einerseits schnellere Bewegungen bei kleineren Drehungen der Motoren ermöglicht, andererseits die Regelung aber erheblich erschwert.

Die Ergebnisse des Simulationslaufes für das zwei-gliedrige Modell zeigen, dass der Motor für ca. 0,75s auf volle Leistung fahren muss, um das erste Glied auf 45° zu fahren (Abb. 5.8). Dies dauert entsprechend länger als bei den vorherigen Simulationen ohne Last. Der Ellbogen fällt dabei von 0° auf -45° und hängt senkrecht nach unten. Bei einer größeren Last (Abb. 5.9) sind die Ergebnisse ähnlich und die gleichen Unterschiede zu Simulationen ohne Last werden noch deutlicher. Der Momentenverlauf unter schwerer Last ist außerdem komplizierter und zeigt, dass die Regelung für steigende Last problematischer wird, da die Leistung der Motoren begrenzt ist. Auch nachvollziehbar ist, dass das nötige Moment, um die Endposition zu halten, deutlich höher liegt, als bei vorherigen Modellen mit keiner oder nur leichter Last.

Die Endposition des Motors für das zwei-gliedrige Modell lässt sich noch nachrechnen:

$$
\begin{aligned}
\theta &= 45° \\
l &= 0,28m \\
F_m &= 0,8 \cdot 9,81N \\
F_g &= 0,04 \cdot 9,81N \\
F_l &= (0,04 + 0,2) \cdot 9,81N \\
r_w &= 0,04m \\
r_m &= 0,0075m \\
x_m &= 0,065m \\
K_{oben} &= 8000N/m \\
K_{unten} &= 1000N/m \\
\rightsquigarrow F_h(\theta) &= 3,6071N \text{ nach (5.2).} \\
\rightsquigarrow q_{soll} &= 243,4445° \text{ nach (5.4).}
\end{aligned}
$$

In guter Übereinstimmung mit $q = 242,8861°$ aus der Simulation.

Analog gilt mit einer Last von 0,25Kg:

$$
\begin{aligned}
\rightsquigarrow F_h(\theta) &= 13,1104N \\
\rightsquigarrow q_{soll} &= 252,5195°
\end{aligned}
$$

Auch in guter Übereinstimmung mit $q = 253,3616°$ aus der Simulation.

Als letzte Bemerkung ist noch zu sagen, dass, um eine einigermaßen gute Regelung zu ermöglichen, bei den Modellen mit Last die Dämpfung leicht erhöht wurde.

5.1.3 Vergleich mit Labormodell

Ein wichtiger Aspekt der Validierung ist der Vergleich mit einem realen Modell. Im Rahmen dieser Arbeit war dies jedoch erheblich erschwert, da keine exakten Modellparameter[5] zur Verfügung standen. Auch wenn diese mit gewissem Aufwand besorgt bzw. exakt errechnet werden könnten, müsste noch die genaue Regelung des Labormodells nachimplementiert werden, bzw. es müssten neue Motormodelle, um die Motoren des Labormodells nachzubilden, ebenfalls implementiert werden. Dies würde eindeutig den Rahmen dieser Arbeit sprengen.

Da aber ein neues verändertes Labormodell demnächst entwickelt werden sollte, und die Module der Bibliothek an dessen Aufbau angelehnt werden sollten, verlor dadurch das Streben nach einem perfekten Vergleich mit dem jetzigen Labormodell an Bedeutung und Relevanz.

Dennoch wurde als Teil der Validierung ein Simulationsmodell erstellt, das dem echten Labormodell soweit wie möglich entspricht. Das Labormodell wurde abgemessen und restliche Parameter sowie Gewicht, Federkonstanten und Trägheitstensoren ohne zu großen Aufwand abgeschätzt. Das Ziel dieses Vergleichs war nur zu zeigen, dass das Simulationsmodell ein ähnliches, vergleichbares Verhalten aufweist.

Das zweite Glied des Labormodells (Ellbogen) wurde aus seiner Ausgangsposition um etwa -3,5° ausgelenkt (Abb. 5.10). Das System wurde dann bei festgehaltenen Motoren ausschwingen gelassen. Dabei bewegte sich natürlich das erste Glied entsprechend mit (Abb. 5.12).

Das Simulationsmodell wurde durch eine Vorspannung in die gleiche Ausgangsposition wie das Labormodell gebracht. Dann wurde die Auslenkung als Anfangsbedingung des zweiten Glieds gesetzt. Vergleichbarkeitshalber wurde das Zeitintervall dem des Labormodells angeglichen (Abb. 5.11 und Abb. 5.13).

[5] Schwerpunkte, Trägheitstensoren, Federkonstanten, Dämpfung.

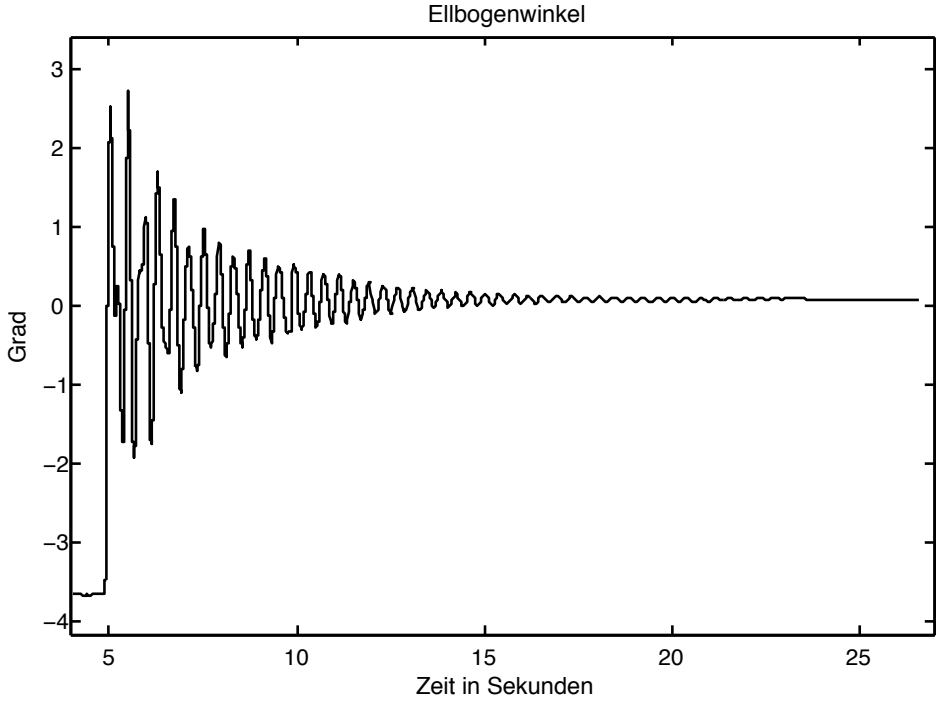


Abbildung 5.10: Verhalten des Labormodells: Zweites Glied

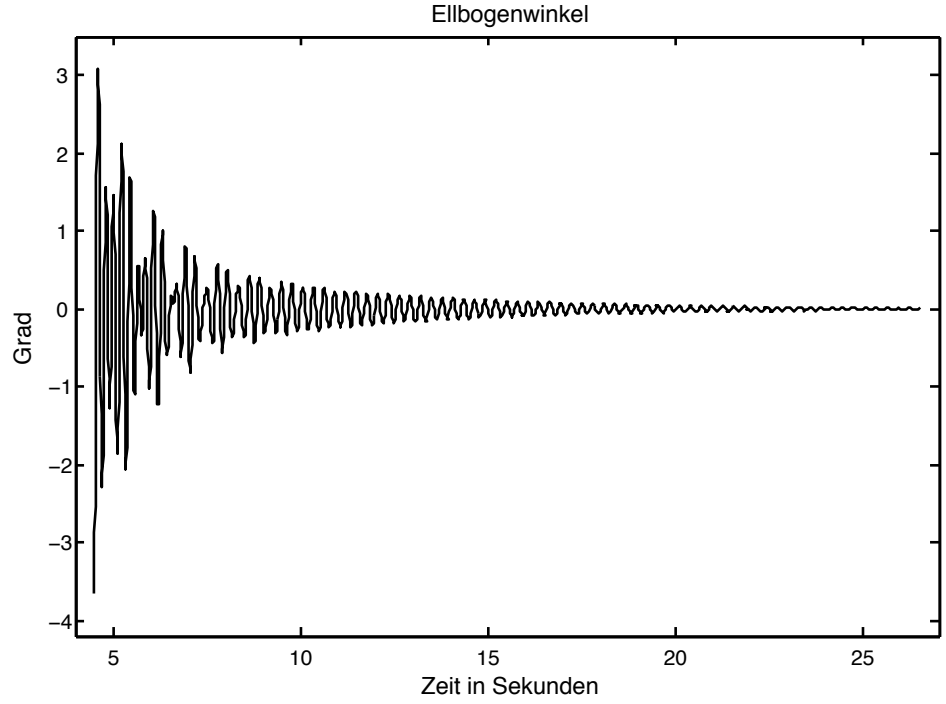


Abbildung 5.11: Verhalten des Simulationsmodells: Zweites Glied

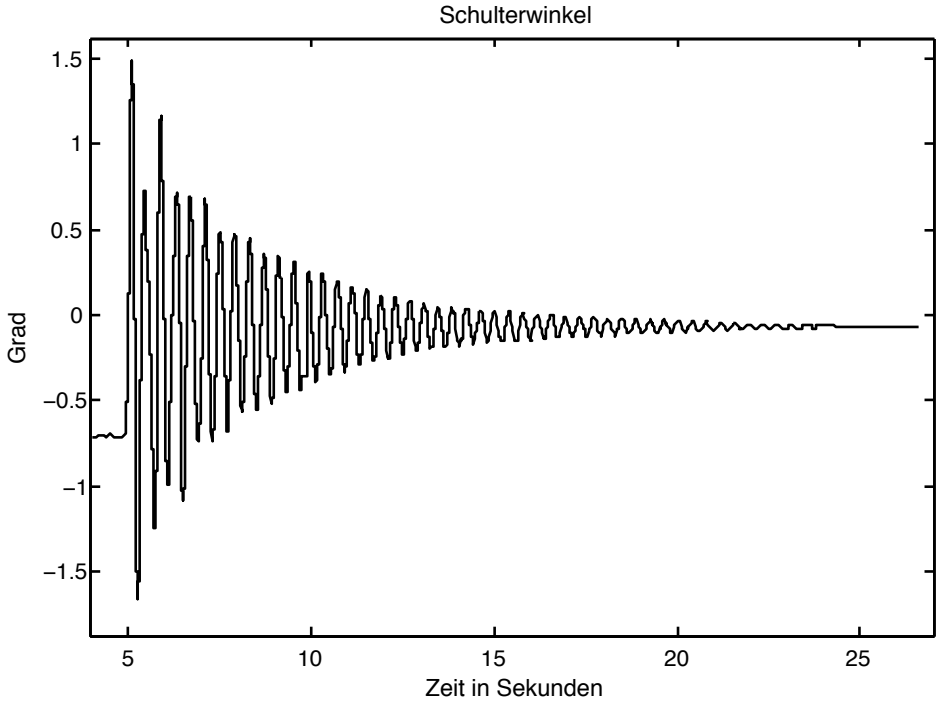


Abbildung 5.12: Verhalten des Labormodells: Erstes Glied

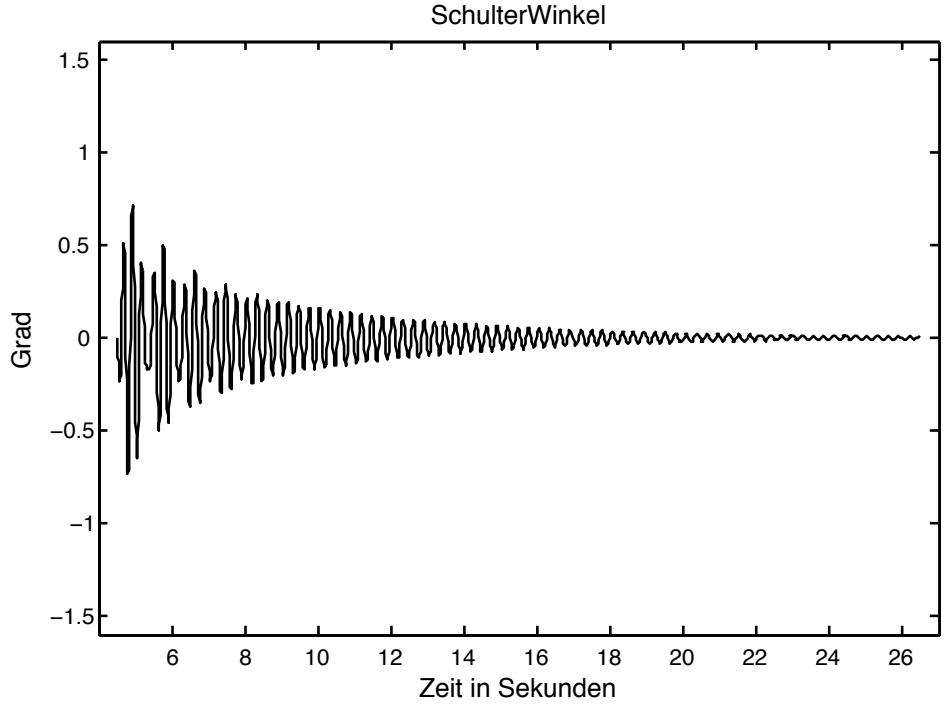


Abbildung 5.13: Verhalten des Simulationsmodells: Erstes Glied

5.1.3.1 Diskussion

Das Verhalten des Labormodells lässt sich gut mit dem des Simulationsmodells vergleichen. Für das zweite Glied, das um etwa -3,5° ausgelenkt wurde, ist bei beiden Ergebnissen die gleiche Ab- und Zunahme an Amplitude zu sehen, die auf eine Übertragung kinetischer Energie, zu und von dem ersten Glied, hindeutet.

Dass jedoch der Verlauf des zweiten Gliedes des Labormodells nicht ganz um 0° symmetrisch ist (Abb. 5.10), deutet auf eine leichte Vorspannung hin, was beim Simulationsmodell fehlt (Abb. 5.11). Möglich ist auch, dass die Federkonstanten leicht auseinanderliegen, was jedoch experimentell nicht bestätigt werden konnte, und deshalb in dem Simulationsmodell nicht eingestellt war.

Ein letztes Problem ist die Tatsache, dass reale Federn – besonders starke Federn – bei kleinen Auslenkungen noch nicht im linearen Bereich sind und sich folglich nicht gemäß des Hookeschen Gesetzes verhalten. Dies ist vielleicht der Grund für die Diskrepanz zwischen dem Verhalten des ersten Gliedes des Labormodells (Abb. 5.12), und dem des Simulationsmodells (Abb. 5.13). Es könnte aber auch an dem Fehler bei der experimentellen Bestimmung der Federkonstanten liegen, der, besonders bei solch kleinen Auslenkungen, einen großen Unterschied bewirkt. Dennoch ist, nach den ersten Ausschlägen, das Verhalten des Simulationsmodells dem des Labormodells doch sehr ähnlich.

5.2 Eignung zur Optimierung

Um zu zeigen, dass ein mit Hilfe der Bibliothek erstelltes Simulationsmodell sich für die Untersuchung und Lösung von Optimierungsaufgaben eignet, wurde ein einfaches Optimierungsproblem betrachtet und gelöst. Im Folgenden wird das Problem motiviert, die genaue Aufgabe und ein konkretes Ziel aufgestellt, die Wahl der Gütefunktion begründet, eine fundierte Auswahl an Optimierungsalgorithmen getroffen und es werden die Ergebnisse präsentiert.

5.2.1 Das Problem

Ein hohes Maß an Elastizität erfordert eine entsprechende Regelung, um unerwünschten Schwingungen entgegenzuwirken. Eine ganze Reihe an Regelungsparametern müssen für jeden Motor so ausgelegt werden, dass der

Bewegungsvorgang möglichst optimal ausgeführt wird. Das Problem dabei ist, dass diese Regelung nur für eine bestimmte Einstellung der Modellparameter, für den bestimmten Bewegungsvorgang und für einen engen Traglastbereich optimal sein wird. Sobald der Aufbau geändert wird, müssen die Regelungsparameter entsprechend angepasst werden, um die Optimalität des Verhaltens zu bewahren.

Schon bei sehr einfachen Modellen und leichten Veränderungen ist das Anpassen der Regelungsparametern eine lästige, zeitaufwendige Arbeit und nur mit viel Erfahrung noch von Hand zu bewältigen.

5.2.2 Das Ziel

Das Ziel ist es, für einen bestimmten Aufbau automatisch, durch einen Optimierungsprozess, die optimalen Regelungsparameter zu bestimmen. Die Frage, was als optimal angesehen wird, taucht sofort auf und muss geklärt werden, da die ganze Optimierung sich danach richten wird.

Natürlich möchte man so schnell wie möglich die erwünschte Postion erreichen - in diesem Fall 45°. Die Motoren unterliegen aber physikalischen Grenzen und sind weder beliebig stark, noch können sie sich unendlich schnell drehen. Außerdem ist eine schnelle Richtungsänderung für die Motoren schädlich. Die Leistung der Motoren wird bereits in dem Motormodell begrenzt, aber die Drehgeschwindigkeit nicht. Daher wäre ein *schnelles* Erreichen von 45°, bei möglichst *gleichmäßigem* Drehen der Motoren optimal.

Die nächste Frage ist, was als Optimierungsparameter zu betrachten ist. Da das Motormodell einen einfachen *pid-Regler*[6] darstellt, sind die drei Regelungsparameter (p, i und d) genau die, die es optimal zu bestimmen gilt.

5.2.3 Die Gütefunktion

Da wir den Manipulator möglichst Motor-schonend steuern wollen, liegt es nahe, den Winkelverlauf des Motors zu optimieren. Der erste Ansatz wäre,

[6]Der Positionsfehler wird mit einem proportionalen, integralen und differentialen Faktor multipliziert und das Ergebnis als Steuerung für den Motor genommen.

den einer 45° Drehung des Armes entsprechenden Winkel als Sollwert zu nehmen:

$$\begin{aligned}
\theta &= 45° \\
F_g &= 0,04 \cdot 9,81N \\
F_l = F_m &= 0N \\
l &= 0,28m \\
r_w &= 0,04m \\
r_m &= 0,0075m \\
K &= 8000N/m \\
\rightsquigarrow F_h(\theta) &= 0,9711N \text{ nach (5.2)} \\
\rightsquigarrow q_{soll} &= 240,9273° \text{ nach (5.4)}
\end{aligned}$$

Da es gilt, diesen Sollwert möglichst schnell zu erreichen, wäre die erste Näherung an eine Gütefunktion das Integral des Quadrats der Differenz zwischen Motorwinkel und Sollwert:

$$f(p,i,d) := \int_0^{t_f} (q(t) - q_{soll})^2 dt$$

Hiermit wird aber noch nicht die Geschwindigkeit des Motors berücksichtigt. Eine Möglichkeit, dies zu tun, wäre, das Quadrat der Ableitung zu minimieren. Um die verschiedenen Terme der Gütefunktion zu bewerten, werden noch zwei Konstanten eingeführt:

$$f(p,i,d) := \int_0^{t_f} [c_1 \cdot (q(t) - q_{soll})^2 + c_2 \cdot \dot{q}_m\ (t)^2] dt$$

Es hat sich als vorteilhaft herausgestellt, den Integrand um die Zeit zu ergänzen: Dies bedeutet einerseits eine große Geschwindigkeit am Anfang der Simulation (für sehr kleines t) weniger schlecht zu bewerten, andererseits Abweichungen vom Zielwert für größer werdendes t immer mehr zu bestrafen[7]. Für die Optimierung wurden folglich die folgenden Konstanten und Gütefunktion verwendet:

$$\begin{aligned}
f(p,i,d) &:= \int_0^{t_f} [c_1 \cdot (q(t) - q_{soll})^2 + c_2 \cdot \dot{q}_m\ (t)^2] \boldsymbol{t} dt \\
c_1 &= 0,1 \\
c_2 &= 0,001 \\
t_f &= 1s
\end{aligned}$$

[7] Dies schließt eine Reihe von eindeutig sub-optimalen Lösungen, bei denen sich der Motorwinkel sehr langsam dem Zielwert nähert, aus.

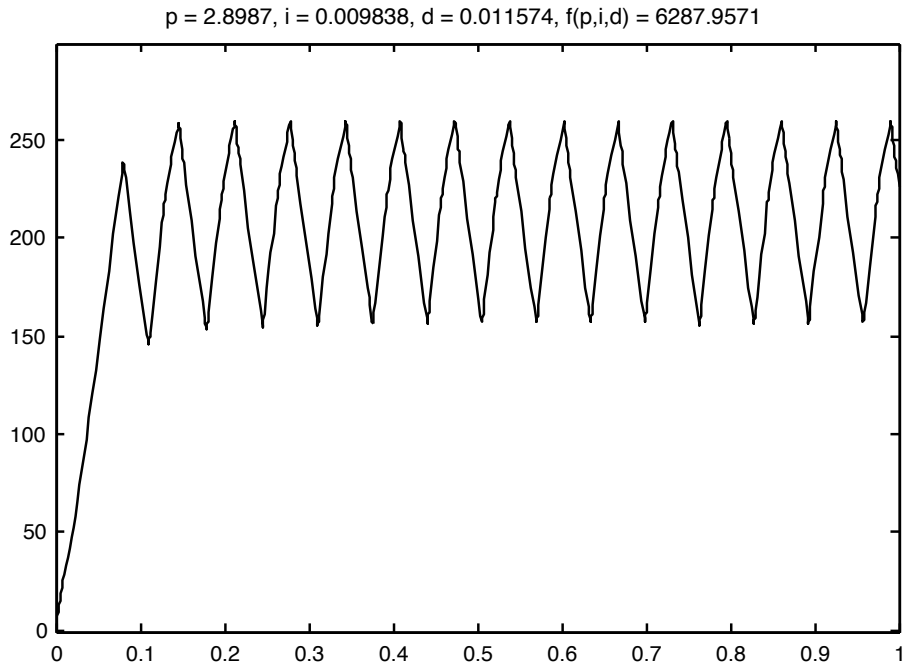


Abbildung 5.14: Unerwünschtes Verhalten: Oszilliert stark um Zielwert.

Um sicher zu gehen, dass die Gütefunktion tatsächlich das erwünschte Verhalten als optimal bewertet, wurden einige Simulationsläufe ausgeführt und die entsprechenden Werte der Gütefunktion betrachtet (Abb. 5.14, 5.15, 5.16 und 5.17).

Als nächster Schritt wurde eine grobe Visualisierung der Gütefunktion erstellt. Aus Erfahrung mit dem zu optimierenden Aufbau ist bekannt, dass gutes Verhalten für Punkte um [0,1 0,001 0,0001] zu erwarten ist. Daher wurde die Gütefunktion auf einem Gitter um diesen Punkt ausgewertet und für festes i ein 3d-Plot erzeugt (Abb. 5.18).

Wie in Abb. 5.18 zu sehen, ist die Gütefunktion in diesem Bereich deutlich konvex, was zugunsten einer Optimierung ist. Wäre die Funktion wellenförmig oder konkav, würde eine Optimierung im ersten Fall nichts bringen und wäre im zweiten Fall unnötig, da man einfach Extremwerte nehmen könnte. Um sich ein noch genaueres Bild der Funktion im „Tal“ zu machen, wurde eine zweite Visualisierung auf einem feineren, aber kleineren Gitter um die besten[8] Punkte (Abb. 5.19) erstellt. Diese zeigt, dass die Funktion auch mit

[8]Basierend auf der ersten Visualisierung.

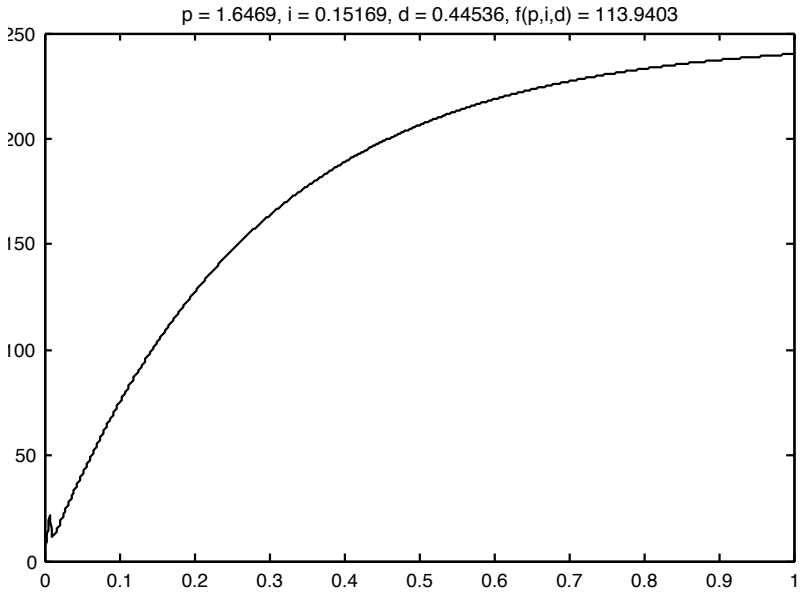


Abbildung 5.15: Unerwünschtes Verhalten: Guter, glatter Verlauf aber sehr langsam.

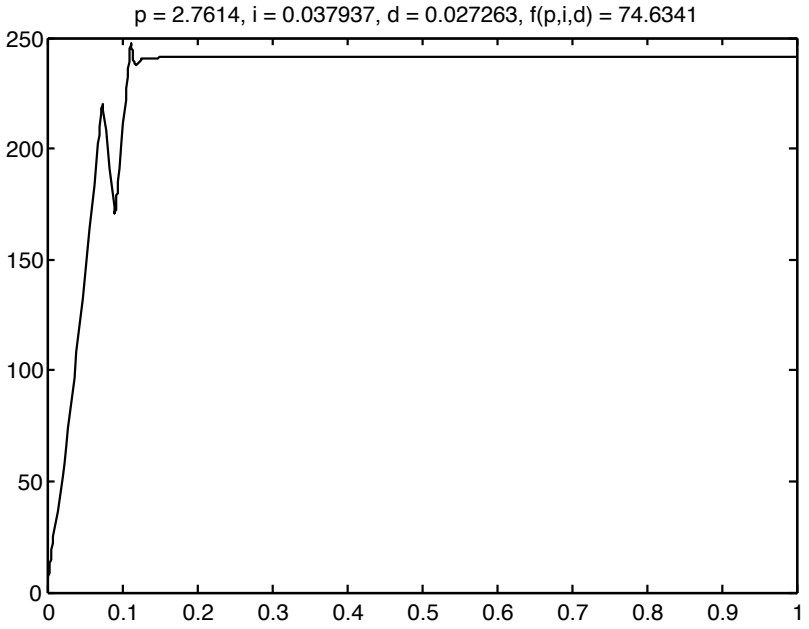


Abbildung 5.16: Unerwünschtes Verhalten: Guter, schneller Verlauf aber mit zu starken Winkelveränderungen.

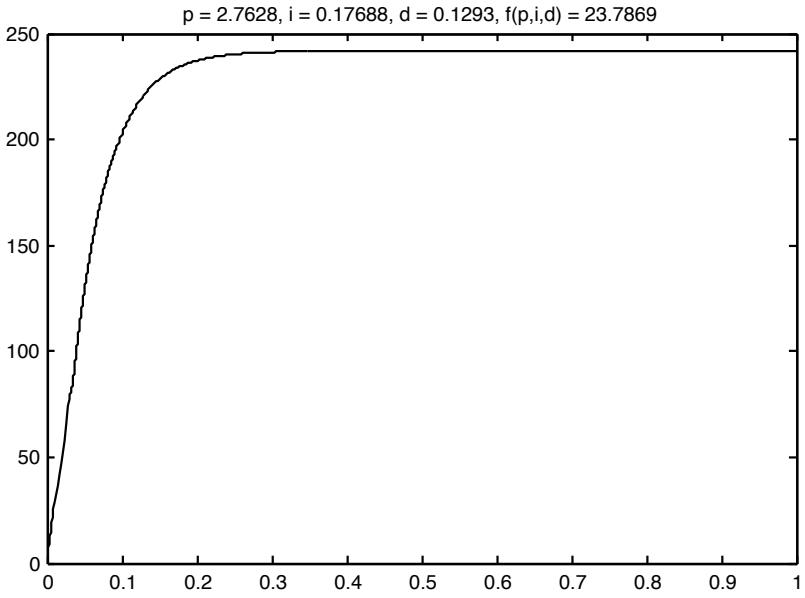


Abbildung 5.17: Erwünschtes Verhalten: Glatte Funktion, Zielwert schnell erreicht.

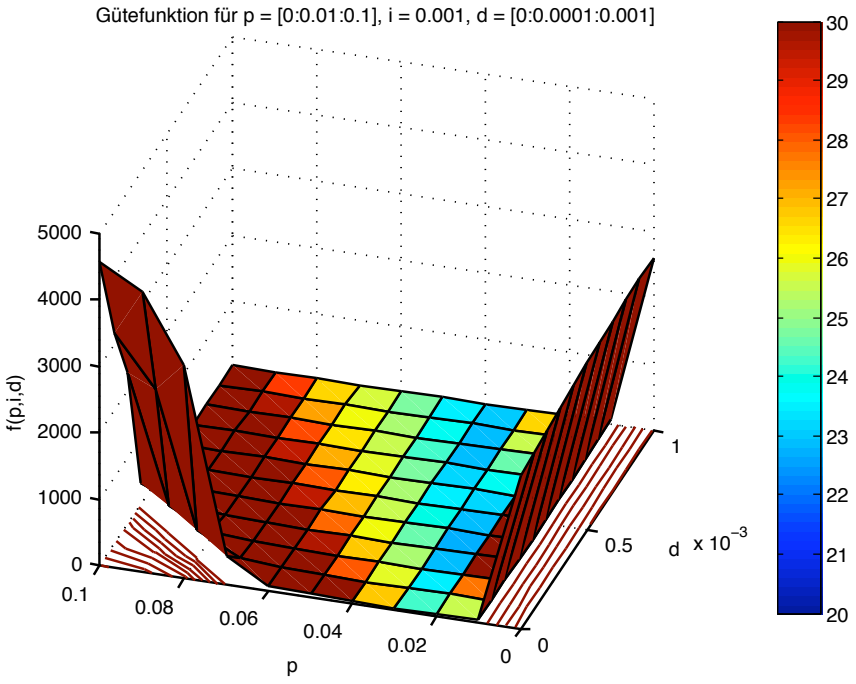


Abbildung 5.18: Visualisierung der Gütefunktion.

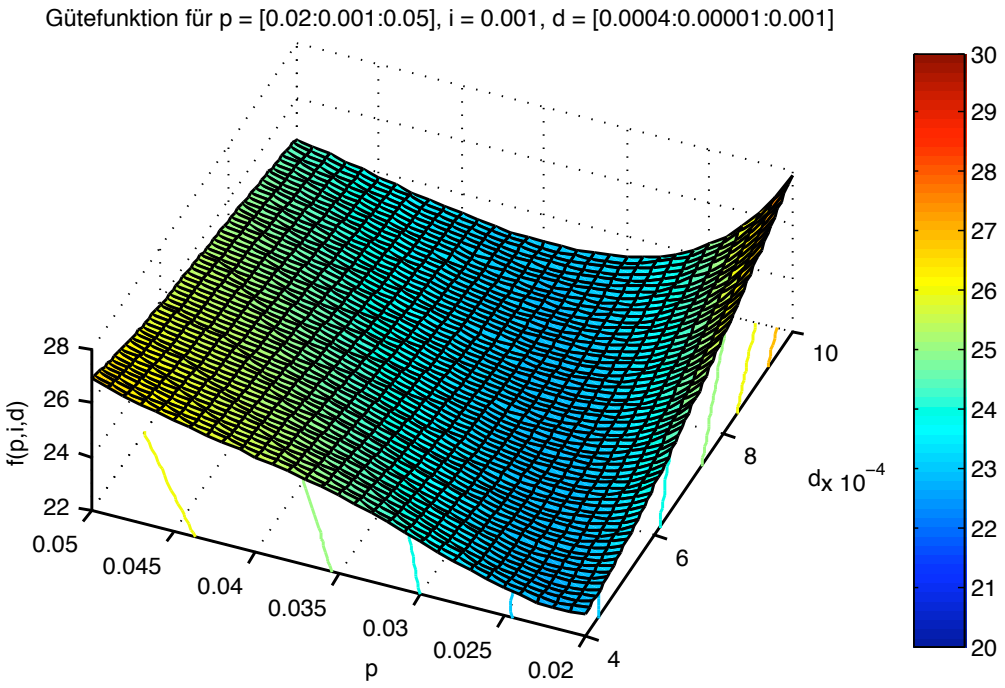


Abbildung 5.19: Visualisierung der Gütefunktion um die besten Punkte.

einer höheren Auflösung keine starken lokalen Schwankungen aufweist, was für eine Optimierung von großem Vorteil ist.

Der Visualisierung der Gütefunktion zufolge ist zu erwarten, dass man mit einem Optimierungsverfahren schnell ins Tal kommen - also nach wenigen Auswertungen[9] der Zielfunktion schon einen guten Wert erzielen - aber danach nur eine im Vergleich geringfügige Verbesserung bekommen kann (Abb. 5.19). Außerdem ist eher das *Verhältnis* zwischen den Parametern wichtig und nicht die genauen Größen, da die Funktion rillenförmige Höhenlinien besitzt und Werte für eine ganze Hyperebene fast gleich sind.

5.2.4 Wahl des Optimierungsalgorithmus

Obwohl die Gütefunktion selber stetig differenzierbar ist, hängt sie vom Motorwinkel ab, der numerisch berechnet wird. Nach einigen Versuchen, den Motorwinkel analytisch zu beschreiben, wurde dies aufgegeben und damit leider auf effiziente, Gradienten-basierte Optimierungsverfahren verzichtet.

[9] Jede Auswertung der Zielfunktion bedeutet einen Simulationslauf auszuführen.

Da das Problem aber lediglich dreidimensional ist, sollte es möglich sein, in vernünftiger Zeit eine gute Lösung mit robusten, Gradienten-freien Verfahren zu erzielen.

Es wurden *Nelder-Mead*, *Direct* und *imfil*[10] als Optimierungsverfahren ausgewählt und mit Implementierungen nach [7] eingesetzt.

5.2.5 Ergebnisse und Diskussion

Wie erwartet war es möglich, nach wenigen – etwa 20 bis 50 – Funktionsauswertungen[11] schon ein recht gutes Ergebnis zu bekommen. Nach weiteren 100 bis 140 Auswertungen und Terminierung des Optimierverfahrens konnte das Ergebnis nur leicht verbessert werden. Das Ergebnis hing dabei entscheidend vom Startwert und geringfügig vom Optimierungsverfahren ab.

Algorithmus	Startwert	Auswertungen	Optimum	Toleranz
Nelder-Mead	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	20 → 30,27 102 → 23,70	23,70	0,01
Direct	[0 1], [0 1], [0 1]	30 → 29,17 139 → 23,66 223 → 23,63	23,63	0,01
Imfil	(0, 0, 0), [0 1], [0 1], [0 1]	55 → 26,26 108 → 23,66	23,66	0,01

Obwohl das Resultat sehr gut war – etwa 23-25 jenach Startwert und Algorithmus – ist in der Visualisierung zu erkennen, dass Tiefstwerte bei etwa 22 liegen (Abb. 5.19).

Mit gezielten, auf der Visualisierung basierenden Startwerten, war es schließlich möglich, das Optimum zu bestimmen (Abb. 5.20).

Dennoch sind die Vorteile einer automatisierten Optimierung unbestreitbar. Die Gitterpunkte für die Visualisierung zu berechnen hat immerhin *zwei Tage Rechenzeit* erfordert, während mit einem der erwähnten Optimierungsverfahren[12] schon nach einigen Minuten ein recht guter Wert zu erzielen ist.

[10] Alles robuste, Gradienten-freie iterative Optimierungsverfahren, geeignet für verrauschte Funktionen - der Fall bei einem numerisch berechneten Simulationsergebnis.

[11] Für dieses Problem dauerte jede Funktionsauswertung im Durchschnitt 10s.

[12] Alle drei funktionierten vergleichbar gut. Je nach Startwert war jedoch imfil sehr empfindlich, während Nelder-Mead sich meist als sehr robust erwies.

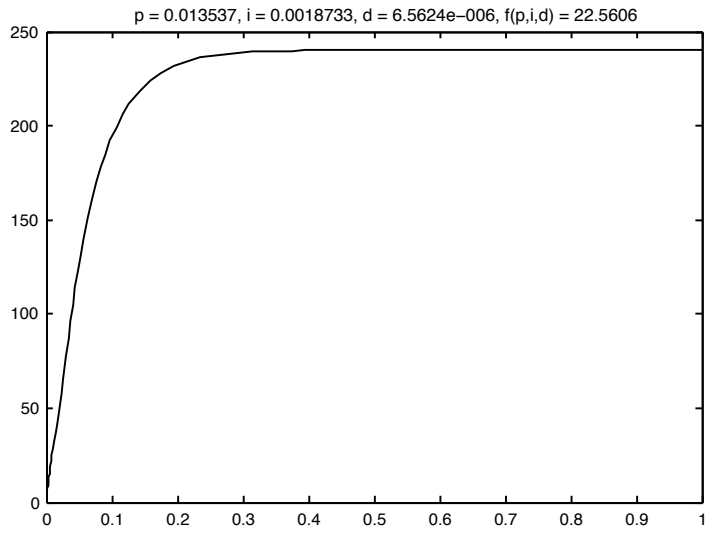


Abbildung 5.20: Optimum mit auf der Visualisierung basierenden Startwerten.

Besonders mit Direct ist es ohne Startwert möglich, einen großen Raum zu durchsuchen, um so herauszufinden, wo gute Startwerte[13] liegen könnten.

In dieser einfachen aber doch repräsentativen Optimierungsphase ist es gelungen zu zeigen, dass man mit einem aus Modulen der Bibliothek bestehenden Simulationsmodell durchaus in der Lage ist, vernünftige Optimierungsaufgaben ausreichend zu untersuchen und zu lösen.

[13]Bei einem komplett neuen Modellaufbau sind nicht einmal Größenordnungen wirklich zu schätzen.

Kapitel 6

Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde eine gut dokumentierte, leicht anpassbare und erweiterbare Ansammlung von modularen Bausteinen entwickelt: eine Bibliothek in SimMechanics, die die Erstellung von Simulationsmodellen verschiedener *elastisch angetriebener* Manipulatoren ermöglicht.

Unter Berücksichtigung der Anforderungen und festgelegten Qualitätsmerkmale wurde ein Dokumentationsframework erstellt, das die leicht umstrukturierbare Dokumentation neuer Module erleichtert.

Darüberhinaus gab es eine ausführliche Validierungsphase, in der die Richtigkeit und Eignung zur Optimierung beispielhaft an einigen Simulationsmodellen demonstriert und teilweise analytisch belegt wurde.

Um die Benutzung der Bibliothek zu unterstützen, wurden zahlreiche Beispiele erstellt, die, unter anderem, Anpassungsmöglichkeiten der Module und den Zusammenbau einiger Manipulatoren erläutern.

Das Motormodul, das die Bibliothek zur Verfügung stellt, ist jedoch nur ein einfacher pid-Regler und modelliert keinerlei mechanische Eigenschaften eines realistischen Motors. Demzufolge sollte das Motormodul als wichtige Erweiterungsstelle der Bibliothek und die schon vorhandene Implementierung lediglich als Beispiel angesehen werden.

Die Vergleiche und Optimierung verschiedener Modelle dienten lediglich der Validierung. Wegen des einfachen Motormoduls und geschätzter Modellparameter (unter anderem Dämpfungskonstanten) sind keine auf den erzielten Ergebnissen basierenden Schlussfolgerungen oder Aussagen möglich.

Eine interessante Ergänzung der Bibliothek wäre die Implementierung des in der Einleitung angesprochenen Kompositantriebs. Dies sollte mit den schon bestehenden Modulen nicht zu schwer zu gestalten sein, da man lediglich ein neues Motormodul für die fein Justierung implementieren und dann vielleicht einige Schnittstellen leicht verändern müsste, um dieses auch in einem Modell integrieren zu können. Ein anderes Antriebsprinzip, das auch mit Modulen aus der Bibliothek zu erreichen sein sollte, wäre die Umsetzung eines sogenannten *antagonistischen* Antriebs, der die gleichzeitige Anspannung beider Aktuatoren – wie beim menschlichen Arm – erlaubt. Dies wäre eine zusätzliche Freiheit, die z.B. neue Lösungen von Optimierungsaufgaben ermöglichen könnte. Da diese Antriebe, zumindest vom Prinzip her, aus schon vorhandenen Bausteinen gebaut werden könnten (Motor-, Adapter- und Federmodulen), wurde auf eine Implementierung in dieser Arbeit verzichtet.

Eine weitere Ergänzung wäre die Benutzung der *MATLAB Virtual Reality Toolbox*, um die Visualisierung der Simulationen zu verschönern. Auch wenn SimMechanics mit äquivalenten Ellipsen eine ausreichende Animationsmöglichkeit bereitstellt, ist diese mit den Möglichkeiten, die die Virtual Reality Toolbox bietet, nicht zu vergleichen. Da diese Toolbox nicht zur Verfügung stand und die Visualisierung niedrig priorisiert wurde, wurde auf den Einsatz dieses viel versprechenden Werkzeugs verzichtet.

Wenn der neue Prototyp feststeht und genaue CAD-Daten zur Verfügung stehen, wäre der nächste Schritt, mit Hilfe der Bibliothek ein präzises Modell des Prototyps zu erstellen. Wenn die genauen Abmessungen, Federkonstanten, Trägheitstensoren, Motoren und Regelung in dem Modell berücksichtigt werden, wären dann die verschiedensten Optimierungsaufgaben und Parameteranpassung bzw. Auslegung möglich.

Literaturverzeichnis

[1] Sebastian Klug, Bernhard Möhl, Oskar von Stryk und Oliver Barth:
Design and Application of a 3 DOF Bionic Robot Arm.

[2] Sebastian Klug, Oskar von Stryk:
Das Verbundprojekt BioRob.
`http://www.biorob.de` (2. Mai 2007)

[3] Andrea Vogel:
Deutschlandfunk - FORSCHUNG AKTUELL: Sanfte Roboter.

[4] Sebastian Klug:
Der bionische Manipulator: Übertragung der Funktionsprinzipien eines biologischen Armmodells auf das ingenieurmäßige Konzept eines "bionischen Roboters".

[5] White, A.S.:
Vibration control in elastic manipulators.
Computing & Control Engineering Journal.

[6] R. Kratz, S. Klug, M. Stelzer, O. von Stryk:
Biologically inspired reflex based stabilization control of a humanoid robot with artificial SMA muscles.

[7] Kelly C.T.:
Iterative Methods for Optimization.
SIAM, Philadelphia, 1999,
Frontiers in Applied Mathematics 18.

[8] Isermann:
Regelungstechnik 1,
Shaker-Verlag, 2002.

Abbildungsverzeichnis

Anhang A

Verzeichnisaufbau

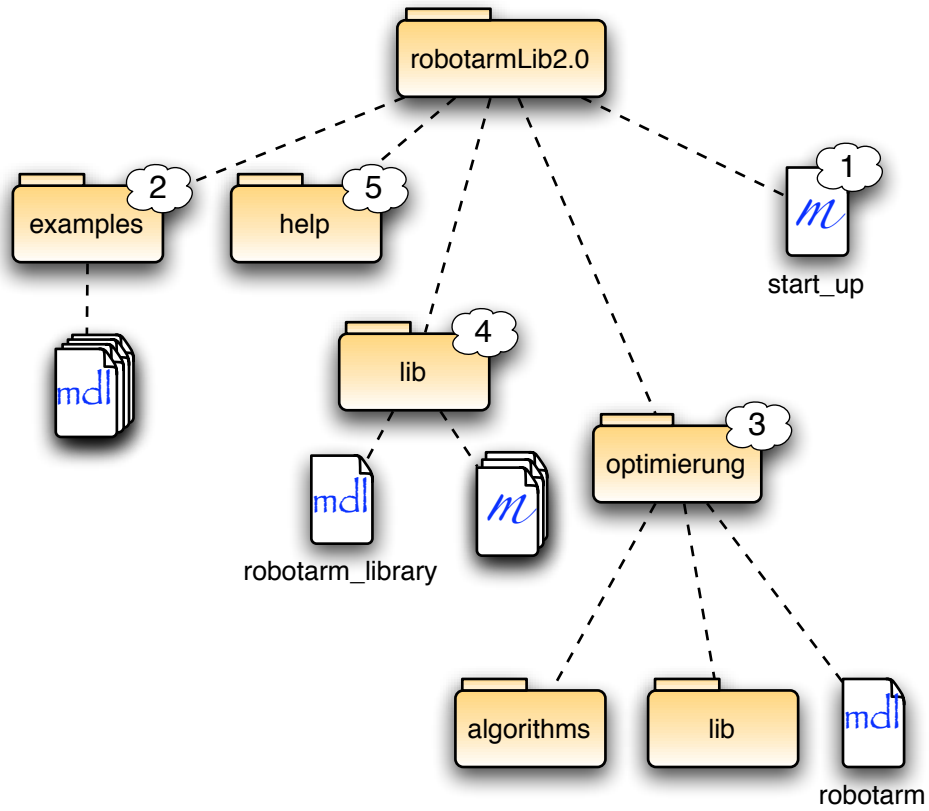


Abbildung A.1: Verzeichnisaufbau der Bibliothek.

1. Um die Bibliothek zu installieren bzw. starten, einfach diese m-Datei[1] ausführen.

2. Enthält einige Beispiele. Anhand dieser Beispiele und der Hilfeseiten lässt sich das Zusammensetzen und die Benutzung der Module schnell nachvollziehen.

3. Dieses Verzeichnis enthält alles, was in der Optimierungsphase benutzt wurde: Implementierungen für Nelder-Mead, Direct und imfil nach [7] und die Gütefunktion in `algorithms`, eine Sammlung praktischer Funktionen in `lib`, um z.B. eine Gütefunktion über einem Gitter zu visualisieren, und das eingestellte, zu optimierende Modell (`robotarm.mdl`).

4. Enthält die SimMechanics-Bibliothekdatei[2] und eine Sammlung wichtiger Funktionen, unter anderem alle[3] Initialisierungsfunktionen und eine Funktion, um Trägheitstensoren zu berechnen. Hilfe zu jeder Funktion ist durch – `help` *`functionname`* – unter MATLAB abrufbar. Dieses Verzeichnis wird durch die Installation dem MATLAB-Pfad hinzugefügt.

5. Enthält alles, was zum Dokumentationsframework gehört. In dem nächsten Abschnitt wird dies in Einzelheiten betrachtet.

[1] MATLAB-Skript
[2] Hier können neue Module hinzugefügt werden.
[3] Auch neue Initialisierungsfunktionen sollten hier gespeichert werden.

Anhang B

Dokumentationsframework

1. Enthält alle Bilder und graffle-Dateien[1], die in den HTML-Hilfeseiten eingebunden werden.

2. Enthält alle HTML-Hilfeseiten für die Module.

3. Enthält alle HTML-Hilfeseiten für die Submodule.

4. Der Index für die ganze Bibliothek. Diese Datei kann ergänzt werden, um eine Übersicht der ganzen Bibliothek zu geben. Besonders wenn die Bibliothek um neue Module ergänzt wird, wäre hier die richtige Stelle, Veränderungen[2] zu dokumentieren.

5. Der gemeinsame Stylesheet für alle Hilfeseiten. Hier kann man Veränderungen an dem Style der Hilfeseiten vornehmen, die sich global auswirken.

6. Ein Eclipse-Projekt, das alles beinhaltet, um automatisch aus XML-Dateien HTML-Hilfeseiten zu generieren.

7. Enthält jar-Dateien, die eine Java-Implementierung des frei verfügbaren xalan XSLT-Parsers zur Verfügung stellen.

8. Ein Ant-Skript, das ausgeführt werden kann, um automatisch die vorhandenen XML-Dateien zu finden[3], den Parser aufzurufen, und mit dem XSLT-Stylesheet die Transformation in HTML auszuführen.

[1] Viele Bilder wurden mit *Omnigraffle* erzeugt.
[2] Was wurde geändert bzw. ergänzt? Von wem? Wann? Wieso?
[3] Skript muss entsprechend um einen *Target* ergänzt werden.

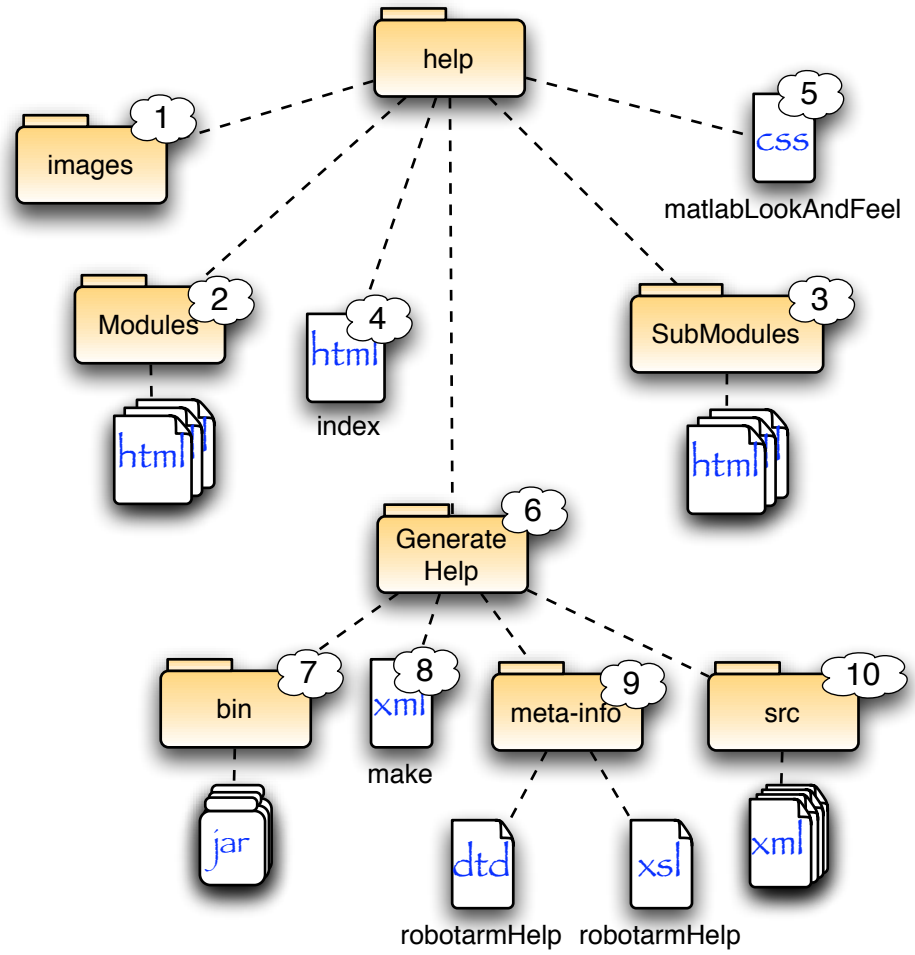


Abbildung B.1: Struktur des Dokumentationsframeworks.

9. Enthält eine DTD-Datei, die die Grammatik und Syntax der XML-Metasprache beschreibt, und den XSLT-Stylesheet, in dem die Transformation XML → deklariert wird.

10. Enthält alle XML-Dateien, die in HTML-Hilfeseiten umgewandelt werden sollen.

Anhang C

Anpassung eines Moduls

1. Als erstes den Link zur Bibliothek brechen. Dies bedeutet einerseits, dass der Block jetzt unabhängig von der Bibliothek verändert werden kann, andererseits, dass Veränderungen an der Bibliothek nicht mehr propagiert werden.

2. Dieser Pfeil wird grau, um zu zeigen, dass der Block jetzt Veränderungen enthalten könnte, und keine Verbindung mehr zur Bibliothek besteht.

3. Als nächstes eine Initialisierungsfunktion, die die Veränderungen vornimmt, implementieren. Manchmal ist es praktisch, vorher die alte aufzurufen, sodass alles andere wie gewohnt initialisiert wird.

4. In der Initialisierungsfunktion kann auf alle Parameter des Blocks zugegriffen werden.

5. Als letztes die Maske des Blocks verändern, sodass die neue Initialisierungsfunktion aufgerufen wird. Es besteht auch die Möglichkeit, für jede Variable eine andere Funktion einzustellen.

6. Der Maske können selbstverständlich auch neue Parameter hinzugefügt werden.

Genau dieses Beispiel und die neue Initialisierungsfunktion sind als `examples/robotarm_2piece_prototype.mdl` bzw. `lib/ini_joint_vertical.m` implementiert.

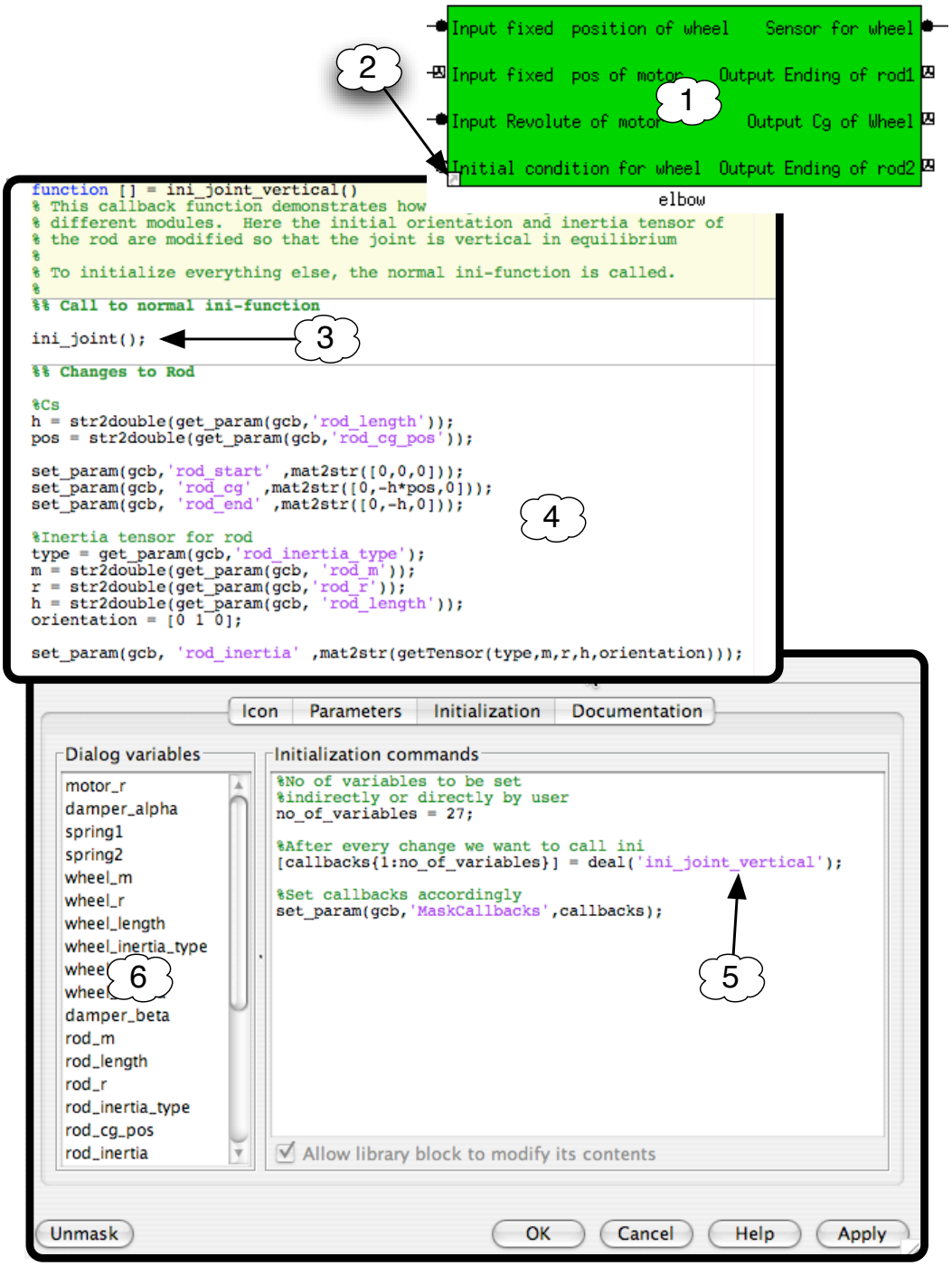


Abbildung C.1: Schritte zur Anpassung eines Moduls

Glossar

Aktuator: Aktoren, oft auch wegen des englischen Begriffs „actuator“ Aktuatoren genannt, setzen Signale einer Regelung in (meist) mechanische Arbeit, das heißt Bewegungen um. Allgemein beschreibt ein Aktor ein Element, das eine Eingangsgröße in eine andersartige Ausgangsgröße umwandelt. Die natürliche Entsprechung des technischen Aktors ist der Muskel.

Block: Mit einem Block werden in Simulink und SimMechanics Systeme bezeichnet, die bestimmte mathematische oder physikalische Modelle darstellen. Der Name Block kommt daher, dass das System meist als Rechteck dargestellt wird, mit Ein- und Ausgängen. Mit Hilfe einer Maske ist es dann möglich, als Abstraktion das System als Blackbox zu betrachten und als eine nicht zerlegbare, primitive Einheit zu benutzen.

Chaotisches Verhalten: Wenn das Verhalten eines Systems unter gewissen Anfangsbedingungen sich stetig verändert und dies deshalb als ungeordnet und zufällig erscheint, wird das System als chaotisch bezeichnet. Es ist dann schwer möglich, die Entwicklung des Systems vorherzusagen.

Copy-and-paste: „Copy and Paste“ ist ein gefährlicher, aber leider wohl bekannter und oft eingesetzter Ansatz, Ähnliches zu implementieren, indem man andere funktionierende Teile kopiert und einfügt. Es wird sogar als „Anti-Pattern“ betrachtet und ist meist ein Zeichen dafür, dass ein Framework, eine Bibliothek oder ein System entweder falsch benutzt wird oder tatsächlich schlecht erweiterbar ist.

Look-and-feel: Mit „Look and Feel“ wird in diesem Dokument das Aussehen und Verhalten von SimMechanics, Simulink oder MATLAB Programmen, Hilfeseiten oder Blöcken bezeichnet. Dies fängt mit sinnvollen Konventionen an und beinhaltet alle Aspekte bis hin zur Farbe.

Manipulator: Ein Manipulator ist typischerweise ein industrieller Roboter, meist in Form eines „Armes“ und mit einem Werkzeug am letzten Glied angehängt.

Maske: Eine Maske in Simulink oder SimMechanics ist ein Dialogfenster, wodurch man vom Entwickler festgelegte Systemparameter einstellen kann. Dadurch ist es möglich, von der konkreten Implementierung eines Systems zu abstrahieren, und den Block als Blackbox zu betrachten und zu benutzen.

Modul: In diesem Dokument ist unter einem Modul eine komplette, selbständige Einheit der Bibliothek zu verstehen – in Form eines SimMechanics Blocks – bestehend aus anderen Modulen bzw. Sub-Modulen der Bibliothek oder anderen eingebauten SimMechanics-Blöcken.

SimMechanics, Simulink, MATLAB: MATLAB ist eine Programmiersprache, die besonders in mathematischen und technischen Kreisen sehr beliebt ist. Sie ist insbesondere für ihren Reichtum an sogenannten „Toolboxes“ bekannt, eine ausgereifte und sehr reiche Ansammlung von nützlichen und auf bestimmten Gebieten spezialisierten Funktionen.

Simulink ist eine Bibliothek in MATLAB, mit der es möglich ist, graphisch - mit sogenannten Blöcken - mathematische Modelle aufzustellen. Dies erlaubt eine gute Abstraktion und fördert die Verständlichkeit und Lesbarkeit komplexer Systeme.

SimMechanics ist eine Erweiterung von Simulink, die die Erstellung *physikalischer* Modelle ermöglicht, indem Blöcke für physikalische Bausteine eines Modells zur Verfügung gestellt werden. Das mathematische System von differentialen Gleichungen wird automatisch aus dem physikalischen Modell errechnet.

(MATLAB)Path: Wenn eine Funktion in MATLAB ausgeführt werden soll, wird nach der Implementierung dieser Funktion gesucht – zuerst im aktuellen Verzeichnis, dann schließlich auf dem sogenannten Pfad, der eine Liste von Verzeichnissen enthält. Wird eine Bibliothek oder Toolbox in MATLAB installiert, so wird meist dieser Pfad um die nötigen Verzeichnissen ergänzt.

Simulink library browser: Der Library Browser in Simulink ist eine nur unter Windows verfügbare, graphische Baumstruktur aller Bibliotheken. Mit dessen Hilfe ist es möglich, Blöcke aus verschiedenen Bibliotheken in das eigene Modell „hineinzuziehen“. Die Möglichkeit besteht, selbstimplementierte Bibliotheken in diesen Browser zu integrieren.

Submodul: Manche Module in der im Rahmen dieser Arbeit erstellten Bibliothek dienen als Bausteine oder gar *Plug-Ins* für andere Module der Bibliothek. Diese Module sind auf einer niedrigeren Abstraktionsebene und werden folglich als Untermodule bzw. Submodule bezeichnet.

Validierung vs. Verifikation: Unter Verifikation versteht man einen formellen Beweis, um die Richtigkeit eines Programms oder Modells nachzuweisen. Die Validierung dagegen ist ein Nachvollziehen und wenn möglich Nachrechnen bzw. Nachprüfen der Ergebnisse eines Programms oder Modells, ohne Anspruch auf Vollständigkeit.