Simulation und Steuerung biomechanischer Mehrkörpersysteme

Diplomarbeit
von
Michael Krieg

Lehrstuhl für Theoretische Astrophysik
an der
Universität Tübingen

Juni 1992

Inhaltsverzeichnis

1 Aufgabenstellung

Mathematische Modelle, die geeignet sind, biomechanische Systeme zu beschreiben, setzen sich im allgemeinen aus mehreren Körpern derart zusammen, daß sich das dynamische Verhalten des Systems nur noch mithilfe der Computersimulation studieren läßt. Das Bestreben der Theorie ist es, das verwendete Modell solange zu verbessern, bis alle betrachteten Eigenschaften des natürlichen Systems mit dem Modell erklärt werden können. So stellt [Gruber] in ihrer Dissertation 1987 ein zweidimensionales, dreigliedriges "Modell mit Schwabbelmassen" zur Untersuchung des Einbeinsprungs des Menschen vor. In diesem Modell, das aus 6 Körpern besteht, wird das menschliche Skelett mit einer kinematischen Kette aus drei Starrkörpern modelliert, an denen jeweils zur Modellierung der Weichteile des Menschen ein weiterer Starrkörper nichtlinear–elastisch angekoppelt ist. Die modellierten Gliedmaßen sind der Rumpf, der Oberschenkel und der Unterschenkel des Menschen. Das Modell wurde von [Widmayer] um einen Arm, bestehend aus Ober– und Unterarm jeweils mit Schwabbelmasse, auf insgesamt 10 Starrkörper erweitert. Das Modell diente zur Simulation der Auswirkungen eines Absturzes aus verschiedenen Fallhöhen.

Ständige Verbesserungen des verwendeten Modells haben zur Folge, daß bei jedem Verbesserungsschritt der gesamte Modellierungszyklus von neuem durchlaufen werden muß. Es müssen die Bewegungsgleichungen des Systems aufgestellt werden. Es muß die Integration der Bewegungsgleichungen programmiert werden und anschließend sind die durch die Integration erhaltenen Daten aufzubereiten, um sie mit einem Experiment zu vergleichen. Aus diesem Vergleich resultieren wiederum Korrekturen am Modell und der nächste Zyklus beginnt. Die Geschwindigkeit, mit der ein Modell weiterentwickelt werden kann, hängt damit entscheidend von der Effizienz ab, mit der ein Modellierungszyklus durchlaufen wird.

Eine besondere Schwierigkeit bei der Modellierung biomechanischer Systeme besteht darin, daß diese Systeme sich aus eigener Kraft koordiniert bewegen und nicht nur wie bewußtlose Körper auf die an ihnen angreifenden Kräfte und Drehmomente reagieren. Um die bei einer speziellen Bewegung auftretenden Effekte im mathematischen Modell erfassen zu können, wird man darauf angewiesen sein, daß auch das Modell den betrachteten Bewegungsablauf beherrscht. Während dieses Problem bei der Robotersteuerung keine unüberwindbare Schwierigkeit darstellt, ist es in der Biomechanik eine sehr große Herausforderung, da die zur Koordination und Steuerung des Systems vorhandenen Kräfte im Vergleich zu Robotern klein sind und außerdem nicht beliebig schnell aufgebracht werden können. Außerdem können diese nicht über beliebige Zeitspannen konstant gehalten werden. Die typische zur Reaktion, d. h. vom Eintreffen eines Signals bis zur Erwiderung durch Kraftaufbau in den Muskeln, benötigte Zeit liegt beim Menschen in der Größenordnung von etwa 100 ms.

Die vorliegende Arbeit beschäftigt sich daher in der Hauptsache damit, eine Arbeitsumgebung auf dem Computer einzurichten, mit der sich die mechanisierbaren Tätigkeiten während eines Modellierungszyklus bei der Erstellung einer Simulation auf ein Minimum reduzieren. Es soll ein flexibles Werkzeug zur Simulation biomechanischer Mehrkörpersysteme entstehen, das außerdem die Möglichkeit zum Einbau von regelnden Elementen zur Steuerung des Modells vorsieht. Dazu wird zunächst die bei der Modellierung benötigte Mechanik kinematischer Ketten und kinematischer Bäume angesprochen. Anschließend werden die an ein Programmsystem gestellten Anforderungen in Form eines Entwurfs aufgeführt, dessen Realisierung anhand eines Beispiels besprochen wird. Das Beispiel wirft das Problem der Steuerung auf, weshalb nach der Modellierung des Beispiels auf die unternommenen Versuche zur Steuerung eingegangen wird.

2 Bewegungsgleichungen und deren Lösung

2.1 Bewegungsgleichungen des Starrkörpers

Ein starrer Körper habe die Masse m, deren Ausdehnung durch den Trägheitstensor Θ beschrieben wird. Auf den Körper wirkt die resultierende Kraft $\boldsymbol{K} = \sum_i \boldsymbol{K}_i$, die sich aus einzelnen Kräften $\boldsymbol{K}_i$ zusammensetzt. Die Bewegung des Schwerpunktes $\boldsymbol{x}$ wird durch folgende Gleichung beschrieben:

$$m \frac{d^2}{dt^2} \boldsymbol{x} = \boldsymbol{K} \tag{1}$$

Es ist $\boldsymbol{r}_i$ der Vektor, der vom Körperschwerpunkt zum Angriffspunkt der Kraft $\boldsymbol{K}_i$ zeigt. Der Körper erfährt durch die angreifenden Kräfte das Drehmoment $\boldsymbol{M}_K = \sum_i \boldsymbol{r}_i \times \boldsymbol{K}_i$. Auf den Körper soll zusätzlich noch das äußere Moment $\boldsymbol{M}_a$ wirken. Die Bewegung der Winkelgeschwindigkeit des Körpers $\boldsymbol{\omega}$ im körperfesten Koordinatensystem wird durch die sog. Eulersche Gleichung beschrieben:

$$\Theta \frac{d}{dt} \boldsymbol{\omega} + \boldsymbol{\omega} \times \Theta \boldsymbol{\omega} = \boldsymbol{M} \tag{2}$$

Hierin setzt sich das Gesamtmoment $\boldsymbol{M} = \boldsymbol{M}_K + \boldsymbol{M}_a$ aus dem durch die Kräfte $\boldsymbol{K}_i$ verursachten Moment und aus dem zusätzlich angreifenden äußeren Moment zusammen.

2.2 Eulersche Winkel

Wie aus den beiden Gleichungen (1) und (2) hervorgeht, benötigt man im Dreidimensionalen 6 Koordinaten, um die Lage und die Orientierung eines starren Körpers in einem festen Bezugssystem zu beschreiben. Dies sind drei kartesische Koordinaten, die die Lage eines körperfesten Punktes (häufig des Körperschwerpunktes) angeben und drei Winkel, die die Orientierung des Körpers beschreiben. Die gebräuchlichsten Winkel hierfür sind die Eulerschen Winkel, die durch drei aufeinanderfolgende, in einer bestimmten Reihenfolge durchzuführende Drehungen definiert sind. Man dreht das ursprüngliche Achsensystem xyz um einen Winkel ϕ gegen den Uhrzeigersinn um die Achse z. Hieraus erhält man das Achsensystem $\xi\eta\zeta$. Die zugehörige Drehmatrix sei mit D bezeichnet. Anschließend dreht man um einen Winkel θ gegen den Uhrzeigersinn um die Achse ξ und erhält so das Koordinatensystem $\xi'\eta'\zeta'$, die Drehmatrix soll C heißen. Man bezeichnet die Achse $\xi \equiv \xi'$ als Knotenlinie. Schließlich dreht man gegen den Uhrzeigersinn um einen Winkel ψ um die Achse ζ' und erhält so das Koordinatensystem $x'y'z'$, die Drehmatrix dieser letzten Drehung sei mit B bezeichnet. Die drei Drehmatrizen sind damit wie folgt definiert:

$$D = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3}$$

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix} \tag{4}$$

$$B = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5}$$

Die Transformation, die das raumfeste System xyz direkt in das körperfeste System $x'y'z'$ überführt, wird durch die Matrix A beschrieben. A ist das Produkt der drei Matrizen B, C und D.

$$A = BCD = \tag{6}$$

$$\begin{pmatrix} \cos\psi\cos\phi - \cos\theta\sin\phi\sin\psi & \cos\psi\sin\phi + \cos\theta\cos\phi\sin\psi & \sin\psi\sin\theta \\ -\sin\psi\cos\phi - \cos\theta\sin\phi\cos\psi & -\sin\psi\sin\phi + \cos\theta\cos\phi\cos\psi & \cos\psi\sin\theta \\ \sin\theta\sin\phi & -\sin\theta\cos\phi & \cos\theta \end{pmatrix}$$

Alle Matrizen sind orthogonal, insbesondere die Matrix A, weshalb die Rücktransformation durch die transponierte Matrix A^T beschrieben wird. Die Matrix A^T beschreibt also die Transformation von körperfesten Koordinaten $x'y'z'$ in raumfeste Koordinaten xyz (abgesehen von einer möglichen Translation). Die Matrix A transformiert raumfeste Koordinaten in körperfeste Koordinaten, ebenfalls von Translationen abgesehen.
Im Zweidimensionalen werden zur Beschreibung der Lage und der Orientierung eines starren Körpers in einem festen Bezugssystem nur drei Koordinaten benötigt: Zwei kartesische Koordinaten eines körperfesten Punktes und ein Drehwinkel zur Angabe der Orientierung des Körpers. Die entsprechende Transformationsmatrix ist in diesem Fall in D aus Gleichung (3) enthalten.

2.3 Transformationen

2.3.1 Eulergleichung in Eulerwinkeln

Die Eulersche Gleichung (2) ist eine Bewegungsgleichung für die Winkelgeschwindigkeit $\boldsymbol{\omega}$ in Koordinaten des körperfesten Systems. Möchte man die körperfesten Koordinaten durch raumfeste Koordinaten ersetzen, so ist die Eulergleichung vollständig in Eulerwinkeln auszudrücken. Dazu wird die Winkelgeschwindigkeit in drei einzelne Drehungen zerlegt, die jeweils um die durch die Eulerwinkel definierten Achsen verlaufen. Die Drehachsen sind die Richtungen der einzelnen Drehungen bzw. Winkelgeschwindigkeiten und es gilt geschrieben in körperfesten Koordinaten:

$$\boldsymbol{\omega} = \boldsymbol{\omega}_\phi + \boldsymbol{\omega}_\theta + \boldsymbol{\omega}_\psi \tag{7}$$

Die Beträge der einzelnen Winkelgeschwindigkeiten sind die Zeitableitungen der Eulerwinkel. Der Winkel ϕ dreht um die raumfeste Achse z. Der Winkel θ dreht um die Achse ξ'. Der Winkel ψ schließlich dreht um die körperfeste Achse z'. Die Achsen, d. h. Richtungen, transformieren sich mit den oben beschriebenen Drehmatrizen der entsprechenden Eulerwinkel in Koordinaten des körperfesten Systems. Es gilt:

$$\boldsymbol{\omega}_\phi = \dot{\phi} A \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{8}$$

$$\boldsymbol{\omega}_\theta = \dot{\theta} B \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \tag{9}$$

$$\boldsymbol{\omega}_\psi = \dot{\psi} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{10}$$

Damit ergibt sich folgende Beziehung zwischen den Eulerwinkeln und der Winkelgeschwindigkeit:

$$\boldsymbol{\omega} = \begin{pmatrix} \dot{\phi} \sin\psi \sin\theta + \dot{\theta} \cos\psi \\ \dot{\phi} \cos\psi \sin\theta - \dot{\theta} \sin\psi \\ \dot{\psi} + \dot{\phi} \cos\theta \end{pmatrix} \tag{11}$$

Um die Eulergleichung vollständig in Eulerwinkeln auszudrücken, ist Gleichung (11) nochmals nach der Zeit abzuleiten.

$$\begin{aligned}
\dot{\boldsymbol{\omega}} &= \begin{pmatrix} \ddot{\phi} \sin\psi \sin\theta + \ddot{\theta} \cos\psi + \dot{\phi}\dot{\psi} \cos\psi \sin\theta - \dot{\theta}\dot{\psi} \sin\psi + \dot{\phi}\dot{\theta} \sin\psi \cos\theta \\ \ddot{\phi} \cos\psi \sin\theta - \ddot{\theta} \sin\psi + \dot{\phi}\dot{\theta} \cos\psi \cos\theta - \dot{\theta}\dot{\psi} \cos\psi - \dot{\phi}\dot{\psi} \sin\psi \sin\theta \\ \ddot{\psi} - \dot{\phi}\dot{\theta} \sin\theta \end{pmatrix} \\
&= \begin{pmatrix} \sin\psi \sin\theta & \cos\psi & 0 \\ \cos\psi \sin\theta & -\sin\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{pmatrix} + \boldsymbol{k} \qquad (12) \\
\boldsymbol{k} &= \begin{pmatrix} \dot{\phi}\dot{\psi} \cos\psi \sin\theta - \dot{\theta}\dot{\psi} \sin\psi + \dot{\phi}\dot{\theta} \sin\psi \cos\theta \\ \dot{\phi}\dot{\theta} \cos\psi \cos\theta - \dot{\theta}\dot{\psi} \cos\psi - \dot{\phi}\dot{\psi} \sin\psi \sin\theta \\ -\dot{\phi}\dot{\theta} \sin\theta \end{pmatrix}
\end{aligned}$$

Betrachtet man als körperfestes Koordinatensystem nur das Hauptträgheitsachsensystem des Körpers, so enthält der Trägheitstensor Θ nur die Elemente der Hauptdiagonalen Θ_x, Θ_y und Θ_z und das Kreuzprodukt der Eulergleichung vereinfacht sich. Man erhält:

$$\begin{pmatrix} \Theta_x & 0 & 0 \\ 0 & \Theta_y & 0 \\ 0 & 0 & \Theta_z \end{pmatrix} \dot{\boldsymbol{\omega}} + \begin{pmatrix} \omega_y \omega_z (\Theta_z - \Theta_y) \\ \omega_x \omega_z (\Theta_x - \Theta_z) \\ \omega_x \omega_y (\Theta_y - \Theta_x) \end{pmatrix} = \boldsymbol{M} \tag{13}$$

Im Zweidimensionalen vereinfacht sich die Eulergleichung erheblich. Es muß nur ein Winkel betrachtet werden und der Trägheitstensor Θ ist ebenso ein Skalar wie das Drehmoment M. Damit läßt sich die Eulergleichung in der Ebene sofort hinschreiben:

$$\Theta \ddot{\phi} = M \tag{14}$$

2.3.2 Zeitableitung

Die aus einem körperfesten System heraus beobachtete zeitliche Änderung eines in einem inertialen System gegebenen Vektors $\boldsymbol{R}$ resultiert zum einen aus der Änderung der Größe selbst, zum anderen aus der eigenen Körperdrehung. Im infinitesimal kleinen Zeitintervall dt gilt daher:

$$d\boldsymbol{R}|_{K\ddot{o}rper} = d\boldsymbol{R}|_{Inert.} + d\boldsymbol{R}|_{Dreh} \tag{15}$$

Für infinitesimale Koordinatendrehungen um den Winkel $\boldsymbol{\Omega}$ gilt die Beziehung:

$$d\boldsymbol{R}|_{Dreh} = \boldsymbol{R} \times d\boldsymbol{\Omega} \tag{16}$$

Setzt man die Gleichungen ineinander ein und dividiert durch das betrachtete Zeitintervall dt, so erhält man die Relation:

$$\left(\left.\frac{d}{dt}\right|_{Inert.} - \left(\left.\frac{d}{dt}\right|_{K\ddot{o}rper} + \boldsymbol{\omega} \ \times \right) \right) \boldsymbol{R} = \mathbf{o} \tag{17}$$

mit

$$\boldsymbol{\omega} = \frac{d\boldsymbol{\Omega}}{dt}$$

Da diese Beziehung für jeden beliebigen Vektor gilt, ist sie als Operatorbeziehung zu verstehen, und man kann den angeschriebenen Vektor weglassen:

$$\left.\frac{d}{dt}\right|_{Inert.} = \left.\frac{d}{dt}\right|_{K\ddot{o}rper} + \boldsymbol{\omega} \ \times \tag{18}$$

Zwischen dem in raumfesten Koordinaten gegebenen Vektor $\boldsymbol{R}$ und seinen körperfesten Koordinaten $\boldsymbol{r}$ besteht die Beziehung $\boldsymbol{R} = A^T\boldsymbol{r}$. Zur Berechnung der Zeitableitung des Vektors $\boldsymbol{R}$ wird die Beziehung (18) verwendet und es gilt daher:

$$\dot{\boldsymbol{R}} = \dot{\boldsymbol{r}} + \omega \times \boldsymbol{r} \tag{19}$$

Soll die Gleichung nochmals abgeleitet werden, gilt:

$$\begin{aligned} \ddot{\boldsymbol{R}} &= \left.\frac{d}{dt}\right|_{Inert.} (\dot{\boldsymbol{r}} + \omega \times \boldsymbol{r}) \\ &= \ddot{\boldsymbol{r}} + 2\boldsymbol{\omega} \times \dot{\boldsymbol{r}} + \dot{\boldsymbol{\omega}} \times \boldsymbol{r} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{r}) \end{aligned} \tag{20}$$

Ist $\boldsymbol{r}$ nicht zeitabhängig, so verschwinden die Glieder mit Zeitableitungen von $\boldsymbol{r}$ und man erhält unter Verwendung der Beziehung $\boldsymbol{a} \times (\boldsymbol{b} \times \boldsymbol{c}) = \boldsymbol{b}(\boldsymbol{ac}) - \boldsymbol{c}(\boldsymbol{ab})$ folgendes Ergebnis:

$$\ddot{\boldsymbol{R}} = \dot{\boldsymbol{\omega}} \times \boldsymbol{r} + \boldsymbol{\omega}(\boldsymbol{\omega r}) - \boldsymbol{r}(\boldsymbol{\omega}^2) \tag{21}$$

2.4 Zwangsbedingungen

Ein Mehrköpersystem ist im allgemeinen ein Verbund aus mehreren Starrkörpern, die sich gegenseitig in ihrer Bewegung durch sog. Zwangsbedingungen einschränken. Man bezeichnet die Zwangsbedingungen als holonom, wenn diese geschlossen in Form einer Gleichung oder in Form eines totalen Differentials darstellbar sind, als anholonom, wenn dies nicht der Fall ist, als skleronom (fest), wenn die Zwangsbedingungen die Zeit nicht explizit enthalten und rheonom (fließend), wenn die Zwangsbedingungen explizit zeitabhängig sind. Für die vorliegende Aufgabenstellung genügt es, im folgenden nur noch holonome, skleronome Zwangsbedingungen zu betrachten.

Jeder Körper des Mehrkörpersystems wird durch ein eigenes Gleichungspaar (1) und (2) beschrieben. Man erhält so für N Körper ein Gleichungssystem aus $6N$ Gleichungen in $6N$ Koordinaten, aus dem hier die Zeilen mit den Gleichungen des Körpers i herausgegriffen sind:

$$\begin{pmatrix} \ddots & & & & & & & \\ & m_i & 0 & 0 & 0 & 0 & 0 & \\ & 0 & m_i & 0 & 0 & 0 & 0 & \\ & 0 & 0 & m_i & 0 & 0 & 0 & \\ & 0 & 0 & 0 & \Theta_{xi}\sin\psi_i\sin\theta_i & \cos\psi_i & 0 & \\ & 0 & 0 & 0 & \cos\psi_i\sin\theta_i & \Theta_{yi} & 0 & \\ & 0 & 0 & 0 & 0 & \cos\theta_i & \Theta_{zi} & \\ & & & & & & & \ddots \end{pmatrix} \begin{pmatrix} \vdots \\ \ddot{x}_i \\ \ddot{y}_i \\ \ddot{z}_i \\ \ddot{\phi}_i \\ \ddot{\theta}_i \\ \ddot{\psi}_i \\ \vdots \end{pmatrix} + \boldsymbol{b} = \begin{pmatrix} \vdots \\ F_{xi} \\ F_{yi} \\ F_{zi} \\ M_{\phi_i} \\ M_{\theta_i} \\ M_{\psi_i} \\ \vdots \end{pmatrix}$$

$$\boldsymbol{b} = \begin{pmatrix} \vdots \\ 0 \\ 0 \\ 0 \\ \dot{\phi}_i\dot{\psi}_i\cos\psi_i\sin\theta_i - \dot{\theta}_i\dot{\psi}_i\sin\psi_i + \dot{\phi}_i\dot{\theta}_i\sin\psi_i\cos\theta_i \\ \dot{\phi}_i\dot{\theta}_i\cos\psi_i\cos\theta_i - \dot{\theta}_i\dot{\psi}_i\cos\psi_i - \dot{\phi}_i\dot{\psi}_i\sin\psi_i\sin\theta_i \\ -\dot{\phi}_i\dot{\theta}_i\sin\theta_i \\ \vdots \end{pmatrix} + \begin{pmatrix} \vdots \\ 0 \\ 0 \\ 0 \\ \omega_{yi}\omega_{zi}(\Theta_{zi} - \Theta_{yi}) \\ \omega_{xi}\omega_{zi}(\Theta_{xi} - \Theta_{zi}) \\ \omega_{xi}\omega_{yi}(\Theta_{yi} - \Theta_{xi}) \\ \vdots \end{pmatrix} \tag{22}$$

Die Komponenten der Winkelgeschwindigkeit von $\boldsymbol{\omega}_i$ gehen aus Gleichung (11) hervor. Die Vektoren dieses Gleichungssystems sind $6N$ dimensional, die Matrix ist eine $6N \times 6N$ Matrix, deren Hauptdiagonale aus N der oben angeschriebenen 6×6 Matrizen besteht. Gesucht sind die Lösungen dieses Gleichungssystems; es sollen jedoch nur solche Lösungen dieses Gleichungssystems zugelassen sein, die alle k Zwangsbedingungen der folgenden Form erfüllen, d. h. mit allen k Zwangsbedingungen verträglich sind:

$$\begin{aligned} f_1(x_1, y_1, \ldots, x_i, y_i, z_i, \phi_i, \theta_i, \psi_i, \ldots, \theta_N, \psi_N) &= 0 \\ \vdots \qquad\qquad\qquad &= \vdots \\ f_k(x_1, y_1, \ldots, x_i, y_i, z_i, \phi_i, \theta_i, \psi_i, \ldots, \theta_N, \psi_N) &= 0 \end{aligned} \tag{23}$$

Das durch Gleichung (22) und (23) beschriebene System hat $f = 6N - k$ Freiheitsgrade. Jede zusätzliche Zwangsbedingung kostet einen Freiheitsgrad. In den beiden folgenden Abschnitten werden zwei verschiedene Wege zum Aufstellen der Bewegungsgleichungen dieses Systems aufgeführt.

2.5 Bewegungsgleichungen in reduzierten Koordinaten

Der eine Weg ist der, daß man die Zwangsbedingungen durch die Einführung geeigneter Koordinaten implizit erfüllt. Man transformiert auf f reduzierte Koordinaten [1] q_i, in denen das Mehrkörpersystem nur noch solche Konfigurationen annehmen kann, die mit den Zwangsbedingungen verträglich sind:

$$\begin{aligned} q_1 &= q_1(x_1, y_1, \ldots, x_i, y_i, z_i, \phi_i, \theta_i, \psi_i, \ldots, \theta_N, \psi_N) \\ \vdots &= \qquad\qquad\qquad \vdots \\ q_f &= q_f(x_1, y_1, \ldots, x_i, y_i, z_i, \phi_i, \theta_i, \psi_i, \ldots, \theta_N, \psi_N) \end{aligned} \tag{24}$$

Jede in den reduzierten Koordinaten auszudrückende Systemkonfiguration ist Lösung der Zwangsbedingungen (23). Man kann damit keine Systemkonfiguration in den reduzierten Koordinaten angeben, die nicht Lösung der Zwangsbedingungen (23) sind. Man eliminiert also durch eine geeignete Koordinatentransformation die Zwangsbedingungen und erhält anstelle der $6N$ Gleichungen in $6N$ Variablen und den k Zwangsbedingungen ein Gleichungssystem mit $f = 6N - k$ Gleichungen in f Koordinaten. Man drückt dazu die alten, vollständigen Koordinaten durch die neuen, reduzierten Koordinaten aus:

$$\begin{aligned} x_1 &= x_1(q_1, \ldots, q_f) \\ y_1 &= y_1(q_1, \ldots, q_f) \\ z_1 &= z_1(q_1, \ldots, q_f) \\ \vdots &= \qquad \vdots \\ \theta_N &= \theta_N(q_1, \ldots, q_f) \\ \psi_N &= \psi_N(q_1, \ldots, q_f) \end{aligned} \tag{25}$$

[1] Analog zur Punktmechanik. Da hier aber ein Starrkörper beschrieben wird, wird nicht der Begriff 'generalisierte Koordinaten' sondern 'reduzierte Koordinaten' verwendet. Neben der impliziten Berücksichtigung der Zwangsbedingungen hat man damit nämlich die Zahl der Koordinaten reduziert.

Und man erhält die Geschwindigkeiten und Beschleunigungen durch Differenzieren:

$$\dot{\xi}_i = \sum_{j=1}^{f} \frac{\partial \xi_i}{\partial q_j} \dot{q}_j \tag{26}$$

$$\ddot{\xi}_i = \sum_{j,l=1}^{f} \frac{\partial^2 \xi_i}{\partial q_j \partial q_l} \dot{q}_j \dot{q}_l + \sum_{j=1}^{f} \underbrace{\frac{\partial \xi_i}{\partial q_j}}_{J_{ij}} \ddot{q}_j \tag{27}$$

Hierin ist $\xi \in \{x_i, y_i, z_i, \phi_i, \theta_i, \psi_i\}$. Schreibt man Glg. (27) in Vektorform, so steht im letzten Summanden die Jakobimatrix J der Koordinatentransformation. Nachdem man in den Gleichungen (22) die Koordinaten, Geschwindigkeiten und Beschleunigungen eliminiert und das Gleichungssystem nach den reduzierten Koordinaten umgruppiert, erhält man $6N$ Gleichungen in den f reduzierten Koordinaten. Die Koeffizientenmatrix aus Glg. (27) — sie sei hier mit T bezeichnet — wird durch die Substitution von rechts mit der Jakobimatrix multipliziert. Man erhält damit die Gleichung:

$$TJ\ddot{\boldsymbol{q}} + \tilde{\boldsymbol{b}} = \boldsymbol{F} \tag{28}$$

Multipliziert man nun diese Gleichung von links mit der transponierten Jakobimatrix J^T, so erhält man die Bewegungsgleichungen in reduzierten Koordinaten:

$$J^T TJ\ddot{\boldsymbol{q}} + \bar{\boldsymbol{b}} = \bar{\boldsymbol{F}} \tag{29}$$

Jede Lösung dieses f-dimensionalen Gleichungssystems erfüllt automatisch die Zwangsbedingungen.

2.6 Bewegungsgleichungen in vollständigen Koordinaten

Der andere Weg zu den Bewegungsgleichungen ist, die Zwangsbedingungen nicht durch die Wahl geeigneter Koordinaten zu berücksichtigen, sondern die Gleichungen (22) unter Beachtung der Zwangsbedingungen (23) zu lösen. Man verwendet hierbei die Methode der Lagrangeschen Multiplikatoren. Aus den rechten Seiten der Gleichungen (22) wird damit folgendes:

$$\text{R.S. aus Glg. (22)} = \begin{pmatrix} \cdots \\ F_{xi} + \lambda_1 \frac{\partial f_1}{\partial x_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial x_i} \\ F_{yi} + \lambda_1 \frac{\partial f_1}{\partial y_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial y_i} \\ F_{zi} + \lambda_1 \frac{\partial f_1}{\partial z_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial z_i} \\ M_{\phi_i} + \lambda_1 \frac{\partial f_1}{\partial \phi_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial \phi_i} \\ M_{\theta_i} + \lambda_1 \frac{\partial f_1}{\partial \theta_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial \theta_i} \\ M_{\psi_i} + \lambda_1 \frac{\partial f_1}{\partial \psi_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial \psi_i} \\ \cdots \end{pmatrix} \tag{30}$$

Mit diesem Schritt hat man sich weitere k unbekannte Größen — nämlich die k λ_l — eingehandelt. Unter Verwendung der k Gleichungen der Zwangsbedingungen

(23) zusätzlich zu den $6N$ Gleichungen (30) lassen sich aber alle $6N+k$ unbekannten Größen bestimmen. Die Bedeutung der λ_l wird klar, wenn man sich veranschaulicht, was die Methode der Lagrangeschen Multiplikatoren macht. Man hat nämlich die Zwangsbedingungen nicht als physikalische Gegebenheiten in Form reduzierter Koordinaten in das System hineingesteckt, sondern man kann sich die Zwangsbedingungen in ihrer Wirkung durch geeignete Kräfte und Momente, sog. Zwangskräfte $\boldsymbol{Z}_i$ und –momente $\boldsymbol{D}_i$, ersetzt denken. Diese Zwangskräfte sind zwar zunächst unbekannt, sie sollen aber stets so wirken, daß zu jedem Zeitpunkt sämtliche Zwangsbedingungen erfüllt sind. Die Zwangskräfte und –momente werden daher zu den rechten Seiten der Gleichungen (22) addiert und man erhält damit:

$$\text{R.S. aus Glg. (22)} = \begin{pmatrix} \cdots \\ F_{xi} + Z_{xi} \\ F_{yi} + Z_{yi} \\ F_{zi} + Z_{zi} \\ M_{\phi_i} + D_{\phi_i} \\ M_{\theta_i} + D_{\theta_i} \\ M_{\psi_i} + D_{\psi_i} \\ \cdots \end{pmatrix} \tag{31}$$

Gleichung (31) erweckt zwar den Eindruck, als sei Gleichung (22) nochmals um $3N$ Unbekannte Z_i und D_i erweitert worden. Dies trifft jedoch nicht zu, da die Zwangskräfte und –momente nicht linear unabhängig sind. Ein Vergleich der Gleichungen (31) und (30) ergibt für die Zwangskräfte und –momente folgende Beziehung:

$$\begin{array}{rcl} \cdots & & \\ Z_{xi} & = & \lambda_1 \frac{\partial f_1}{\partial x_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial x_i} \\ Z_{yi} & = & \lambda_1 \frac{\partial f_1}{\partial y_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial y_i} \\ Z_{zi} & = & \lambda_1 \frac{\partial f_1}{\partial z_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial z_i} \\ D_{\phi_i} & = & \lambda_1 \frac{\partial f_1}{\partial \phi_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial \phi_i} \\ D_{\theta_i} & = & \lambda_1 \frac{\partial f_1}{\partial \theta_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial \theta_i} \\ D_{\psi_i} & = & \lambda_1 \frac{\partial f_1}{\partial \psi_i} + \ldots + \lambda_k \frac{\partial f_k}{\partial \psi_i} \\ \cdots & & \end{array} \tag{32}$$

2.7 Vollständige vs. reduzierte Koordinaten

An dieser Stelle sollen die Bewegungsgleichungen in vollständigen Koordinaten kurz mit den Bewegungsgleichungen in reduzierten Koordinaten im Hinblick auf die vorliegende Aufgabenstellung verglichen werden. Es wird nach wie vor von einem Mehrkörpersystem bestehend aus N Starrkörpern ausgegangen, die k holonomen, skleronomen Zwangsbedingungen unterworfen sind.

Das Gleichungssystem in reduzierten Koordinaten besteht aus $3N - k$ gekoppelten Differentialgleichungen in ebensovielen Unbekannten. Die Gleichungen sind linear in der zweiten Zeitableitung und nichtlinear in der ersten Zeitableitung der reduzierten Koordinaten. Die Gleichungen werden im allgemeinen durch die Transformation auf die reduzierten Koordinaten sehr unanschaulich und ein Algorithmus zur systematischen Durchführung dieser Transformation bzw. die direkte Erstellung des Gleichungssystems in reduzierten Koordinaten ist sehr aufwendig und deshalb während der Entwicklung schlecht zu überprüfen oder zu testen. Abgesehen davon wird im Falle einer numerischen Integration mittels Mehrschrittverfahren zur Berechnung einer jeden Stützstelle die Lösung des Gleichungssystems erforderlich. Ausschlaggebend für die dafür aufzubringende Rechenzeit ist weniger der Rang des zu lösenden Gleichungssystems als vielmehr die Anzahl der von Null verschiedenen Elemente der Koeffizientenmatrix. Durch die Transformation auf das Minimalsystem hängt die Rechenzeit also stark davon ab, wie gut die Minimalkoordinaten entkoppeln. Je komplizierter die Koeffizientenmatrix ist, desto mehr Rechenzeit wird dies in Anspruch nehmen. Durch die Transformation auf das Minimalsystem geht ferner jede Information über Zwangskräfte verloren. Diese müssen nachträglich für jeden Körper aus der Differenz der Beschleunigungen mal Körpermasse und den am Körper angreifenden Kräften berechnet werden.

In vollständigen Koordinaten bleibt das Gleichungssystem sehr übersichtlich und systematisch. Wenngleich die Anzahl der Gleichungen entsprechend der Anzahl Zwangsbedingungen höher ist als die des Minimalsystems, so ist die numerische Lösung der Gleichungen aufgrund der einfacheren Form der Koeffizientenmatrix gleichschnell oder schneller durchzuführen. Außerdem werden die Zwangskräfte bei der Lösung automatisch mitberechnet und sie stehen dann zur Abschätzung von erforderlichen Belastbarkeiten einzelner Teile des Mehrkörpersystems zur Verfügung. Ein Problem bei der Verwendung vollständiger Koordinaten kann darin bestehen, daß die Zwangsbedingungen selbst Gegenstand numerischer Fehler sind, d. h. daß sie ebenfalls nur bis auf die Größenordnung der Integrationsgenauigkeit eingehalten werden. Dies wirkt sich nachteilig aus bei der Simulation von Anordnungen, bei denen die Belastbarkeit einzelner Bauteile keine Rolle spielt und es auf hohe Präzision ankommt. In der Biomechanik ist die Beschaffenheit der simulierten Systeme jedoch ebenfalls so unpräzise wie die zur Simulation verwendete Numerik, weshalb dieses Argument belanglos wird.

Vor allem aufgrund der einfachen und systematischen Form der Bewegungsgleichungen und der damit verknüpften Möglichkeit, die Gleichungen maschinell zu generieren, werden im folgenden nur noch die vollständigen Koordinaten betrachtet. In den nächsten Abschnitten werden eine Reihe von in der Biomechanik besonders wichtigen Zwangsbedingungen aufgeführt, anhand denen der Vorteil vollständiger Koordinaten ersichtlich wird. Ein Teil der vorstehenden Argumentation wird jedoch erst in entsprechenden Abschnitten zur numerischen Integration und zur maschinellen Gleichungsgenerierung aufgegriffen und veranschaulicht.

2.8 Kinematische Ketten

Kinematische Ketten sind über Gelenke miteinander verbundene Starrkörper. Folgende Gelenktypen sollen hier näher betrachtet werden: Das Scharniergelenk und das Kugelgelenk. Im Zweidimensionalen sind beide Gelenke identisch. Im Dreidimensionalen sind noch weitere Gelenktypen denkbar, beispielsweise ein Schraubengelenk oder eine Achshalterung. Auf die beiden letzten Gelenktypen soll hier aber nicht eingegangen werden.

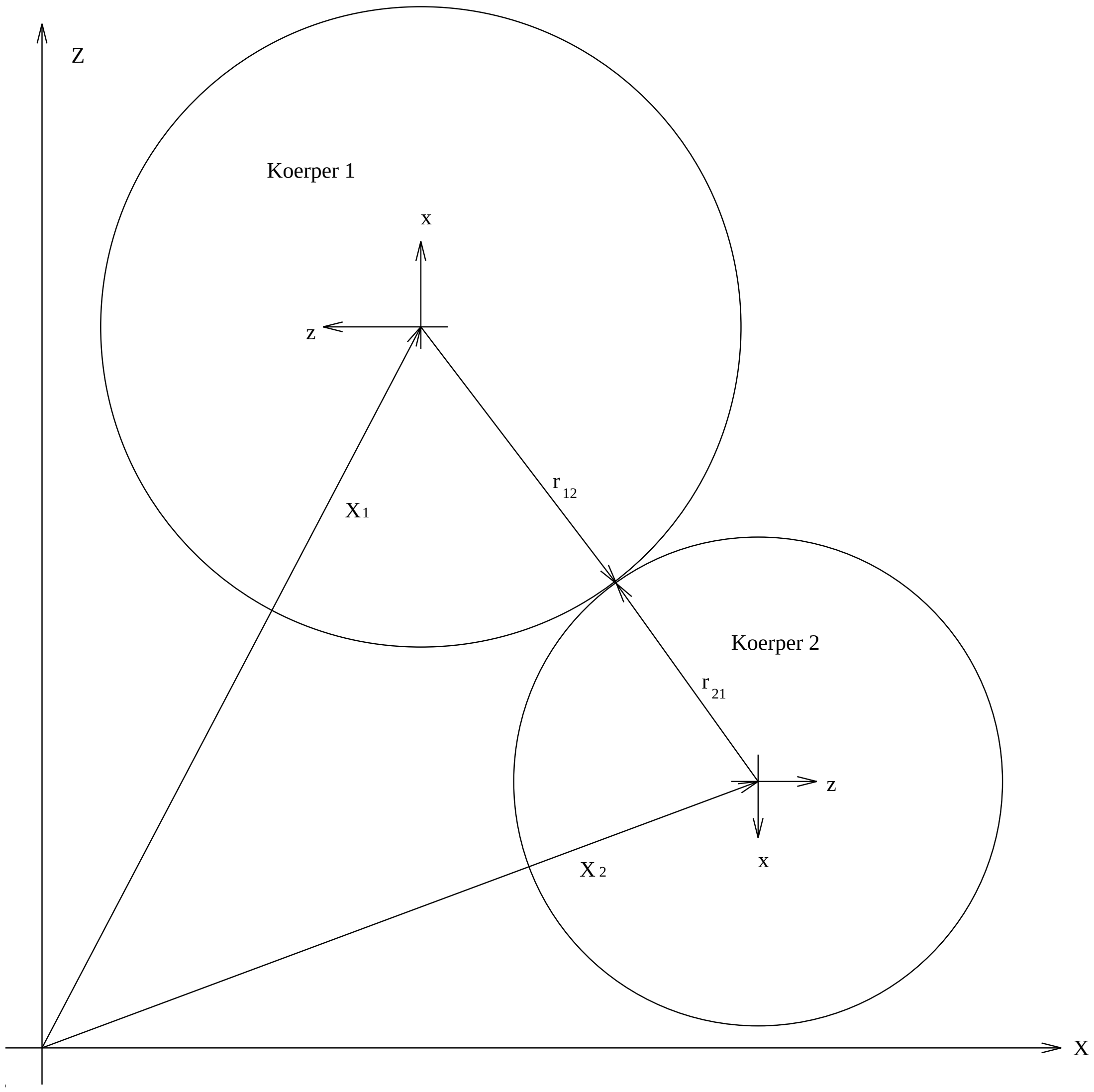

Abbildung 1: Formulierung der Zwangsbedingungen

Die Zwangsbedingungen werden anhand eines einfachen, zweigliedrigen Modells aus Abb. 1 formuliert und später durch Anpassung der Nomenklatur auf kinematische Ketten und Bäume angewendet. Man betrachte dazu zwei Körper, deren Schwerpunkt durch die Ortskoordinaten $\boldsymbol{X}_i$ im raumfesten System beschrieben wird. Im jeweiligen körperfesten Koordinatensystem sitzt an der Stelle $\boldsymbol{r}_{12}$ des Körpers 1 bzw. $\boldsymbol{r}_{21}$ des Körpers 2 das beide Körper verbindende Gelenk. Die Vektoren $\boldsymbol{r}_{ij}$ werden im folgenden als Hebel oder Gelenkhebel bezeichnet. Der erste

Index ist der Körper, von dessen Schwerpunkt der Hebel wegführt, der zweite Index trägt die Nummer desjenigen Körpers, der über das betreffende Gelenk mit dem ersten Körper verbunden ist. Vektoren in körperfesten Koordinaten stehen in Kleinbuchstaben, Vektoren in raumfesten Koordinaten stehen in Großbuchstaben. Die Transformation von körperfesten Vektoren auf raumfeste Vektoren erfolgt durch die Matrix A^T aus Gleichung (6). Da die Orientierung eines jeden Körpers durch drei eigene Eulerwinkel beschrieben wird, wird an die Matrix noch ein Index mit der Körpernummer angefügt. Es gilt:

$$\begin{aligned} \boldsymbol{R}_{ij} &= A_i^T \boldsymbol{r}_{ij} && (33) \\ \boldsymbol{r}_{ij} &= A_i \boldsymbol{R}_{ij} && (34) \end{aligned}$$

Das Gelenk sei eine idealisierte punktförmige Verbindung ohne Ausdehnung. Zur Veranschaulichung der Nomenklatur dient die Abbildung 1.

2.8.1 Kugel- und Scharniergelenk

In diesem Abschnitt werden die Zwangsbedingungen zur Modellierung eines Kugel- und eines Scharniergelenks aufgestellt. Ein Kugelgelenk ist nichts anderes als zwei an genau einem gemeinsamen Punkt verbundene Körper. Dieser gemeinsame Punkt wird in jedem Körperkoordinatensystem durch einen Gelenkhebel ausgezeichnet. Ausgedrückt in raumfesten Koordinaten ergibt sich daraus folgende Bedingung:

$$\boldsymbol{X}_1 + A_1^T \boldsymbol{r}_{12} = \boldsymbol{X}_2 + A_2^T \boldsymbol{r}_{21} \qquad (35)$$

Möchte man aus einem Kugelgelenk ein Scharniergelenk machen, muß man zusätzlich eine Gelenkachse angeben, die die Richtung der Gelenkbewegung festlegt. Die Angabe zweier nicht identischer Verbindungspunkte in der Art von Glg. (35), die voneinander einen festen Abstand haben, wäre prinzipiell dazu geeignet, diese Achse zu definieren, vorausgesetzt, man eliminiert eine Gleichung unter Verwendung der Eigenschaft des festen Abstandes. Dies ist erforderlich, da die Definition des Scharniergelenks nur über fünf Zwangsbedingungen erfolgen darf und in der hier zugrundegelegten Formulierung der Mechanik alle Zwangsbedingungen linear unabhängig sein müssen. Weil diese Elimination jedoch zusätzlichen Aufwand darstellt, soll hier eine andere Methode zur Beschreibung des Scharniergelenks verwendet werden. Man definiert in beiden Körpersystemen eine Gelenkebene durch einen Normalenvektor, der gleichzeitig die Achsrichtung des Scharniers festlegt. Die Definition der Ebene erfolgt durch je zwei vom Gelenkpunkt eines jeden Körpers ausgehende, nicht parallele Vektoren $\boldsymbol{\sigma}_{1ij}$ und $\boldsymbol{\sigma}_{2ij}$, deren Kreuzprodukt die Achse des Scharniergelenks $\boldsymbol{n}_{ij}$ darstellt:

$$\boldsymbol{n}_{ij} = \boldsymbol{\sigma}_{1ij} \times \boldsymbol{\sigma}_{2ij} \qquad (36)$$

Die Endpunkte der Vektoren $\boldsymbol{\sigma}$ sollen im folgenden Stützpunkte heißen und sind vom jeweiligen Körperschwerpunkt aus durch den Ortsvektor $\boldsymbol{s}$ definiert:

$$\boldsymbol{s} = \boldsymbol{r}_{ij} + \boldsymbol{\sigma} \tag{37}$$

Die Formulierung zweier linear unabhängiger Zwangsbedingungen erfolgt nun durch die Forderung, daß beide Achsen identisch, d. h. parallel, sein müssen. Beide Körper sind sind bereits durch den Gelenkpunkt miteinander verbunden, was eine parallelverschobene, unzusammenhängende Bewegung verhindert. Wie man also sieht, ist diese Forderung ausreichend zur Modellierung eines Scharniergelenks. Erfüllt ist diese Forderung dadurch, daß jeder Stützpunkt eines Körpers in der durch den Achsvektor des anderen Körpers und den gemeinsamen Gelenkpunkt definierten Gelenkebene liegt, d. h. ihre Ebenengleichung erfüllt. Große Buchstaben stehen für die entsprechenden Vektoren in raumfesten Koordinaten. Alle Punkte $\boldsymbol{Y}$ der Gelenkebene in raumfesten Koordinaten erfüllen die (skalare) Ebenengleichung:

$$(\boldsymbol{Y} - \boldsymbol{X}_i)\boldsymbol{N}_{ij} = 0 \tag{38}$$

Dies gilt insbesondere wie eben gefordert für die beiden Stützpunkte $\boldsymbol{s}_{ji}$ des anderen Körpers. Man erhält damit neben Gleichung (35) folgende zusätzliche Zwangsbedingungen für das Scharniergelenk:

$$(\boldsymbol{X}_j + \boldsymbol{S}_{1ji} - \boldsymbol{X}_i)\boldsymbol{N}_{ij} = 0 \tag{39}$$

$$(\boldsymbol{X}_j + \boldsymbol{S}_{2ji} - \boldsymbol{X}_i)\boldsymbol{N}_{ij} = 0 \tag{40}$$

Das System aus zwei Starrkörpern hätte ohne Gelenk 12 Freiheitsgrade. Mit einem Kugelgelenk, dessen Zwangsbedingungen in Form von 3 Gleichungen (35) formuliert sind, hat es nur noch $12 - 3 = 9$ Freiheitsgrade. Das Kugelgelenk "kostet" also 3 Freiheitsgrade. Als reduzierte Koordinaten zur Beschreibung des Systems würde man beispielsweise die 3 Schwerpunktskoordinaten des ersten Körpers, seine 3 Eulerwinkel und die 3 Eulerwinkel des zweiten Körpers verwenden. Die Position des zweiten Körpers ist durch die Vorgabe des Kugelgelenkes bereits bestimmt. Betrachtet man nun das Scharniergelenk, so "kostet" dies zwei weitere Freiheitsgrade. Ist nämlich die Orientierung und die Lage des ersten Körpers beschrieben, ist der Gelenkpunkt des zweiten Körpers fest und der zweite Körper kann sich nur noch innerhalb der Gelenkfläche orientieren. Als reduzierte Koordinaten bieten sich die 3 Schwerpunktskoordinaten und 3 Eulerwinkel des ersten Körpers an, die Orientierung und die Lage des zweiten Körpers ist durch die Angabe des Gelenkwinkels als weitere Koordinate vollständig beschrieben.

2.8.2 Dreidimensionale Bewegungsgleichungen mit Kugelgelenk

Um die oben angegebenen Zwangsbedigungen in das Gleichungssystem (22) im Sinne von Glg. (23) miteinzubeziehen, muß man bei Verwendung vollständiger Koordinaten sowohl die in Gleichung (32) aufgeführten partiellen Ableitungen aus den Gleichungen (35) bzw. (39) und (40) errechnen als auch die zweiten

Zeitableitungen der letztgenannten Gleichungen aufstellen. Man erhält so ein $3N + k$ dimensionales Differentialgleichungssystem der Form:

$$Q\boldsymbol{y} = \boldsymbol{u} \tag{41}$$

Hierbei ist Q die noch aufzustellende Koeffizientenmatrix und $\boldsymbol{y}$ sind die Unbekannten, die im Falle des zweigliedrigen Modells ($N = 2$) mit einem Kugelgelenk ($k = 3$) die folgende Form haben:

$$\boldsymbol{y}^T = \begin{pmatrix} \ddot{x}_1 & \ddot{y}_1 & \ddot{z}_1 & \ddot{\phi}_1 & \ddot{\theta}_1 & \ddot{\psi}_1 & \ddot{x}_2 & \ddot{y}_2 & \ddot{z}_2 & \ddot{\phi}_2 & \ddot{\theta}_2 & \ddot{\psi}_2 & \lambda_1 & \lambda_2 & \lambda_3 \end{pmatrix} \tag{42}$$

In $\boldsymbol{u}$ sind alle verbleibenden Glieder der Gleichung (22) zusammengefaßt. Als erstes soll auf die Errechnung der zweiten Zeitableitung der Zwangsbedingungen eingegangen werden. Dazu werden zunächst die Gleichungen (35) zweimal abgleitet und mithilfe der Beziehung (21) umgeformt. Man erhält:

$$\ddot{\boldsymbol{X}}_1 - \ddot{\boldsymbol{X}}_2 + \dot{\boldsymbol{\omega}}_1 \times \boldsymbol{r}_{12} - \dot{\boldsymbol{\omega}}_2 \times \boldsymbol{r}_{21} + \boldsymbol{\omega}_1(\boldsymbol{r}_{12}\boldsymbol{\omega}_1) - \boldsymbol{r}_{12}(\boldsymbol{\omega}_1^2) - \boldsymbol{\omega}_2(\boldsymbol{r}_{21}\boldsymbol{\omega}_2) + \boldsymbol{r}_{21}(\boldsymbol{\omega}_2^2) = \mathbf{o} \tag{43}$$

Hierin tragen die ersten vier Summanden zu folgenden drei Zeilen der gesuchten Koeffizientenmatrix Q bei:

$$\begin{pmatrix} E & D_{12} & -E & -D_{21} & O \end{pmatrix} \tag{44}$$

Es sind alle Symbole E, D, und O 3×3 Matrizen. E ist die Einheitsmatrix und O ist die Nullmatrix. Die Matrix D hat folgende Form (die jeweils erste Indexziffer aus der oberen Gleichung ist an den Winkeln anzubringen, jeweils beide Ziffern an den Komponenten von $\boldsymbol{r}$):

$$D = \begin{pmatrix} r_z \cos\psi \sin\theta & -r_z \sin\psi & -r_y \\ -r_z \sin\psi \sin\theta & -r_z \cos\psi & r_x \\ r_y \sin\psi \sin\theta - r_x \cos\psi \sin\theta & r_y \cos\psi + r_x \sin\psi & 0 \end{pmatrix} \tag{45}$$

Die übrigen Summanden sind die zu diesen Gleichungen gehörenden Komponenten des Vektors $\boldsymbol{u}$. In Vektorschreibweise lauten mit $\varrho_{ij} = \boldsymbol{\omega}_i \boldsymbol{r}_{ij}$ und $\omega_i^2 = \boldsymbol{\omega}_i \boldsymbol{\omega}_i$ die Komponenten:

$$\varrho_{12}\boldsymbol{\omega}_1 + \omega_1^2 \boldsymbol{r}_{12} - \varrho_{21}\boldsymbol{\omega}_2 - \omega_2^2 \boldsymbol{r}_{21} + \begin{pmatrix} k_x r_z - r_x k_z \\ k_z r_x - r_z k_x \\ k_x r_y - r_x k_y \end{pmatrix}_{12} - \begin{pmatrix} k_x r_z - r_x k_z \\ k_z r_x - r_z k_x \\ k_x r_y - r_x k_y \end{pmatrix}_{21} \tag{46}$$

mit

$$\boldsymbol{k} = \begin{pmatrix} \dot{\phi}\dot{\psi} \cos\psi \sin\theta - \dot{\theta}\dot{\psi} \sin\psi + \dot{\phi}\dot{\theta} \sin\psi \cos\theta \\ \dot{\phi}\dot{\theta} \cos\psi \cos\theta - \dot{\theta}\dot{\psi} \cos\psi - \dot{\phi}\dot{\psi} \sin\psi \sin\theta \\ -\dot{\phi}\dot{\theta} \sin\theta \end{pmatrix} \tag{47}$$

aus Gleichung (12). Mit den Umformungen, die bis hierher gemacht wurden, sind drei der 15 Gleichungen aufgestellt. Von den restlichen 12 Gleichungen ist der Teil der Koeffizientenmatrix Q, der nur die 12 Koordinaten betrifft, bereits durch die linke Seite der Gleichung (22) gegeben. An dieser Stelle muß also nur noch der Teil der Matrix angegeben werden, der die drei λ_1, λ_2 und λ_3 betrifft. Es sind dies von den ersten 12 Zeilen die drei letzten Spalten der Matrix Q, die drei übrigen Zeilen sind bereits angegeben worden. Die ersten 12 Zeilenenden berechnen sich wie in Gleichung (30) angegeben zu:

$$\left.\begin{array}{cccc}
\cdots & 1 & 0 & 0 \\
\cdots & 0 & 1 & 0 \\
\cdots & 0 & 0 & 1 \\
\cdots & a_{\phi_1 1} & a_{\phi_1 2} & a_{\phi_1 3} \\
\cdots & a_{\theta_1 1} & a_{\theta_1 2} & a_{\theta_1 3} \\
\cdots & a_{\psi_1 1} & a_{\psi_1 2} & a_{\psi_1 3} \\
\cdots & -1 & 0 & 0 \\
\cdots & 0 & -1 & 0 \\
\cdots & 0 & 0 & -1 \\
\cdots & -a_{\phi_2 1} & -a_{\phi_2 2} & -a_{\phi_2 3} \\
\cdots & -a_{\theta_2 1} & -a_{\theta_2 2} & -a_{\theta_2 3} \\
\cdots & -a_{\psi_2 1} & -a_{\psi_2 2} & -a_{\psi_2 3}
\end{array}\right) \tag{48}$$

mit

$$\begin{aligned}
\boldsymbol{a}_{\phi_1} &= \frac{\partial}{\partial\phi} A_1 \boldsymbol{r}_{12} \\
\boldsymbol{a}_{\theta_1} &= \frac{\partial}{\partial\theta} A_1 \boldsymbol{r}_{12} \\
\boldsymbol{a}_{\psi_1} &= \frac{\partial}{\partial\psi} A_1 \boldsymbol{r}_{12}
\end{aligned}$$

Analog dazu sind die anderen Matrixelemente definiert, jedoch mit den Winkeln und Gelenkhebelkomponenten des anderen Körpers. Es gilt im einzelnen:

$$\begin{aligned}
\boldsymbol{a}_{\phi} &= \begin{pmatrix} -r_x(\cos\psi\sin\phi + \cos\theta\cos\phi\sin\psi) + r_y(\cos\psi\cos\phi - \cos\theta\sin\phi\sin\psi) \\ r_x(\sin\psi\sin\phi - \cos\theta\cos\phi\cos\psi) + r_y(\sin\psi\cos\phi - \cos\theta\sin\phi\cos\psi) \\ r_x\sin\theta\cos\phi + r_y\sin\theta\sin\phi \end{pmatrix} \\
\boldsymbol{a}_{\theta} &= \begin{pmatrix} r_x\sin\theta\sin\phi\sin\psi - r_y\sin\theta\cos\phi\sin\psi + r_z\sin\psi\cos\theta \\ r_x\sin\theta\sin\phi\cos\psi - r_y\sin\theta\cos\phi\cos\psi + r_z\cos\psi\cos\theta \\ r_x\cos\theta\sin\phi - r_y\cos\theta\cos\phi - r_z\sin\theta \end{pmatrix} \\
\boldsymbol{a}_{\psi} &= \begin{pmatrix} -r_x(\sin\psi\cos\phi + \cos\theta\sin\phi\cos\psi) - r_y(\sin\psi\sin\phi - \cos\theta\cos\phi\cos\psi) \\ -r_x(\cos\psi\cos\phi - \cos\theta\sin\phi\sin\psi) - r_y(\cos\psi\sin\phi + \cos\theta\cos\phi\sin\psi) \\ 0 \end{pmatrix} \\
&+ \begin{pmatrix} r_z\cos\psi\sin\theta \\ -r_z\sin\psi\sin\theta \\ 0 \end{pmatrix}
\end{aligned} \tag{49}$$

Die Bewegungsgleichungen für zwei über ein Kugelgelenk verbundene Körper sind damit aufgestellt. Im nächsten Abschnitt wird dasselbe Problem in zwei Dimensionen gelöst.

2.8.3 Zweidimensionale Bewegungsgleichungen mit Gelenk

In zwei Dimensionen sind Kugel- und Scharniergelenk identisch. Es müssen daher nur die Überlegungen des Kugelgelenks ins Zweidimensionale übertragen werden. Auch im Zweidimensionalen stellt man das Gleichungssystem ganz analog zum Dreidimensionalen auf. Um zwei über ein Gelenk verbundene Körper in dem hier dargestellten Formalismus zu erfassen, werden $3N + 2k = 8$ Gleichungen benötigt, wenn N die Körperzahl und k die Gelenkzahl bedeuten. Man benötigt also 8 Gleichungen. Diese werden prinzipiell wie im Dreidimensionalen aufgestellt, sie sind jedoch weniger umfangreich. Aufgrund der einfachen Gestalt der zweidimensionalen Drehmatrix A, die nur von einem einzigen Winkel ϕ abhängt, kann man bei der Herleitung auf die Beziehung (21) verzichten und stattdessen eine einfachere Beziehung verwenden. Es transformiere sich der in Körperkoordinaten gegebene (zweidimensionale) Vektor $\boldsymbol{r}$ mit der Matrix A in raumfeste Koordinaten $\boldsymbol{R}$:

$$\boldsymbol{R} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \boldsymbol{r} \tag{50}$$

Unter der Voraussetzung, daß $\dot{\boldsymbol{r}} = 0$ verschwindet, lautet die Zeitableitung des Vektors $\boldsymbol{R}$:

$$\dot{\boldsymbol{R}} = \dot{\phi} \begin{pmatrix} -\sin\phi & -\cos\phi \\ \cos\phi & -\sin\phi \end{pmatrix} \boldsymbol{r} = \dot{\phi} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} A\boldsymbol{r} \tag{51}$$

Man erreicht also die Zeitableitung des Vektors $\boldsymbol{R}$ durch Multiplikation mit der Winkelgeschwindigkeit und einer antisymmetrischen Matrix P, wobei gilt:

$$P^2 = -1$$

Die zweite Zeitableitung berechnet sich damit wie folgt:

$$\ddot{\boldsymbol{R}} = \ddot{\phi} P \boldsymbol{R} - \dot{\phi}^2 \boldsymbol{R} \tag{52}$$

Die Zeitableitungen der Zwangsbedingungen schreiben sich deshalb im Zweidimensionalen wie folgt:

$$\ddot{\boldsymbol{X}}_1 + \ddot{\phi}_1 P \boldsymbol{R}_{12} - \ddot{\boldsymbol{X}}_2 - \ddot{\phi}_2 P \boldsymbol{R}_{21} = \dot{\phi}_1^2 \boldsymbol{R}_{12} - \dot{\phi}_2^2 \boldsymbol{R}_{21} \tag{53}$$

Die beiden letzten Spalten der ersten 6 Zeilen der Koeffizientenmatrix Q errechnen sich ebenfalls analog zum Dreidimensionalen und man erhält für das Gleichungssystem:

$$
\begin{pmatrix}
m_1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & m_1 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & \Theta_1 & 0 & 0 & 0 & R_{12y} & -R_{12x} \\
0 & 0 & 0 & m_2 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & m_2 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & \Theta_2 & -R_{21y} & R_{21x} \\
1 & 0 & -R_{12y} & -1 & 0 & R_{21y} & 0 & 0 \\
0 & 1 & R_{12x} & 0 & -1 & -R_{21x} & 0 & 0
\end{pmatrix} \boldsymbol{y} = \begin{pmatrix}
F_{1x} \\
F_{1y} \\
M_1 \\
F_{2x} \\
F_{2y} \\
M_2 \\
\dot{\phi}_1^2 R_{12x} - \dot{\phi}_2^2 R_{21x} \\
\dot{\phi}_1^2 R_{12y} - \dot{\phi}_2^2 R_{21y}
\end{pmatrix} \tag{54}
$$

Hierin ist

$$
\boldsymbol{y}^T = \begin{pmatrix} \ddot{x}_1 & \ddot{y}_1 & \ddot{\phi}_1 & \ddot{x}_2 & \ddot{y}_2 & \ddot{\phi}_2 & \lambda_1 & \lambda_2 \end{pmatrix}
$$

Hiermit sind die Bewegungsgleichungen für zwei über ein Gelenk verbundene Körper im Zweidimensionalen aufgestellt.

2.8.4 Feste Aufhängung

Um kinematische Ketten an raumfesten Punkten aufzuhängen oder zu befestigen, muß eine entsprechende Zwangsbedingung formuliert werden. Ein raumfester Punkt ist im Prinzip ein Kugelgelenk mit einem fixierten Körper. In der Gleichung des Kugelgelenkes sind also die Koordinaten des fixierten Körpers durch Konstanten zu ersetzen. Die Nomenklatur soll von den Gelenken übernommen werden, um die raumfesten Punkte als Sonderfall von Gelenken behandeln zu können. Der Index j am Körper i bezieht sich daher auf den fixierten (und daher durch Konstanten ersetzten) Körper j analog zur Gelenkdefinition zwischen den Körpern i und j. Ist $\boldsymbol{Y}_i$ der festzuhaltende Punkt in raumfesten Koordinaten, so ergibt sich für die Zwangsbedingung des am Hebelendpunkt $\boldsymbol{r}_{ij}$ festgehaltenen Körpers i:

$$
\boldsymbol{X}_i + A_1^T \boldsymbol{r}_{ij} = \boldsymbol{Y}_i \tag{55}
$$

Mit dieser Art der Zwangsbedingung läßt sich beispielsweise ein sphärisches Pendel realisieren. Möchte man daraus ein ebenes Pendel machen, sind hierzu ähnliche Überlegungen erforderlich wie beim Übergang vom Kugelgelenk zum Scharniergelenk. Die Gelenkebene läßt sich jedoch einfacher angeben, da es sich hier um einen raumfesten Bezugspunkt handelt. Man gibt die Ebene durch einen Normalenvektor $\boldsymbol{N}_i$ in raumfesten Koordinaten an. Man erhält:

$$
(\boldsymbol{X}_i - \boldsymbol{Y}_i)\boldsymbol{N}_i = 0 \tag{56}
$$

$$
(\boldsymbol{X}_i + A_j^T \boldsymbol{s}_{ji} - \boldsymbol{Y}_i)\boldsymbol{N}_{ij} = 0 \tag{57}
$$

Im Zweidimensionalen fallen beide Fälle zusammen und daher lautet die Zwangsbedingung genauso wie die aus Gleichung (55), jedoch nur in zwei Dimensionen.

2.8.5 Dreidimensionale Bewegungsgleichungen mit fester Aufhängung

Das Aufstellen der Bewegungsgleichungen erfolgt genauso wie im Falle des Kugelgelenks. Im Unterschied dazu sind jedoch keine zwei Körper beteiligt, sondern nur einer. Daher reduziert sich die Gleichungszahl von 15 auf 9 Gleichungen. Aufzustellen ist wieder das Gleichungssystem (41), wobei hier gilt:

$$\boldsymbol{y}^T = \begin{pmatrix} \ddot{x} & \ddot{y} & \ddot{z} & \ddot{\phi} & \ddot{\theta} & \ddot{\psi} & \lambda_1 & \lambda_2 & \lambda_3 \end{pmatrix} \tag{58}$$

Die gesuchte Koeffizientenmatrix Q ist eine 9×9 Matrix. Die Inhomogenität $\boldsymbol{u}$ hat ebenfalls 9 Komponenten. Aufgrund der Ähnlichkeit der Zwangsbedingungen mit der des Kugelgelenks lassen sich die durch Ableiten der Zwangsbedingung gefundenen Gleichungen sofort hinschreiben. Die drei letzten Zeilen der Koeffizientenmatrix lauten:

$$\begin{pmatrix} E & D & O & O & O \end{pmatrix} \tag{59}$$

Es sind die Symbole E, D, und O dieselben Matrizen wie in Glg. (44) und (45). Die zugehörigen Komponenten des Vektors u lauten mit $\boldsymbol{k}$ aus Glg. (47), $\varrho = \boldsymbol{\omega r}$ und $\omega^2 = \boldsymbol{\omega\omega}$:

$$\varrho\boldsymbol{\omega} + \omega^2\boldsymbol{r} + \begin{pmatrix} k_x r_z - r_x k_z \\ k_z r_x - r_z k_x \\ k_x r_y - r_x k_y \end{pmatrix} \tag{60}$$

Durch das Differenzieren der Zwangsbedingungen ist der konstante Vektor $\boldsymbol{Y}$ nun nicht mehr Bestandteil der Bewegungsgleichungen. Um dennoch die Zwangsbedingung jederzeit zu erfüllen, müssen die Anfangsbedingungen des Gleichungssystems entsprechend gewählt werden. Die Bewegungsgleichungen "sorgen" dann dafür, daß der betroffene Körper am entsprechenden Punkt ruht. Welche Lage dieser Punkt im Raum hat, bestimmen die Anfangsbedingungen und damit der Vektor $\boldsymbol{Y}$. Die letzten drei Spalten der ersten 6 Zeilen der Koeffizientenmatrix errechnen sich wieder durch partielles Ableiten der Zwangsbedingungen und man erhält mit den $\boldsymbol{a}_i$ aus Glg. (49):

$$\left. \begin{array}{cccc} \cdots & 1 & 0 & 0 \\ \cdots & 0 & 1 & 0 \\ \cdots & 0 & 0 & 1 \\ \cdots & a_{\phi 1} & a_{\phi 2} & a_{\phi 3} \\ \cdots & a_{\theta 1} & a_{\theta 2} & a_{\theta 3} \\ \cdots & a_{\psi 1} & a_{\psi 2} & a_{\psi 3} \end{array} \right) \tag{61}$$

Hiermit sind die Bewegungsgleichungen für eine an einem festen Punkt aufgehängte kinematische Kette in drei Dimensionen aufgestellt. Im nächsten Abschnitt werden entsprechende Gleichungen fürs Zweidimensionale aufgestellt.

2.8.6 Zweidimensionale Bewegungsgleichungen mit fester Aufhängung

Auch hier ist der Weg zu den Bewegungsgleichungen wieder derselbe wie im Falle des Gelenks. Das hier abzuleitende Gleichungssystem besteht jedoch nur aus 5 Gleichungen, da nur ein einzelner Körper betrachtet werden muß. Das Gleichungssystem lautet:

$$\begin{pmatrix} m & 0 & 0 & -1 & 0 \\ 0 & m & 0 & 0 & -1 \\ 0 & 0 & \Theta & R_y & -R_x \\ 1 & 0 & -R_y & 0 & 0 \\ 0 & 1 & R_x & 0 & 0 \end{pmatrix} \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\phi} \\ \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} F_x \\ F_y \\ M \\ \dot{\phi}^2 R_x \\ \dot{\phi}^2 R_y \end{pmatrix} \tag{62}$$

2.9 Numerische Integration der Gleichungen

Um das Differentialgleichungssystem eines Mehrkörpersystems zu lösen, ist man auf numerische Methoden angewiesen, da man für die nichtlinearen Gleichungen nur in ganz wenigen Spezialfällen noch analytische Lösungen finden kann. Während das Ergebnis analytischer Lösungen zeitabhängige Funktionen sind, erhält man durch die numerische Integration für jeden Satz von Anfangswerten bestimmte Endwerte, die ebenfalls Lösung des Systems sind. Grundsätzliche Zusammenhänge bleiben allerdings verborgen und man ist auf das Integrieren einer ganzen Reihe von verschiedenen Anfangswerten angewiesen, um ein Gefühl für das simulierte System zu bekommen.
Die numerische Integration der Bewegungsgleichungen funktioniert prinzipiell wie folgt. Zu Beginn der numerischen Integration eines Zeitintervalls wird ein kompletter Satz von Anfangskoordinaten und -geschwindigkeiten benötigt. Abhängig vom verwendeten Verfahren wird das Integrationsintervall durch die Einführung von unterschiedlich vielen Stützstellen in kleinere Intervalle zerlegt. An der ersten Stützstelle werden die Koordinaten und Geschwindigkeiten aus den Anfangswerten extrapoliert und mit diesen Werten werden die neuen Beschleunigungen errechnet. Im vorliegenden Fall sind die Bewegungsgleichungen zu jedem Zeitpunkt ein lineares Gleichungssystem in den Beschleunigungen und den Lagrangeschen Multiplikatoren. Dieses wird faktorisiert, d. h. in Dreiecksform gebracht und anschließend gelöst. Mit den so erhaltenen Beschleunigungen an einer Stützstelle werden die Geschwindigkeiten und Koordinaten an der nächsten Stützstelle errechnet. Die letzten Schritte wiederholen sich solange, bis das Ende des geforderten Integrationsintervalls erreicht ist. Die gängigen Integrationsverfahren unterscheiden sich in der Wahl der Stützstellen (feste oder variable Schrittweite, Anzahl Stützstellen) und in der Berechnung der Koordinaten und Geschwindigkeiten aus den an den Stützstellen erhaltenen Beschleunigungen (linear oder höhere Ordnungen, in der Ordnung variabel, d. h. fehlerabhängig).

Die numerische Integration macht verfahrensbedingt Fehler, die im Einzelfall zu völlig falschen Ergebnissen führen können. Es empfiehlt sich deshalb eine ständige Überwachung des Integrationsfehlers. Da in den Bewegungsgleichungen die Zwangsbedingungen mitintegriert werden, machen sich die Integrationsfehler dadurch bemerkbar, daß die eigentlich am Gelenkpunkt verknüpften Körper am Gelenkpunkt Abstände von der Größenordung des Integrationsfehlers haben. Für hochpräzise, technische Anwendung mag dies von Bedeutung sein, nicht jedoch

für die Simulation von biomechanischen Modellen, da die hier simulierten Gelenke von Natur aus nicht so präzise arbeiten wie sie in den Zwangsbedinungen modelliert sind. Die Bewegungsgleichungen in vollständigen Koordinaten und deren numerische Lösung ist also grundsätzlich geeignet, biomechanische Mehrkörpersysteme zu simulieren.
Für die Integration der Bewegungsgleichungen wurde die Integrationsroutine *de_()*[2] verwendet, die als FORTRAN *SUBROUTINE DE()* am Institut verfügbar ist. Die Routine *de_()* arbeitet nach einem Verfahren von L. F. Shampine und M. K. Gordon, das fehlerabhängig automatisch die Ordnung und die Schrittweite steuert. Die Routine *de_()* integriert jedoch nur Differentialgleichungssysteme 1. Ordnung, weshalb die $6N$ Differentialgleichungen 2. Ordnung $\ddot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t)$ durch Einführen von Geschwindigkeiten $\boldsymbol{v}$ in $12N$ Differentialgleichungen 1. Ordnung umgeschrieben werden:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{v} \\ \dot{\boldsymbol{v}} &= \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{v}, t) \end{aligned} \tag{63}$$

Die Routine *de_()* ruft zur Berechnung der Beschleunigungen an jeder Stützstelle eine ihr zu übergebende Funktion auf, übergibt ihr Koordinaten und Geschwindigkeiten und erwartet von ihr die Rückgabe der errechneten Beschleunigungen und Geschwindigkeiten. Innerhalb dieser Funktion wurden zum Lösen des linearen Gleichungssystems in den Beschleunigungen die ebenfalls in FORTRAN vorliegenden Routinen *dgefa_()* und *dgesl_()* der Funktionensammlung LINPACK verwendet. Die Routine *dgefa_()* benutzt einen Gaußalgorithmus zur Faktorisierung (fa steht für factorize) des Gleichungssystems und die Routine *dgesl_()* löst das faktorisierte Gleichungssystem (sl steht für solve).

2.10 Maschinelle Gleichungsgenerierung

Ein großer Vorteil der Verwendung von vollständigen Koordinaten zur Beschreibung des MKS ist der, daß die resultierenden Bewegungsgleichungen eine systematisch einfache Gestalt haben. Die Bewegungsgleichungen sind daher prädestiniert dafür, von einem Computerprogramm maschinell generiert zu werden. Dieses Programm wird im folgenden als Bewegunggleichungsgenerator (*bgg*) bezeichnet. Ausgehend von einer normierten Beschreibung des Aufbaus eines MKS soll der Bewegungsgleichungsgenerator die Bewegungsgleichungen aufstellen. Dadurch lassen sich Flüchtigkeitsfehler bei der manuellen Generierung der Gleichungen vermeiden und die Geschwindigkeit bei der Umsetzung eines Modells von der Idee in die Simulation erheblich erhöhen. Die normierte Beschreibung des MKS soll es ermöglichen, aus Starrkörpern in beliebiger Weise verkettete Modelle zu definieren, die über die in den letzten Abschnitten diskutierten Primitiven verfügen:

---

[2] Alle im Rahmen dieser Arbeit geschriebenen Funktionen und Programme sind in der Programmiersprache C erstellt. Die Einbindung von FORTRAN Funktionen bedingt das Anhängen des Unterstrichs an den sonst gleichlautenden Funktionsnamen

Kugelgelenksverbindung und feste Aufhängung. Der Bewegungsgleichungsgenerator liest die Definitionen und setzt diese in Gleichungen um.

Der bisher betrachtete Fall von Körpern mit jeweils nur einem Gelenk läßt sich leicht verallgemeinern. Dazu wird die zugrundeliegende Nomenklatur der Gelenkhebel angepaßt. Der erste Index bezeichnet wie bisher den Körper, der zweite Index die Nummer des Gelenks am Körper. Die Gelenknummer ergibt sich aus der Reihenfolge der nach ihrer Körpernummer aufsteigend sortierten Nachbarkörper. Zur Illustration dient Abb. 2, in der die Körpernummern mit einem Kästchen umrahmt sind, die Gelenknummern eingekreist sind und die zugehörigen Hebelbezeichnungen eingetragen sind.

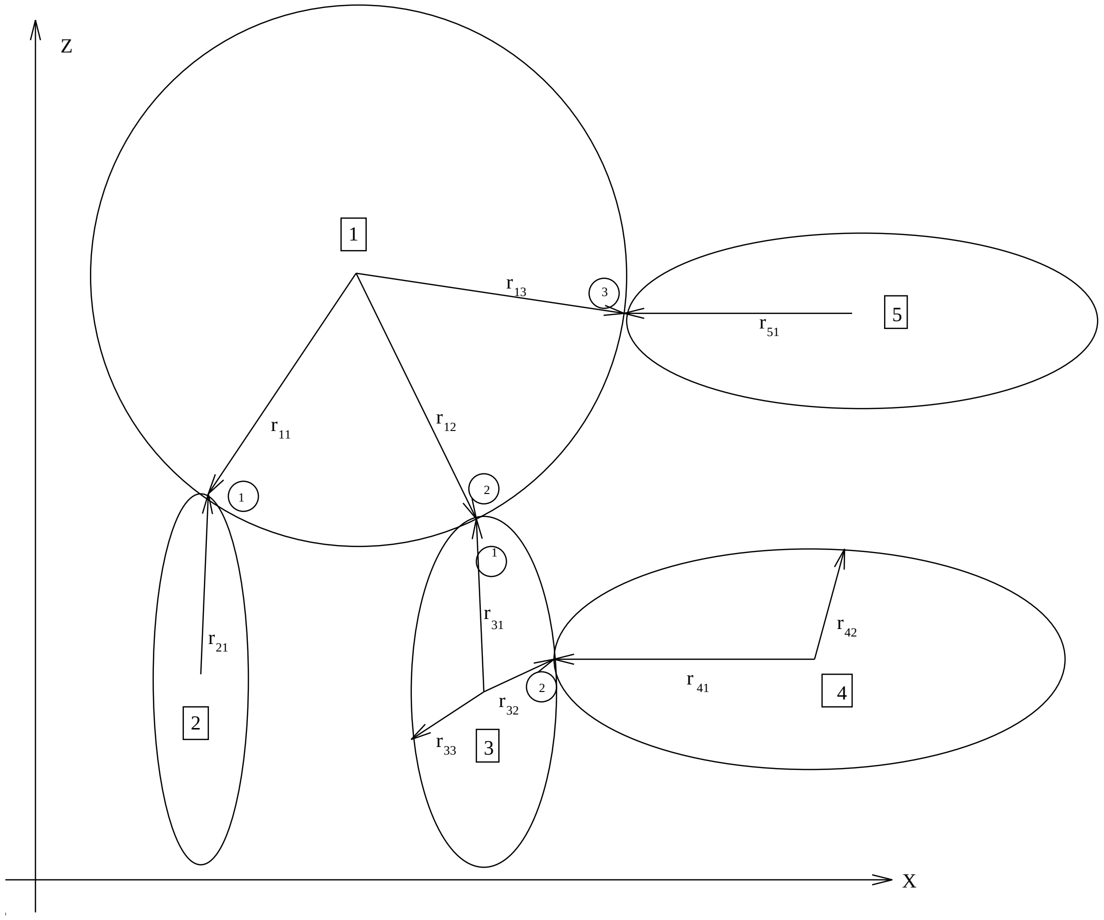

Abbildung 2: Nomenklatur der Gelenkhebel

Das Erstellen der zugehörigen Bewegungsgleichungen hat folgende Systematik: Besteht das MKS aus N Körpern, so sind zunächst $6N$ ungekoppelte Gleichungen der N frei beweglichen Körper aufzustellen. Anschließend betrachtet man die Primitiven. Jedes Kugelgelenk stellt eine Kopplung zwischen den beiden beteiligten Körpern dar und das Gleichungssystem muß erweitert werden. Es kommen 3 Lagrangesche Multiplikatoren dazu und die Koeffizientenmatrix erhält drei neue Zeilen und drei neue Spalten. In die Spalten werden in den Zeilen der betreffenden Körper die Kopplungsglieder eingetragen. Ein Kugelgelenk schlägt sich also durch drei eigene, zusätzliche Zeilen und drei eigene, zusätzliche Spalten in der Koeffizientenmatrix nieder. Für die feste Aufhängung gilt dasselbe: Die Koeffizientenmatrix erhält auch hier drei zusätzliche Spalten und Zeilen und es müssen ebenfalls drei Lagrangesche Multiplikatoren hinzugefügt werden. Der Unterschied zwischen Kugelgelenk und fester Aufhängung ist der, daß die Spalten des Kugelgelenks eine Kopplung zwischen den beteiligten Körpern darstellen, während die feste Aufhängung die Sache eines einzelnen Körpers ist.

Ausgehend von dem in diesem Abschnitt stets betrachteten Problem mit nur zwei Körpern gelangt man zum Gleichungssystem von vielen Körpern, indem man entsprechende Blöcke der Koeffizientenmatrix bei deren Vergrößerung "verschiebt". Zur Illustration dieser Systematik dient Abb. 3. Die Abbildung zeigt, welche Blöcke der Koeffizientenmatrix zweier über ein Gelenk verbundener Körper wie verschoben werden müssen, um die entsprechende Koeffizientenmatrix eines MKS mit mehr als zwei Körpern zu erhalten.

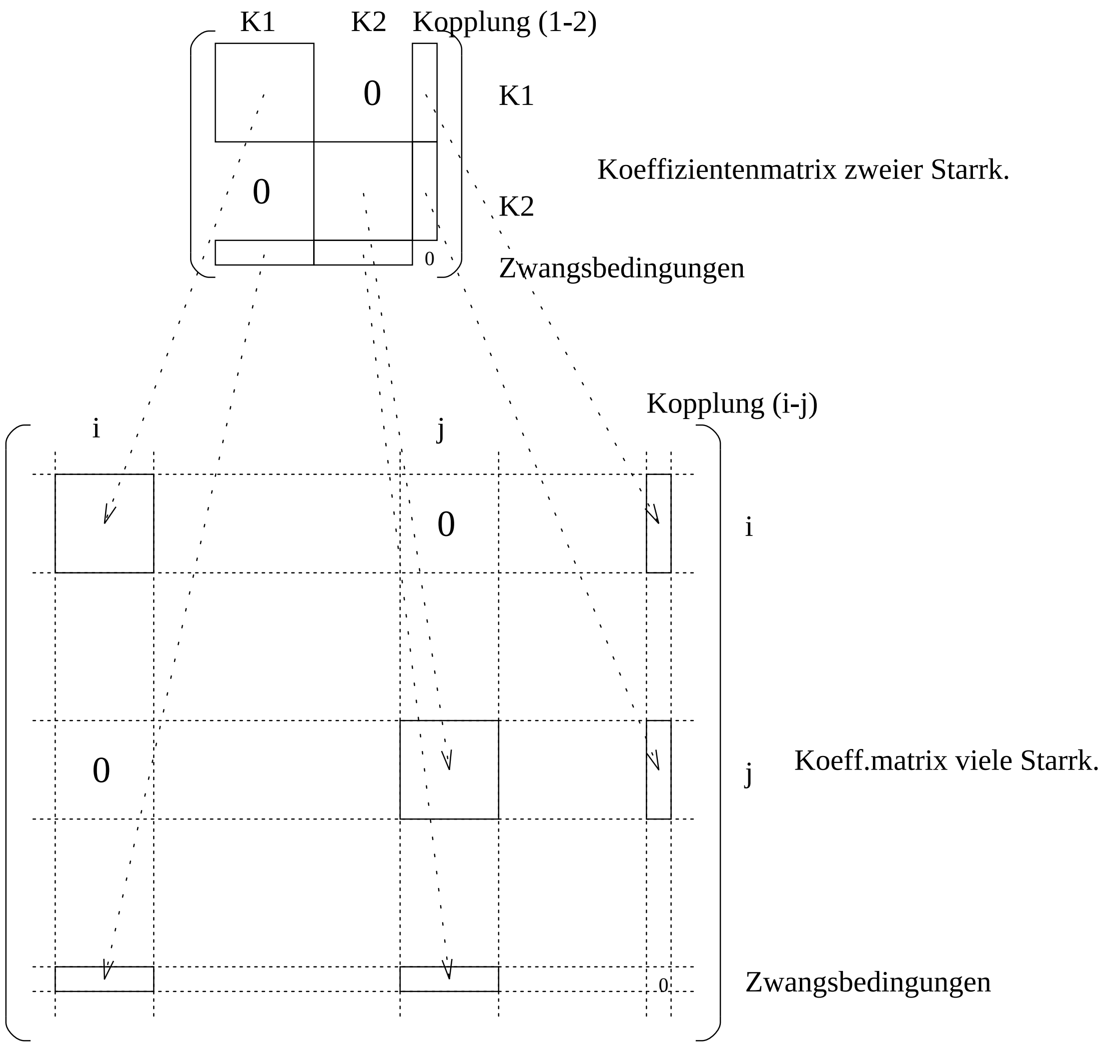

Abbildung 3: Systematik bei der maschinellen Gleichungsgenerierung

3 Entwurf eines Programmsystems

Nachdem sich das vorangegangene Kapitel mit der Herleitung der Bewegungsgleichungen der in der Biomechanik benötigten Formen kinematischer Ketten beschäftigt hat, soll sich dieses Kapitel mit grundsätzlichen Anforderungen an ein Programmsystem beschäftigen, die bei der praktischen Arbeit an einer biomechanischen Simulation anfallen. Dazu werden zunächst einige allgemein wichtige Problempunkte umrissen und daraus wünschenswerte Eigenschaften eines Programmsystems abgeleitet. Anschließend werden einige in verschiedenen Simulationen immer wiederkehrende Anforderungen an das simulierte System besprochen. Alle in diesem Kapitel angeführten Rechnungen beschränken sich auf zweidimensionale Modelle.

3.1 Allgemeine Anforderungen

3.1.1 Von der Idee zur Simulation

Um bestimmte Eigenschaften eines biomechanischen Systems zu simulieren, wird man sich zunächst ein einfaches, biomechanisches Modell machen, von dem man glaubt, daß es die gewünschten Eigenschaften besitzt. Anschließend wird man versuchen, die gewünschten Eigenschaften durch die Simulation zu validieren. Glaubt man, auf der richtigen Spur zu sein, wird man solange an der Verfeinerung des Modells arbeiten, bis die gewünschte Eigenschaft zur eigenen Zufriedenheit simulierbar ist. Zeigt der erste Entwurf nicht die gewünschten Eigenschaften, wird man solange mit anderen Modellen von vorne beginnen, bis sich ein gewisser Erfolg einstellt.

Die dabei durchzuführenden Tätigkeiten sind die Modellierung des Systems in Form kinematischer Ketten wie im vorangegangen Kapitel besprochen, die Erstellung der dazugehörigen Bewegungsgleichungen, die Programmierung der numerischen Integration, die Erzeugung der erforderlichen Daten und die (aufbereitete) Ausgabe der Daten in Form von Diagrammen, Plots oder Bewegungsabläufen auf dem Bildschirm. Jede Änderung am Modell, die sich bis hin zu den kinematischen Ketten auswirkt, hat eine komplette Überarbeitung, mindestens aber eine genaue Kontrolle der darauf aufbauenden Integration und Datenaufbereitung zur Folge. Wenn sich beispielsweise die Körperzahl des Modells erhöht oder wenn man die Anordnung verändert, müssen die Gleichungen entsprechend durchgesehen und angepaßt werden. Bei dieser Durchsicht wird der bestehende Programmcode dem verbesserten Modell angeglichen, was eine Fehlerquelle darstellt, die sich auch mit großer Sorgfalt nicht ausschließen läßt. Programmfehler, die sich aufgrund inkonsistenter Änderungen einzelner Programmteile oder Datenstrukturen ergeben, sind sehr schwer zu finden und sind sehr ärgerlich. Je grundlegender daher die Änderung und je größer und komplizierter das System ist, desto größer ist der Aufwand zur Anpassung. Als Folge dieser Arbeitsweise verschlingt die Umsetzung und Verfeinerung des Modells in ein ablauffähiges Programm zur Simulation erhebliche Zeit, die im Sinne der Modellierung oder Simulation nichts einbringt. Entscheidend für die Produktivität bei der Erstellung der Simulation ist also die schnelle Implementierbarkeit einer Idee oder Vermutung.

3.1.2 Konsequenzen

Um die schnelle Implementierbarkeit eigener Ideen und damit die eigene Produktivität zu verbessern, sind daher folgende Punkte wichtig:

Vollständiger Quellcode Die Generierung der Bewegungsgleichungen muß automatisch aufgrund einer normierten Beschreibung der Gestalt des Systems erfolgen. Die erzeugten Bewegungsgleichungen dürfen keine manuelle Nachbearbeitung mehr benötigen, da bei einem abermaligen Generieren (aufgrund notwendiger Änderungen am Systemaufbau) die bereits durchgeführten Anpassungen verloren gehen oder aber von Hand in die neu erzeugten Bewegungsgleichungen übernommen werden müßten. Zusammengefaßt heißt dies, daß die normierte Beschreibung in Bezug auf die Bewegungsgleichungen im Sinne einer Programmiersprache der vollständige Quellcode ist und alle zur Erstellung benötigten Informationen enthält. Der Bewegungsgleichungsgenerator ist der dazu passende Compiler.

Universelle Integrationsroutine Die maschinell erzeugten Bewegungsgleichungen sollen die Form einer normierten, parametrisierten Schnittstelle haben, über die eine universelle Integrationsroutine jede Art der so erzeugten Bewegungsgleichungen integrieren kann, unabhängig von der Gestalt des zu simulierenden Systems. Dies läßt sich am besten realisieren, wenn die Bewegungsgleichung in Form von Funktionen im Quellcode einer Programmiersprache erzeugt werden und die Integrationsroutine auf die Funktionen zurückgreift. Ist diese Bedingung erfüllt, kann man alleine aufgrund der oben erwähnten normierten Beschreibung des Systems ohne manuellen Eingriff eine Simulation durchführen. Der Bewegungsgleichungsgenerator übernimmt damit zusätzlich die Funktion eines Codegenerators.

Schnittstelle für individuellen Code Automatisch generierter Code, in den man nicht mehr manuell eingreifen kann, hat den Nachteil, daß in ihm nur die Fähigkeiten berücksichtigt sind, die der Programmierer des Codegenerators im Design berücksichtigt hat. Um dieses Manko zu umgehen, muß eine flexible Schnittstelle geschaffen werden, die das Einfügen von eigenem (individuell auf das Problem abgestimmtem) Code ermöglicht, ohne jedoch vom Prizip des vollständigen Quellcodes abzukommen. Dies wird erreicht, indem man an bestimmten Schlüsselstellen des Integrationsablaufs in vom Codegenerator automatisch generierte Funktionen springt, deren Inhalt der Codegenerator aus bestimmten Teilen der normierten Systembeschreibung entnimmt. Damit steht im Quellcode der Simulation neben der eigentlichen Systemgestalt auch noch "wirklicher" Quellcode einiger Funktionen.

Biomechanische Specials Schließlich sollten die in vielen Simulationen immer wieder benötigten Eigenheiten der Biomechanik nicht explizit in der eben besprochenen Form kodiert werden müssen, sondern entweder als parametrisierte Funktionen zur Verfügung stehen, deren Parameter zwischen einzelnen Simulationsläufen verändert werden, oder aber ebenfalls durch den Codegenerator erzeugt werden. Diese Eigenheiten werden im nächsten Abschnitt erläutert.

3.2 Biomechanische Eigenheiten

Es hat sich gezeigt, daß einige Eigenschaften speziell biomechanischer Modelle in vielen Anwendungen benötigt werden. Um diese im Entwurf des Programmsystems entsprechend zu berücksichtigen, werden sie in den folgenden Abschnitten einzeln besprochen.

3.2.1 Schwabbelmassen

Ein bei der Modellierung biomechanischer Systeme immer wieder auftretendes Problem ist, daß die Zusammensetzung der Körpermasse des simulierten Modells für die Qualität der Ergebnisse von entscheidender Bedeutung ist. Die biomechanische Körpermasse darf nicht nur aus über Gelenke verbundenen Starrkörpern modelliert werden, da dies dann unrealistisch wird, wenn hohe Beschleunigungen auftreten wie z. B. bei einem Aufprall. Man muß bei der Modellierung berücksichtigen, daß diese sowohl aus Knochenteilen als auch aus Weichteilen wie Fettgewebe, Muskulatur und inneren Organen bestehen. In diesem Fall wird nicht die gesamte Körpermasse auf einen Schlag abgebremst, sondern nur das Skelett. Die Weichteile werden sanfter abgebremst, sie werden durch den Aufprall aus ihrer ursprünglichen Ruhelage ausgelenkt und führen um diese gedämpfte Schwingungen aus bzw. "schwabbeln". Diese Eigenschaft wird in einem Modell dadurch berücksichtigt, daß die einzelnen Gliedmaßen je aus einem knöchernen Anteil und eine an den Knochen gekoppelte "Schwabbelmasse" bestehen. Die Modellierung der Schwabbelmasse kann auf verschiedene Arten erfolgen.

Schwabbelnder Starrkörper In den Arbeiten von [Gruber] oder [Widmayer] wurde die Schwabbelmasse als Starrkörper modelliert, die am zugehörigen Knochen linear verschiebbar und drehbar aufgehängt ist. Dazu wurden die Schwabbelmassen über innere Kräfte [3] an die Knochen gekoppelt. Die Kräfte werden getrennt aus einem longitudinalen (Index l) und einen transversalen Anteil (Index t) berechnet und sollen mit der dritten Potenz der Auslenkung $\Delta\boldsymbol{r} = \Delta\boldsymbol{r}_l + \Delta\boldsymbol{r}_t = \boldsymbol{X}_{Schwabbel} - \boldsymbol{X}_{Knochen}$ der Schwabbelmassen wachsen. Die Kräfte sind ferner abhängig von der Querschnittsfläche f der Schwabbelmasse und werden durch einen zur Geschwindigkeit proportionalen Dämpfungsterm gedämpft. Für den Betrag der transversalen Kopplungskraft F_t gilt:

$$F_t = c_t \Delta r_t f + d_t |\Delta r_t| \Delta \dot{r}_t f \qquad (64)$$

[3] Die inneren Kräfte wirken sowohl auf den Knochen als auch auf die Schwabbelmasse mit gleichem Betrag aber unterschiedlichem Vorzeichen

Für den longitudinalen Term gilt dieselbe Gleichung sinngemäß. Ferner wird bei Verdrehung der Schwabbelmasse um den Winkel $\Delta\phi$ mit der Winkelgeschwindigkeit $\Delta\dot{\phi}$ gegenüber dem Knochen das innere Moment[4] M übertragen:

$$M = c_m \Delta\phi + d_m \Delta\dot{\phi} \tag{65}$$

Die Konstanten c_t, c_l, c_m, d_t, d_l, und d_m dienen zur Anpassung der Kopplungskräfte und –momente an entsprechende Experimente. Durch die getrennte Modellierung der Kopplungskräfte als longitudinalen und transversalen Anteil wirken die ins raumfeste System transformierten Kräfte $\boldsymbol{F}$ nicht zwangsläufig längs des Verschiebungsvektors $\Delta\boldsymbol{r}$ und es entsteht deshalb fälschlicherweise ein Moment, das korrigiert wird durch:

$$M_{korr} = (\Delta r)_x F_y - (\Delta r)_y F_x \tag{66}$$

Eine andere Methode zur Modellierung der Kopplungskräfte der Schwabbelmassen ist die, die Schwabbelmasse mithilfe von Federelementen (siehe 3.2.5) am Knochen zu befestigen. Dies ist eine sehr flexible Methode, da man die Aufhängung sehr frei gestalten kann. Auf diese Art können an einen Knochen mehrere voneinander unabhängige Schwabbelmassen angebracht werden, wodurch man sehr detailliert modellieren kann.

Schwabbelnde Punktmassen Der Extremfall mehrerer Schwabbelmassen an einem Knochen ist der, daß die Schwabbelmasse anstatt aus Starrkörpern aus Punktmassen besteht, die untereinander und am Knochen über Federelemente verbunden bzw. befestigt sind. Die dafür notwendigen Bewegungsgleichungen sind sehr einfach, da die Punktmasse keine Orientierung besitzt und folglich durch nur drei Koordinaten beschrieben wird. Da die Ankopplung über Federn funktioniert, deren Kräfte allein vom Abstand der Endpunkte und deren Relativgeschwindigkeit zueinander abhängen, entkoppelt bei der Berechnung einer Stützstelle der Teil des linearen Gleichungssystems, der den Punktmassen zugeordnet ist. Die Berechnung der Punktmassen ist damit parallelisierbar, und der Nachteil einer evtl. höheren Anzahl Körper aufgrund der Punktmassen wird dadurch wettgemacht, daß die zugehörigen Gleichungen einfach nach den Beschleunigungen aufzulösen sind:

$$\ddot{\boldsymbol{x}} = \frac{1}{m}\boldsymbol{F} \tag{67}$$

3.2.2 Kontaktproblem

Ein weiteres, immer wieder auftretendes Problem bei der Simulation ist das Kontaktproblem. Die zu simulierenden Körper haben eine Ausdehnung, aufgrund der es zu Stößen mit anderen Körpern oder mit der Umgebung der Körper kommt. Die Ausdehnung kann im Einzelfall die Simulation entscheidend beeinflussen,

[4]Innere Momente wirken auf beide Körper mit dem gleichen Betrag, jedoch mit verschiedenem Vorzeichen.

wenn man z. B. an einen Aufprall denkt. In den bisher modellierten Gleichungen ist diese Tatsache jedoch nicht berücksichtigt. Die Gleichungen gehen von Starrkörpern oder Punktmassen aus, was zur Folge hat, daß die so dargestellten Körper keine Ausdehnung haben, höchstens eine Orientierung, deren Änderungsgeschwindigkeit mit dem Trägheitsmoment zu beeinflussen ist. Um die Ausdehnung eines Körpers in der Simulation zu berücksichtigen, ist seine Gestalt in die Simulation miteinzubeziehen. An jedem Punkt der Körperoberfläche müssen Kontaktkräfte definiert sein, die normal und tangential zur Oberfläche jeweils als Beziehung zwischen Kraft, Deformation, und Deformationsgeschwindigkeit des Körpers modelliert sind.[5] Die Kräfte treten nur auf, wenn ein Körper einen anderen Körper bzw. einen Gegenstand der Umgebung berührt. Neben der Gestalt der Oberfläche der einzelnen Körper ist daher noch die Gestalt der Umgebung zu berücksichtigen. Der dadurch entstehende Rechenaufwand ist beträchtlich, da alle Punkte aller Oberflächen auf Berührung mit irgendeiner anderen Oberfläche geprüft werden müssen. Nur um herauszufinden, ob sich einzelne Körper berühren, sind daher an jeder Stützstelle des Integrationsintervalls umfangreiche Kontaktprüfungen vorzunehmen.
Während die vollkommene Berücksichtigung der Gestalt aller an der Simulation beteiligten Körper wohl zum grundsätzlichen Verständnis eines biomechanischen Systems und seiner Eigenschaften nicht sehr wesentlich ist, kommt dennoch mindestens das Problem des Bodenkontakts in allen Simulationen vor ([Gruber], [Hospach], [Widmayer]). Während bei [Gruber] nur der Fersenkontakt modelliert wurde, war bei den Sturzsimulationen von [Widmayer] bereits das Auftreffen von Kopf, Gesäß und Knie im Modell realisiert. Hierzu wurde die von [Gruber] empirisch gefundene Beziehung zwischen der normalen Eindringtiefe Δz und der Normalkraftkomponente F_N

$$F_N = A_N \Delta z^{3.5} + B_N \Delta \dot{z} \Delta z^{3.5} \tag{68}$$

sinngemäß auf die Tangentialkomponente F_T und die tangentiale Körperdeformation Δx übertragen:

$$F_T = A_T \Delta x^{3.5} + B_T \Delta \dot{x}^{3.5} \tag{69}$$

Bei [Widmayer] wurde die Kontaktrechnung für einige exponierte Stellen des Modells explizit berücksichtigt, wodurch also bereits die stark vereinfachte Gestalt des Modells mitsimuliert wurde.

3.2.3 Gelenkanschläge

Die im vorigen Kapitel modellierten Systeme basieren auf im Raum orientierten Punktmassen ohne Ausdehnung, die über masselose Hebelarme an den Gelenkpunkten aneinandergekettet sind. In einer Simulation hat dies die Konsequenz,

[5] Und man wird zunächst annehmen, daß der Körper keine Veränderung seiner Gestalt erfährt, da diese sonst ebenfalls mitberechnet werden müßte; die Änderung der Körpergestalt ist in einem solchen Modell natürlich nicht mehr zu berücksichtigen und man wird zu grundsätzlich anderen Methoden wie den Finiten Elementen greifen müssen.

daß anatomisch unmögliche Bewegungen und Stellungen auftreten: Überdrehen eines Gelenks oder Durchschlagen eines Gelenks und die damit verbundene Durchdringung der über das Gelenk verbundenen Körper. Dies soll unterbunden werden, indem winkelabhängige Anschlagsmomente verwendet werden.

Kommt ein Gelenk in den Bereich seines minimalen oder maximalen Winkels, wird im Gelenk ein Begrenzungs- oder Anschlagsmoment aufgebaut, das das Überdrehen oder Durchschlagen des Gelenks verhindern soll. Die Anschlagsmomente dienen zur Modellierung der nicht dem Willen oder Reflexen unterworfenen Gelenkkräfte, die von Sehnen, Knorpel oder der Gelenkbeschaffenheit herrühren. Sie sind also eine feinere Berücksichtigung anatomischer Gegebenheiten. Bleibt der Gelenkwinkel während einer Bewegung zwischen dem minimalen und maximalen Gelenkwinkel, d. h. im Arbeitsbereich, verschwindet das Anschlagsmoment.

Auf den Arbeitsbereich des Gelenks folgen die Anschlagsbereiche mit der Breite $\Delta\varphi$, innerhalb denen das Anschlagsmoment allmählich auf den Endwert beim oberen Anschlagswinkel φ_o und beim unteren Anschlagswinkel φ_u ansteigt.

Im Arbeitsbereich $\varphi < \varphi_o - \Delta\varphi_o$ verschwinden die Momente. Die Breite des Anschlagsbereichs wird durch $\Delta\varphi_o > 0$ und durch $\Delta\varphi_u > 0$ festgelegt. Im Anschlagsbereich und vor allem über den Anschlag hinaus soll das Anschlagsmoment eine streng monoton steigende Funktion sein. Nur so kann man ein Durchschlagen der Gliedmaßen verhindern. Der Übergang in den Anschlagsbereich soll stetig sein und soll am Punkt $\varphi = \varphi_o - \Delta\varphi_o$ wegen der numerischen Integration möglichst oft stetig differenzierbar sein. Diese Forderungen werden von folgender Funktion erfüllt:

$$M(\varphi) = \begin{cases} a_o e^{(\varphi-\varphi_o+\Delta\varphi_o)} - \sum\limits_{n=0}^{N} \frac{a_o}{n!}(\varphi - \varphi_o + \Delta\varphi_o)^n & : \quad \varphi < \varphi_o - \Delta\varphi_o \\ 0 & : \quad \varphi \\ \sum\limits_{n=0}^{N} \frac{a_u}{n!}(\varphi_u + \Delta\varphi_u - \varphi)^n - a_u e^{(\varphi_u+\Delta\varphi_u-\varphi)} & : \quad \varphi > \varphi_u + \Delta\varphi_u \end{cases} \tag{70}$$

Die Funktion aus Glg. (70) ist am Punkt $\varphi = \varphi_0 - \Delta\varphi$ N mal stetig differenzierbar. Die k. Ableitung lautet für $k <= N$:

$$\frac{d^k}{(d\varphi)^k} M = \begin{cases} a_o e^{(\varphi-\varphi_o+\Delta\varphi_o)} - \sum\limits_{n=0}^{N-k} \frac{a_o}{n!}(\varphi - \varphi_o + \Delta\varphi_o)^n & : \quad \varphi < \varphi_o - \Delta\varphi_o \\ 0 & : \quad \varphi \\ \sum\limits_{n=0}^{N-k} \frac{a_u}{n!}(\varphi_u + \Delta\varphi_u - \varphi)^n - a_u e^{(\varphi_u+\Delta\varphi_u-\varphi)} & : \quad \varphi > \varphi_u + \Delta\varphi_u \end{cases} \tag{71}$$

Die Ableitung bleibt also stetig. Die Normierung a_i auf ein maximales Anschlagsmoment M_{max} bei Erreichen des Anschlagswinkels ergibt sich mit Gleichung (70) aus

$$M(\varphi = \varphi_i) = M_{max} \tag{72}$$

zu

$$a_i = \frac{M_{max}}{\exp(\Delta\varphi_i) \pm \sum_{n=0}^{N} \frac{1}{n!}(\Delta\varphi_i)^n} \tag{73}$$

3.2.4 Aktive Systemkomponenten

Mit den bisher angesprochenen Eigenschaften kann man nur passive Systeme simulieren. Die Fähigkeit zur Simulation einzelner Bewegungen oder ganzer Bewegungsabläufe ist alleine durch Gelenke, Schwabbelmassen und Gelenkanschläge nicht erreicht. Es fehlen aktive Komponenten, die die Bewegung und die Steuerung der Glieder des Systems "aus eigener Kraft" durchführen und so die Simulation typischer Abläufe ermöglichen. Erst wenn das simulierte System eine Bewegung durchführen kann, lassen sich zuverlässige Angaben über die in dieser Bewegung simulierten Größen machen und mit experimentell ermittelten Daten vergleichen. Es ist daher von entscheidender Bedeutung für die Simulation, Bewegungen und Bewegungsabläufe im Modell durchführen zu können.
Eine Möglichkeit zur Berücksichtigung aktiver Komponenten ist die Simulation einzelner Muskeln oder Muskelgruppen. Dazu soll eine innere Kraft zwischen zwei Körpern wirken, die durch irgendeinen im Einzelfall zu definierenden Algorithmus bzw. durch geeignete Differentialgleichungen errechnet wird. Eine andere Möglichkeit ist die, die Summe aller an einem Körper angreifenden Kräfte durch Momente an den Gelenken auszudrücken und dann diese anstelle einzelner Muskelkräfte im Hinblick auf die beabsichtigte Bewegung koordiniert zu verändern.

3.2.5 Federelemente

Eine bequeme Methode zur Modellierung von Kräften zwischen zwei Körpern ist der Einsatz von Federelementen. Diese sind an beiden Körpern an je einem Punkt fixiert und entfalten ihre Kraft stets in Richtung des Verbindungsvektors beider Punkte. Mithilfe solcher Federn lassen sich z. B. Schwabbelmassen in einfacher Weise an die knöchernen Teile eines Modells ankoppeln. Sieht man vor, daß solche Federn auch an einem Ende irgendwo im Raum fixiert werden, lassen sich elastische Halterungen und Aufhängungen realisieren. Die Federkraft soll sich wie folgt berechnen:

$$F = a(l - l_0)^b + cv^d(l - l_0) \tag{74}$$

Die Federkraft F wirkt dabei stets in Richtung des Verbindungsvektors beider Endpunkte, wobei l_0 die Ruhelänge der Feder ist, l ist die Federlänge und $v = \dot{l}$ ist die Geschwindigkeit, mit der sich die Federlänge verändert. Die Parameter a, b, c und d dienen zur Einstellung der übrigen Federeigenschaften. Der Term $(l - l_0)$ am Dämpfungsglied hat die Funktion, eine Dämpfung in der Ruhelage zu verhindern.

3.3 Zusammenfassung der Anforderungen

Aus den vorhergehenden Abschnitten dieses Kapitels ergeben sich einige Anforderungen an ein Programmsystem zur Simulation von biomechanischen Systemen, die an dieser Stelle zusammengefaßt werden. An dieser Stelle nicht mehr erwähnt sind die ganz zu Anfang dieses Kapitels aufgeführten allgemeinen Forderungen, da diese mehr Richtlinien zur Implementierung darstellen als konkrete Eigenschaften des Systems beschreiben. Die Implementierung folgender Eigenschaften wären sinnvoll:

1. Parametrisierte Beschreibung der Körpergestalt in Form von kinematischen Ketten und Bäumen aus Starrkörpern.
2. Parametrisierte, abschaltbare Gelenkanschläge zum Festlegen der Anschlagwinkel und der Anschlagmomente.
3. Parametrisierte Beschreibung der Starrkörpereigenschaften wie Masse, Trägheitsmoment, Hebelgeometrie und Oberflächenverlauf sowie Oberflächenbeschaffenheit im Hinblick auf die Kontaktkräfte.
4. Parametrisiertes Einhängen von Federelementen an beliebiger Stelle der kinematischen Ketten.
5. Einbau von Kontaktkräften, insbesondere Bodenkontakt.
6. Parametrisierte Eingabe der Anfangsbedingungen aller Körper in reduzierten Koordinaten. Die vollständigen Koordinaten errechnet das System aus der Zusammensetzung der kinematischen Ketten.
7. Implementierung einer Schnittstelle für individuellen Code zur Datenausgabe, für innere und äußere Kräfte und Momente. Die Schnittstelle muß über eine beliebige Anzahl frei definierbarer Parameter verfügen. Das System muß eine beliebige Anzahl eigener Größen mitintegrieren können.
8. Berücksichtigung von Gravitation in Stärke und Richtung.
9. Optionale Ausgabe der gängigen Daten einer Simulation wie vollständige Koordinaten, Drehimpuls, Schwerpunktskoordinaten, Schwerpunktsgeschwindigkeiten, Gelenkwinkel und -geschwindigkeiten.
10. Zur Kontrolle wird der Drehimpuls sowohl über die Momente aufintegriert als auch direkt aus den Winkelgeschwindigkeiten berechnet. Beide Werte müssen übereinstimmen.

Die hier aufgezählten Punkte stellen Maximalforderungen dar, deren Implementierung bereits im ersten Entwurf nicht sinnvoll gewesen wären, die jedoch bei der Verbesserung des Programmsystems berücksichtigt und ergänzt werden sollten. Im nächsten Kapitel wird die Realisierung einer Reihe dieser Forderungen vorgestellt.

4 Realisierung

Dieses Kapitel beschäftigt sich mit der Umsetzung des im letzten Kapitel beschriebenen Entwurfs eines Programmsystems zur Durchführung biomechanischer Simulationen. Zunächst werden in einer Übersicht alle das System ausmachenden Komponenten vorgestellt und deren Zusammenwirken aufgezeigt; im Anschluß daran wird die Implementierung in der Programmiersprache C unter dem Betriebssystem UNIX erläutert.

4.1 Übersicht

4.1.1 Namensgebung

Um nicht immer umständlich vom Programmsystem zur Simulation biomechanischer Mehrkörpersysteme schreiben zu müssen, wird dieses mit *simsys* bezeichnet.

4.1.2 Aufgliederung in Komponenten

Das System besteht aus folgenden Komponenten:

- Der Bewegungsgleichungsgenerator *bgg* erfüllt die Forderungen aus dem Abschnitt 3.3 und generiert nicht nur die Bewegungsgleichungen in C, sondern setzt gleichzeitig individuellen Code in "leere Funktionen" ein, um den generierten C–Code nicht nachträglich von Hand bearbeiten zu müssen. Die Ausgabe des Bewegungsgleichungsgenerators ist eine komplette Quelltextdatei in der Programmiersprache C.

- Die Programmbibliothek *libsimsys.a* beinhaltet die universelle Integrationsroutine, die jeden vom Bewegungsgleichungsgenerator erzeugten Code integrieren kann. In der Bibliothek sind ferner alle Funktionen zum Lesen der Parameterdateien, die Funktionen zur Berechnung der Federkräfte und Bodenreaktionskräfte sowie Gelenkanschläge enthalten. Weitere Bestandteile sind ein universelles Ausgabemodul, das die Ausgabe der wichtigsten Daten in binärer Form erzeugt und der Integrierer *de_()*, der für die Verwendung in diesem Zusammenhang auf die Integration von 100 anstatt 20 Gleichungen abgeändert wurde. Die Programmbibliothek enthält auch die Funktion *main()*, mit der in jedem C–Programm die Ausführung begonnen wird.

- Das eigentliche Simulationsprogramm *simsys* — der Name steht stellvertretend für jeden beliebigen unter UNIX zulässigen Programmnamen — wird dadurch erzeugt, daß die vom Bewegungsgleichungsgenerator erstellte Quelltextdatei kompiliert und mit der Programmbibliothek zusammengelinkt wird. Das so erstellte Programm führt die Simulation durch und erzeugt die Ausgabedaten in binärer Form. Die Ausgabedaten werden nicht in ASCII erzeugt, weil binäre Daten weniger Platz auf der Festplatte benötigen und die Rechenzeit des Konvertierens so nicht während der Simulation verbraucht wird.

- Das Umsetzprogramm *blowup* dient dazu, die in binärer Form vorliegenden Ausgabedaten in ASCII umzusetzen. Erst nach Umsetzung der Daten mithilfe dieses Programms sind diese mit einem konventionellen ASCII Editor zu bearbeiten.

- Das Anzeigeprogramm *simxwin* dient dazu, den in den erzeugten Binärdaten steckenden Bewegungsablauf unter X–Windows (X11.4) anzuzeigen. Die Darstellung der Körper auf dem Bildschirm erfolgt skizzenhaft durch Strichzeichnungen der Gelenkhebel.

- Eine Vielzahl von Dateien, aus denen der Bewegungsgleichungsgenerator, das Simulationsprogramm und das Anzeigeprogramm alle Informationen zum simulierten System entnehmen. Hierin ist sowohl der individuelle Code untergebracht als auch sämtliche während der Simulation benötigten Parameter. Es gibt drei unterschiedliche Typen von Dateien:

  – Parameterdateien. Von dieser Art gibt es im Augenblick nur eine. Diese Datei enthält die Dateinamen aller Beschreibungsdateien und die während des Ablaufs einer Simulation benötigten Parameter wie Integrationszeitraum, Integrationsschrittweite, usw.

  – Beschreibungsdateien. In diesen wird die Beschaffenheit und der Aufbau des zu simulierenden Systems beschrieben. Die Dateien beschreiben, welche Starrkörper über Gelenke miteinander verbunden sind, welche Massen und Trägheitsmomente die Körper haben usw.

  – Quelltextdateien in C. Von dieser Art gibt es im Augenblick nur eine. Diese muß den Code enthalten, der die Integration der für eine Simulation spezifischen Variablen veranlaßt und die parametrisierte Ausgabe durchführt.

- UNIX Utilities und die zugehörigen Dateien wie Makefile und SCCS[6]–Dateien unterstützen die Arbeit zur Entwicklung und Verbesserung von Modellen. Diese sind nicht unmittelbarer Bestandteil von *simsys*, sie erleichtern jedoch die Arbeit mit *simsys* erheblich.

Um mit dem System an der Verfeinerung eines Modells arbeiten zu können oder ein Modell entwerfen zu können, ist es wichtig, verschiedene Ideen simultan umzusetzen und diese miteinander zu vergleichen. Dazu muß man während dieser Arbeitsphase quasi parallel an verschiedenen Varianten desselben Modells arbeiten und die Ergebnisse dokumentieren. Dies wird von *simsys* dadurch unterstützt, daß sich alle Programme und Daten in den für alle simulierten Modelle identischen allgemeinen Teil und in den modellspezifischen Teil unterteilen. Der allgemeine Teil besteht aus den Programmen *bgg*, *blowup* und *simxwin*, sowie aus der Programmbibliothek *libsimsys.a*. Der modellspezifische Teil umfaßt das eigentliche Simulationsprogramm *simsys* und die Parameter–, Beschreibungs– und Quelltextdateien. Man kann ein bestimmtes Modell also variieren, indem man

[6] Source Code Controlling System

ausgehend von der identischen Kopie des modellspezifischen Teils Veränderungen anbringt und mit diesen experimentiert. Das gleiche Vorgehen empfiehlt sich zum Entwurf eines neuen Modells: Man beginnt ebenfalls mit der Kopie eines bereits fertigen Modells und verändert den modellspezifischen Teil. Das Zusammenwirken der einzelnen Komponenten und die Unterteilung in modellspezifischen und allgemeinen Teil ist in Abb. 4 dargestellt.

4.1.3 Verzeichnisstruktur

Die gleichzeitige Arbeit an verschiedenen Modellen oder verschiedenen Varianten desselben Modells erfordert eine sinnvolle und übersichtliche Verzeichnisstruktur. Diese ist in Abb. 5 dargestellt und erläutert. Die Verzeichniswurzel des dort dargestellten Baums kann jeden beliebigen Namen tragen, sofern die entsprechenden Makefiles und Pfade angepaßt werden. Eine sinnvolle Unterteilung der einzelnen Modellverzeichnisse in Unterverzeichnisse ist empfehlenswert. Da Notwendigkeit und Art der Unterteilung jedoch von der Komplexität eines Modells abhängen, wird an dieser Stelle keine Unterteilung vorgeschlagen.

4.1.4 Verwendung von *make* und *sccs*

Zur Erstellung des Simulationsprogramms und zur Durchführung der Simulationen sind zwei UNIX Dienstprogramme sehr hilfreich: *make* und *sccs*. An dieser Stelle soll kein Auszug aus dem UNIX Handbuch wiedergegeben werden, sondern die Funktion beider Programme insofern erklärt werden, als dies zur Begründung ihrer Verwendung ausreicht.
Das UNIX Dienstprogramm *make* hat den Sinn, die zur Erzeugung von Dateien (sog. *targets*) notwendigen Kommandos und Programme automatisiert auszuführen und damit die Dateien zu erzeugen, ohne zur Erledigung dieser Aufgabe eigene Programme mit umfangreichen Prüfungen zu schreiben. Die Philosophie von *make* lautet: Jedes *target* muß dann neu erzeugt werden, wenn die Dateien, die Einfluß auf die Gestalt des *targets* haben (die sog. *dependencies*), jüngeren Datums sind als das *target*. Nach welchen Regeln das *target* erstellt werden muß, und welche *dependencies* das *target* hat, wird im *Makefile* mit einer speziellen Syntax definiert. Da jedes *dependency* auch als *target* mit eigenen *dependencies* definiert werden kann, können somit umfangreiche Befehlsfolgen durch alleinige Eingabe des Kommandos *make* automatisch ablaufen. Das Dienstprogramm *make* wird dazu verwendet, das Simulationsprogramm nach Änderungen an den Dateien ggf. neu zu erstellen und automatisch Simulationen durchführen zu lassen.
Das andere Dienstprogramm *sccs* dient dazu, erfolgreich durchgeführte Simulationen zu konservieren, um auf diese auch nach einer Weiterentwicklung der Simulation jederzeit wieder zurückgreifen zu können.

4.1.5 Funktion von *simsys*

An dieser Stelle soll kurz auf die Funktion des eigentlichen Simulationsprogrammes *simsys* eingegangen werden. Zu Beginn liest das Programm die notwendigen Parameter und initialisiert seine programminternen Variablen. Anschließend wird

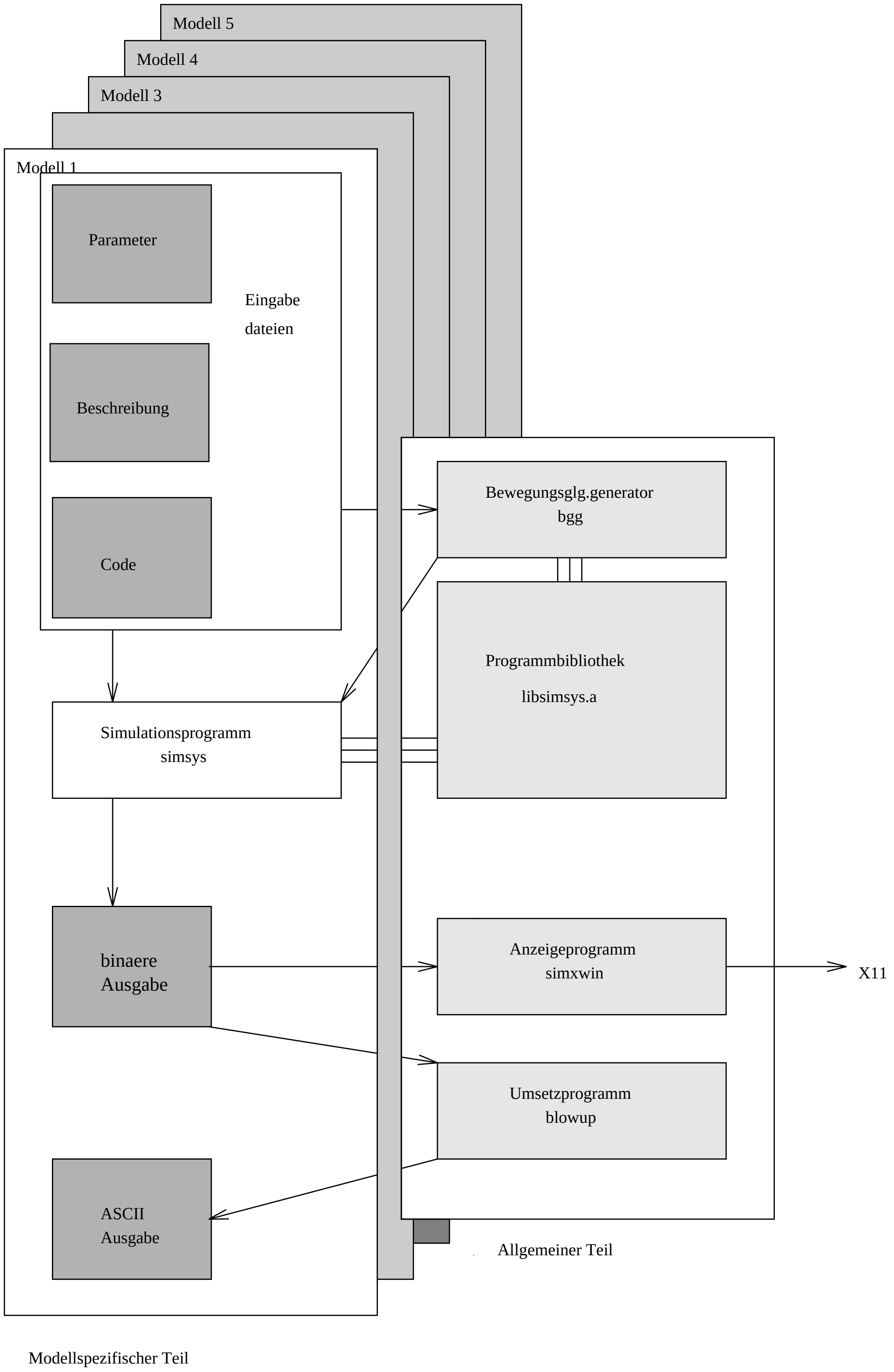

Abbildung 4: Zusammenwirken der einzelnen Komponenten des Programmsystems

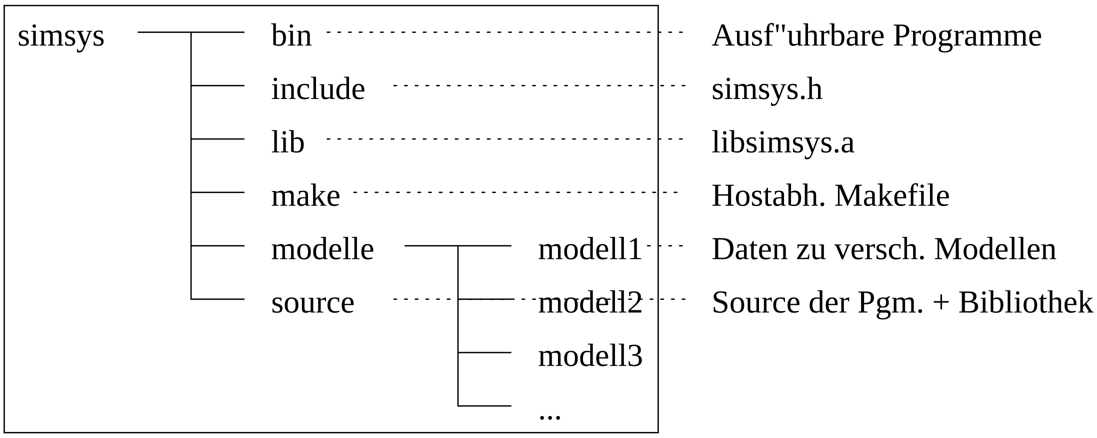

Abbildung 5: Verzeichnisstruktur von *simsys*

die Datenausgabe initialisiert. Danach wird die Integration *de_()* mit der vorgegebenen Schrittweite gestartet. Der Integrator *de_()* ruft zur Berechnung jeder Stützstelle die Funktion auf, die das Gleichungssystem der Beschleunigungen löst. Nach Integration des ersten Zeitintervalls erfolgt die Datenausgabe. Anschließend wird *de_()* zur Integration des nächst folgenden Zeitintervalls aufgerufen und die letzten Punkte wiederholen sich solange, bis das Ende des vorgegebenen Integrationszeitraums erreicht ist. Danach werden die Ausgabedateien geschlossen und das Programm wird ordnungsgemäß beendet.
Das Programm arbeitet in zwei Dimensionen und benutzt zur Beschreibung die Koordinaten X und Y, wobei diese in einem Rechtssystem definiert sind. Die Orientierung des Systems geht im Zweidimensionalen nur bei Winkelangaben in Form des Vorzeichens ein. Sieht man die Ebene als Sonderfall des Raumes, wird dies durch die Vorzeichen aller Kreuzprodukte sofort ersichtlich. In allen Quelltextdateien und allen Parametern wird die Koordinate Y jedoch fälschlicherweise mit Z bezeichnet. Dies ist eine fehlerhafte Nomenklatur und bedarf der Überarbeitung.

4.2 Erstellung einer Simulation

4.2.1 Allgemeines Vorgehen

Der erste Schritt bei der Erstellung einer Simulation ist die Anfertigung eines Modells in Form einer Zeichnung wie in Abb. 6 anhand eines Beispiels demonstriert.
In diesem Entwurf werden alle Körper beginnend bei 1 in aufsteigender Folge lückenlos durchnumeriert. Alle Gelenke eines Körpers werden ebenfalls beginnend bei 1 in aufsteigender Folge in der Art lückenlos durchnumeriert, daß die Gelenknumerierung die Reihenfolge der aufsteigend nach ihrer Nummer sortierten Nachbarkörper repräsentiert. Anschließend wird die Zahl, Richtung und Numerierung aller zusätzlich zu den Gelenkhebeln benötigten Hebel jedes Körpers bestimmt. Kräfte können auf die Körper nur über Hebel einwirken. Zusätzliche Hebel sind damit zusätzliche Kraftangriffspunkte. Die Numerierung der Hebel

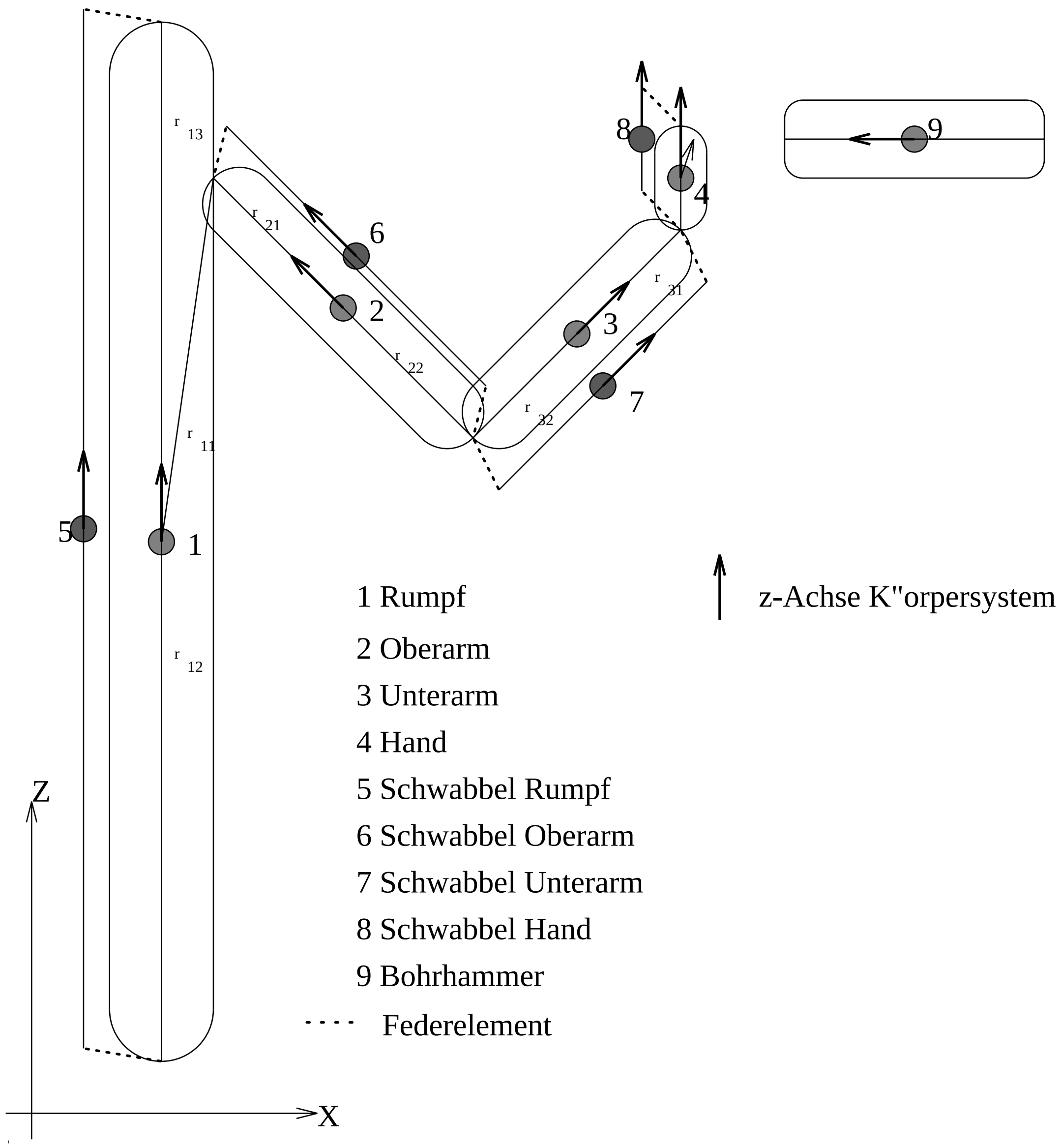

Abbildung 6: Entwurf eines einfachen Modells als Beispiel

erfolgt nach der folgenden Systematik: Jeder Hebel ist durch zwei Indizes gekennzeichnet. Der erste Index ist identisch mit der Nummer des Körpers, zu dem der Hebel gehört. Der zweite Index ist identisch mit der Nummer des Gelenks, zu dem der Hebel innerhalb des Körpers führt. Alle Hebel eines Körpers, die nicht zu einem Gelenk führen, erhalten als zweiten Index eine lückenlos aufsteigende Nummer beginnend bei der auf das Gelenk mit der höchsten Nummer folgenden Nummer.
Anschließend wird der so erstellte Entwurf umgesetzt, indem folgende Daten wie im Abschnitt 4.3 dargelegt eingegeben werden:

- Gelenkverbindungen aller Körper.
- Hebel aller Körper.
- Gelenkanschläge und –anschlagsmomente.
- Massen und Trägheitsmomente.
- Verwendete Federelemente.
- Dateinamen der oben stehenden Daten in der Parameterdatei.

Hat man alle diese Daten eingegeben, kann durch Aufruf des UNIX-Programms *make* — vorausgesetzt, das Makefile ist bereits richtig gepflegt worden — das Simulationsprogramm erstellt werden. Anschließend kann mit der eigentlichen Simulation begonnen werden, nachdem man folgende Daten wie im Abschnitt 4.3 angegeben erzeugt hat:

- Anfangswerte, d. h. Koordinaten und Geschwindigkeiten aller Körper.
- Gelenkwinkel und –geschwindigkeiten aller Körper. Die daraus resultierenden und zur Rechnung benötigten vollständigen Koordinaten errechnet das System zu Beginn jeder Simulation.
- Dateinamen der oben stehenden Daten in der Parameterdatei.
- In der Parameterdatei müssen ferner noch Integrationsparameter und gewünschte Ausgabedateien angegeben werden.

4.2.2 Aufgabenstellung des Beispiels

Als Beispiel zur Verwendung von *simsys* dient die Simulation eines Menschen, der einen Bohrhammer bedient. Der Mensch wird modelliert durch Rumpf, Oberarm, Unterarm und Hand als Skelett mit angekoppelter Schwabbelmasse. Der Bohrhammer liegt an der Hand an und koppelt an die Hand über eine Kraft–Deformationsbeziehung ähnlich dem Bodenkontakt. Von Interesse ist die Belastung der Gelenke infolge der periodischen Bohrerbewegung. Um die Problematik so einfach wie möglich zu machen, wird anstelle der Schwerkraft, des Bodenkontakts und einem daraus resultierenden Moment auf den Rumpf eine äußere Kraft und ein äußeres Moment bei Schwerelosigkeit direkt auf den Rumpf gegeben, damit die im zeitlichen Mittel vom Bohrer auf den Rumpf übertragene Kraft und das zugehörige Moment kompensiert wird. Der Rumpf soll im zeitlichen Mittel in Ruhe bleiben.

Das Modell muß daher folgende Eigenschaften besitzen:

- Das Modell wird ohne Schwerkraft simuliert, da die Schwerkraft bei einem in der Wand steckenden Bohrhammer im Vergleich zu den durch den Andruck und den Rückschlag auftretenden Kräften zu vernachlässigen ist.
- Auf den Rumpf wirkt ein äußeres Moment und eine äußere Kraft, die unter realen Bedingungen durch die Füße und den Boden auf den Rumpf übertragen werden.
- Zwischen Hand und Bohrhammer muß eine der Bodenkraft ähnliche Kraft–Deformationsbeziehung (evtl. auch geschwindigkeitsabhängig) modelliert werden. Die Kraft ist durch Messungen entsprechend zu bestätigen.
- Der menschliche Arm ist derart zu steuern, daß er den Bohrhammer im zeitlichen Mittel leicht nach vorne bewegt (resp. bohrt), jedoch nicht die beim Rückschlag durch die Hammerbewegung des Bohrers auftretenden Auslenkungsspitzen verhindert. Die Steuerung darf also nicht mit der Geschwindigkeit folgen, mit der die Spitzen auftreten.

Während die ersten Punkte einfach zu lösen sind, ist die Simulation einer realistischen Steuerung des Armes das zentrale Problem an dieser Modellierung und wird daher nach Erstellen des Modells als erstes angegangen. Während zur Lösung des Problems lediglich ein Freiheitsgrad notwendig wäre, sind hier drei Freiheitsgrade redundant zu gebrauchen.

In den folgenden Abschnitten wird jedoch anhand dieser Modellierung zunächst nur die Funktion von *simsys* erläutert und erst im nächsten Kapitel wird das Problem der Steuerung wieder aufgegriffen.

4.3 Verwendung von *simsys*

In diesem Abschnitt wird anhand des Beispiels beschrieben, wie man mit dem Programmpaket *simsys* eine Simulation eines Problems erstellt. Die Bedienung der Programme *simsys*, *bgg*, *blowup* und *simxwin* wird im Anhang im Stil der UNIX Manual Pages beschrieben. Hier werden die Formate der Dateien beschrieben, die zur Erstellung einer Simulation mit *simsys* erforderlich sind. Ausgehend vom Beispiel aus Abschnitt 4.2 wird die Bedeutung eines jeden Eintrags in den zur Erstellung der zugehörigen Simulation benötigten Dateien erläutert.

4.3.1 Die Parameterdatei

Die Parameterdatei wird von den Programmen *simsys*, *simxwin* und *bgg* gelesen, um bestimmte Programmvariablen ohne Veränderung der Programme selbst mit Werten zu belegen. Die Parameterdatei ist eine normale ASCII–Datei, die aus Kommentarzeilen und Zeilen mit Schlüsselworten und zugehörigen Werten besteht. Eine Kommentarzeile beginnt in der ersten Spalte stets mit dem Zeichen '#' und bewirkt, daß die restlichen Zeichen der Zeile vom Programm ignoriert werden. Bei allen übrigen Zeilen wird das erste Wort der Zeile als Schlüsselwort angesehen, hinter dem ein oder mehrere Werte stehen. An diese Werte gelangt das Programm dadurch, daß es die ersten Worte aller Zeilen der Parameterdatei (außer Kommentarzeilen) mit einem im Programm definierten Schlüsselwort vergleicht und im Fall der Übereinstimmung die dahinterstehenden Werte (sie werden in einer bestimmten Form erwartet und ggf. konvertiert) programminternen Variablen zuweist. Wird das gesuchte Schlüsselwort nicht gefunden, werden die entsprechenden programminternen Variablen mit im Programm fest kodierten, sog. *default*–Werten belegt. Während für die Programme die Abfolge der einzelnen Zeilen keine Rolle spielt, empfiehlt sich zur besseren Übersicht eine Sortierung der Schlüsselworte wie sie unten vorgeschlagen ist.
Zu dem Beispiel aus Abschnitt 4.2 gehört folgende Parameterdatei:

```
########### Eingabedateien ##################
########### Erstellen und Ablauf von simsys #
#---------- Gelenkdefinitionen --
EingGelenke   gelenke.dat
#---------- Innere Kraefte --
EingMuskeln   muskeln.dat
#---------- Gelenkmomente --
EingMomente   momente.dat
#---------- Aeussere Kraefte --
EingKraefte   kraefte.dat
#---------- Aeussere Momente --
EingAussMom   aussmom.dat
#---------- Parameter Programmierschnittstelle --
EingParamet   param.dat
########### Nur Ablauf von simsys ###########
#---------- Massen und Traegheitsmomente --
EingMasTrg    mastrg.dat
#---------- Hebelarme --
EingHebel     hebel.dat
#---------- Federelemente --
EingFedern    federn.dat
#---------- Anfangswerte --
EingAnfWerte  anfwerte.dat
EingAnfWinkel anfwink.dat
############### Integrationsparameter ############
IntMaxErrTol 1
IntMaxErrSteps 1
IntVarZahl 3
#---------- Integrationsdauer in s ----
IntDauer 0.05
#---------- Integrationsschrittweite in ms -----
IntSchrWeite 0.7
#---------- Absolute Integrationsgenauigkeit ---
IntGenauAbs 1.0e-6
#---------- Relative Integrationsgenauigkeit ---
IntGenauRel 1.0e-6
############### Berechnungsparameter ###############
#---------- Gravitation in m/(s*s) ----------
GravitatX 0.0
```

```
GravitatZ 0.0
#---------- Gelenkanschlaege mitberechnen ---
GelAnschlOrdnung 0
#---------- Hoehe des Bodens ----------------
BodenHoeheZ 0.0
#---------- Eigene Parameter ----------------
BohrKraft 300.0
BohrFrequenz 5.0
VortriebKraft 0.0
HandgelP 2.0
HandgelI 400.0
HandgelD 20.0
EllengelP 2.0
EllengelI 400.0
EllengelD 20.0
SchulgelP 2.0
SchulgelI 400.0
SchulgelD 20.0
HandKraft 0.0
RumpfKraft 0.0
RumpfMoment 1000.0
############### Ausgabeparameter ###################
AusgIntervall 0.001
#-------------- Ausgabedateien ---------------------
#-------------- vollst. Dateiname (d. h. mit Pfad) der Ausgabedateien
AusgZwKraefte
AusgGelAusdehn
AusgHblKoord
AusgKoordin  out/Koordin.dat
AusgEnergie
AusgDrehimp
AusgSchwerp
AusgKraefte
AusgMomente
AusgGelWinkel  out/GelWinkel.dat
AusgSimGelWin  out/GelWin.txt
AusgSimGelMom  out/GelMom.txt
############### Anzeigeparameter ###################
#-------------- Sleep in ms zwischen zwei Bildern --
AnzSleep 2
```

Die Datei gliedert sich in die Blöcke 'Eingabedateien', 'Integrationsparameter', 'Berechnungsparameter', 'Ausgabeparameter' und 'Anzeigeparameter'. Die Schlüsselwörter werden im Anhang bei der Beschreibung eines jeden Programms nochmals genau erläutert. An dieser Stelle soll nur ein Überblick verschafft werden.

Eingabedateien Hinter jedem der oben aufgeführten Schlüsselwörter werden die Dateinamen der entsprechenden Dateien mit Verzeichnispfad erwartet. Die ersten Dateien enthalten die sowohl zur Erstellung von *simsys* als auch zum Ablauf der Simualation notwendigen Daten. Die anderen Dateien enthalten nur Daten, die während des Ablaufs der Simulation gebraucht werden.

Integrationsparameter Hier kann Einfluß auf die numerische Integration genommen werden. Es kann die Integrationsdauer, die –schrittweite, der absolute und relative numerische Fehler und die Art der Fehlerbehandlung eingestellt werden. Ferner wird hierin die Anzahl eigener Integrationsvariablen festgelegt. Die Integrationsparameter werden nur vom Simulationsprogramm verwendet.

Berechnungsparameter Hierin wird die Stärke und die Richtung der Gravitation eingetragen. Man beachte, daß die mit Z bezeichnete Richtung die Richtung Y ist. Ferner wird hier eingestellt, ob und wie die Gelenkanschlagsmomente in der Simulation berücksichtigt werden und auf welcher Höhe der Boden liegt, der stets parallel zur Richtung X verläuft. Daran anschließend kommen die eigenen Parameter, deren Verwendung in der Beschreibung der Programmierschnittstelle (siehe 4.3.9) beschrieben wird.

Ausgabeparameter Hierin werden die Dateinamen der Ausgabedateien angegeben. Wird hinter einem Schlüsselwort kein Dateiname angegeben, entfällt die entsprechende Ausgabe. Das genaue Format der Ausgabedateien ist im Anhang aufgeführt.

Anzeigeparameter Die hier eingebenen Parameter haben auf die Simulation keinen Einfluß, sie werden nur vom Anzeigeprogramm verwendet.

4.3.2 Format der Beschreibungsdateien

Eine Beschreibungsdatei besteht aus einem oder mehreren Blöcken. Innerhalb eines Blocks ist eine für den jeweiligen Block spezifische Syntax vorgeschrieben, außerhalb des Blocks können Zeilen beliebigen Inhalts stehen. Ein Block beginnt stets mit einer Zeile, in der ab Spalte 1 das Schlüsselwort steht. Das Schlüsselwort hat stets das Format `@@NAME@`, worin `NAME` stellvertretend für den Blocknamen steht. Ein Block endet stets mit einer Zeile, in der ab Spalte 1 die Zeichenfolge `@@@` steht. Eine Kommentarzeile innerhalb eines Blocks beginnt mit dem Zeichen `#` in der ersten Spalte und kann an jeder Stelle innerhalb des Blocks eingefügt werden.

Mit dieser Vereinbarung lassen sich je nach Bedarf und Übersichtlichkeit die Blöcke in Dateien zusammenfassen oder alternativ einzeln in je eine Datei schreiben. Der Dateiname der Datei, in der dann der entsprechende Block steht, muß in der Parameterdatei angegeben werden. In den folgenden Abschnitten wird daher nur noch die Syntax der einzelnen Blöcke erläutert.

4.3.3 Gelenke und Gelenkwinkel

Der Blockanfang des Blocks, in dem die Gelenke definiert werden, beginnt mit dem Schlüsselwort `@@GELENKE@`. Die zugehörigen Gelenkwinkeldefinitionen sind in derselben Datei untergebracht und der entsprechende Block beginnt mit dem Schlüsselwort `@@WINKEL@`. Der Gelenkblock hat im Beispiel aus Anschnitt 4.2 die folgende Gestalt:

```
@@GELENKE@
# dieser Block besteht aus einer symmetrischen Matrix.
# Jede Spalte und jede Zeile repraesentieren einen Starrkoerper,
# jedes von Null verschiedene Matrixelement ein Gelenk zwischen den
```

```
# beiden Starrkoerpern (Zeile, Spalte). Fuer Gelenkverbindungen werden
# nur die Werte rechts der Hauptdiagonale eingetragen. Zulaessige
# Werte hierfuer sind 0 oder 1.
#
# In die Hauptdiagonale wird die Anzahl der zusaetzlich benoetigten
# Hebelarme je Koerper eingetragen.
#
# Soll ein Koerper nicht durch ein Gelenk an einen anderen Koerper
# angebunden werden, so ist fuer ihn mindestens ein Hebelarm
# einzutragen, da der Koerper sonst nicht beruecksichtigt wird.
#
# Alle Koerper des Modells sind lueckenlos beginnend bei 1 in
# aufsteigender Reihenfolge durchzunumerieren. Andernfalls wird
# u. U. fehlerhaft simuliert.
#
#                    11  13  15  17  19  21  23  25  27  29
# 1 2 3 4 5 6 7 8 9 10  12  14  16  18  20  22  24  26  28  30
#
  4 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    1
    0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    2
      0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    3
        3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    4
          2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    5
            2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    6
              2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    7
                2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    8
                  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    9
                    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   10
                      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   11
                        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   12
                          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   13
                            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   14
                              0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   15
                                0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   16
                                  0 0 0 0 0 0 0 0 0 0 0 0 0 0   17
                                    0 0 0 0 0 0 0 0 0 0 0 0 0   18
                                      0 0 0 0 0 0 0 0 0 0 0 0   19
                                        0 0 0 0 0 0 0 0 0 0 0   20
                                          0 0 0 0 0 0 0 0 0 0   21
                                            0 0 0 0 0 0 0 0 0   22
                                              0 0 0 0 0 0 0 0   23
                                                0 0 0 0 0 0 0   24
                                                  0 0 0 0 0 0   25
                                                    0 0 0 0 0   26
                                                      0 0 0 0   27
                                                        0 0 0   28
                                                          0 0   29
                                                            0   30
@@@

@@WINKEL@
# Gelenkwinkeldefinitionen ueber Gelenkvorzeichen sowie Anschlagswinkel und
# Anschlagsmomente.
# Ein positives Gelenkvorzeichen steht fuer den
# Winkel, der vom Koerper mit der kleineren Ordnungszahl im mathema-
# tisch positiven Drehsinn (gegen Uhrzeiger) zum Koerper mit der
# groesseren Ordnungszahl fuehrt. Ein negatives Vorzeichen steht fuer
# Winkel, die vom Koerper mit groesserer Ordnungszahl zum Koerper
# kleinerer Ordnungszahl fuehren. Ein Gelenkwinkel, der hier nicht angegeben
# ist, hat automatisch positives Vorzeichen.
# Anschlagsmomentart:
# 0
# Syntax:
# (GelenkNr:Gelenkvorzeichen); [(UAn, UAnB: UMom); (OAn, OAnB: OMom);]
# UAn  - Unterer Anschlagswinkel
# UAnB - Breite des unteren Anschlagsbereichs
# UMom - Unteres Anschlagsmoment
#
(1:1);  (-13, 3: 50); (175, 5: 50);
(2:-1); (4, 2: 45); (179, 4: 45);
```

```
(3:1); (90, 3: 40); (270, 5: 40);
@@@
```

Die Syntax ist in den Kommentarzeilen zu Beginn der Blöcke erklärt. In diesem Beispiel wird ein Modell mit 9 Körpern erzeugt, da im Gelenkblock in 9 Zeilen von Null verschiedene Einträge gemacht wurden. Körper 1 hat ein Gelenk, das ihn mit Körper 2 verbindet. Zusätzlich besitzt Körper 1 noch 4 Hebelarme, die nicht zu Gelenken führen. Die Hebel dienen als Kraftangriffspunkte bei der Simulation. Die Bodenkraft wird automatisch an jedem Hebel berücksichtigt. Andere Kräfte müssen explizit kodiert oder im Fall der Feder nur definiert werden, für alle Kräfte wird damit verbundene Drehmoment automatisch berechnet und berücksichtigt.

Körper 2 hat zwei Gelenke, eines, das ihn mit Körper 2 verbindet, und eines, das ihn mit Körper 3 verbindet. Gelenke eines Körpers, die zu Körpern mit niedrigerer Ordnungszahl führen, wären eigentlich links der Hauptdiagonale zu suchen; aus Symmetriegründen ist dies äquivalent mit den Eintragungen in der Spalte des betreffenden Körpers. Körper 2 besitzt keine zusätzlichen Hebel außer den Gelenkhebeln.

Ähnliches gilt für die Körper 3 und 4. Die Körper 5 bis 9 besitzen keine Gelenke und müssen deshalb mindestens einen Hebel bekommen, damit sie überhaupt in der Simulation berücksichtigt werden. Die Körper sind teilweise als Schwabbelmassen gedacht, die über Federkräfte angekoppelt werden, und benötigen daher keine Gelenke.

Die Syntax der Definition der 3 benötigten Gelenkwinkel und der Gelenkanschläge ist im Quelltext erläutert.

4.3.4 Der generierte Quelltext

Der generierte Quelltext befindet sich im Anhang. Eine kurze Übersicht und Strukturierung bietet Abb. 7. Die in den Kästen stehenden Programmteile werden aus den in den folgenden Abschnitten besprochenen Dateien durch den Bewegungsgleichungsgenerator extrahiert, einige Variablen modifiziert und an die entsprechenden Stellen im Quelltext eingefügt. Aus der Abbildung soll ersichtlich werden, an welchen Stellen der Stützpunktberechnung man Einfluß auf die Simulation nehmen kann.
Der Code, der aus den in den folgenden 4 Abschnitten besprochenen Blöcken stammt, kann alle Variablen verwenden, die im generierten Quelltextmodul als modulglobale Variablen definiert sind, siehe dazu die entsprechende Anmerkung in Abb. 7.
In den nächsten 4 Abschnitten wird beschrieben, in welcher Form man den Teil des Quelltextes, der durch den Bewegungsgleichungsgenerator an die dafür vorgesehene Stelle im erzeugten Quelltext geschrieben wird, kodieren muß. Um den Code so effektiv wie möglich kodieren zu können werden in diesem Abschnitt

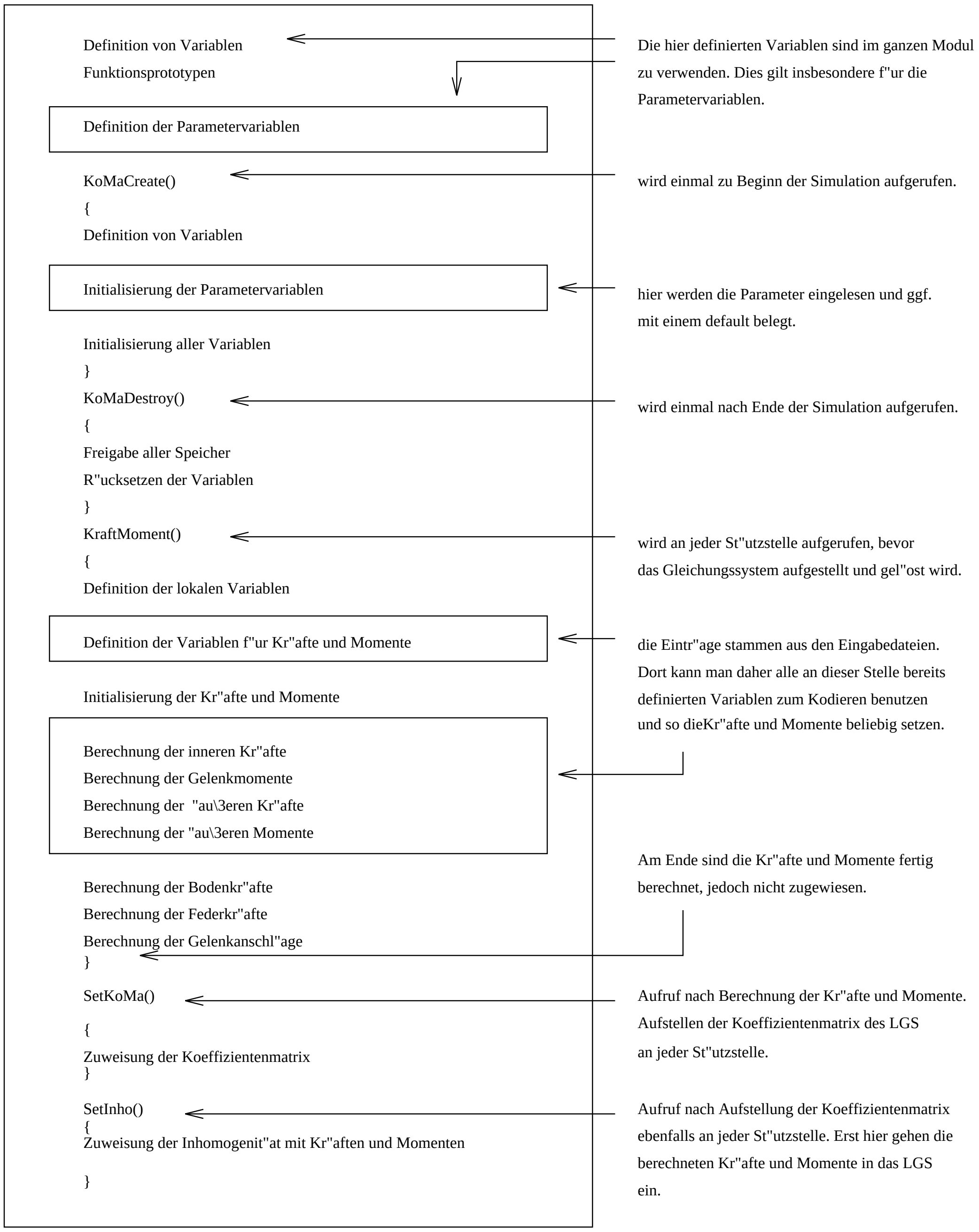

Abbildung 7: Struktur des generierten Quelltextes

die zur Verfügung stehenden Variablen erläutert. Die Definitionen der Variablen zusammen mit ihren Typdefinitionen sind im folgenden dargestellt und erläutert:

```
#define PI 3.14159265358979323846264 34       /* wg. Umrechnung ins Bogenmass */
typedef enum {False, True} Bool;
typedef int  intd[2];
typedef struct
{
    Bool   Touching;  /* True == Bodenkontakt */
    double TouchX,    /* Beruehrpunkt X Koordinate */
           TouchZ,    /* dto. Z Koordinate */
           TouchVX,   /* Geschwindigkeit der Unterlage */
           TouchVZ,   /* am Beruehrpunkt */
           BodenhoeheZ, /* Hoehe des waagrechten Bodens */
           ax, az,    /* elast. Bodenkraftkonstanten in X und Z */
           bx, bz,    /* dissi. Bodenkraftkonstanten in X und Z */
           cb;        /* nicht verwendet */
    int    OrdElast,  /* nicht verwendet */
           OrdDissi;  /* nicht verwendet */
} HDLBoden;

typedef struct
{
    double  x,      /* x - Koordinate */
            z,      /* z - Koordinate */
            phi,    /* Drehwinkel um y - Achse im Bogenmass */
            vx,     /* Geschwindigkeit in x - Richtung */
            vz,     /* Geschwindigkeit in z - Richtung */
            vphi;   /* Winkelgeschwindigkeit im Bogenmass */
} tKoordSatz;

typedef struct
{
    Bool      NichtBerechnen; /* True == Gelenk ohne Anschlag */
    double    PhiOben,        /* PhiAnschlagOben - DeltaPhiOben */
              PhiUnten,       /* PhiAnschlagUnten + DeltaPhiUnten */
              Oben,           /* MaxMoment / (exp(DeltaPhiOben) - expser (DeltaPhiOben, 3)) */
              Unten;          /* MaxMoment / (expser (DeltaPhiUnten, 3) - exp(DeltaPhiUnten)) */
} HDLGelAnschlag;

typedef struct
{
    int    VonK,/* Nummer des ersten Koerpers beg. bei 0 */
           VonH,/* Nummer des Hebels am ersten Koerper beg. bei 0*/
           BisK,/* Nummer des zweiten Koerpers beg. bei 0*/
           BisH;/* Nummer des Hebels am zweiten Koerper beg. bei 0*/
    double len, /* Ruhelaenge der Feder */
           a,   /* Federkonstante als Faktor */
           b,   /* Exponent fuer Laengenaenderung */
           c,   /* Faktor fuer Dissipation */
           d;   /* Exponent fuer Dissipation */
} tFeder;

typedef struct
{
    double  hx,     /* Koerperfeste x-Komponente des Hebelarms */
            hz;     /* Koerperfeste z-Komponente des Hebelarms */
} tHebel;

typedef double      tVektor[2];                /* Zweikompon. Vektor */
typedef double      tRotMatrix[2][2];          /* Rotationsm. e. Koerpers */
typedef int         tGelMatrix[MAXGM][MAXGM];  /* Gelenkmatrix */
typedef int         tGelVektor[MAXGM];         /* Spalte/ Zeile d.Gelenkm */

typedef double      tKoMatrix;


static tKoMatrix    a[33][33];  /* Koeffizientenmatrix des LGS, wird gesetzt in SetKoMa() */
```

```
static double       b[33],       /* Inhomogener Gleichungsanteil, wird gesetzt in SetInho() */
                    ac[27],      /* Beschleunigungen nach Lsg. des LGS, zugewiesen in SetBeschl() */
                    mt[9],       /* Momente: Wird gesetzt in KraftMoment() */
                    gx, gz;      /* Gravitationsbeschleunigungen in x und z Richtung */
static tVektor      kt[9],       /* Kraftvektoren: Wird gesetzt in KraftMoment() */
                    zk[3];       /* Langrange Mult. nach Lsg. des LGS, zugewiesen in SetZwKr() */
static HDLBoden hBoden[45];      /* Fuer jeden Hebel einen Variablensatz fuer moegl. Bodenkontakt */
static const double *m,          /* Referenz der Koerpermassen */
                    *t,          /* Referenz der Traegheitsmomente */
                    *sb,         /* Referenz der Sinusse der Drehwinkel */
                    *cb;         /* Referenz der Cosinusse der Drehwinkel */
static const tHebel *ri,         /* Referenz der raumfesten Hebelarme */
                    *rv,         /* Referenz der raumfesten Hebelgeschwindigkeiten */
                    *r;          /* Referenz der koerperfesten Hebelarme */
static int    GelAnschlOrdnung, /* Ordnung der Reihe f. Momente */
              VarZahl,          /* Anz. benutzerdef. Integvars */
              FedZahl;          /* Anzahl simulierter Federn */
static const intd   *GelNachbar;/* Jedes Gelenk hat ein Element, in dem */
                                 /* die Nachbarkoerpernummern eines Gelenks abgelegt sind */
                                 /* erste Zahl: Koerper 1, zweite Zahl Koerper 2 */
                                 /* Antwort auf: Welche Koerper sind ueber Gelenk x verbunden? */
                                 /* Vorsicht: Koerper- und Gelenknummern beginnen bei 0 */
static const int    *GelWinVorz,/* Gelenkwinkelvorzeichen wie im Gelenkblock definiert */
                    KoeZahl = 9,/* Koerperzahl */
                    HebPKoe = 5,/* Max. benoetigte Hebel eines Koerpers */
                    *KoeHebel;  /* Zahl der Hebel eines jeden Koerpers */
static const double *GelWinkel, /* Berechnete Winkel der Gelenke */
                    *GelWinGeschw, /* dto. Winkelgeschwindigkeiten */
                    *KoefWin;   /* Hebelwinkel im jeweiligen Koerpersystem */
static const HDLGelAnschlag *GelAnschlag; /* Fuer jedes Gelenk einen */
                                 /*  Variablensatz fuer moegl. Gelenkanschlag */
static const tFeder *Feder;      /* Fuer jede Feder einen Variablensatz */
```

Bei der Berechnung von Kräften und Momenten steht zusätzlich eine Variable zur Verfügung, die die eigenen integrierten Größen enthält.

```
double *var; /* Sind eigene Groessen mitintegriert worden, werden diese */
             /* ueber var[0], var[1], etc. angesprochen. Wurden keine */
             /* eigenen Groessen integriert, ist var == NULL */
```

Zusätzlich zu den oben hinter den Definitionen der Variablen angebrachten Kommentaren sind noch folgenden Punkte wichtig zum Verständnis:

- *Numerierung in C:* Aufgund der Konvention in der Programmiersprache C, den Arrayindex bei 0 beginnen zu lassen, sind die Daten des ersten Elements eines Arrays mit dem Index `[0]` anzusprechen.

- *Dimensionen:* An dieser Stelle sei daran erinnert, daß der hier beschriebene Quelltext problemspezifisch generiert wird. Deshalb sind die Dimensionen einzelner Arrays und Matrizen in zu verschiedenen Problemen gehörenden Dateien möglicherweise unterschiedlich.

- *Referenzen:* Viele Arrays sind nur als `const` verfügbar, um zu verhindern, daß in diesem Modul einzelne Werte verändert und so die Funktion des Programms möglicherweise beeinträchtigt wird. Die betreffenden Adressen werden in `KoMaCreate()` gesetzt.

- *Hebel:* Obwohl die einzelnen Körper unterschiedlich viele Hebel besitzen, ist für jeden Körper zur Speicherung seiner Hebel im Array `r` gleichviel

Platz reserviert wie für den Körper mit den meisten Hebeln. Alle Hebel sind in einem eindimensionalen Array untergebracht und die Hebel eines Körpers werden wie folgt zugegriffen. Der Körper habe im Gelenkblock die Nummer n (beginnend bei 1), der Körper mit den meisten Hebeln (jedes Gelenk zählt einfach, zzgl. der nicht zu einem Gelenk führenden Hebel) habe l Hebel. Der erste Hebel des Körpers n hat damit den Arrayindex $[(n-1)l]$, der zweite Hebel desselben Körpers hat $[(n-1)l+1]$, usw. Im Programm steht zur Adressierung im Array für die maximale Hebelzahl eines Körpers die Variable `HebPKoe` zur Verfügung. Gleiches gilt für die Hebelgeschwindigkeiten `rv`, für die raumfesten Hebelkoordinaten `ri`, für die Winkel der Hebel im körperfesten System `KoefWin` und für die Variablen zur Bodenkraftberechnung `hBoden`. Die tatsächliche Zahl an Hebeln eines Körpers ist im Array `KoeHebel` gespeichert.

- *Parameter:* Zusätzlich zu den hier aufgeführten Variablen stehen sämtliche im Parameterblock angegebenen Variablen als `double` zur Verfügung. Siehe Abschnitt 4.3.9.

4.3.5 Innere Kräfte

Im vorliegenden Beispiel wurden keine inneren Kräfte (Muskeln) benötigt. Aus diesem Grund ist der entsprechende Block auskommentiert. Da jedoch sinngemäß dasselbe gilt wie für die Gelenkmomente, wird auf den nächsten Abschnitt verwiesen.

```
#@@MUSKELN@
# Definition der inneren Kraefte, die zwischen zwei Koerpern wirken.
# Jede Kraft ist durch einen Muskel repraesentiert und muss
# zwischen zwei Hebeln wirken. Die Kraft wirkt stets laengs der
# Verbindungslinie beider Hebelendpunkte. Die Kraft hat positives
# Vorzeichen in Richtung vom ersten auf den zweiten Hebel.
# Jeder Eintrag hat die folgende Form:
# Variablenname(Koerper#1,Hebel#1-Koerper#2,Hebel#2)
# {
#     /* Hier kommen C Statements */
#     /* Diese Statements werden einschliesslich Klammer uninterpretiert */
#     /* in den Code uebernommen. Der Hebelabstand = Muskellaenge steht als */
#     /* Variable double dr == sqrt (dx*dx + dz*dz) zur Verfuegung */
# }
#@@@
```

4.3.6 Gelenkmomente

Im folgenden Beispiel ist demonstriert, wie in den drei Gelenken Gelenkmomente aufgebaut werden. Zu jedem Gelenk wird ein eigener Block erstellt, der mit dem Variablennamen des zu definierenden Gelenkmoments beginnt. Anschließend folgt in runden Klammern die Nummer des betreffenden Gelenks beginnend bei 1. Der Bewegungsgleichungsgenerator definiert die hier angegebene Variable automatisch an passender Stelle und übernimmt den im Block {} angegebenen Code Block für Block in derselben Reihenfolge wie hier aufgeführt. Im untenstehenden Beispiel sind deshalb Initialisierungsläufe nur im Code des zuerst definierten Gelenkmoments enthalten. Am Ende eines jeden Codeblocks wird der berechnete

Wert der entsprechenden Gelenkvariable zugewiesen. Werte von Gelenkmoments-variablen aus weiter oben stehenden Blöcken sind unverändert zu verwenden. Im folgenden Beispiel wird zur Berechnung der Gelenkmomente eine eigene Funktion aufgerufen.

```
@@MOMENTE@
# Definition der Gelenkmomente, die in einem Gelenk aufgebaut werden.
# Die Gelenkmomente koennen anstelle der Muskeln oder zusaetzlich zu
# den Muskeln zur Bewegung des MKS verwendet werden. Das Gelenkmoment
# wirkt im selben Sinn wie die Definition des zugehoerigen Gelenkwinkels,
# d.h. positiv in Richtung des Gelenkwinkels.
# Die Eintragungen in diesem Block haben folgende Form:
# Variablenname(Gelenk#)
# {
#     Variablenname = ...; /* Hier steht C Code */
# }
Handgelenk(3)
{
    int ii;
    if (zt == 0.0)
        for (ii = 0; ii < 3; ++ii)
            GelWinSoll[ii] = GelWinkel[ii];
    Handgelenk = GelMomPID (GelWinSoll[2], GelWinkel[2], GelWinGeschw[2],
                            var[2], HandgelP, HandgelI, HandgelD);
    HandgelM = Handgelenk;
}
Ellenbogengelenk(2)
{
    Ellenbogengelenk = GelMomPID (GelWinSoll[1], GelWinkel[1], GelWinGeschw[1],
                                  var[1], EllengelP, EllengelI, EllengelD);
    EllengelM = Ellenbogengelenk;
}
Schultergelenk(1)
{
    Schultergelenk = GelMomPID (GelWinSoll[0], GelWinkel[0], GelWinGeschw[0],
                                var[0], SchulgelP, SchulgelI, SchulgelD);
    SchulgelM = Schultergelenk;
}
@@@
```

Wie der Bewegungsgleichungsgenerator den hier angegebenen Code im einzelnen umsetzt, ist aus dem in Anhang abgedruckten Quelltext zu entnehmen.

4.3.7 Äußere Kräfte

Die Definition des Codes, der an dieser Stelle eingegeben werden soll, erfolgt in Analogie zum vorangegangenen Abschnitt. Da äußere Kräfte nur über Hebel an Körpern angreifen, ist hier in runden Klammern zusätzlich noch der betreffende Hebel (beginnend bei 1) anzugeben. Am Ende eines Blocks sind außerdem beide Komponenten der Kraft zu setzen, da die Kraft ein Vektor ist.
Die Programmzeilen, die das durch den Angriffspunkt im Körper erzeugte Drehmoment berechnen und zuweisen, werden vom Bewegungsgleichungsgenerator nach den letzten hier angegebenen Zeilen eines Blocks automatisch eingefügt und müssen daher nicht kodiert werden.

```
@@KRAEFTE@
# Definition der aeusseren Kraefte. Jede Kraft greift ueber einen
# Hebel an einem einzigen Koerper an. Dadurch wird zusaetzlich ein
# Moment erzeugt, das aufgrund dieser Eintragungen ebenfalls erzeugt
# wird. Die Eintragungen haben folgende Form:
# Variablenname(Koerper#:Hebel#)
```

```
# {
#     Variablenname[0] = ....; /* Hier steht C Code */
#     Variablenname[1] = ....;
# }
BohrHammer(5:2)
{
    double arg;

    arg = sin (BohrFrequenz * 2 * PI * zt);
    BohrHammer[0] = - BohrKraft * 0.5; /* * arg * arg; */
    BohrHammer[1] = 0.0;
}
HandKontakt(4:3)
{
    HandKontakt[0] = - (BohrKraft * 0.5 + VortriebKraft);
    HandKontakt[1] = 0.0;
}
Rumpf(1:5)
{
    Rumpf[0] = 0.5 * BohrKraft + VortriebKraft;
    Rumpf[1] = 0.0;
}
BohrerKontakt(5:1)
{
    BohrerKontakt [0] = - HandKontakt[0];
    BohrerKontakt [1] = 0.0; /* Fuehrungskraefte der Wand */
}
@@@
```

4.3.8 Äußere Momente

Auch hier erfolgen die Definitionen in Analogie zu den vorangegangenen Abschnitten.

```
@@AUSSMOM@
# Definition der aeusseren Momente. Jedes aeussere Moment greift an einem
# einzigen Koerper an.
# Variablenname(Koerper#)
# {
#     Variablenname = ...; /* Hier steht C Code */
# }
#
RumpfMom(1)
{
    static double phi0;
    double hx, hz;
    if (zt == 0.0)
        phi0 = z[0].phi;
    HebelRaufKoord (&hx, &hz, 3, 2);
    RumpfMom = - RumpfMoment * (z[0].phi - phi0);
}
@@@
```

4.3.9 Parameter

Will man die Berechnung der Kräfte und Momente abändern, werden die in den vorangegangenen vier Abschnitten erläuterten Dateien editiert und danach ein neues Simulationsprogramm erstellt. Dieses Vorgehen ist relativ Zeitaufwendig, da dazu der Bewegungsgleichungsgenerator, der Compiler und der Linker ablaufen müssen und ist nur bei grundsätzlichen Änderungen an den Routinen gerechtfertigt. Kleine Änderungen wie die Variation von Faktoren sollten parametrisiert und ohne Neukompilation durchführbar sein. Es besteht daher die

Möglichkeit, Variablen zu definieren, die ihren Wert aus der Parameterdatei unter einem ebenfalls definierten Schlüsselwort lesen, oder den Wert mit einem default belegen, falls das Schlüsselwort nicht in der Parameterdatei enthalten ist. Die Definition eines solchen Parameters erfordert also die Angabe des Schlüsselwortes, des Variablennamens und des default–Wertes. Der Bewegungsgleichungsgenerator fügt im generierten Code an passender Stelle die Definition der Variablen ein und generiert in der Funktion `KoMaCreate()` Code, der das Lesen der Werte aus der Parameterdatei bewirkt. Die hier definierten Variablen sind in allen zuvor besprochenen Blöcken definiert und man kann damit parametrisierte Berechnungen der Momente und Kräfte durchführen. Die Syntax der dazu notwendigen Definitionen lautet wie folgt.

```
@@PARAMETER@
# Alle hierin aufgefuehrten Variablen werden im erzeugten File als
# static definiert und mit den hier voreingestellten Werten initialisiert.
# Die initialisierten Werte koennen jederzeit durch die in der
# Parameterdatei stehenden Werte ueberschrieben werden.
# Jede Zeile hat folgende Form:
# Schluessel:Variable = Initializer;
#
BohrKraft:BohrKraft=0.0;
BohrFrequenz:BohrFrequenz=5.0;
VortriebKraft:VortriebKraft=0.0;
RumpfMoment:RumpfMoment = 100;
HandgelM:HandgelM = 0.0;
HandgelP:HandgelP = 0.0;
HandgelI:HandgelI = 0.0;
HandgelD:HandgelD = 0.0;
EllengelM:EllengelM = 0.0;
EllengelP:EllengelP = 0.0;
EllengelI:EllengelI = 0.0;
EllengelD:EllengelD = 0.0;
SchulgelM:SchulgelM = 0.0;
SchulgelP:SchulgelP = 0.0;
SchulgelI:SchulgelI = 0.0;
SchulgelD:SchulgelD = 0.0;
@@@
```

4.3.10 Masse und Trägheitsmomente

Nachdem in den vorangegangenen Abschnitten erläutert wurde, wie eine Simulation zu erstellen ist, folgen ab hier Dateien, die die Daten einer Simulation definieren. Die Massen und Trägheitsmomente aller an einer Simulation beteiligten Körper sind in zwei getrennten Blöcken wie folgt anzugeben:

```
@@MASSEN@
# F"ur jeden K"orper wird in einer eigenen Zeile seine Masse angegeben.
#   Rumpf
125.0
#   Oberarm
8.0
#   Unterarm
4.0
#   Hand
0.8
#   Bohrer
20.0
#   Schwabbelmassen
25.0
4.0
2.0
```

```
0.1
@@@

@@TRAEGHEITSMOMENTE@
# Das Tr"agheitsmoment eines jeden K"orpers wird wie die K"orpermasse
# in je einer eigenen Zeile angegeben.
#   Rumpf
10.7
#   Oberarm
0.5
#   Unterarm
0.3
#   Hand
0.1
#   Bohrer
2.0
#   Schwabbelmassen
1.5
0.2
0.1
0.05
@@@
```

4.3.11 Hebel

Alle Hebel eines Körpers müssen in körperfesten Koordinaten angegeben werden. Die Reihenfolge der Hebel entspricht ihrer Numerierung wie im Abschnitt 4.2.1 vereinbart und die Anzahl muß genau mit den Angaben der Gelenkverbindungen übereinstimmen. Die Hebel werden wie folgt angegeben.

```
@@HEBEL@
# Fuer jeden der Koerper werden die Hebelarme (= Ortsvektoren) der
# Gelenke im koerperfesten Haupttraegheitsachsensystem angegeben. Die
# Numerierung der Gelenke erfolgt dabei wie in der Matrix eingetragen
# von links nach rechts und von oben nach unten. Die Hebelarme aller Gelenke
# eines Koerpers m"ussen in aufsteigender Folge der Ordnungszahlen der Gelenke
# angegeben werden. Die Abfolge der einzelnen K"orper hat in
# aufsteigender Reihenfolge ihrer Ordnungszahlen jeweils zeilenweise zu erfolgen.
# Fuer jeden Koerper werden alle Hebelarme angegeben, fuer jeden
# davon zwei Komponenten.
#
#        ------------------ reserviert, muss z. Zt. stets 0 sein
#        |   -------------- Ausgabe der raumfesten Koordinaten des Endpunkts
#        |   |              0 (nein) 1 (ja)
#        |   |    --------- x und z Koordinate (eigentlich y) des Hebels
#        |   |    |   |     im koerperfesten System
#        |   |    |   |
#        V   V    V   V
#       (0, Ausg, x1, z1); (0, Ausg, x2, z2); ...
#
#   Rumpf(1)
        (0, 0, 0.02, 0.60); (0, 0, 0.0, -0.90); (0, 0, 0.0, 0.90); (0, 0, 0.0, 0.60); (0, 0, 0.0, 0.0);
#   Oberarm(2)
        (0, 0, 0.0, 0.25);  (0, 0, 0.0, -0.25);
#   Unterarm(3)
        (0, 0, 0.0, -0.20);  (0, 0, 0.0, 0.20);
#   Hand(4)
        (0, 0, 0.0, -0.02);  (0, 0, 0.0, 0.14); (0, 1, 0.0, 0.06);  (0, 0, 0.0, 0.0);
#   Boschhammer(5)
        (0, 1, 0.0, 0.18);  (0, 0, 0.0, -0.18);
#   Schwabbel Rumpf(6)
        (0, 1, 0.0, 0.60);   (0, 0, 0.0, -0.90);
#   Schwabbel Oberarm(7)
        (0, 0, 0.0, 0.25);  (0, 0, 0.0, -0.25);
#   Schwabbel Unterarm(8)
        (0, 0, 0.0, -0.20);  (0, 0, 0.0, 0.20);
```

```
#   Schwabbel Hand(9)
        (0, 0, 0.0, -0.02);  (0, 0, 0.0, 0.14);
@@@
```

4.3.12 Federn

Die Federn sind ein bequemes Instrument zur Ankopplung von Schwabbelmassen. Die mögliche Anzahl von Federelementen ist unbegrenzt und die Definition erfolgt wie unten angegeben.

```
@@FEDERN@
# In diesem Block werden alle Federn eingegeben. Eine Feder verbindet
# zwei beliebige Hebelenden zweier K"orper. Die Federkonstanten
# werden danach festgelegt.
# Syntax: (KoerperNr1:HebelNr1-KoerperNr2:HebelNr2) = (l, a, b, c, d);
#           l - Ruhelaenge in Meter
#           a - Federkonstante bei Dehnung
#           b - Exponent fuer Federkraft bei Dehnung
#           c - Konstante bei Dissipation
#           d - Exponent fuer Federkraft bei Dissipation
# Federkraft = a (1000(L - l))^b + c (1000v)^d (L - l)
#           v = Zeitableitung von L
#           L = Federlaenge
(1:4-6:1) = (0.0, 0.0, 1.0, 0.0, 1.0);
(1:2-6:2) = (0.0, 0.0, 1.0, 0.0, 1.0);
(2:1-7:1) = (0.0, 0.0, 1.0, 0.0, 1.0);
(2:2-7:2) = (0.0, 0.0, 1.0, 0.0, 1.0);
(3:1-8:1) = (0.0, 0.0, 1.0, 0.0, 1.0);
(3:2-8:2) = (0.0, 0.0, 1.0, 0.0, 1.0);
(4:1-9:1) = (0.0, 0.0, 1.0, 0.0, 1.0);
(4:2-9:2) = (0.0, 0.0, 1.0, 0.0, 1.0);
@@@
```

4.3.13 Anfangswerte und –winkel

Die Anfangsbedingungen der Simulation werden in reduzierten Koordinaten eingegeben, da vollständige Koordinaten zur Eingabe der Anfangsbedingungen ungeeignet sind. Für jede kinematische Kette ist als Bezugspunkt ein Punkt in raumfesten Koordinaten anzugeben. Für jeden Körper ist zusätzlich die Orientierung in Form eines Winkels anzugeben. Als Bezugspunkt zugelassen sind der Körperschwerpunkt oder jeder beliebige Hebelendpunkt des Körpers. Koordinatenangaben werden in raumfesten Koordinaten entweder bezüglich des Koordinatenursprungs oder bezüglich des Schwerpunktes eines jeden anderen Körpers angegeben. Die Anfangswerte werden in folgender Form angegeben:

```
@@ANFANGSWERTE@
#   Hierin werden die Aufpunktskoordinaten bei Simulationsbeginn definiert.
#   Fuer jede kinematische Kette ist in einer eigenen Zeile der Bezugspunkt
#   zu definieren, seine beiden Koordinaten x und z sowie seine beiden
#   Geschwindigkeiten vx und vz einzugeben. Der Bezugspunkt ist ueber die
#   Koerpernummer beginnend bei 1 und den verwendeten Hebel (beginnend bei 1)
#   einzugeben. Soll der jeweilige
#   Koerperschwerpunkt der Bezugspunkt sein, wird der verwendete Hebel mit
#   0 bezeichnet. Angaben bezueglich des Koordinatenursprungs erhalten anstelle
#   der Koerpernummer eine 0.
#   Folgende Syntax ist hierbei zu verwenden:
#   Koerpernummer-Hebelnummer, Bezugskoerper: (x, vx); (z, vz);
4-3, 0: (1.0, 0.0); (1.6, 0.0);
5-1, 0: (1.0, 0.0); (1.6, 0.0);
6-0, 1: (0.0, 0.0); (0.0, 0.0);
7-0, 2: (0.0, 0.0); (0.0, 0.0);
```

```
8-0, 3: (0.0, 0.0); (0.0, 0.0);
9-0, 4: (0.0, 0.0); (0.0, 0.0);
@@@
```

Die Eingabe der Anfangswinkel hat folgende Syntax:

```
@@ANFANGSWINKEL@
# Fuer jeden Starrkoerper ist hier in einer eigenen Zeile der Neigungswinkel
# Phi und die zugeh"orige Winkelgeschwindigkeit, mit denen die Simulation
# gestartet werden soll, einzugeben. Die Winkel werden in Grad
# eingegeben und bezeichnen die Drehung des k"orperfesten Koordinatensystems
# gegen das raumfeste Koordinatensystem.
# Mit den nachfolgenden Winkeln und -geschwindigkeiten sind die Formen
# und die Formveraenderungen aller kinematischen Ketten zum Anfangszeitpunkt
# der Simulation bekannt. Zur vollstaendigen Angabe der Anfangsbedingungen
# ist jedoch fuer jede kinematische Kette noch die Eingabe der raumfesten
# Koordinaten eines beliebigen Punktes erforderlich. Dieser wird in
# der Sektion @@ANFANGSWERTE@ eingegeben, siehe dort.
# Syntax einer Zeile:
#   (Phi, PhiPunkt);
# Rumpf (1)
0:    (0.0, 0.0);
# Oberarm (2)
0:    (45.0, 0.0);
# Unterarm (3)
0:    (-45.0, 0.0);
# Hand (4)
0:    (0.0, 0.0);
# Boschhammer (5)
0:    (90.0, 0.0);
# Schwabbelmassen (6, 7, 8, 9)
1:    (0.0, 0.0);
2:    (0.0, 0.0);
3:    (0.0, 0.0);
4:    (0.0, 0.0);
@@@
```

Die hier eingegebenen reduzierten Koordinaten werden zu Beginn der Simulation in vollständige Koordinaten umgerechnet.

4.3.14 Integration eigener Größen

Mit den bisher vorgestellten Fähigkeiten ist es nicht möglich, eigene Größen simultan zur laufenden Integration mitzuintegrieren. Um dies zu ermöglichen, wurde in den generierten Quelltext direkt nach Definition der Variablen und Funktionsprototypen die Zeile `#include ''simuser.hc''` eingefügt. Diese Zeile wird vor dem eigentlichen Übersetzungsvorgang durch den Compiler durch den Quelltext aus der Datei `simuser.hc` ersetzt.
In der Datei `simuser.hc` steht die folgende Funktion, die im Fall des Bohrhammerbeispiels die Differenz zwischen den Soll– und Istwerten der drei Gelenkwinkel des Armes integriert und damit Bestandteil der noch zu besprechenden Steuerung ist.

```
/*--- NAME BerechneVars() ----------------------------------------------------
    Berechnung der Zeitableitungen der benutzerdef. Variablen
----- SYNOPSIS ---------------------------------------------------------------
#include "simsys.h"
----- PARAMETER --------------------------------------------------------------
varp     Hierhinein muessen die Zeitableitungen der Variablen.
var      Hier steht der Loesungsvektoren der Variablen zur Zeit zt.
zs       Vektor der berechneten Geschwindigkeiten und Beschleunigungen des
```

```
        Gesamtsystems. Die Anzahl der Koerper ist in der Variablen KoeZahl
 abgelegt.
z       Vektor z der Koordinaten und Geschw. des Gesamtsystems.
zt      Zeitkoordinate
IntegVar Anzahl benutzerdefinierter Variablen.
----- DESCRIPTION ------------------------------------------------------------
    Wie von der Integrationsroutine de_() verlangt, muessen hier die
    Zeitableitungen der benutzerdefinierten Integrationsvariablen
    auf den gewuenschten Wert gesetzt  werden.
-----------------------------------------------------------------------------*/
void BerechneVars (varp, var, zs, z, zt, IntegVar)
double *varp;
const double *var;
const tKoordSatz *zs;
const tKoordSatz *z;
const double zt;
int IntegVar;
{
    int ii;
    for (ii = 0; ii < 3 && ii < IntegVar; ++ii)
varp[ii] = GelWinkel[ii] - GelWinSoll[ii];
} /* BerechneVars() */
```

Man kann die oben genannte Funktion genauso als eigenes Modul mitführen, vorausgesetzt, man ändert den Makefile entsprechend ab und stellt sicher, daß das Modul ebenfalls übersetzt und gelinkt wird. Die Einbindung eines Includefiles bietet unabhängig von der hier vorgestellten Nutzungsmöglichkeit jedoch zusätzliche Fähigkeiten, die die "unsaubere" Lösung dieser Art rechtfertigen.[7]

4.4 Erreichter Stand, Einschränkungen

Die im vorangegangenen Abschnitt vorgestellten Fähigkeiten von *simsys* genügen fast allen Forderungen aus 3.3. Verbesserungsbedürftig sind allerdings noch folgende Punkte:

1. Die Körperform und die entsprechenden Kontaktkräfte werden nicht berücksichtigt.
2. Der Boden ist bislang immer horizontal (parallel zur Richtung X). Die Eingabe jeder beliebigen Bodenform wäre wünschenswert.
3. Raumfeste Punkte und raumfeste Federenden sind nicht implementiert.
4. Massenpunkte zur Modellierung von Schwabbelmassen sind nicht implementiert.
5. Der Integrierer *de_()* kann im Augenblick nur 100 Gleichungen 1. Ordnung simultan integrieren. Diese Einschränkung ist bei der Erweiterung der Modelle stets im Auge zu behalten.
6. Durch die Form der Eingabe der Gelenke lassen sich bisher maximal 30 Körper simulieren.

[7]Unter Verwendung des C Preprocessors lassen sich z. B. bedingte Compilation oder gemeinsame Macros benutzen, ferner sind alle Variablen des Moduls verwendbar

7. Die Koordinate Y ist fälschlicherweise stets mit Z bezeichnet.

8. Alle Modelle sind zweidimensional.

4.5 Erweiterung auf drei Dimensionen

Die Erweiterung auf drei Dimensionen ist prizipiell wünschenswert. Dazu wären abgesehen vom einfachen Umsetzen der in zwei Dimensionen implementierten Eigenschaften vor allem folgende zusätzliche Eingaben zu machen:

- Bei Eingabe der Gelenkverbindungen muß zwischen Kugel– und Scharniergelenk unterschieden werden. Ein Scharniergelenk bedarf der zusätzlichen Angabe der Gelenkebene. Hierfür muß eine geeignete Form gefunden werden.

- Die Behandlung des Gelenkwinkels bei Kugelgelenken erfordert zwei Winkelangaben. Diese müssen zum Vergleich normiert sein. Auch hierfür fehlen bisher geeignete Definitionen.

- Die Definition der Gelenkanschläge eines Kugelgelenks gestaltet sich sehr schwierig, da statt den Anschlagswinkeln (d. h. eindimensionale Grenzen) nun geschlossene Anschlagslinien anzugegeben sind. Hierfür muß eine geeignete, pflegbare Eingabeform gefunden werden.

5 Versuche der Steuerung

5.1 Problemstellung

Mit den im vorangegangenen Kapitel beschriebenen Programmsystem *simsys* ist die Erstellung des Grundgerüsts einer Simulation in sehr kurzer Zeit möglich. Ausgehend von einer Modellskizze läßt sich sehr rasch das "Pinocchio"–Stadium des Modells erreichen. Man hat dann im Prinzip eine regungslose (aber allen äußeren Kräften und Momenten gehorchende) Puppe geschaffen. Um aber etwas über Kräfte und Belastungen in speziellen Bewegungsabläufen zu erfahren, muß sich die Puppe bewegen können. Man muß daher versuchen, dem modellierten System die zu untersuchende Bewegung beizubringen.

In diesem Kapitel soll nun genau dieses Problem für das Beispiel aus Abschnitt 4.2.2 angegangen werden. Um die Auswirkungen der Arbeit mit einem Bohrhammer auf das menschliche Skelett mithilfe von Simulationen untersuchen zu können, muß das Modell den dazu notwendigen Bewegungsablauf ausführen können. Das Verhalten des Bohrhammers soll in Form des Kraftverlaufs auf die Hand des Modells als äußere Kraft berücksichtigt werden und das Modell soll den Bohrhammer letztlich führen können. Dazu ist zunächst das Problem zu untersuchen, wie der Arm gegen eine vom Bohrhammer übertragene Kraft seine Haltung beibehalten kann. Anschließend muß daran gearbeitet werden, diese Haltung im Sinne der gewünschten Bewegung kontrolliert zu verändern.

Die genaue Implementierung des Bohrhammerverhaltens und eine experimentell gestützte Simulation der Kraft–Deformationsbeziehung in der Hand ist zum Verständnis der Armsteuerung zunächst nicht wichtig und wird deshalb im Rahmen dieser Arbeit nicht ausgeführt. Es soll hier der Kraftverlauf des Bohrhammers stark vereinfacht mit einer zeitabhängigen Funktion $f(t) = a\sin^2(\omega t)$ angenommen werden. Die im zeitlichen Mittel auf die Hand übertragene Kraft ist dann $0.5a$. Der Arm soll zunächst nur diese Kraft aufbringen und seine Haltung beibehalten.

Im vorliegenden Kapitel soll nun versucht werden, das Modell dahingehend zu verbessern, daß es die Armhaltung unter Einwirkung einer (nochmals vereinfachten) konstanten Kraft beibehält.

5.2 PID–Regelmechanismus

Um die in den Anfangsbedingungen vorgegebene Armhaltung gegen die auf die Hand einwirkende Kraft beizubehalten, werden Gelenkmomente im Handgelenk, Ellenbogengelenk und Schultergelenk erzeugt. Hierfür wird mit einem PID-Regler experimentiert. Die Regelgröße des Gelenks ist der jeweilige Gelenkwinkel φ, die Stellgröße ist das Gelenkmoment M, die Regelvorgabe ist der Anfangswinkel des Gelenks φ_0 zum Anfangszeitpunkt t_0. Der Regler regelt das Gelenkmoment mithilfe dreier Summanden, einem Proportionalteil, einem Integralteil und einem

Differentialteil und die Stellgröße setzt sich damit wie folgt zusammen:

$$M(t) = P(\varphi(t) - \varphi_0) + I \int_{t_0}^{t} \varphi(\tau) - \varphi_0 d\tau + D\dot{\varphi}(t) \tag{75}$$

Die drei Größen P, I und D sind im Einzelfall individuell festzulegende Konstanten, um die verschiedenen Regelanteile bedarfsgemäß einzustellen. Die Vorzeichen sind so zu wählen, daß die Wirkung des jeweiligen Regelteils im richtigen Sinn auf die Regelstrecke (das Gelenk) einwirkt. Für den Fall, daß die Regelvorgabe zeitabhängig ist, muß der Differentialteil noch um die Zeitableitung der Regelvorgabe sinngemäß erweitert werden.

Der Proportionalteil bewirkt eine Erhöhung bzw. Erniedrigung der Stellgröße in Abhängigkeit der Abweichung der Sollgröße von der Regelvorgabe. Der Proportionalteil alleine kann jedoch nicht verhindern, daß sich ein Gleichgewicht einstellt, bei dem die Sollgröße trotzdem noch von der Regelgröße abweicht. Die Aufgabe, diese Regelabweichung zu verhindern, hat der Integralteil. Die Abweichung wird aufintegriert und je länger sie besteht, desto größer wird der Integralteil und damit die Stellgröße. Der Differentialteil hat die Aufgabe, Veränderungen der Regelgröße entsprechend dem Vorzeichen von D entgegenzuwirken.

5.3 Implementierung

Die Modellierung der drei für die betreffenden Gelenke notwendigen Regler ist in Abschnitt 4.3.6 als Quelltext abgedruckt. Die Stellgrößen werden durch Aufruf der Funktion `GelMomPID()` wie im vorangegangenen Abschnitt dargelegt berechnet, wobei für jedes Gelenk die drei Größen P, I und D aus Glg. (75) als Parameter vorgesehen ist (siehe Abschnitt 4.3.9). Die Funktion `GelMomPID()` zur Berechnung der Gelenkmomente ist in der Datei `simuser.hc` kodiert und in Anhang A.7 abgedruckt. Dort ist das Vorzeichen der Stellgröße ebenfalls angegeben.

Während bei der Modellierung auf alle oben angegebenen Details Wert gelegt worden ist (Schwabbelmassen, Bohrhammer und Einbau der Kraft–Deformationsbeziehung zwischen Hand und Bohrhammer sowie eine periodische Kraft auf den Bohrer), spielen diese Feinheiten beim grundlegenden Problem, gegen eine plötzlich auftretende, konstante Kraft die Armhaltung beizubehalten, nur eine untergeordnete Rolle. Aus diesem Grund werden die Kopplungen zwischen den Schwabbelmassen und den Knochenteilen abgeschaltet und die Kraft–Deformationsbeziehung zwischen dem Bohrhammer und der Hand wird ersetzt durch eine direkt an der entsprechenden Stelle angreifende Kraft von $150N$. Die Bewegung des Bohrhammers wird dadurch unnötig und kann außer Betracht bleiben. Um zu verhindern, daß sich der Rumpf aufgrund des steif werdenden Armes verdreht, wird dieser mithilfe einer (linearen) Torsionsfeder am Schwerpunkt ausgerichtet und mit einem hohen Trägheitsmoment versehen. Damit sich der Rumpf nicht durch die an der Hand angreifende Kraft bewegt, wird dieser im Schwerpunkt durch eine gleichgroße, entgegengerichtete Kraft gehalten. Die Torsionsfeder und die eben erwähnte Kraft sind die Modellierung des unter realen Bedingungen über den Fußkontakt zum Boden und die Bodenreaktionskraft aufgebrachten Drehmoments auf den Rumpf und haben daher ihre Berechtigung. Das vereinfachte Modell zeigt Abb. 8.

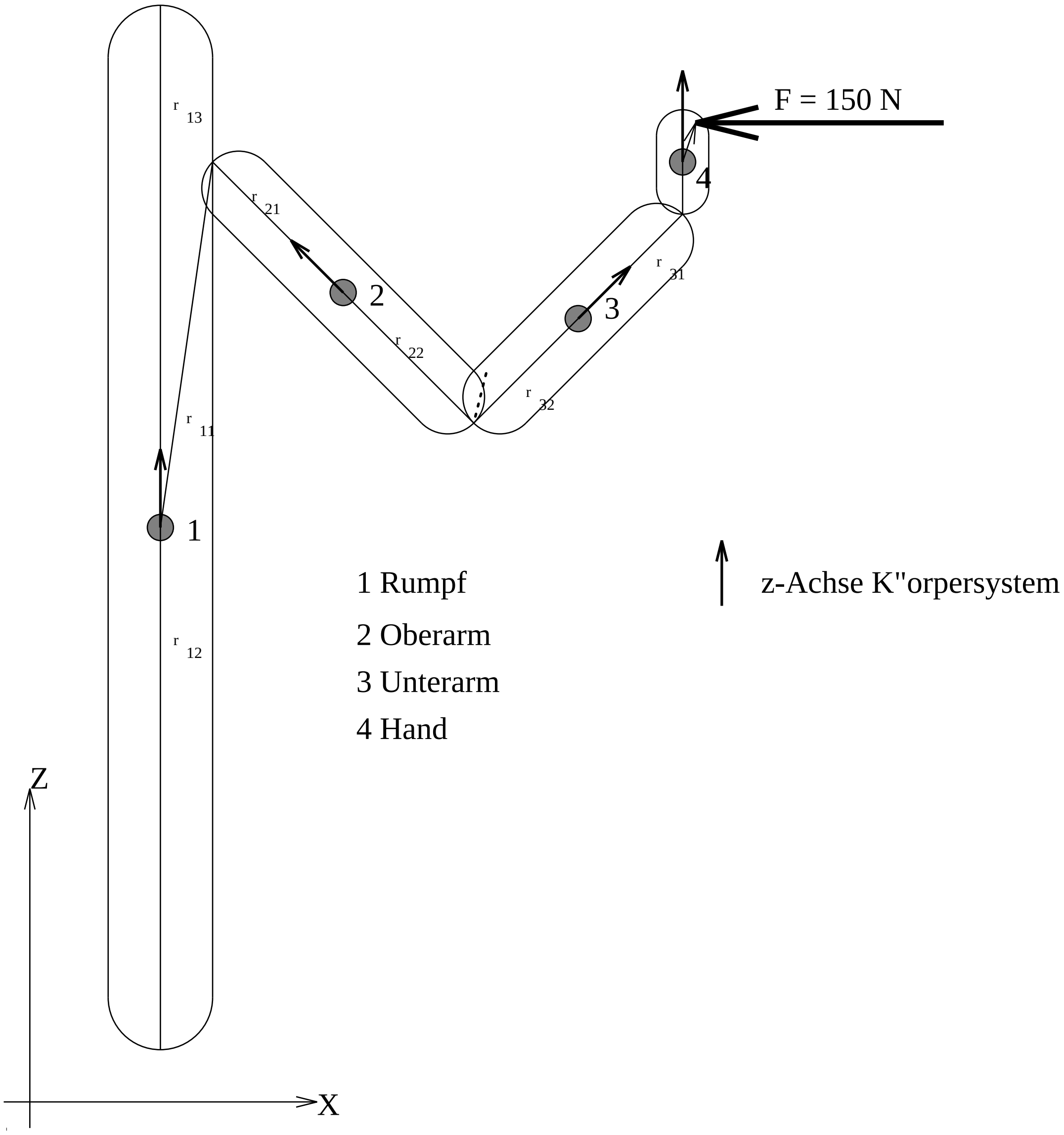

Abbildung 8: Simuliertes Modell

5.4 Ergebnis

Mit diesem nochmals stark vereinfachten Modell wird nun versucht, im zunächst kräfte– und momentenfreien Skelett die zum Anfangszeitpunkt der Simulation auftretende Kraft durch den Aufbau von Gelenkmomenten zu kompensieren. Der Aufbau der Gelenkmomente erfolgt mittels des oben dargestellten PID-Reglers, bei dem die drei Konstanten so einzustellen sind, daß der Arm seine Ausgangshaltung wieder einnimmt, die Momente nicht zu groß werden und der Momentenaufbau sich in charakteristischen Zeiten von etwa 100 ms vollzieht. Diese

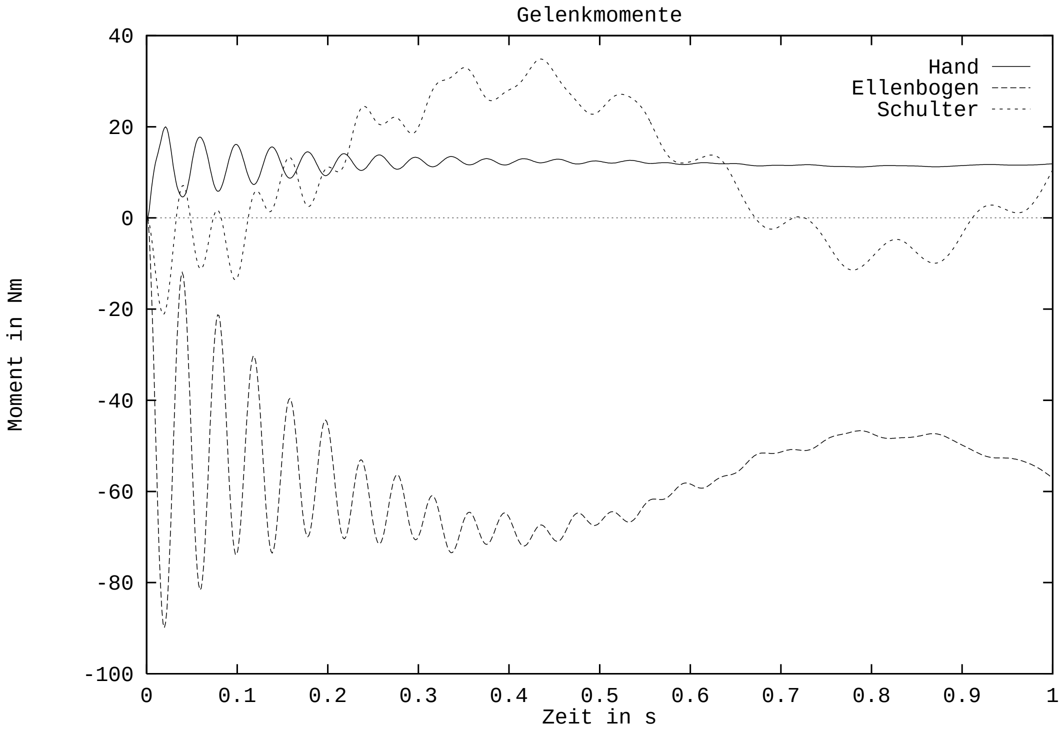

Abbildung 9: Gelenkmomente bei $P = 200$, $I = 400$, $D = 20$

"Reaktionszeit" beim plötzlichen Aufbau von Momenten ergibt sich beim Menschen durch die Signallaufzeit von der Hand in das Rückenmark und zurück bzw. vom Gehirn in die entsprechende Muskulatur sowie durch die Signalverarbeitung durch Neuronen und ist ein wesentlicher Unterschied zu gesteuerten Maschinen und Roboterarmen, bei denen diese Zeit entfällt. Die Beschränkung der Momentengröße auf ein natürliches Maß ist ein weiterer, wichtiger Unterschied zwischen Mensch und Maschine.

Im folgenden werden die Gelenkmomente und die Gelenkwinkel im Verlauf der ersten nach Eintritt der Krafteinwirkung simulierten Sekunde dargestellt. In den verschiedenen Diagrammen werden unterschiedliche Kombinationen der Konstanten des PID–Reglers ausprobiert.

Hierbei werden ausgehend von der Wertekombination von P, I und D aus Abb. 9 jeweils zwei der Werte festgehalten und der dritte verändert.

Auffallend an der Einstellung aus Abb. 9 ist, daß die Gelenkwinkel zwar gegen die plötzlich einwirkende Kraft in ihre ursprüngliche Lage zurückstreben und entsprechende (in der Größe auch plausible) Gelenkmomente aufgebaut werden, es ist jedoch die Zeit des Momentenaufbaus viel zu kurz. Innerhalb von ca. 0.02 Sekunden nach Krafteinwirkung haben alle Momente bereits ihr Maximum erreicht. Es ist also anzustreben, diese Zeit zu verlängern. Aufgrund dieser schnellen Reaktion sind die Winkelamplituden auch unrealistisch klein.

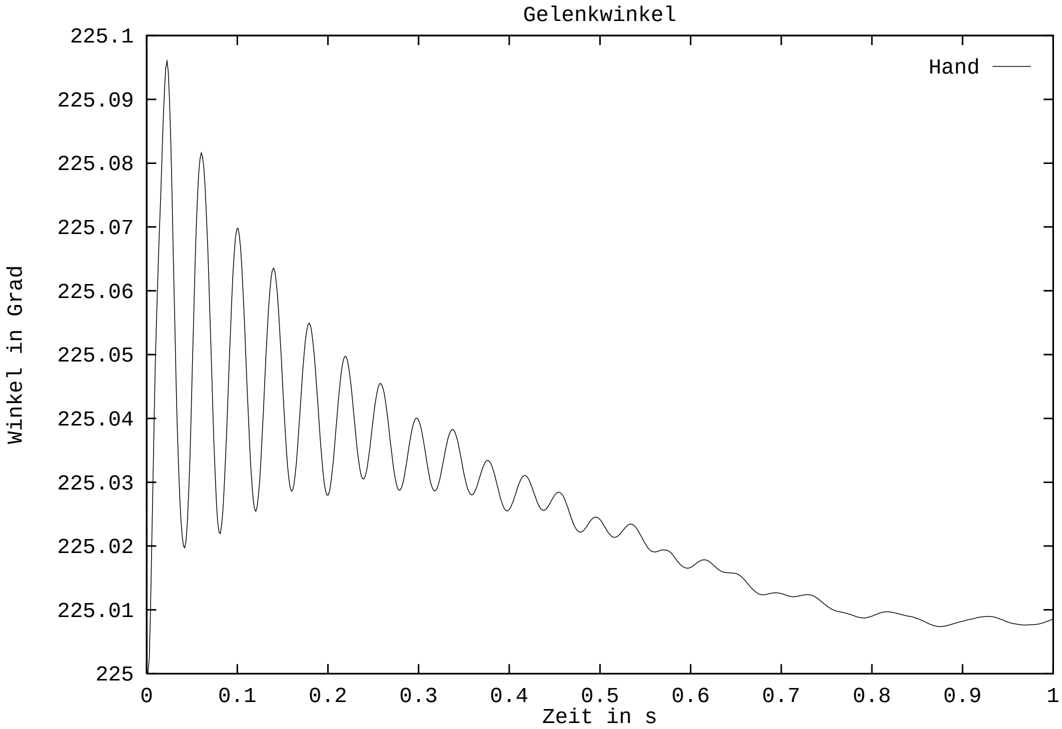

Abbildung 10: Handgelenkwinkel bei $P = 200$, $I = 400$, $D = 20$

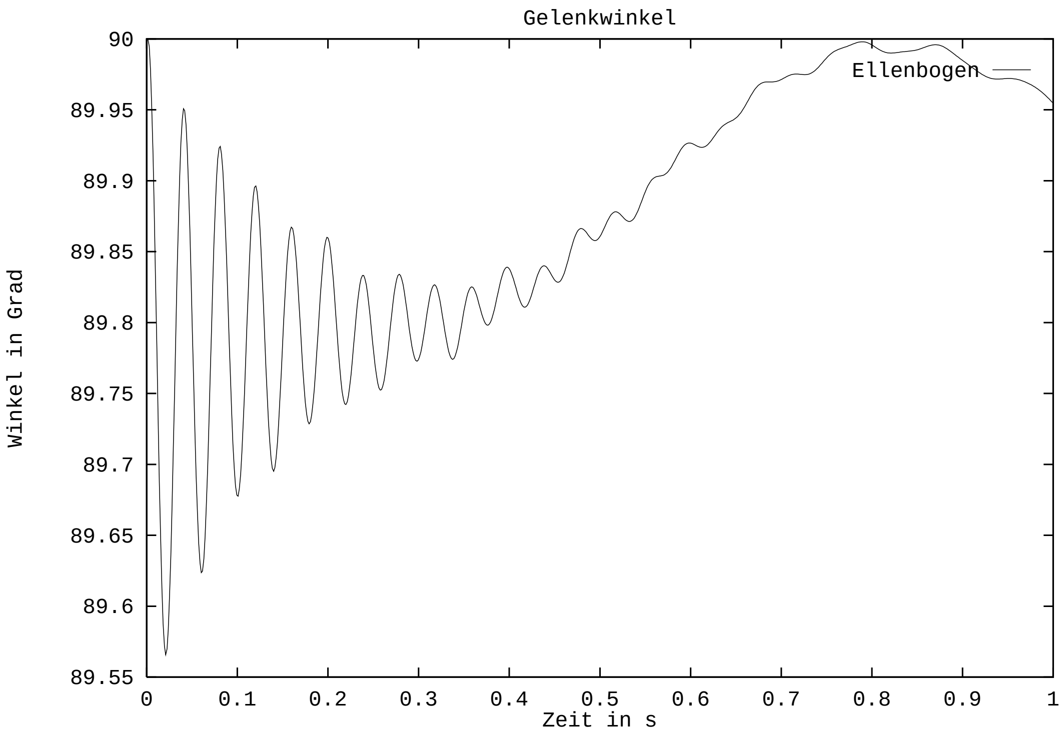

Abbildung 11: Ellenbogenwinkel bei $P = 200$, $I = 400$, $D = 20$

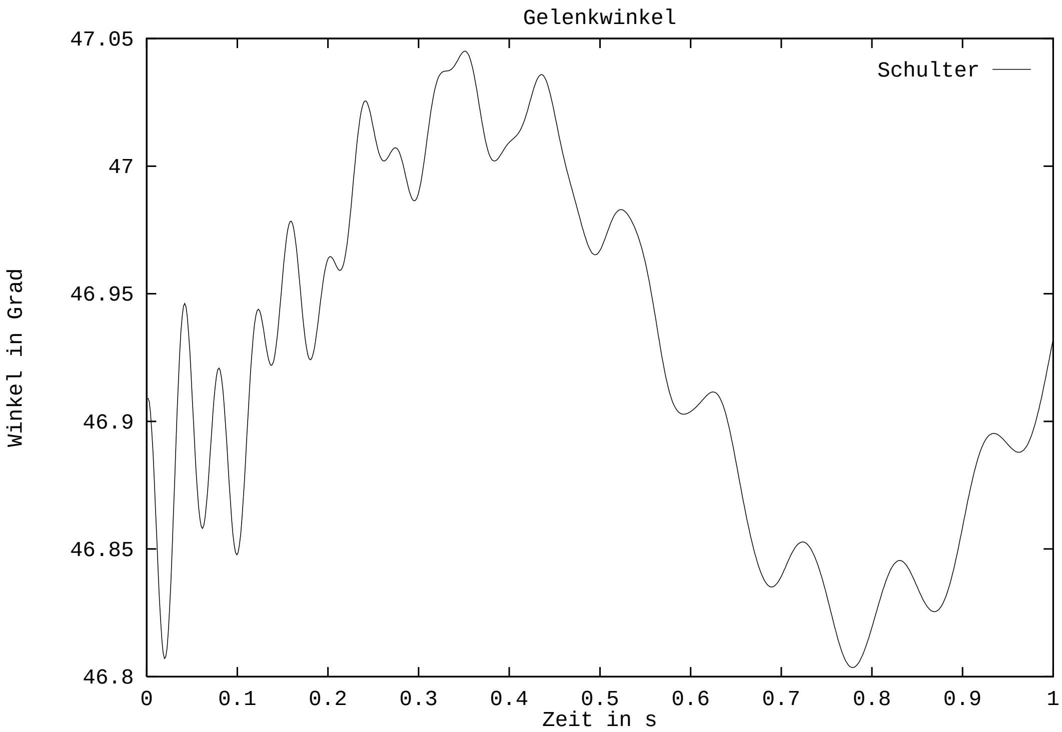

Abbildung 12: Schulterwinkel bei $P = 200$, $I = 400$, $D = 20$

Im Vergleich zum vorangegangenen Simulationslauf wurde im Simulationslauf zu Abb. 13 der Proportionalanteil um den Faktor 10 verkleinert. Dies hat zur Folge, daß die unerwünscht schnelle Reaktionszeit auf ein plausibles Maß zurückgeht. Es muß jedoch angezweifelt werden, ob diese Einstellung in der Lage ist, den Arm zu stabilisieren, wenn man die beiden Peaks am Ende der Simulationszeit im Ellenbogen– und im Schultergelenk betrachtet. Es scheint, als würde die Schulter die Beruhigung der anderen beiden Gelenke verhindern und diese neu aufschaukeln.

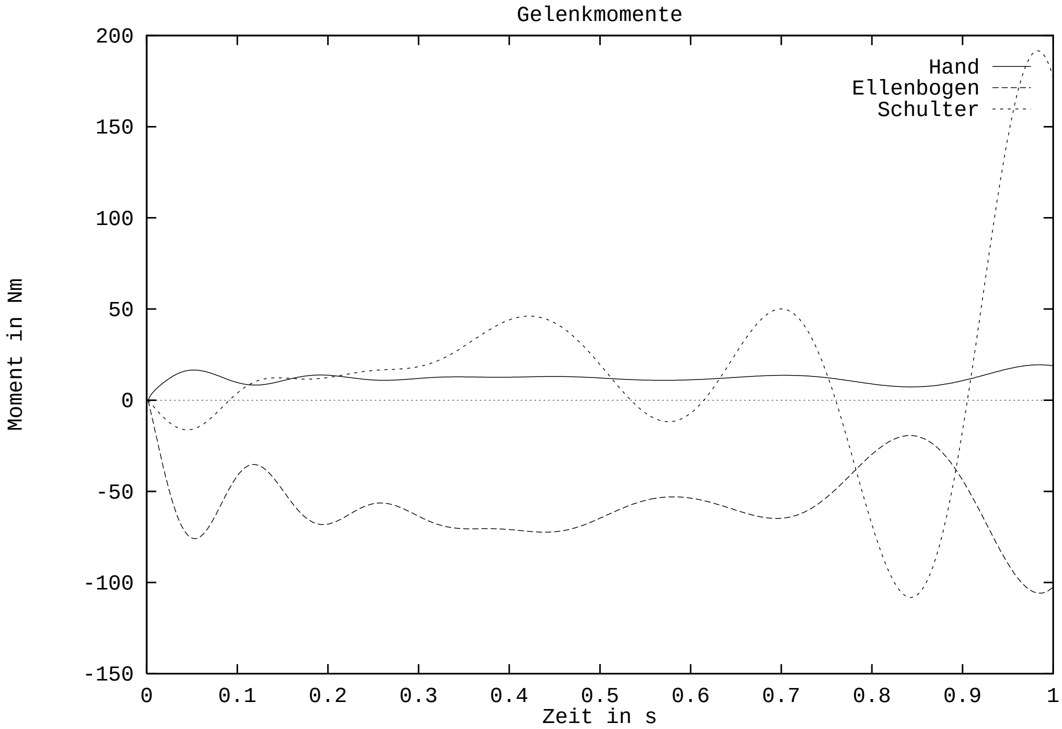

Abbildung 13: Gelenkmomente bei $P = 20,\ I = 400,\ D = 20$

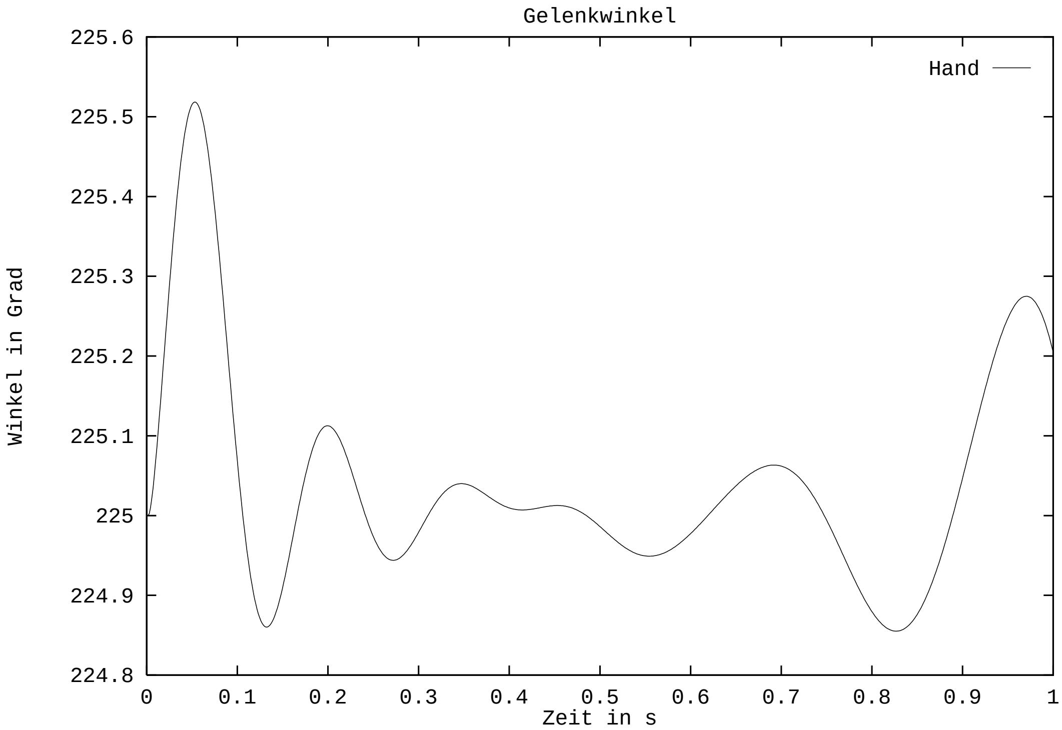

Abbildung 14: Handgelenkwinkel bei $P = 20,\ I = 400,\ D = 20$

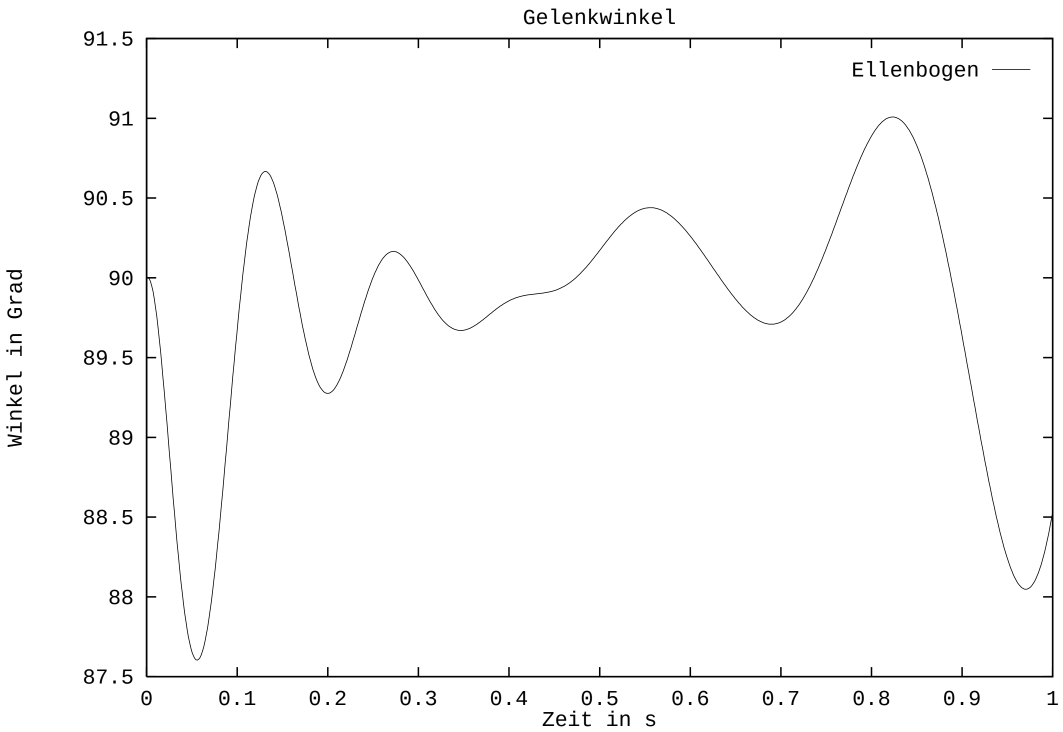

Abbildung 15: Ellenbogenwinkel bei $P = 20,\ I = 400,\ D = 20$

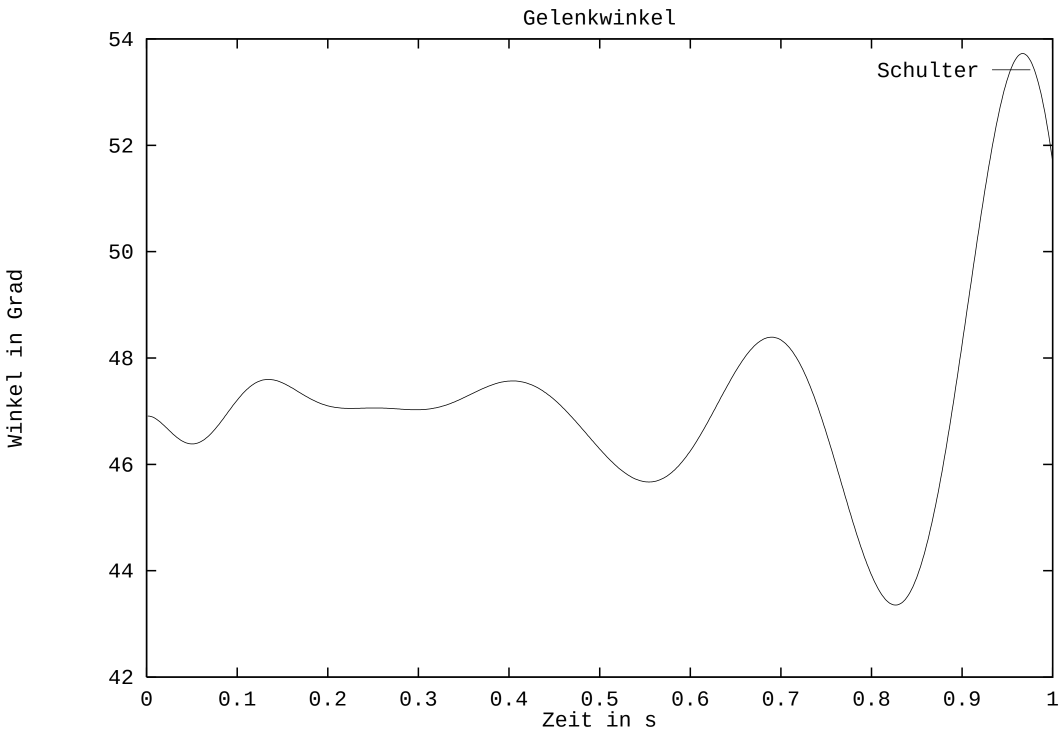

Abbildung 16: Schulterwinkel bei $P = 20,\ I = 400,\ D = 20$

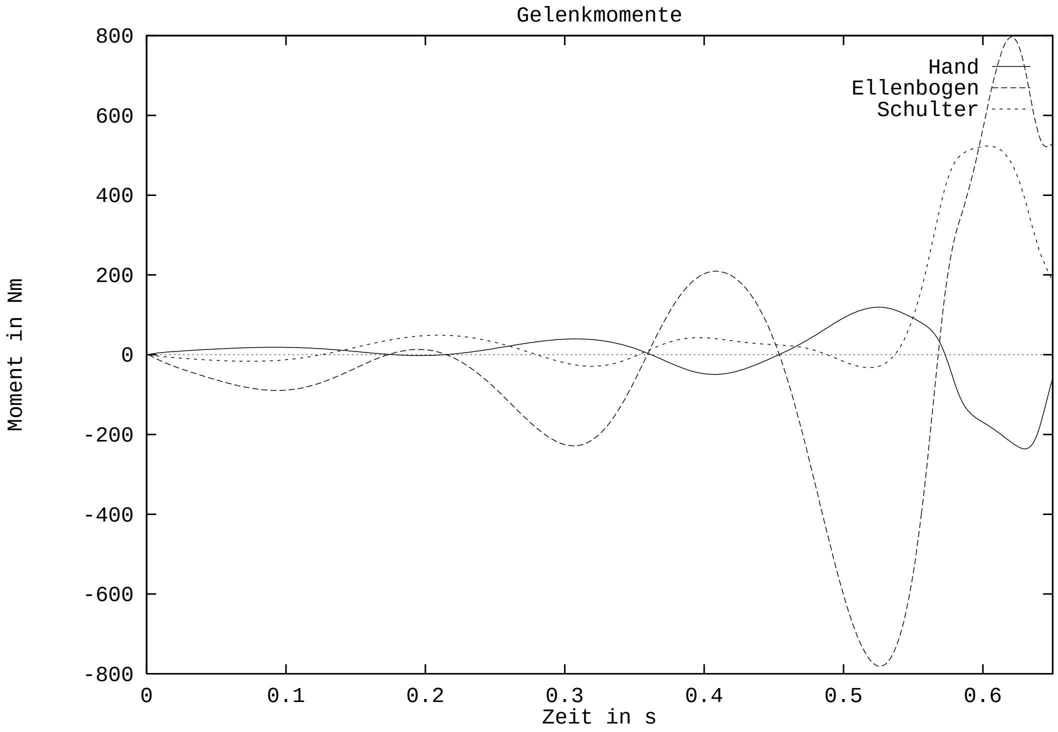

Abbildung 17: Gelenkmomente bei $P = 2$, $I = 400$, $D = 20$

Die mit den Einstellungen zu Abb. 17 durchgeführte Simulation ist nach 0.65 Sekunden wegen mehrfachen Toleranzüberschreitungen abgebrochen worden. Die Bewegung des Armes ist danach völlig aus der Kontrolle geraten. Es wird deutlich, daß die Gelenkmomente Nullpunktsschwingungen ausführen, die sich gegenseitig aufschaukeln. Die Einstellungens scheinen daher überhaupt nicht geeignet.

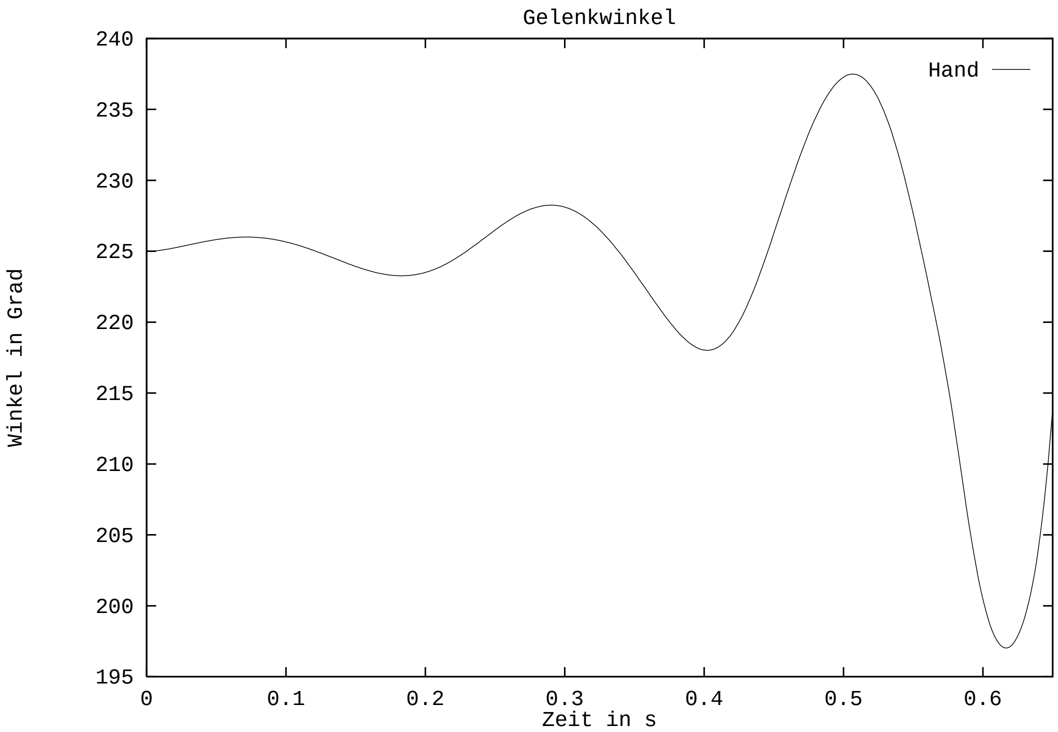

Abbildung 18: Handgelenkwinkel bei $P = 2$, $I = 400$, $D = 20$

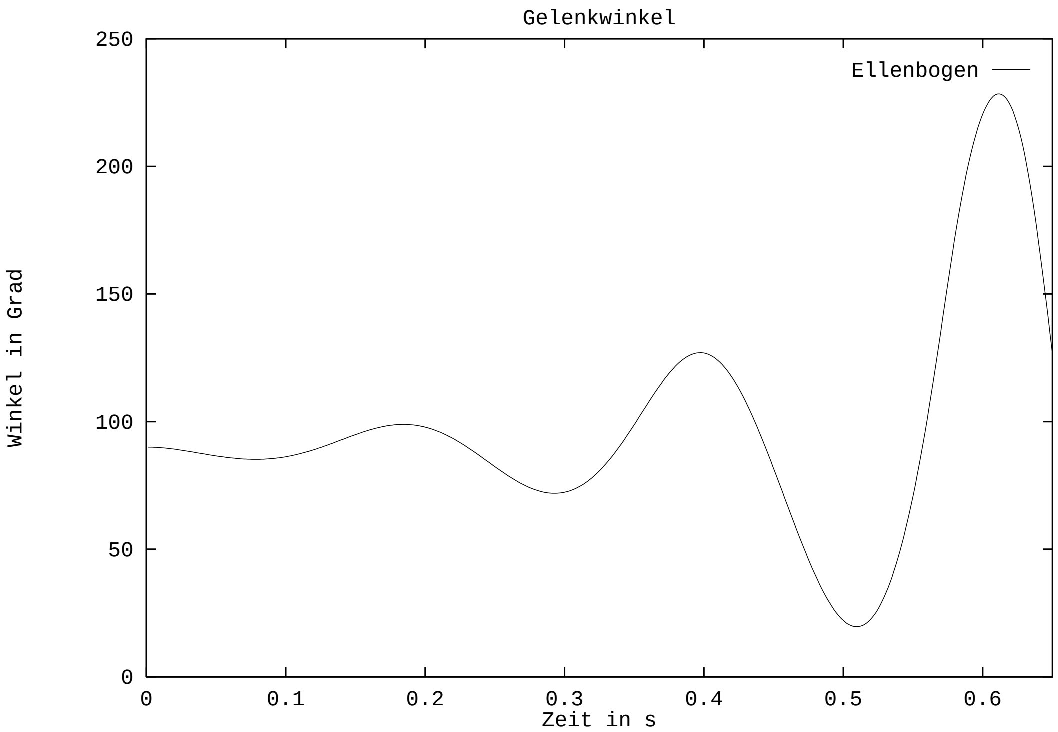

Abbildung 19: Ellenbogenwinkel bei $P = 2$, $I = 400$, $D = 20$

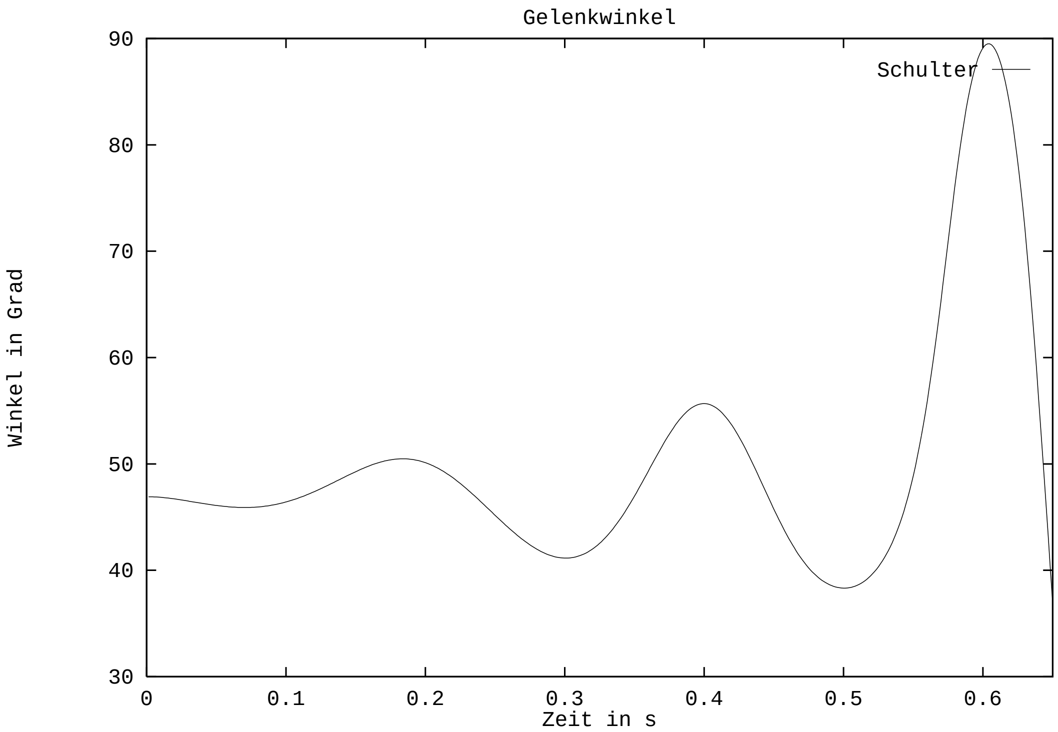

Abbildung 20: Schulterwinkel bei $P = 2$, $I = 400$, $D = 20$

Bei der in Abb. 21 dargestellten Simulation besteht dasselbe Problem wie in der vorangegangenen Einstellung. Die Gelenkmomente führen Nullpunktsschwingungen aus, die sich gegenseitig aufschaukeln. Auch diese Einstellungen sind daher ungeeignet.

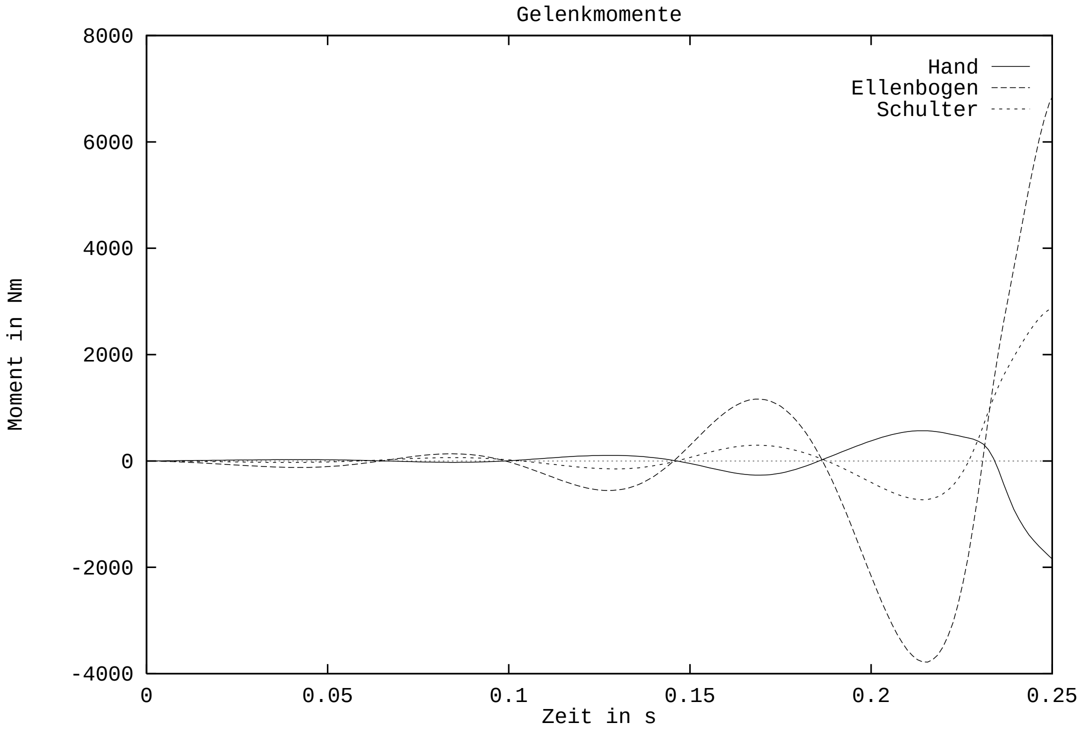

Abbildung 21: Gelenkmomente bei $P = 200$, $I = 4000$, $D = 20$

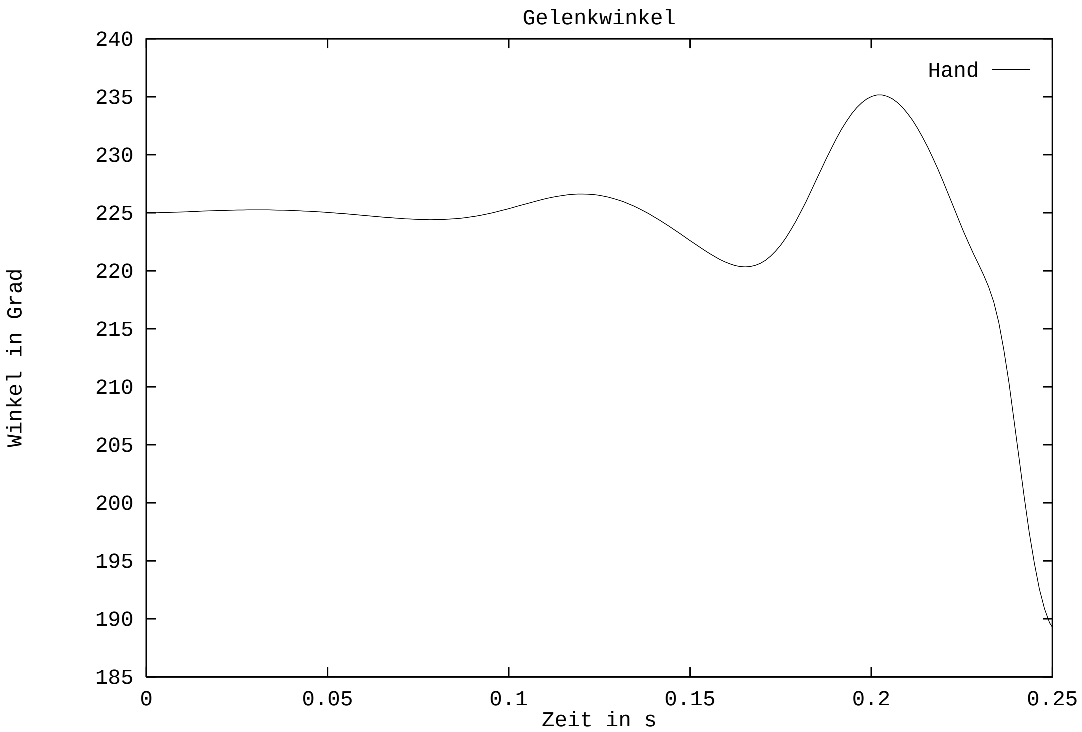

Abbildung 22: Handgelenkwinkel bei $P = 200$, $I = 4000$, $D = 20$

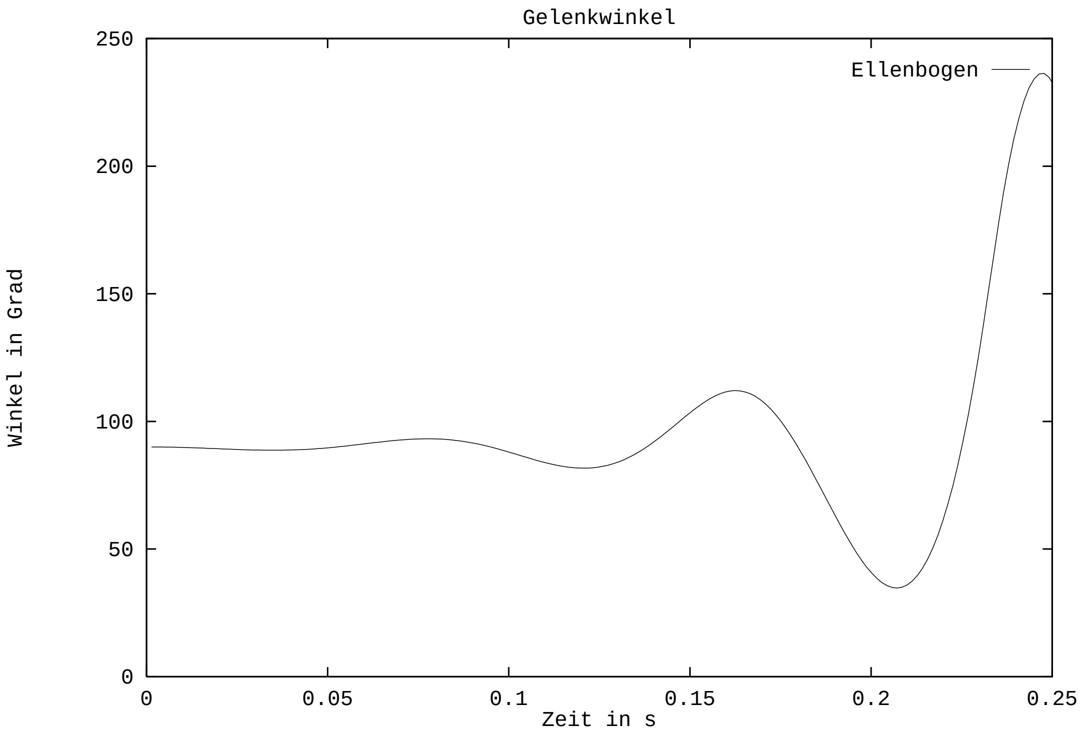

Abbildung 23: Ellenbogenwinkel bei $P = 200,\ I = 4000,\ D = 20$

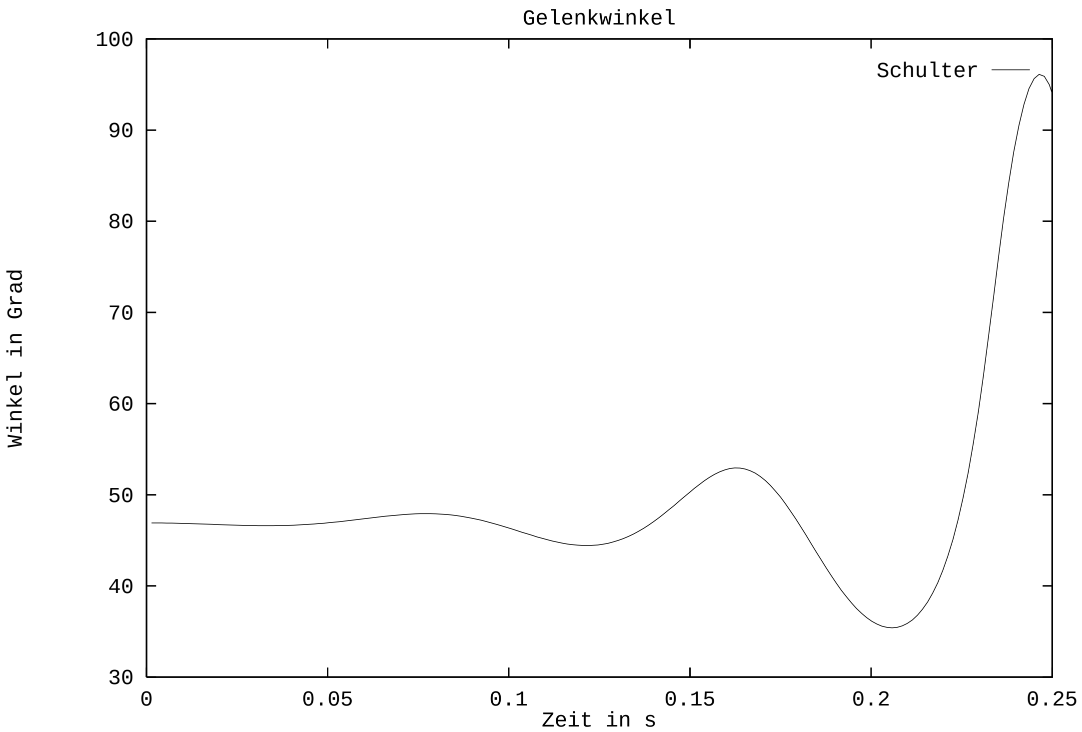

Abbildung 24: Schultergelenkwinkel bei $P = 200,\ I = 4000,\ D = 20$

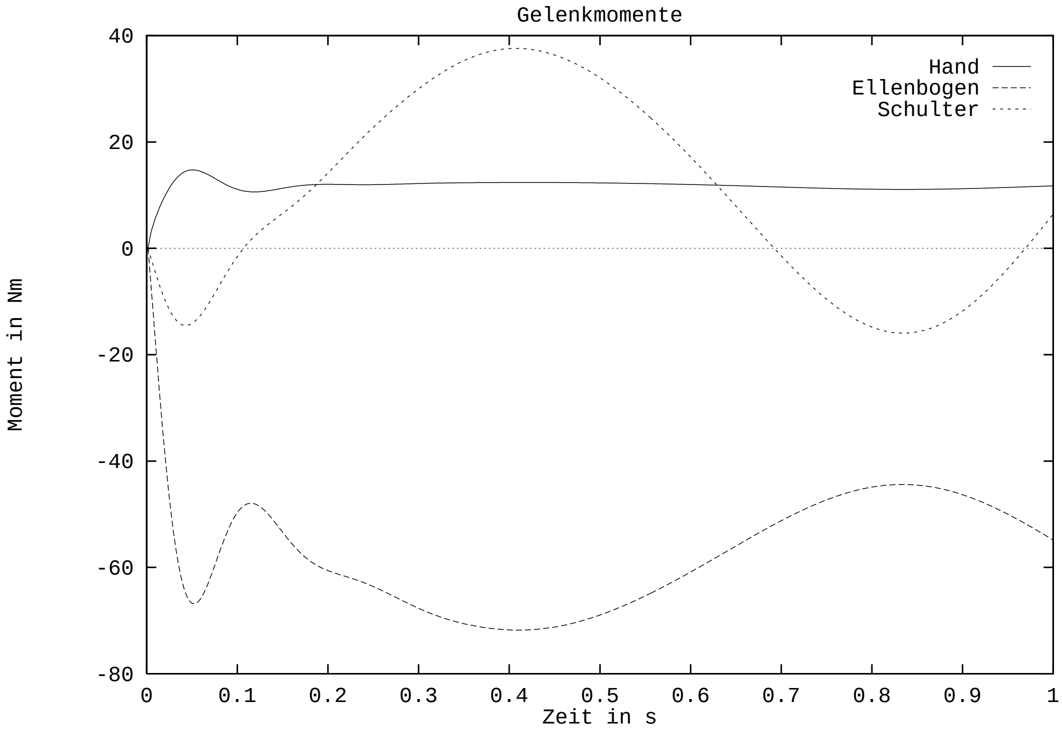

Abbildung 25: Gelenkmomente bei $P = 200$, $I = 40$, $D = 20$

Obwohl die in Abb. 25 verwendeten Parameter Momentenverläufe ergeben, die sowohl vom Momentenaufbau als auch von der Größe der Momente das gewünschte Ergebnis liefern, bleibt anzuzweifeln, ob sich die Schwingungen des Schultermoments auch bei längerer Simulationszeit wieder beruhigt hätten. Wie sich aus den Abb. 27 und 28 ersehen läßt, scheinen die Schwingungen des Schultergelenks das Ellenbogengelenk aufzuschwingen. Auch hier wäre daher eine längere Integrationszeit von Interesse. Die Kurven zur Simulation aus Abb. 9 und die hier vorliegenden Kurven unterscheiden sich nur durch einen überlagerten, höherfrequenten Teil, der offensichtlich auf den dort höheren Integralteil zurückgeht. Die grundsätzliche Kurvenform ist identisch. Durch den kleineren Integralteil ist also der Einschwingvorgang erheblich verkürzt worden.

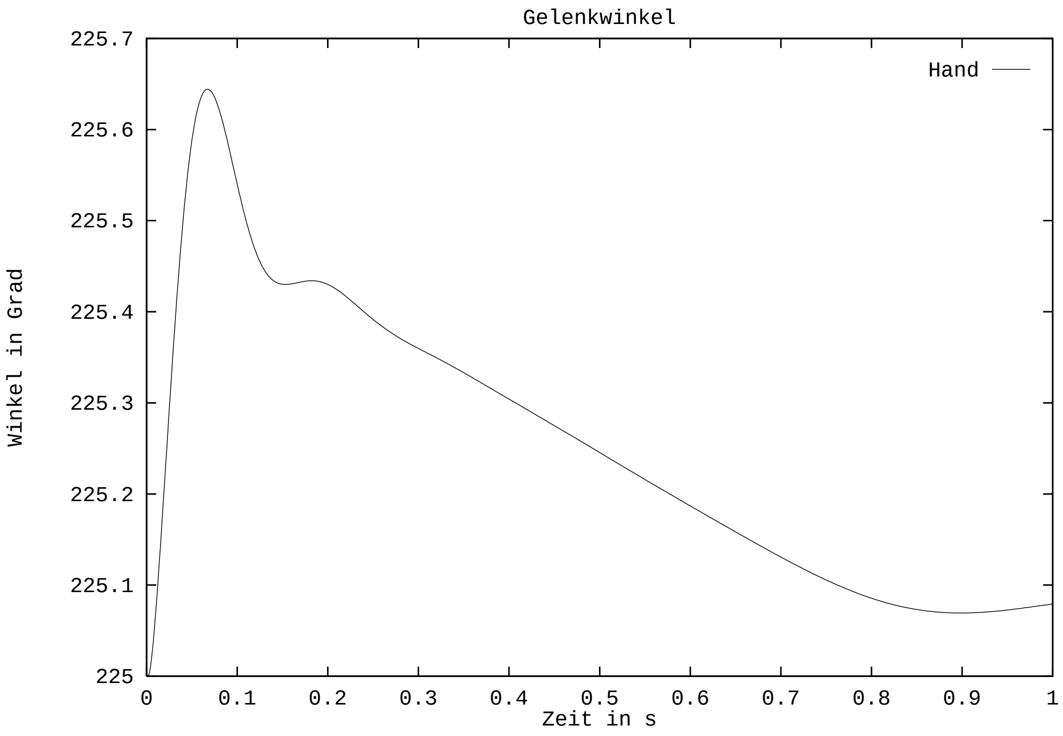

Abbildung 26: Handgelenkwinkel bei $P = 200$, $I = 40$, $D = 20$

Die Einstellungen $P = 200$, $I = 400$, $D = 200$ führten sofort zu steifen Gleichungen und die Simulation wurde deshalb ohne Ergebnis abgebrochen.

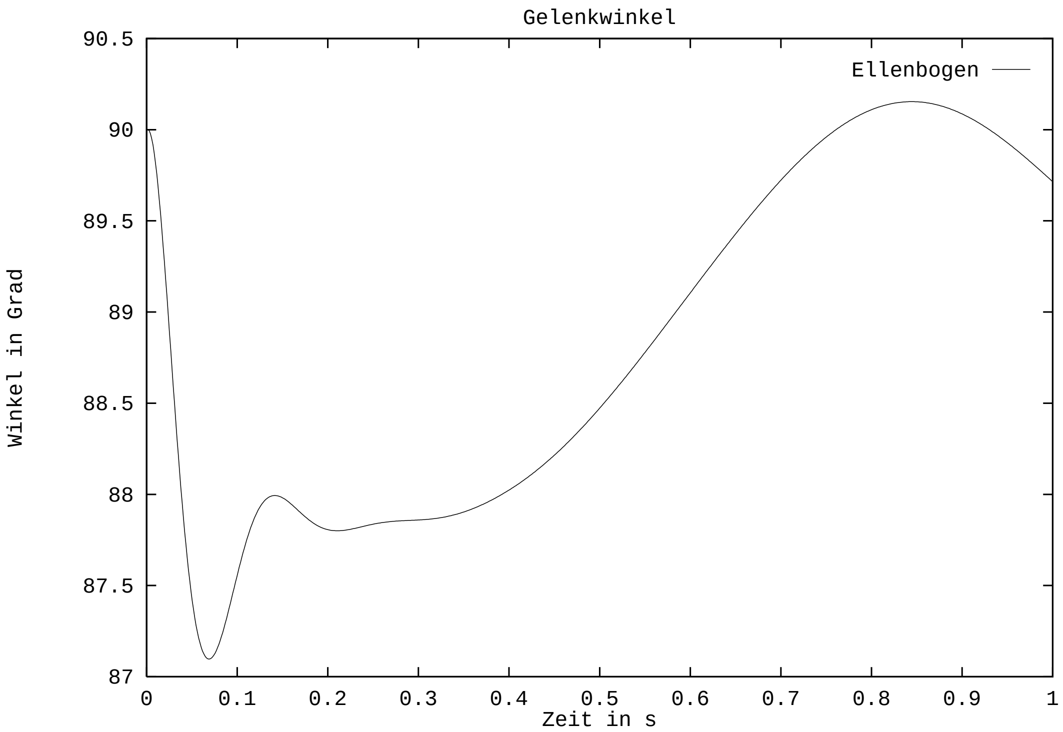

Abbildung 27: Ellenbogenwinkel bei $P = 200,\ I = 40,\ D = 20$

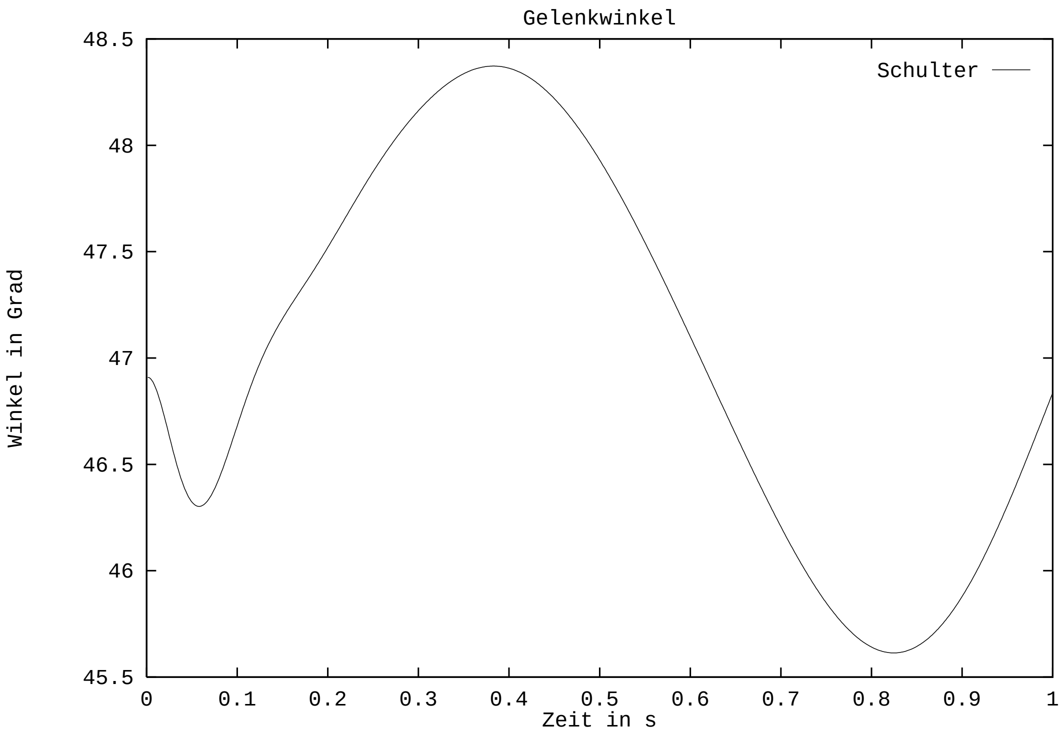

Abbildung 28: Schulterwinkel bei $P = 200,\ I = 40,\ D = 20$

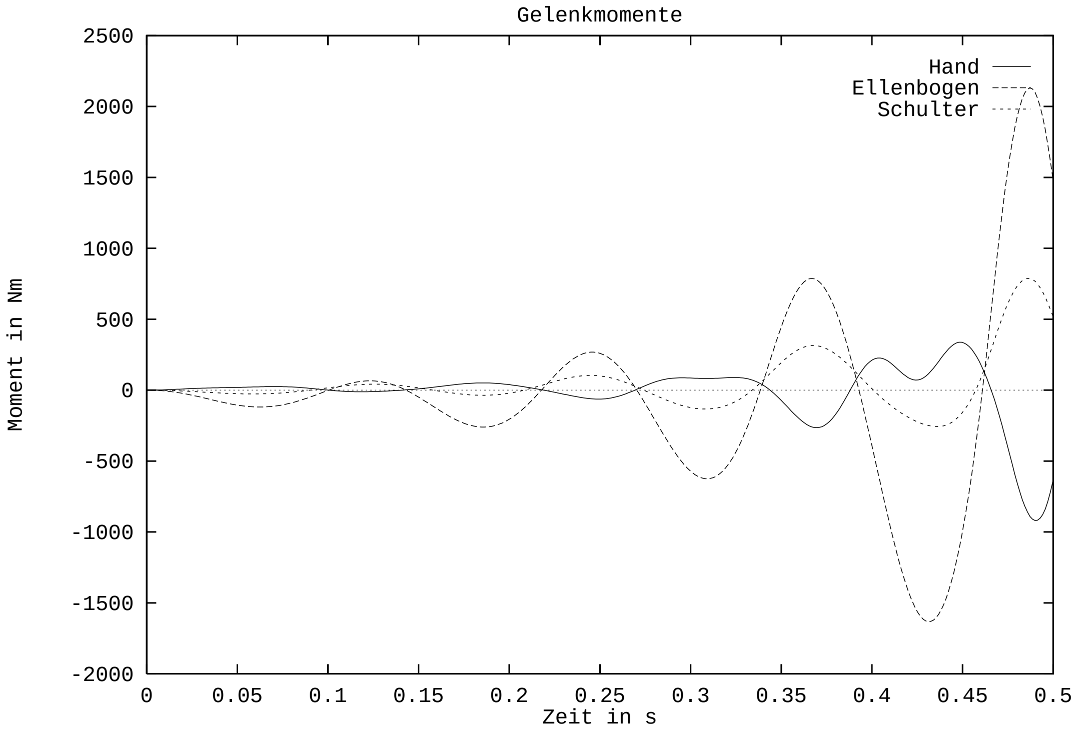

Abbildung 29: Gelenkmomente bei $P = 200$, $I = 400$, $D = 2$

Die Einstellungen aus Abb. 29 haben denselben Fehler wie einige Einstellungen zuvor: Nullpunktsschwingungen schaukeln sich auf und das System gerät außer Kontrolle. Die Einstellungen sind daher ebenfalls nicht brauchbar.

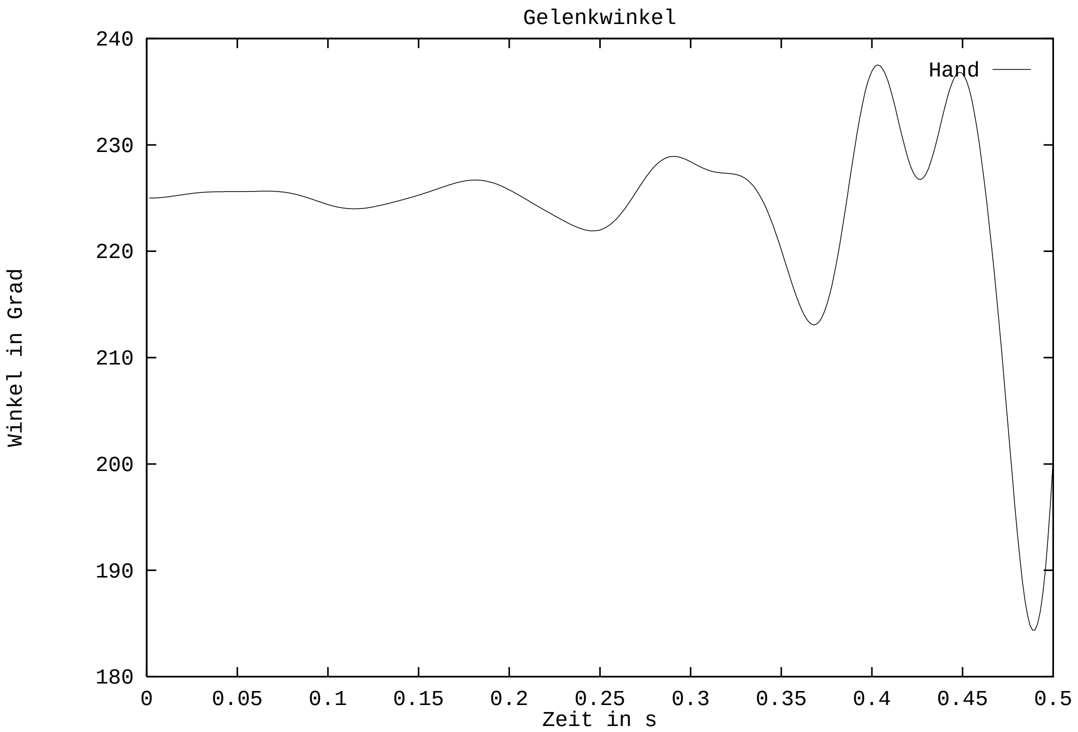

Abbildung 30: Handgelenkwinkel bei $P = 200,\ I = 400,\ D = 2$

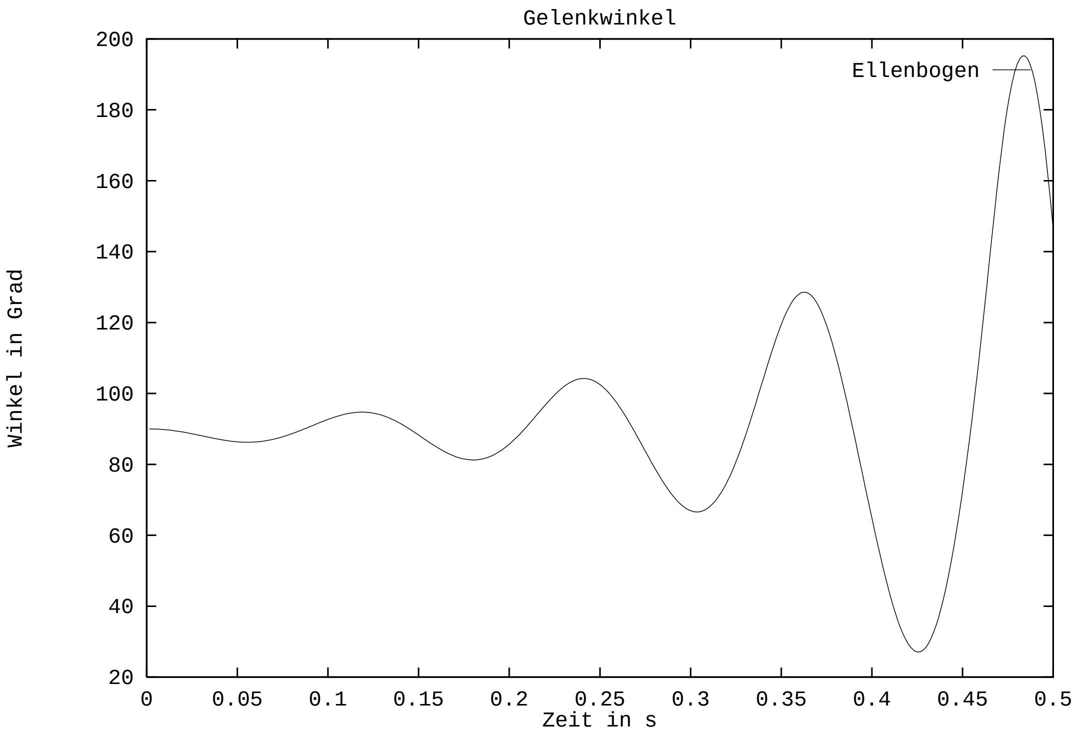

Abbildung 31: Ellenbogenwinkel bei $P = 200,\ I = 400,\ D = 2$

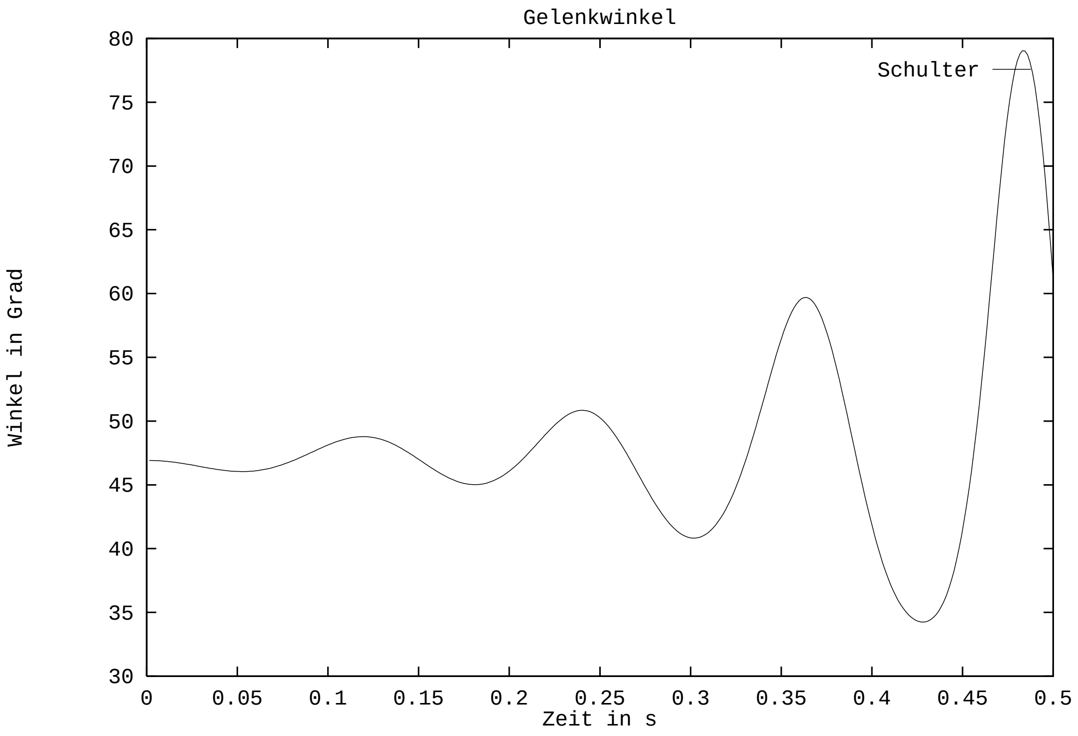

Abbildung 32: Schulterwinkel bei $P = 200,\ I = 400,\ D = 2$

5.5 Erreichter Stand

Wie aus den vorstehenden Abbildungen ersichtlich ist, ist es ein sehr mühsames Unterfangen, einen so großen Parameterraum nach zufällig geeigneten Lösungen zu untersuchen. Die Ergebnisse der durchgeführten Simulationen sind dahingehend unbefriedigend, als sie keine Erkenntnis über die zur Steuerung notwendigen Maßnahmen in Abhängigkeit der Systemparameter liefern und auch keine sichere Methode zu irgendeiner (unverstandenen aber funktionierenden) Lösung darstellen. Es fehlen ferner experimentell ermittelte Daten desselben Versuchs, die als Anhaltspunkt dienen könnten.

Das in dieser Arbeit vorgestellte Werkzeug zur Erstellung von Simulationen hat allerdings seine Aufgabe dahingehend erfüllt, als das Experimentieren und Einbringen neuer Ideen dadurch erheblich vereinfacht wird. Anstatt sich auf dem Weg zur Simulation bereits zu verzetteln, kann man die Zeit statt dessen in die eigentliche Problematik investieren. Dies ändert jedoch nichts an der Tatsache, daß *simsys* als (nützliches) Werkzeug nur ein Wegbereiter der Lösung, nicht aber die Lösung selbst ist.

Das Problem, aktive Systemkomponenten in den Griff zu bekommen, muß daher noch eingehender studiert werden. Hierzu bieten sich folgende Möglichkeiten an:

- Ankopplung eines neuronalen Netzes, das die Steuerung durch Vorgabe einer Zielfunktion selbst erlernt. Beherrscht das System die Bewegung, ist allerdings noch immer nicht bekannt, wie einzelne Parameter in die Bewegung eingehen.

- Verwendung von fuzzy logic zur Steuerung.

- Experimentelle Untersuchungen an vergleichbaren Anordnungen in der Wirklichkeit sollen Aufschluß geben über Momentenverläufe bei der Bewegung einzelner Gliedmaßen. Man müßte durch ein Experiment das Antwortverhalten auf einen Kraftsprung (wie oben simuliert) der Gelenkwinkel und Gelenkmomente bestimmen und dann versuchen, die gemessenen Werte mit der Simulation zu reproduzieren.

- Verbesserung des PID-Reglers durch Einführung zeitabhängiger Regelgrößen, womit Bewegungsabläufe möglich werden.

A Programmbeschreibungen

A.1 Bewegungsgleichungsgenerator *bgg*

Die Beschreibung findet sich nach dem Literaturverzeichnis.

A.2 Simulationsprogramm *simsys*

Die Beschreibung findet sich nach dem Literaturverzeichnis.

A.3 Anzeigeprogramm *simxwin*

Die Beschreibung findet sich nach dem Literaturverzeichnis.

A.4 Umsetzprogramm *blowup* und Ausgabedateien

Die Beschreibung findet sich nach dem Literaturverzeichnis.

A.5 Makefiles

```
#**************************************************************************
#
#    nova.mak
#    @(#) SID 1.4 2/19/92
#
#    Lehr- und Forschungsbereich Theoretische Astrophysik Tuebingen
#    Biomechanik
#
#    Hostabhaengiges Make Include File. Entahelt die hostabhaengigen Pfade
#    und Bezeichnungen.
#
#-------- Revision History (neuester Eintrag zuerst) -------------------------
#
#    Datum       Autor   Bemerkung
#
#    12.12.91    kri     Kodierung
#
#****************************************************************************/

SIMSYS=$(HOME)/simsys
BIN=$(SIMSYS)/bin
LIB=$(SIMSYS)/lib

CC=/usr/bin/cc
CFLAGS= -g -I$(SIMSYS)/include -I/usr/include -I/usr/openwin/share/include
LDFLAGS= -L/usr/lib -L/usr/local/lib -L$(SIMSYS)/lib

ARCH=ar rcv
RANLIB= echo

COMPILE.c=$(CC) $(CFLAGS) -c
LINK.c=$(CC) $(CFLAGS) $(LDFLAGS) -o

FC=/usr/bin/f77
COMPILE.f=$(FC) -static $(CFLAGS) -c

LIBSX= -lXt -lXext -lX11
LIBS= -lsimsys -llinpack -lF77 -lm
```

```
#***************************************************************************
#
#    makefile
#    @(#) SID 1.1 3/18/92
#
#    Lehr- und Forschungsbereich Theoretische Astrophysik Tuebingen
#    Biomechanik
#
#
#-------- Revision History (neuester Eintrag zuerst) ------------------------
#
#    Datum       Autor   Bemerkung
#
#    05.02.91    kri     Kodierung
#
#**************************************************************************/
MAKEDIR=../../make
BINDIR=../../bin
include $(MAKEDIR)/$(HOST).mak

BGG=$(BINDIR)/bgg
DEPEND_ALL= makefile $(MAKEDIR)/$(HOST).mak

EXE1=simsys
OBJ1=$(EXE1).o
SRC1=$(EXE1).c
CTRL1=$(EXE1).dat


all: $(EXE1)

new: clean all

clean:
# \rm -f *.o $(EXE1)
\rm -f *.o $(SRC1) $(EXE1)

$(EXE1): $(CTRL1) $(SRC1) $(OBJ1) $(DEPEND_ALL)
$(LINK.c) $@ $(OBJ1) $(LIBS)

$(SRC1): $(CTRL1) $(BGG) simuser.hc gelenke.dat kraefte.dat aussmom.dat muskeln.dat momente.dat
$(BGG) $(CTRL1) $@

.c.o: $(DEPEND_ALL)
$(COMPILE.c) $<

.f.o:
$(COMPILE.f) $<
```

A.6 Generierter Quellcode des Beispiels

```
/* Hebel pro Koerper: 5 */
/* Koerperzahl: 9 */
/* Gelenkzahl: 3 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <simsys.h>


static tKoMatrix    a[33][33]; /* Koeffizientenmatrix des Gleichungssystems in den Beschleunigungen */
static double       b[33],     /* Inhomogener Gleichungsanteil */
                    ac[27],    /* Beschleunigungen */
                    mt[9],    /* Momentenarray: geht in die Inhomogenitaet ein */
                    gx, gz;    /* Gravitationsbeschleunigungen in x und z Richtung */
static tVektor      kt[9],    /* Kraftvektor: geht in die Inhomogenitaet ein */
```

```
                    zk[3];     /* Zwangskraft: Erhaelt man nach Loesung des LGS */
static HDLBoden hBoden[45];
static const double *m,        /* Referenz der Koerpermassen */
                    *t,        /* Referenz der Traegheitsmomente */
                    *sb,       /* Referenz der Sinusse der Drehwinkel */
                    *cb;       /* Referenz der Cosinusse der Drehwinkel */
static const tHebel *ri,       /* Referenz der raumfesten Hebelarme */
                    *rv,       /* Referenz der raumfesten Hebelgeschwindigkeiten */
                    *r;        /* Referenz der koerperfesten Hebelarme */
static int     GelAnschlOrdnung,              /* Ordnung der Reihe f. Momente */
               VarZahl,                       /* Anz. benutzerdef. Integvars */
               FedZahl;                       /* Anzahl simulierter Federn */
static const intd   *GelNachbar;
static const int    *GelWinVorz,
                    KoeZahl = 9,
                    HebPKoe = 5,
                    *KoeHebel;
static const double *GelWinkel,
                    *GelWinGeschw,
                    *KoefWin;
static const HDLGelAnschlag *GelAnschlag;
static const tFeder *Feder;
static void SetzeFederKraefte();
static void SetzeBodenKraefte();
static void SetzeAnschlagsMomente();
static void SetzeKraftMoment();
static void SetzeGelenkMoment();


static double BohrKraft;
static double BohrFrequenz;
static double VortriebKraft;
static double RumpfMoment;
static double HandgelM;
static double HandgelP;
static double HandgelI;
static double HandgelD;
static double EllengelM;
static double EllengelP;
static double EllengelI;
static double EllengelD;
static double SchulgelM;
static double SchulgelP;
static double SchulgelI;
static double SchulgelD;
const double *GetKoMa()    {return (&a[0][0]);}  /* GetKoMa()   */
const double *GetInho()    {return (b);}         /* GetInho()   */
const double *GetMomt()    {return (mt);}        /* GetMomt()   */
const tVektor *GetZwKr()   {return (zk);}        /* GetZwKr()   */
const double *GetBeschl()  {return (ac);}        /* GetBeschl() */
const tVektor *GetKrft()   {return (kt);}        /* GetKrft()   */
int GetKoeZ()              {return (9);}        /* GetKoeZ()   */
int GetHebZ()              {return (5);}        /* GetHebZ()   */
int GetGelZ()              {return (3);}        /* GetGelZ()   */
const HDLBoden *GethBoden(){return (hBoden);}    /* GethBoden() */


#include "simuser.hc"
void KoMaCreate (EnvParmDatei)
const char *EnvParmDatei;
{
double ax, bx, az, bz, cbr, BodenhoeheZ;
int    ii, OrdElast, OrdDissi;
    BohrKraft = EnvGetDouble (EnvParmDatei, "BohrKraft", 0.0);
    BohrFrequenz = EnvGetDouble (EnvParmDatei, "BohrFrequenz", 5.0);
    VortriebKraft = EnvGetDouble (EnvParmDatei, "VortriebKraft", 0.0);
    RumpfMoment = EnvGetDouble (EnvParmDatei, "RumpfMoment", 100);
    HandgelM = EnvGetDouble (EnvParmDatei, "HandgelM", 0.0);
    HandgelP = EnvGetDouble (EnvParmDatei, "HandgelP", 0.0);
    HandgelI = EnvGetDouble (EnvParmDatei, "HandgelI", 0.0);
```

```
    HandgelD = EnvGetDouble (EnvParmDatei, "HandgelD", 0.0);
    EllengelM = EnvGetDouble (EnvParmDatei, "EllengelM", 0.0);
    EllengelP = EnvGetDouble (EnvParmDatei, "EllengelP", 0.0);
    EllengelI = EnvGetDouble (EnvParmDatei, "EllengelI", 0.0);
    EllengelD = EnvGetDouble (EnvParmDatei, "EllengelD", 0.0);
    SchulgelM = EnvGetDouble (EnvParmDatei, "SchulgelM", 0.0);
    SchulgelP = EnvGetDouble (EnvParmDatei, "SchulgelP", 0.0);
    SchulgelI = EnvGetDouble (EnvParmDatei, "SchulgelI", 0.0);
    SchulgelD = EnvGetDouble (EnvParmDatei, "SchulgelD", 0.0);
    BodenhoeheZ = EnvGetDouble (EnvParmDatei, "BodenHoeheZ", 0.0);
    GelAnschlOrdnung = EnvGetInt (EnvParmDatei, "GelAnschlOrdnung", DEFAULT_GELANSCHLORDNUNG);
    ax = EnvGetDouble (EnvParmDatei, "BodKraftFaktX", 500.0);
    bx = EnvGetDouble (EnvParmDatei, "BodKraftFaktVX", 250.0);
    az = EnvGetDouble (EnvParmDatei, "BodKraftFaktZ", 500.0);
    bz = EnvGetDouble (EnvParmDatei, "BodKraftFaktVZ", 250.0);
    cbr = EnvGetDouble (EnvParmDatei, "BodKraftExpoV", 10000.0);
    OrdElast = EnvGetInt (EnvParmDatei, "BodKraftOrdElast", 3);
    OrdDissi = EnvGetInt (EnvParmDatei, "BodKraftOrdDissi", 3);
    gx = EnvGetDouble (EnvParmDatei, "GravitatX", 0.0);
    gz = EnvGetDouble (EnvParmDatei, "GravitatZ", 0.0);
    VarZahl = GetIntegVar();
    m = GetMasse();
    t = GetTrgMo();
    sb = GetSinPhi();
    cb = GetCosPhi();
    ri = GetRaufHbl();
    rv = GetRaufVHbl();
    r = GetKoefHbl();
    KoefWin = GetKoefWin();
    KoeHebel = GetKoeHebel();
    GelWinVorz = GetGelWinVorz();
    GelWinkel = GetGelWinkel();
    GelWinGeschw = GetGelWinGeschw();
    GelAnschlag = GetGelAnschlag();
    GelNachbar = GetGelNachbar();
    FedZahl = GetFedZahl();
    Feder = GetFedern();
    for (ii = 0; ii < 45; ++ii)
    {
        hBoden[ii].Touching = False;
        hBoden[ii].BodenhoeheZ = BodenhoeheZ;
        hBoden[ii].ax = ax;
        hBoden[ii].az = az;
        hBoden[ii].bx = bx;
        hBoden[ii].bz = bz;
        hBoden[ii].cb = cbr;
        hBoden[ii].OrdElast = OrdElast;
        hBoden[ii].OrdDissi = OrdDissi;
    }
} /* KoMaCreate() */


void KoMaDestroy()
{
    m = NULL;
    t = NULL;
    sb = NULL;
    cb = NULL;
    ri = NULL;
    rv = NULL;
    r = NULL;
    KoeHebel = NULL;
    GelNachbar = NULL;
    GelWinkel = NULL;
    GelAnschlag = NULL;
    Feder = NULL;
} /* KoMaDestroy() */


void KraftMoment (z, zt)
```

```
const tKoordSatz *z;
const double zt;
{
const double *var = (VarZahl > 0) ? &z[KoeZahl].z : NULL; /* Benutzerdef. Integvar */
double  Handgelenk;
double  Ellenbogengelenk;
double  Schultergelenk;
tVektor  BohrHammer;
tVektor  HandKontakt;
tVektor  Rumpf;
tVektor  BohrerKontakt;
double  RumpfMom;
int ii;
    for (ii = 0; ii < 9; ++ii)
    {
        kt [ii][0] = m [ii]*gx;
        kt [ii][1] = m [ii]*gz;
        mt [ii] = 0.0;
    }
    { /* Handgelenk (2) */
        int ii;
        if (zt == 0.0)
            for (ii = 0; ii < 3; ++ii)
                GelWinSoll[ii] = GelWinkel[ii];
        Handgelenk = GelMomPID (GelWinSoll[2], GelWinkel[2], GelWinGeschw[2],
                                var[2], HandgelP, HandgelI, HandgelD);
        HandgelM = Handgelenk;
        SetzeGelenkMoment (2, Handgelenk);
    }

    { /* Ellenbogengelenk (1) */
        Ellenbogengelenk = GelMomPID (GelWinSoll[1], GelWinkel[1], GelWinGeschw[1],
                                      var[1], EllengelP, EllengelI, EllengelD);
        EllengelM = Ellenbogengelenk;
        SetzeGelenkMoment (1, Ellenbogengelenk);
    }

    { /* Schultergelenk (0) */
        Schultergelenk = GelMomPID (GelWinSoll[0], GelWinkel[0], GelWinGeschw[0],
                                    var[0], SchulgelP, SchulgelI, SchulgelD);
        SchulgelM = Schultergelenk;
        SetzeGelenkMoment (0, Schultergelenk);
    }

    { /* BohrHammer (4:1) */
    double    RiX = z[4].x + ri[21].hx,
              RiZ = z[4].z + ri[21].hz,
              RvX = z[4].vx + rv[21].hx,
              RvZ = z[4].vz + rv[21].hz;
        double arg;

        arg = sin (BohrFrequenz * 2 * PI * zt);
        BohrHammer[0] = - BohrKraft * 0.5; /* * arg * arg; */
        BohrHammer[1] = 0.0;
        SetzeKraftMoment (4, 1, BohrHammer[0], BohrHammer[1]);
    }

    { /* HandKontakt (3:2) */
    double    RiX = z[3].x + ri[17].hx,
              RiZ = z[3].z + ri[17].hz,
              RvX = z[3].vx + rv[17].hx,
              RvZ = z[3].vz + rv[17].hz;
        HandKontakt[0] = - (BohrKraft * 0.5 + VortriebKraft);
        HandKontakt[1] = 0.0;
        SetzeKraftMoment (3, 2, HandKontakt[0], HandKontakt[1]);
    }

    { /* Rumpf (0:4) */
    double    RiX = z[0].x + ri[4].hx,
              RiZ = z[0].z + ri[4].hz,
```

```
                RvX = z[0].vx + rv[4].hx,
                RvZ = z[0].vz + rv[4].hz;
        Rumpf[0] = 0.5 * BohrKraft + VortriebKraft;
        Rumpf[1] = 0.0;
        SetzeKraftMoment (0, 4, Rumpf[0], Rumpf[1]);
    }

    { /* BohrerKontakt (4:0) */
    double      RiX = z[4].x + ri[20].hx,
                RiZ = z[4].z + ri[20].hz,
                RvX = z[4].vx + rv[20].hx,
                RvZ = z[4].vz + rv[20].hz;
        BohrerKontakt [0] = - HandKontakt[0];
        BohrerKontakt [1] = 0.0; /* Fuehrungskraefte der Wand */
        SetzeKraftMoment (4, 0, BohrerKontakt[0], BohrerKontakt[1]);
    }

    { /* RumpfMom (0) */
        static double phi0;
        double hx, hz;
        if (zt == 0.0)
            phi0 = z[0].phi;
        HebelRaufKoord (&hx, &hz, 3, 2);
        RumpfMom = - RumpfMoment * (z[0].phi - phi0);
        mt [0] += RumpfMom;
    }

    SetzeBodenKraefte(z);
    if (FedZahl)
        SetzeFederKraefte(z);
    if (GelAnschlOrdnung)
        SetzeAnschlagsMomente(z);
} /* KraftMoment() */
void SetKoMa (z)
const tKoordSatz *z;

{
int   ii, jj;

    for (ii = 0; ii < 33; ++ii)
    {
        for (jj = 0; jj < 33; ++jj)
            a[ii][jj] = 0;
        b[ii] = 0;
    }
    a[0][0] = m [0];
    a[1][1] = m [0];
    a[2][2] = t [0];

    a[3][3] = m [1];
    a[4][4] = m [1];
    a[5][5] = t [1];

    a[6][6] = m [2];
    a[7][7] = m [2];
    a[8][8] = t [2];

    a[9][9] = m [3];
    a[10][10] = m [3];
    a[11][11] = t [3];

    a[12][12] = m [4];
    a[13][13] = m [4];
    a[14][14] = t [4];

    a[15][15] = m [5];
    a[16][16] = m [5];
    a[17][17] = t [5];

    a[18][18] = m [6];
```

```
    a[19][19] = m [6];
    a[20][20] = t [6];

    a[21][21] = m [7];
    a[22][22] = m [7];
    a[23][23] = t [7];

    a[24][24] = m [8];
    a[25][25] = m [8];
    a[26][26] = t [8];

    a [0][27] = 1;
    a [2][27] = -ri[0].hz;
    a [3][27] = -1;
    a [5][27] = ri[5].hz;

    a [27][0] = -1;
    a [27][2] = ri[0].hz;
    a [27][3] = 1;
    a [27][5] = -ri[5].hz;

    a [1][28] = 1;
    a [2][28] = ri[0].hx;
    a [4][28] = -1;
    a [5][28] = -ri[5].hx;

    a [28][1] = -1;
    a [28][2] = -ri[0].hx;
    a [28][4] = 1;
    a [28][5] = ri[5].hx;
    a [3][29] = 1;
    a [5][29] = -ri[6].hz;
    a [6][29] = -1;
    a [8][29] = ri[10].hz;

    a [29][3] = -1;
    a [29][5] = ri[6].hz;
    a [29][6] = 1;
    a [29][8] = -ri[10].hz;

    a [4][30] = 1;
    a [5][30] = ri[6].hx;
    a [7][30] = -1;
    a [8][30] = -ri[10].hx;

    a [30][4] = -1;
    a [30][5] = -ri[6].hx;
    a [30][7] = 1;
    a [30][8] = ri[10].hx;
    a [6][31] = 1;
    a [8][31] = -ri[11].hz;
    a [9][31] = -1;
    a [11][31] = ri[15].hz;

    a [31][6] = -1;
    a [31][8] = ri[11].hz;
    a [31][9] = 1;
    a [31][11] = -ri[15].hz;

    a [7][32] = 1;
    a [8][32] = ri[11].hx;
    a [10][32] = -1;
    a [11][32] = -ri[15].hx;

    a [32][7] = -1;
    a [32][8] = -ri[11].hx;
    a [32][10] = 1;
    a [32][11] = ri[15].hx;
} /* SetKoMa() */
```

```
void SetInho (z)
const tKoordSatz *z;
{
    b [0] = kt [0][0];
    b [1] = kt [0][1];
    b [2] = mt [0];
    b [3] = kt [1][0];
    b [4] = kt [1][1];
    b [5] = mt [1];
    b [6] = kt [2][0];
    b [7] = kt [2][1];
    b [8] = mt [2];
    b [9] = kt [3][0];
    b [10] = kt [3][1];
    b [11] = mt [3];
    b [12] = kt [4][0];
    b [13] = kt [4][1];
    b [14] = mt [4];
    b [15] = kt [5][0];
    b [16] = kt [5][1];
    b [17] = mt [5];
    b [18] = kt [6][0];
    b [19] = kt [6][1];
    b [20] = mt [6];
    b [21] = kt [7][0];
    b [22] = kt [7][1];
    b [23] = mt [7];
    b [24] = kt [8][0];
    b [25] = kt [8][1];
    b [26] = mt [8];
    b [27] = z[0].vphi * z[0].vphi * ri[0].hx
           - z[1].vphi * z[1].vphi * ri[5].hx;
    b [28] = z[0].vphi * z[0].vphi * ri[0].hz
           - z[1].vphi * z[1].vphi * ri[5].hz;
    b [29] = z[1].vphi * z[1].vphi * ri[6].hx
           - z[2].vphi * z[2].vphi * ri[10].hx;
    b [30] = z[1].vphi * z[1].vphi * ri[6].hz
           - z[2].vphi * z[2].vphi * ri[10].hz;
    b [31] = z[2].vphi * z[2].vphi * ri[11].hx
           - z[3].vphi * z[3].vphi * ri[15].hx;
    b [32] = z[2].vphi * z[2].vphi * ri[11].hz
           - z[3].vphi * z[3].vphi * ri[15].hz;
} /* SetInho() */


void   SetZStr (zs, z)
tKoordSatz             *zs;
const tKoordSatz   *z;
{
int ii;

    for (ii = 0; ii < 9; ++ii)
    {
        zs[ii].x = z[ii].vx;
        zs[ii].z = z[ii].vz;
        zs[ii].phi = z[ii].vphi;
        zs[ii].vx = b[ii * 3];
        zs[ii].vz = b[ii * 3 + 1];
        zs[ii].vphi = b[ii * 3 + 2];
    }
    zs [9].x = 0;
    for (ii = 0; ii < 9; ++ii)
        zs[9].x += mt[ii] + z[ii].x*kt[ii][1] - z[ii].z*kt[ii][0];
} /* SetZStr() */


void SetBeschl()
{
    memcpy (ac, b, 27*sizeof(double));
```

```
} /* SetBeschl() */
void SetZwKr()
{
int    ii, jj;

    for (ii = 0; ii < 3; ii++)
        for (jj = 0; jj < 2; ++jj)
            zk [ii][jj] = b [27 + ii*2 + jj];
} /* SetZwKr() */


static void SetzeKraftMoment (KoeNr, HebNr, KraftX, KraftZ)
const int    KoeNr,
             HebNr;
const double KraftX,
             KraftZ;
{
register int HNr = KoeNr*5 + HebNr;

    kt [KoeNr][0] += KraftX;
    kt [KoeNr][1] += KraftZ;
    mt [KoeNr] += ri [HNr].hx * KraftZ - KraftX * ri [HNr].hz;
} /* SetzeKraftMoment() */


static void SetzeAnschlagsMomente(z)
const tKoordSatz *z;
{
int ii;

    GelBerWinkel (z);
    for (ii = 0; ii < /*GelZahl*/3; ++ii)
    {
    double phi;

        if (GelAnschlag[ii].NichtBerechnen)
            continue;
        if ((phi = GelWinkel[ii] - GelAnschlag[ii].PhiOben) <= 0.0)
        {
            if ((phi = GelAnschlag[ii].PhiUnten - GelWinkel[ii]) <= 0.0)
                continue;
            else
                SetzeGelenkMoment (ii, (exp (phi) - expser (phi, GelAnschlOrdnung)) * GelAnschlag[ii].Unten);
        }
        else
            SetzeGelenkMoment (ii, (exp (phi) - expser (phi, GelAnschlOrdnung)) * GelAnschlag[ii].Oben);
    }
} /* SetzeAnschlagsMomente() */


static void SetzeGelenkMoment (GelNr, Moment)
const int    GelNr;
const double Moment;
{
    mt [GelNachbar [GelNr][0]] += ((double )GelWinVorz [GelNr])*Moment;
    mt [GelNachbar [GelNr][1]] -= ((double )GelWinVorz [GelNr])*Moment;
} /* SetzeGelenkMoment() */


static void SetzeBodenKraefte(z)
const tKoordSatz *z;
{
int kk, hh;

    for (kk = 0; kk < 9; ++kk)
        for (hh = 0; hh < KoeHebel[kk]; ++hh)
        {
        tVektor BodenKraftHebel;

            if (BerBodenKraft (&hBoden[kk*5 + hh], BodenKraftHebel, z, kk, hh))
```

```
                SetzeKraftMoment (kk, hh, BodenKraftHebel[0], BodenKraftHebel[1]);
        }
} /* SetzeBodenKraefte() */


static void SetzeFederKraefte (z)

const tKoordSatz *z;

{

int ii;


    for (ii = 0; ii < FedZahl; ii++)

    {

    tVektor FederKraft;


        BerFederKraft (FederKraft, z, &Feder[ii]);

        SetzeKraftMoment (Feder[ii].VonK, Feder[ii].VonH, FederKraft[0], FederKraft[1]);

        SetzeKraftMoment (Feder[ii].BisK, Feder[ii].BisH, - FederKraft[0], -FederKraft[1]);

    }

} /* SetzeFederKraefte() */
```

A.7 Parametrisierte Ausgabe, simuser.hc

Als Beispiel für die parametrisierte Ausgabe und der Vollständigkeit wegen wird der Quellcode der Datei simuser.hc hier abgedruckt.

```
/******* Modulverwaltung ****************************************************

    simuser.hc
    @(#) SID 1.1 3/18/92

    Lehr- und Forschungsbereich Theoretische Astrophysik Tuebingen
    Biomechanik

-------- Kurzbeschreibung des Modulinhalts ----------------------------------

    Setzen der Zeitableitungen der benutzerdefinierten Variablen.
    Datei wird in den vom Bewegungsgleichungsgenerator erzeugten Code
    mit #include "simuser.hc" eingebunden.

-------- Revision History (neuester Eintrag zuerst) -------------------------

    Datum       Autor   Bemerkung

    02.06.92    kri     Verbesserung der Ausgabe PID
    18.03.92    kri     Testende des PID
    09.03.92    kri     Kodierung

****************************************************************************/
#include <string.h>
#define RKreuzF(rx,rz,fx,fz) ((rx)*(fz)-(fx)*(rz))
static double GelMomPID (double Soll,    /* einzuregelnder Sollwert */
                         double Ist,     /* Istwert */
 double IstP,    /* Zeitableitung des Istwertes */
 double IstI,    /* Integral der Differenz Istwert - Sollwert */
 double Prop,    /* Proportionalstaerke */
 double Integ,   /* Staerke des Integralteils */
```

```
 double Diff);   /* Staerke des Differentialanteils */
static FILE *OpenFile (const char *ParmDatName, const char *Parm);
static double GelWinSoll[3];
static FILE *hFileGelMom,
    *hFileGelWin;
/*--- NAME BerechneVars() ---------------------------------------------------
    Berechnung der Zeitableitungen der benutzerdef. Variablen
----- SYNOPSIS -------------------------------------------------------------
#include "simsys.h"
----- PARAMETER ------------------------------------------------------------
varp     Hierhinein muessen die Zeitableitungen der Variablen.
var      Hier steht der Loesungsvektoren der Variablen zur Zeit zt.
zs       Vektor der berechneten Geschwindigkeiten und Beschleunigungen des
         Gesamtsystems. Die Anzahl der Koerper ist in der Variablen KoeZahl
 abgelegt.
z        Vektor z der Koordinaten und Geschw. des Gesamtsystems.
zt       Zeitkoordinate
IntegVar Anzahl benutzerdefinierter Variablen.
----- DESCRIPTION ----------------------------------------------------------
    Wie von der Integrationsroutine de_() verlangt, muessen hier die
    Zeitableitungen der benutzerdefinierten Integrationsvariablen
    auf den gewuenschten Wert gesetzt  werden.
--------------------------------------------------------------------------*/
void BerechneVars (varp, var, zs, z, zt, IntegVar)
double *varp;
const double *var;
const tKoordSatz *zs;
const tKoordSatz *z;
const double zt;
int IntegVar;
{
    int ii;
    for (ii = 0; ii < 3 && ii < IntegVar; ++ii)
varp[ii] = GelWinkel[ii] - GelWinSoll[ii];
} /* BerechneVars() */

/*--- NAME SimAusgCreate() --------------------------------------------------
    Initialisierung der benutzerprog. Ausgabe
----- SYNOPSIS -------------------------------------------------------------
#include "simsys.h"
----- PARAMETER ------------------------------------------------------------
ParmDatName Name der Parameterdatei, aus der die Parameter zu lesen sind.
----- DESCRIPTION ----------------------------------------------------------
    An dieser Stelle sollen die benutzerdefinierten Dateien zur Ausgabe
    geoeffnet werden.
--------------------------------------------------------------------------*/
void SimAusgCreate (ParmDatName)
const char *ParmDatName;
{
    hFileGelMom = OpenFile (ParmDatName, "AusgSimGelMom");
    hFileGelWin = OpenFile (ParmDatName, "AusgSimGelWin");
} /* SimAusgCreate() */

/*--- NAME SimAusgDestroy() -------------------------------------------------
    Schliessen aller Files der benutzerprog. Ausgabe
----- SYNOPSIS -------------------------------------------------------------
#include "simsys.h"
----- DESCRIPTION ----------------------------------------------------------
    Mit dem Aufruf sollte die Wirkung des Aufrufs von SimAusgCreate()
    rueckgaengig gemacht werden.
--------------------------------------------------------------------------*/
void SimAusgDestroy()
{
    if (hFileGelMom)
    {
        fclose (hFileGelMom);
        hFileGelMom = NULL;
    }
    if (hFileGelWin)
    {
```

```
        fclose (hFileGelWin);
        hFileGelWin = NULL;
    }
} /* SimAusgDestroy() */

/*--- NAME SimAusgabe() -----------------------------------------------------
    Benutzerprogrammierte Ausgabe
----- SYNOPSIS -------------------------------------------------------------
#include "simsys.h"
----- PARAMETER ------------------------------------------------------------
z   vollstaendige Loesung des Systems zur Zeit zt
zt  Endzeitpunkt des soeben angeschlossenen Integrationsintervalls
----- DESCRIPTION ----------------------------------------------------------
    In dieser Funktion kann beliebiger Code eingebaut werden, um eigene
    Groessen nach Wunsch in beliebigem Format auszugeben.
--------------------------------------------------------------------------*/
void SimAusgabe (z, zt)
const tKoordSatz *z;
const double     zt;
{
    if (hFileGelMom && zt > 0.0)
    {
        fprintf (hFileGelMom, "%f %f %f %f\n", zt, HandgelM, EllengelM, SchulgelM);
    }
    if (hFileGelWin && zt > 0.0)
    {
        fprintf (hFileGelWin, "%f %f %f %f %f %f %f\n", zt, GelWinkel[2], GelWinGeschw[2],
                                                           GelWinkel[1], GelWinGeschw[1],
                                                           GelWinkel[0], GelWinGeschw[0]);
    }
} /* SimAusgabe() */

static double GelMomPID (Soll, Ist, IstP, IstI, Prop, Integ, Diff)
double Soll, /* einzuregelnder Sollwert */
    Ist,     /* Istwert */
    IstP,    /* Zeitableitung des Istwertes */
    IstI,    /* Integral der Differenz Istwert - Sollwert */
    Prop,    /* Proportionalstaerke */
    Integ,   /* Staerke des Integralteils */
    Diff;    /* Staerke des Differentialanteils */
/* Wichtig hierbei ist: ein  positiver Returnwert muss verkleinernd auf
   die Istgroesse einwirken.
   Bei Gelenkwinkeln bewirkt ein positives Moment ein Zusammenziehen
   des Gelenkwinkels. Ist der Istwinkel > Sollwinkel --> ret > 0. Bei
   positiver Winkelgeschw. muss ebenfalls ein positives Moment erfolgen.
*/
{
    double ret;
    ret =  Prop * (Ist - Soll) + Diff * IstP + Integ * IstI;
    return (ret);
} /* GelMomPID() */

static FILE *OpenFile (ParmDatName, EnvVar)
const char  *ParmDatName,
            *EnvVar;
{
FILE  *ret = NULL;
char  Parm[180];

    if (EnvGet (Parm, ParmDatName, EnvVar, sizeof (Parm)) != NULL
    && strlen (Parm))
    {
        ret = fopen (Parm, "wt");
    }
    return (ret);
} /* OpenFile() */
```

B Aufbau des Systems

B.1 Grundgedanken

Folgende Grundgedanken sind mehr oder weniger konsistent bei der Umsetzung des Systems verfolgt worden:

- *Modularisierung:* Die Bibliothek besteht aus einigen Modulen, die jeweils etwa aus 500 - 1000 Zeilen Source bestehen. Die Aufteilung der Funktionen in die Module erfolgte nach Gesichtspunkten gemeinsamer Datenstrukturen und funktioneller Verwandtschaft. Die Namen aller zu *simsys* gehörenden Module beginnen mit `si...`, um die Zugehörigkeit kenntlich zu machen. Nach den ersten beiden Buchstaben folgt eine Bezeichnung, die die ungefähre Aufgabe des Moduls benennt.

- *Funktionen:* Die meisten Module, die nicht die Funktion `main()` enthalten, bestehen nach außen hin aus mindestens zwei Funktionen: `...Create()`, die sämtliche Intialisierungen vornimmt, und `...Destroy()`, die alle reservierten Speicher freimacht und evtl. offene Dateien schließt. In manchen Fällen existiert eine weitere Funktion, die z. B. bei der Ausgabe die eigentliche Tätigkeit des Moduls ausführt. Dies ist dann der Fall, wenn die Tätigkeit erst auf Abruf (ggf. mehrfach) und nicht bereits beim Initialisieren (nur einfach) ausgeführt werden darf.

- *Information Hiding:* Der Gültigkeitsbereich von Variablen und die Sichtbarkeit von Funktionen ist auf das notwendige Mindestmaß beschränkt. Es gibt keine programmglobalen Variablen. Alle Funktionen, die nur innerhalb eines Moduls aufgerufen werden, sind `static` deklariert und damit von keiner Funktion außerhalb des Moduls aufzurufen. Funktionen, die sich gemeinsame Variablen teilen bzw. diese gemeinsam bearbeiten, sind im selben Modul untergebracht, wenn die Variablen nicht als Funktionsparameter übergeben werden. Solche gemeinsamen Variablen sind im entsprechenden Modul als `static` definiert. Variablen innerhalb von Funktionen sind nur in den Blöcken definert und sichtbar, in denen sie auch gebraucht werden. Die Variablen, die im Kopf einer Funktion definiert sind, werden also in der ganzen Funktion verwendet. Funktionsparameter, die in einer Funktion nicht verändert werden, sind als `const` deklariert, um eine entsprechende Compilerprüfung (sofern er es kann) zu veranlassen.

- *Variablen:* Arrays von Daten werden nur in einem Modul gepflegt, d. h. erstellt und verändert. Dies dient der leichteren Durchführung konzeptioneller Änderungen. Funktionen außerhalb des Moduls erfahren die Adresse des Arrays über einen Funktionsaufruf `Get...()`, der einen `const *` zurückgibt. Damit können diese Funktionen die Daten zwar einsehen, nicht aber verändern. Die Stellen, an denen Datenänderungen stattfinden, sind dadurch im Programm leicht zu finden und überschaubar.

- *Revision History:* Die im Modulkopf angebrachten Einträge dienen der Übersicht über die am Modul durchgeführten Änderungen und sind wichtig bei Programmpflege.
- *Funktionsköpfe:* Die einheitlichen Funktionsköpfe ermöglichen die Extraktion des dort untergebrachten Kommentars und der Beschreibung mit *nawk*. Dient zur Erstellung von Dokumentation.

B.2 Übersicht über die Quelltextdateien

Folgende Biliotheksmodule sind Bestandteile von *libsimsys.a*:

`bgg.c`	`main()` des Bewegungsgleichungsgenerators.
`blowup.c`	`main()` des Umwandlungsprogramms.
`sianfbed.c`	Funktionen, die die Anfangsbedingungen aus den Blöcken `ANFBED` und `ANFWINKEL` lesen, deren Syntax prüfen und aus den eingelesenen, reduzierten Koordinaten vollständige Koordinaten errechnen. Verwaltung der Koordinaten.
`siausg.c`	Standardausgabe. Öffnet und schließt die entsprechenden Dateien und berechnet diverse Ausgabedaten. Von hier aus erfolgt der Aufruf der benutzerdefinierten Ausgaberoutinen.
`sienv.c`	Funktionen zum Lesen der Parameter aus beliebigen Dateien in Form der Parameterdatei.
`sifeder.c`	Einlesen, Interpretation und Verwaltung der entsprechenden Variablenarrays des Blocks `FEDERN`.
`sigelenk.c`	Einlesen, Interpretation und Verwaltung entsprechender Variablenarrays der Blöcke `GELENKE` und `GELWINKEL`.
`sihebel.c`	Einlesen, Interpretation und Verwaltung entsprechender Variablenarrays der Blöcke `HEBEL`. Ferner Verwaltung von Variablen, die die Sinusse und Cosinusse aller Winkel enthalten, Berechnung derselben und Berechnung der raumfesten Hebelkoordinaten.
`siinteg.c`	Aufruf der Integrationsroutine `de_()` sowie Behandlung der Integrationsfehler. Enthält ferner die Funktion, die die Stützstellenberechnung durchführt.
`silies.c`	Lesefunktionen zum Einlesen einer Zeile eines Datenblocks mit autom. Ausblendung von Kommentarzeilen. Aufbau einfacher Arrays aus eingelesenen Daten für Massen und Trägheitsmomente.
`simain.c`	`main()` des Simulationsprogramms.
`simastrg.c`	Einlesen, Interpretation und Verwaltung entsprechender Variablenarrays der Blöcke `TRAEGHEITSMOMENTE` und `MASSEN`.
`simath.c`	Funktionen zur Berechnung der Federkraft und der Bodenreaktionskraft.
`sixwin.c`	`main()` des Anzeigeprogramms *simxwin*.
`sleep.c`	Rechnerabhängige Verzögerungsroutine, die bei Aufruf den Programmlauf für die eingestellte Zeit unterbricht. Verwendung in *simxwin* zum Verlangsamen von schnellen Bewegungsabläufen.
`xmain.c`	Programm mit X–Window–Funktionen, die freundlicherweise von Birgitt Schönfisch zur Verfügung gestellt wurden.
`xwindow.c`	Testprogramm für das X–Window–Modul, es stammt ebenfalls von Birgitt Schönfisch.

Andere Module, die zur Erzeugung einer Simulation nötig sind:

je Modell:

`simsys.c`	Datei wird durch bgg erzeugt und enthält den Code zur Berechnung der Kräfte und Momente, sowie die Zuweisung der Koeffizientenmatrix und des inhomogenen Anteils des LGS in den Beschleunigungen.
`simsys.hc`	Enthält die Funktionen, die von Hand mit Code gefüllt werden müssen, um eigene Größen zu integrieren und individuelle Ausgabe zu veranlassen.

include:

`simsys.h`	Enthält alle Typvereinbarungen und sämtliche Funktionsprototypen.

B.3 Modularer Ablauf von *simsys*

Der Ablauf von simsys wird in folgender Abbildung erläutert:

Literatur

[Bronstein] Bronstein, Semendjajew: Taschenbuch der Mathematik
Verlag Harri Deutsch Thun, Frankfurt (Main), 1981, 21. Auflage

[Goldstein] Herbert Goldstein: Klassische Mechanik
AULA-Verlag Wiesbaden, 1987, 9. Auflage

[Gruber] Karin Gruber: Entwicklung eines Modells zur Berechnung der Kräfte im Knie- und Hüftgelenk bei sportlichen Bewegungsabläufen mit hohen Beschleunigungen.
Dissertation 1987, Univ. Tübingen, ETH Zürich

[Hospach] Frank Hospach: unveröffentlicht, Arbeiten zur Dissertation
Lehrstuhl für Theoretische Astrophysik Univ. Tübingen

[Jelitto] Rainer Jelitto: Theor. Physik 2: Mechanik II
AULA-Verlag Wiesbaden, 1987, 2. korr. Auflage

[K&R] Brian W. Kernighan, Dennis M. Ritchie: The C Programming Language
Second Edition, ANSI C
Prentice-Hall International, 1978, 1988

[Volz] Helmut Volz: Einführung in die Theor. Mechanik
Akademische Verlagsgesellschaft, 1971

[Widmayer] Karin Widmayer: Simulation von Bewegungsabläufen beim Menschen mit Hilfe von Mehrkörpersystemen
Diplomarbeit am Lehrstuhl für Theoretische Astrophysik Univ. Tübingen, 1990