Simulation biomechanischer Bewegungsabläufe, gesteuert durch neuronale Netze

Diplomarbeit

von

Helmut Mutschler

Lehr- und Forschungsbereich Theoretische Astrophysik der
Eberhard-Karls-Universität Tübingen

25. Februar 1999

Zusammenfassung

In der Biomechanik setzen sich neuronale Steuerungkonzepte immer mehr durch. In dieser Arbeit wird der grundsätzliche Aufbau einer solchen Simulation dargestellt. Entwickelt wurde ein Bewegungsgleichungs-Generator, mit dessen Hilfe eine 3-dimensionale Mehrkörpersimulation (*MKS*) eines vereinfachten Modells von einem menschlichen Arm durchgeführt wird. Die Ansteurung der Gelenkmomente des MKS-Systems wird mittels neuronaler Netze bewerkstelligt. Dieses Steuerungskonzept wird in einer Simulation des Weitwurfes getestet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Um die Aufgabe einer neuronal gesteuerten Bewegung zu lösen, bedarf es verschiedener Simulationsprogramme. Als erstes müssen die mechanischen Eigenschaften der verwendeten Körper simuliert werden, im folgenden benötigt man noch ein Simulationsprogramm für neuronale Netze. Es ist aber auch denkbar ein Simulationsprogramm zu verwenden, das beliebige physikalisch Simulationen erlaubt. Davon wurde abgesehen, da solch ein Programm (z.B. MATLAB[1]) jeweils eine Anpassung (Lib) für den physikalischen Sachverhalt benötigt, und sich die Eingabe für die Beschreibung des zu simulierenden Sachverhalts teilweise sehr schwierig gestaltet (für die Simulation von Bewegungen wird z.B. die Differentialgleichung der Bewegung erwartet).

- Mechanisches Simulationsprogramm :

  In [Sch93] wird eine Vielzahl von MKS-Simulationsprogrammen beschrieben, doch mangels freier Verfügbarkeit dieser mechanischen Simualtionsprogramme für 3-dimensionale Körper [2] wurde ein MKS-Simualtionsprogramm unter folgenden Gesichtspunkten entwickelt :

  Die Idee, ein Problem nicht als ganzes lösen zu wollen, sondern das Modell mittels kleinster Bausteine nachzumodellieren, ist nicht besonders neu, aber hier wird versucht, Bewegungsgleichungen für kleine Mehrkörpersysteme auf genau diese Art

[1] MATLAB ist eine kommerzielle, graphisch bediente Mathematik-Utility-Sammlung

[2] bei Beginn der Arbeit war DADS, ein kommerzielles Starrkörpersimulationsprogramm mit graphischer Bedieneroberfläche, nicht verfügbar

zu berechnen. Als Bausteine dienen die Translation und die Rotation. Modelliert werden kann eine mechanische Kette, Kräfte und Momente können auf die Kette wirken, Führungen können aber nicht modelliert werden (keine Zwangskräft). Das Programm wurde ausgelegt um 2, 3 und 4 - dimensionale Starrkörper zu simulieren. Der Rechenaufwand beschränkt das Programm (386 PC) auf ca. 7 Freiheitsgrade pro Kette, verteilt auf beliebig viele Körper. Das Ergebnis des Programms ist ein C-Programm, welches die Bewegungsgleichung integriert, und abschließend die Bewegung graphisch darstellt.

- Simulationsprogramm für neuronale Netze :

  Das Verfügbarkeitsproblem verhält sich hier genau umgekehrt, es ist eine Vielzahl von Simulationsprogrammen frei verfügbar [Zel94]. Die Wahl fiel auf *SNNS*, den "Stuttgarter Neuronale Netze Simulator". Das Programm wurde entwickelt am Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR), und ist auf diversen FTP-Servern verfügbar. Ausgewählt wurde es aufgrund seiner Leistungsfähigkeit, und der Möglichkeit, ein trainiertes neuronales Netz in Form eines C-Programms auszugeben.

- Kopplung beider Simulationsprogramme :

  Da beide Simulationsprogramme ihre Ergebnisse in Form eines C-Programms ausgeben, wird durch das Linken ein Programm erzeugt, das beide Simulationen in sich vereinigt. Die mechanische Simulation hat damit Zugriff auf die Funktionen, die das Simulationsprogramm für neuronale Netze zur Verfügung stellt.

Kapitel 2

Aufstellen von Bewegungsgleichungen für gekoppelte Starrkörper

Es gibt, abhängig von den Anforderungen die zu erfüllen sind, vielfältige Möglichkeiten Bewegungsgleichungen für mechanische Systeme zu erstellen. Deshalb erfolgt vorab eine kurzer Überblick über die geläufigsten Verfahren :

> Alle Beispiele beziehen sich auf den 3 dimensionalen Raum,
> n Körper sind gekoppelt,
> Das System besitzt f Freiheitsgrade

Jeder einzelne Starrkörper wird als *frei* betrachtet. Die Kopplung der Körper untereinander wird durch implizite algebraische Gleichungen (Bindungsgleichungen) festgelegt. Werden diese algebraischen Gleichungen durch zweimaliges Ableiten in die Differenzialgleichung eingefügt, entstehen f gekoppelte Dgl's 2. Ordnung. Dadurch ist das Starrkörpersystem in Minimalkoordinaten beschrieben. Behält man die algebraischen Gleichungen bei, dann wird es als "differential algebraisches Gleichungssystem" (*DAG*) bezeichnet

Sind die Minimalkoordinaten des mechanischen Systems bekannt, kann die Dgl mit z.B. einem der folgenden Verfahren berechnet werden :

- Newton - Euler

- D'Alambert
- Hamilton
- Lagrange

Für das entwickelte MKS-System wurde eine Beschreibung in Minimalkoordinaten gewählt, zur Erstellung der Bewegungsgleichung wird wahlweise das Lagrange-, oder das Newton-Euler-Verfahren verwendet. Diese Vorgehensweise erlaubt eine kompakte Formulierung der physikalischen Gegebenheiten. Der Nachteil dieses Methode ist jedoch, daß implizite Größen wie z.B. Zwangskräfte innerhalb des Mehrkörpersystems nachträglich berechnet werden müssen, da sie beim Erstellen der Bewegungsgleichung eleminiert werden. Umgekehrt betrachtet, bleiben die Zwangsbedinungen dafür aber die gesamte Integrationszeit exakt erfüllt. Alle, zur Erstellung des MKS-System relevanten, physikalischen Zusammenhänge werden im Folgenden diskutiert.

2.1 Generierung der Bewegungsgleichung unter Verwendung von Minimal-Koordinaten

Betrachtet man Translation und Rotation als 'Erzeugende' von mechanischen Bewegungen von Starrkörpern, so führt dies zu einer Parametrisierung der mechanischen Freiheitsgrade von gekoppelten Starrkörpern in Minimal-Koordinaten. Damit anschließend die Dgl. erstellt werden kann, wird zuerst eine genaue Definition aller auftretenden Größen gegeben:

α - η	beliebige Skalare Größen
q	Generalisierte Koordinaten
t	Zeit
a - **z**	Vektoren
A - **Z**	Matrizen (Tensoren)
a - *z*	Funktionen

Unter Minimal-Koordinaten wird hier die kleinste Anzahl an notwendigen Variablen verstanden, die nötig sind, um ein konkretes mechanisches Problem zu parametrisieren. Generalisierte Koordinaten sind entweder Winkel- oder Längen- Variablen, eine genaue Unterscheidung ist bei der Berechnung nicht nötig, da sie sich genau gleich behandeln lassen. [1]

2.1.1 Anatomie der Translation

Eine Translation der i-ten Komponente eines Vektors wird erzeugt durch Funktionen a_i, die mindestens zweifach stetig differenzierbar sind.

$$\left[\ldots \quad , a_i\left(q_k\left(t\right)\right), \quad \ldots \right] = \mathbf{a}\ \left(\mathbf{q_1}\left(\mathbf{t}\right), ..., \mathbf{q_i}\left(\mathbf{t}\right), ..., \mathbf{q_n}\left(\mathbf{t}\right)\right) \tag{2.1}$$

2.1.2 Anatomie der Rotation

Eine Rotation wird durch orthonormale Transformations-Matrizen beschrieben. Die Matrix $\mathbf{R}_{IK}$ beschreibt die Transformation vom körperfesten Koordinatensystem (K) zum Interialkoordinatensystem (I).

$$\mathbf{r}_I = \mathbf{R}_{IK} * \mathbf{r}_K \tag{2.2}$$

Folgende Eigenschaft kennzeichnet orthonormale Matrizen

$$\mathbf{R}_{KI} = \mathbf{R}_{IK}^{-1} = \mathbf{R}_{IK}^{T} \Leftrightarrow \mathbf{R}_{IK} * \mathbf{R}_{KI} = \mathbf{1}$$

,sodaß die Rücktransformation gegeben ist durch

$$\mathbf{r}_K = \mathbf{R}_{KI} * \mathbf{r}_I = \mathbf{R}_{IK}^{T} * \mathbf{r}_I$$

Im Allgemeinen wird die Rotation auf eine Drehachse bezogen, z.B. in 3 Dimensionen die Drehung um die x, y und z-Achse

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(q_i(t)) & \sin(q_i(t)) \\ 0 & -\sin(q_i(t)) & \cos(q_i(t)) \end{pmatrix} \tag{2.3}$$

[1] In der anglistischen Literatur wird stellenweise keine Unterscheidung zwischen diesen Begriffen vorgenommen [dad94] .

$$\mathbf{R}_y = \begin{pmatrix} \cos(q_i(t)) & 0 & -\sin(q_i(t)) \\ 0 & 1 & 0 \\ \sin(q_i(t)) & 0 & \cos(q_i(t)) \end{pmatrix} \tag{2.4}$$

$$\mathbf{R}_z = \begin{pmatrix} \cos(q_i(t)) & 0 & \sin(q_i(t)) \\ -\sin(q_i(t)) & 0 & \cos(q_i(t)) \\ 0 & 1 & 0 \end{pmatrix} \tag{2.5}$$

Die Drehmatrizen sind so organisiert, daß sich für ein positives $q_i(t)$ eine mathematisch positive Drehung ergibt. Durch wiederholtes rotieren um verschiedene Drehachsen ist es möglich, alle Freiheitsgrade eines Körpers auszuschöpfen. Dabei ist die Reihenfolge der angewendeten Drehachsen von großer Wichtigkeit, da im allgemeinen die Rotationsmatrizen nicht vertauschen.

$$\mathbf{R}_\alpha * \mathbf{R}_\beta \neq \mathbf{R}_\beta * \mathbf{R}_\alpha$$

Aus diesem Grund existieren viele, aber gleichberechtigte, Beschreibungsmöglichkeiten, wobei die Euler - und Kardanwinkel aber am geläufigsten sind. Soll ein, durch Scharniergelenke, zusammengestelltes mechanisches System simuliert werden, ist die Beschreibung durch die Abfolge der Glenke festgelegt. Die Verwendung von Kugelgelenken legt die Reihenfolge der Drehmatrizen jedoch nicht fest, da den Freiheitsgraden des Kugelgelenk keine definierte Abfolge zugeordnet werden kann.

- Eulerwinkel:
  $\mathbf{R_{gesamt}}[\alpha, \beta, \gamma] = \mathbf{R_x}[\alpha] * \mathbf{R_z}[\beta] * \mathbf{R_x}[\gamma]$
  Diese Winkel wurden benannt, da sie anschaulich zu interpretieren sind:
  $\gamma \rightarrow Rotorwinkel\ \beta \rightarrow Elevationswinkel\ \alpha \rightarrow Azimuthwinkel$
- Kardanwinkel:
  $\mathbf{R_{gesamt}}[\alpha, \beta, \gamma] = \mathbf{R_x}[\alpha] * \mathbf{R_y}[\beta] * \mathbf{R_z}[\gamma]$

2.2 Starrkörpermodell

Die auf Abb.2.1 bezogenen Bezeichnungnen

$\mathrm{Index}\ K$: Darstellung im körperfesten Koordinatensystems

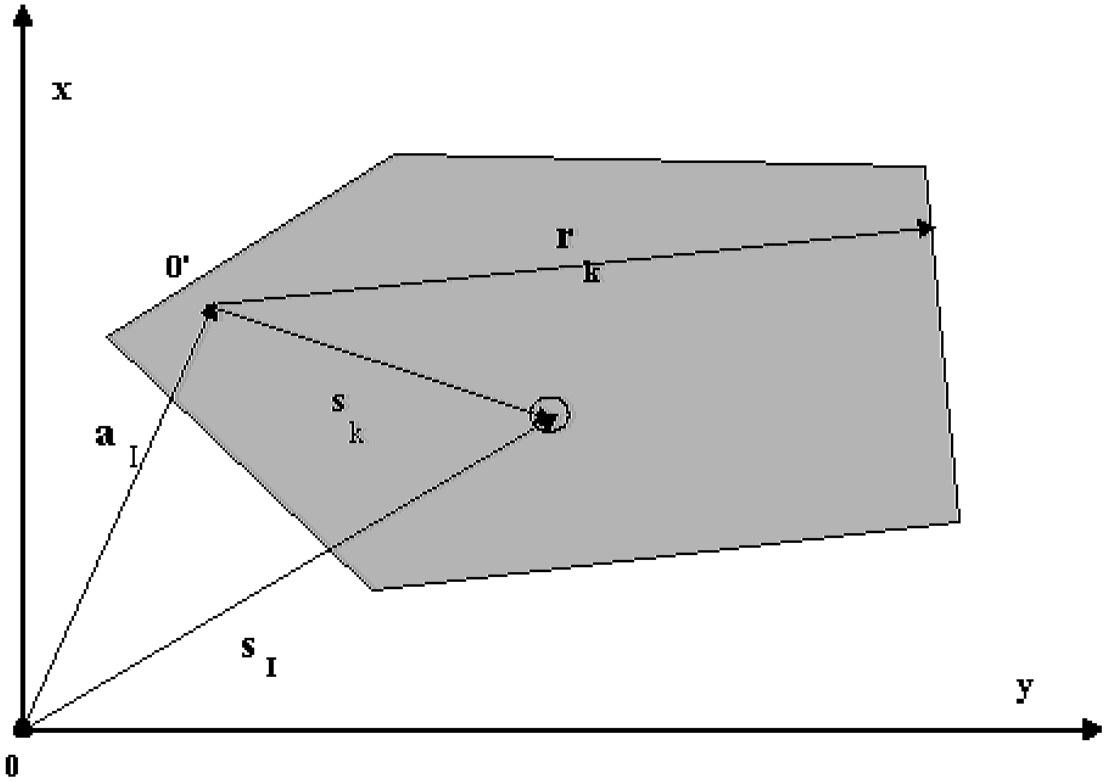

Abbildung 2.1: Definition des Starrkörpers

Index I	: Darstellung im inertialen Koordinatensystems
$\mathbf{a}_I$	: Aufpunkt
$\mathbf{r}_K$	: Referenzpunkt
$\mathbf{s}_K$	: Schwerpunkt
$\mathbf{0}$	: Nullpunkt des inertialen Koordinatensystems
$\mathbf{0}'$	: Nullpunkt des körperfesten Koordinatensystems

Somit läßt sich die Bewegung des Starrörpers durch folgende Gleichung charakterisieren :

$$\mathbf{s}_I = \mathbf{a}_I(q_1(t), ..., q_n(t)) + \mathbf{R}_{IK}(q_{n+1}(t), ..., q_f(t)) * \mathbf{s}_K \tag{2.6}$$

n : Anzahl der translatorischen Freiheitsgrade

$f - n$: Anzahl der rotatorischen Freiheitsgrade

$$\mathbf{s}_I = \mathbf{a}_I + \mathbf{R}_{IK} * \mathbf{s}_K \qquad \text{(Kurzform von Gl.2.6)} \tag{2.7}$$

Durch Ableiten der Abbildung erhält man die Geschwindigkeit des Schwerpunktes :

$$\begin{aligned} \mathbf{v}_I^s &= \tfrac{d}{dt}\mathbf{s}_I = \tfrac{d}{dt}\left[\mathbf{a}_I + \mathbf{R}_{IK} * \mathbf{s}_K\right] \\ \mathbf{v}_I^s &= \tfrac{d}{dt}\mathbf{a}_I + \left[\tfrac{d}{dt}\mathbf{R}_{IK}\right] * \mathbf{s}_K \end{aligned} \tag{2.8}$$

da $\frac{d}{dt}\mathbf{s}_K = 0$
Verändert der Körper seine Ausdehnung, ist dies nicht mehr zutreffend. Da es sich hier um einen Starrkörpermodell handelt, ist die Annahme aber zutreffend.

$$\mathbf{v}_K^s = \mathbf{R}_{KI} * \mathbf{v}_I^s = \mathbf{R}_{KI} * \frac{d}{dt}\mathbf{a}_I + \mathbf{R}_{KI} * \left[\frac{d}{dt}\mathbf{R}_{IK}\right] * \mathbf{s}_K \tag{2.9}$$

$$\text{Def.:} \qquad \mathbf{\Omega}_{KK} = \mathbf{R}_{KI} * \left[\tfrac{d}{dt}\mathbf{R}_{IK}\right] \tag{2.10}$$

$$\text{Def.:} \qquad \mathbf{v}_K^a = \mathbf{R}_{KI} * \tfrac{d}{dt}\mathbf{a}_I \tag{2.11}$$

$$\mathbf{v}_K^s = \mathbf{v}_K^a + \mathbf{\Omega}_{KK} * \mathbf{s}_K \tag{2.12}$$

$\mathbf{v}_K^a$: ist die Geschwindigkeit des Schwerpunkts in körperfesten Koordinaten

$\mathbf{\Omega}_{KK}$: ist ein Tensor 2. Stufe und gibt die Geschwindigkeit um die Drehachsen an. Handelt es sich um ein 3. dimensionales Problem, läßt er sich als Vektor schreiben. Dies funktioniert nur deshalb, da immer schiefsymmetrisch ist, und somit im 3. dimensionalen durch 3 Zahlen bestimmt ist.

$$\omega_K := \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \Leftarrow \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

Da die Rotationsmatrizen bekannt sind, läßt sich jeder Vektor und jeder Tensor in das jeweilige andere Koordinatensystem transformieren. Die später benutzten Terme werden hier aufgeführt :

$$\mathbf{v}_I^s = \mathbf{R}_{IK} * \mathbf{v}_K^s \tag{2.13}$$

$$\mathbf{\Omega}_{II} = \mathbf{R}_{IK} * \mathbf{\Omega}_{KK} * \mathbf{R}_{KI} \tag{2.14}$$

Nochmaliges Ableiten führt natürlich zu den Beschleunigungen. Um die Bewegungsgleichungen zu erstellen, ist dies nicht unbedingt erforderlich.

2.3 Kopplung von Starrkörpern

Ein Starrkörper für sich alleine genommen ist meist nicht so interessant, da seine Berechnungen schon ganze Bücher füllen (Kreiselgleichungen, usw). Die erste Kopplung von Starrkörpern ist das aneinanderhängen , sodaß eine mechanische Kette entsteht. "Kette" ist dabei nicht wörtlich zu verstehen, "Baum" wäre treffender, der Begriff hat sich aber so etabliert. Diese Vorgehensweise läßt sich mathematisch einfach ausdrücken: Der Aufpunkt, also der Punkt, an dem der Körper aufgehängt ist, wird an einen beliebigen Referenzpunkt des Vorgängers angehängt. Dabei ist aber unbedingt darauf zu achten, daß dieser Körper alle Rotationen aller Vorgänger ausführen wird. Damit läßt sich dann, wie oben angegeben, die Geschwindigkeit **v** und die Drehgeschwindigkeit ω für jeden Körper berechnen. Das körperfeste Koordinatensystem bezieht sich dabei auf den jeweiligen Körper.

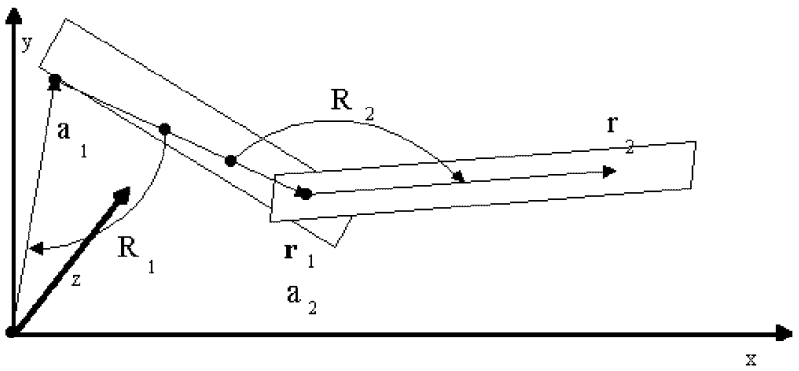

Abbildung 2.2: Mein zweites Bild

	Aufpunkt	Rotation
Körper 1	$\mathbf{a}_I^1$	$\mathbf{R}_{IK}^1$
Körper 2	$\mathbf{a}_I^2 = \mathbf{a}_I^1 + \mathbf{R}_{IK^1}^1 * \mathbf{r}_{K^1}^1$	$\mathbf{R}_{IK^1}^1 \mathbf{R}_{K^1K^2}^2$
usw.		

2.4 Bindungen

Die Kopplung von Starrkörpern kann auch mittels Bindungen beschrieben werden. Dabei werden allen n Körpern alle 6 n Freiheitsgrade belassen, die 6 n beschreibenden Parameter werden in einem Vektor zu den Systemkoordinaten zusammengefaßt.

$$\mathbf{r_I} = \mathbf{r_I}(\overline{\mathbf{z}}) \qquad ,\mathbf{r_I} \in \mathbf{R^3} \qquad ,\ \overline{\mathbf{z}} \in \mathbf{R^{6n}}$$

Die m Bindungen zwischen den Körpern werden durch implizite Gleichungen angegeben.

$$\phi(\overline{\mathbf{z}}, \mathbf{t}) = \mathbf{0} \qquad ,\phi \in \mathbf{R^m} \qquad ,\mathbf{m} \leq \mathbf{6n}$$

Diese Art der Bindung wird holonom genannt. Falls die Bindung explizit von der Zeit abhängt, heißt sie zusätzlich rheonom, bei zeitlicher Unabhängigkeit skleronom. Nun werden noch die Minimalkoordinaten s benötigt:

$$\overline{\mathbf{z}} = \overline{\mathbf{z}}(\mathbf{s}) \qquad ,\mathbf{s} \in \mathbf{R^f} \qquad ,\mathbf{f} = \mathbf{6n} - \mathbf{m}$$

Aus der Ableitung dieser Gleichung leitet man folgendes ab:

$$\frac{d}{dt}\overline{\mathbf{z}} = \frac{\partial \overline{\mathbf{z}}}{\partial \mathbf{s}}\dot{\mathbf{s}} \qquad * \frac{\partial}{\partial \dot{\mathbf{s}}}$$

$$\frac{\partial \overline{\mathbf{z}}}{\partial \mathbf{s}} = \frac{\partial \dot{\overline{\mathbf{z}}}}{\partial \dot{\mathbf{s}}} \qquad \in R^{6n\,,f}$$

Ist nun die letzere Gleichung immer umkehrbar (dies ist bei eindeutiger Bindungsgleichung immer gegeben), ist folgendes gültig:

$$\dot{\mathbf{s}} = \begin{bmatrix} \dot{\mathbf{s}} \\ \dot{\overline{\mathbf{z}}} \end{bmatrix} \dot{\overline{\mathbf{z}}}$$

Als letztes wird die Bindungsgleichung zur Lösung herangezogen :

$$\frac{d}{dt}\phi = \frac{\partial \phi}{\partial \overline{\mathbf{z}}}\dot{\overline{\mathbf{z}}} + \frac{\partial \phi}{\partial t} = B(\overline{\mathbf{z}}, t)\,\dot{\overline{\mathbf{z}}} + c(\overline{\mathbf{z}}, t) = 0 \qquad * \frac{\partial}{\partial \dot{\mathbf{s}}}$$

$$B(\overline{\mathbf{z}}, t)\frac{\partial \dot{\overline{\mathbf{z}}}}{\partial \dot{\mathbf{s}}} = 0 \tag{2.15}$$

Vorgehensweise :

- Aufstellen der Bindungsgleichungen $B(\overline{\mathbf{z}}, t)\,\dot{\overline{\mathbf{z}}} = 0$

- Eine Lösungsmatrix so suchen, daß gilt $B\left(\overline{\mathbf{z}},t\right)\frac{\partial\overline{\dot{\mathbf{z}}}}{\partial\dot{\mathbf{s}}}=0$

- Die Minimalgeschwindigkeiten ergeben sich durch
  $\dot{\mathbf{s}}=\left[\left[\frac{\partial\overline{\dot{\mathbf{z}}}}{\partial\dot{\mathbf{s}}}\right]^{T}*\left[\frac{\partial\overline{\dot{\mathbf{z}}}}{\partial\dot{\mathbf{s}}}\right]\right]^{-1}*\left[\frac{\partial\overline{\dot{\mathbf{z}}}}{\partial\dot{\mathbf{s}}}\right]^{T}*\ \overline{\dot{\mathbf{z}}}$

2.5 Iterative Berechnung von $\boldsymbol{\Omega}$

Soll sich ein Körper um mehr als eine Achse drehen, so erfordert dies mehrere hintereinander ausgeführte Rotationen. Nach obiger Formel läßt sich $\boldsymbol{\Omega}$ direkt auswerten. Dies erfordert aber einen beträchtlichen Rechenaufwand, sodaß es günstiger erscheint, eine Iterative Formel zur Berechnung heranzuziehen.

$$\begin{aligned}
\boldsymbol{\Omega} &= \mathbf{R}_{gesamt}^{T}*\left(\frac{d}{dt}\mathbf{R}_{gesamt}\right)\\
&= \left[\mathbf{R}_1*\mathbf{R}_2\right]^{T}*\left(\frac{d}{dt}\mathbf{R}_1*\mathbf{R}_2\right)\\
&= \mathbf{R}_2^{T}*\mathbf{R}_1^{T}*\left[\left(\frac{d}{dt}\mathbf{R}_1\right)*\mathbf{R}_2\ +\mathbf{R}_1*\left(\frac{d}{dt}\mathbf{R}_2\right)\right]\\
&= \mathbf{R}_2^{T}*\left(\mathbf{R}_1^{T}\frac{d}{dt}\mathbf{R}_1\right)*\mathbf{R}_2\ +\mathbf{R}_2^{T}\left(\frac{d}{dt}\mathbf{R}_2\right)\\
\boldsymbol{\Omega}\left(\mathbf{R}_1*\mathbf{R}_2\right) &= \mathbf{R}_2^{T}*\boldsymbol{\Omega}\left(\mathbf{R}_1\right)*\mathbf{R}_2\ +\boldsymbol{\Omega}\left(\mathbf{R}_2\right) \qquad (2.16)
\end{aligned}$$

In dieser Formulierung kann für jede Elementardrehung ein konstantes $\boldsymbol{\Omega}$ berechnet werden, damit sie später, in der durch die Rotationen zugeordneten Reihenfolge, summiert werden.

Diese Vorgehensweise läßt sich im 3 - dimensionalen Raum auf ω anwenden. Der Vektor wird nach Elementardrehgeschwindigkeiten entwickelt

$$\omega_{\mathbf{K}}=\mathbf{R}_{\gamma}*\mathbf{R}_{\beta}*\left[\mathbf{e}_{\mathbf{R}_{\alpha}}\right]+\mathbf{R}_{\gamma}*\left[\mathbf{e}_{\mathbf{R}_{\beta}}\right]+\left[\mathbf{e}_{\mathbf{R}_{\gamma}}\right]$$

$$(e_{\mathbf{R}_\alpha})_x = \begin{bmatrix} \dot{\alpha} \\ 0 \\ 0 \end{bmatrix} \qquad (e_{\mathbf{R}_\alpha})_y = \begin{bmatrix} 0 \\ \dot{\alpha} \\ 0 \end{bmatrix} \qquad (e_{\mathbf{R}_\alpha})_z = \begin{bmatrix} 0 \\ 0 \\ \dot{\alpha} \end{bmatrix}$$

Der Einheitsvektor der Elementardrehgeschwindigkeit bezieht sich auf die Drehachse der Rotation.

2.6 Die System-Energie

Sollen die Dgl's mit dem Lagrangeverfahren ermittelt werden, ist es nötig, die kinetische und die potentielle Energie des Starrkörpers zu kennen. Die kinetische Energie wird unterteilt in den translatorischen - und den rotatorischen Anteil. Für diese Unterteilung gibt es mehrere Möglichkeiten, da der rotatorische Anteil vom Bezugspunkt der Drehachse abhängig ist.

$T_{rot} = f(\mathbf{\Theta}^D_{KK}(\mathbf{P})$

PD : Bezugspunkt der Drehachse

Der Steiner'sche Satz transformiert den Trägheitstensor auf einen anderen Bezugspunkt. Um sich diese Transformation zu sparen, wird die rotatorische kinetische Energie im weiteren immer auf den Schwerpunkt des Starrkörpers bezogen. Eine Veränderung des Aufpunkts des Starrkörpers ändert deshalb immer nur die translatorische kinetische Energie.

Die translatorische kinetische Energie in körperfesten Koordinaten:

$$T_{trans} = \frac{1}{2}\mathbf{v}_K\, m \mathbf{v}_K^T \tag{2.17}$$

m : Masse

$\mathbf{v}_K$: Geschwindigkeit des Schwerpunkts nach Formel 2.12

Die rotatorische kinetische Energie im 3. dimensionalen Raum mit vektoriellem w hat äußerlich genau dieselbe Form :

$$T_{rot} = \frac{1}{2}\omega_K \Theta^S_{KK} \omega^T_K \tag{2.18}$$

Θ^S_{KK} : Trägheitstensor des Körpers bezogen auf den Schwerpunkt
ω_K : Drehgeschwindigkeit nach Formel 2.10

Versucht man, das Starrkörpersimulationsprogramm auf beliebige Dimensionen auszuweiten, stößt man auf das folgende Problem. Erweitert man den Ausdruck in Formel 2.18 einfach auch auf das tensorielle $\mathbf{\Omega}$, ergibt sich eine tensorielle kinetische Energie, der Lagrangeformalismus ist aber auf skalare Energien angewiesen, sodaß diese Vorgehensweise nicht richtig sein kann. Da zu diesem Thema in der mir verfügbaren Literatur keinerlei Information zu finden ist, wurde folgender Ausdruck, mittels Analogieschluß von $\frac{\mathbf{p}^2}{2m}$, entwickelt, der für beliebige Dimensionen die skalare rotatorische Energie liefert.

$$T_{rot} = \mathrm{Tr}\left[\Theta^{-1}_{KK}\mathbf{\Omega}\left[\Theta^{-1}_{KK}\mathbf{\Omega}\right]^T\right] * \frac{1}{4}\mathrm{Det}\left[\Theta_{KK}\right] \tag{2.19}$$

$\mathrm{Det}[\mathbf{\Theta}_{KK}]$: Determinante von $\mathbf{\Theta}_{KK}$
$\mathrm{Tr}[\mathbf{A}]$: Spur des Tensors A

Die kinetischen Energien T_{rot} und T_{trans} lassen sich auf die gleiche Weise auch im inertialen Koordinatensystem berechnen, vorausgesetzt alle Terme wurden entsprechend transformiert. Der hierfür erforderliche Rechenaufwand, vorallem für , sollte dabei aber nicht vernachlässigt werden, zumal das skalare Ergenis unabhängig von der Koordinatendarstellung ist.

Das Potential des Schwerefeldes hat folgende Form :

$$V_{\mathbf{g}} = m\mathbf{g}\mathbf{r}^s_I \tag{2.20}$$

$\mathbf{g}$: Erdbeschleunigung, typischerweise (0, 0, - 9.81)
$\mathbf{r}^s_I$: Position des Schwerpunkts in inertialen Koordinaten

Die Gesamtenergie des Körpers ergibt sich zu :

$$E_{ges} = \sum_{i=1}^{n} [V_{\mathbf{g}} + T_{rot} + T_{trans}]_i \tag{2.21}$$

n : Anzahl der Körper

2.7 Generalisierte Kräfte

Die Verwendung von generalisierten Kräften hat den Vorteil, daß sie sich direkt auf die ihnen zugeordneten Freiheitsgrade beziehen. Wird das zu simulierende System nicht mit generalisierten Koordinaten beschrieben, müssend die eingeprägten Kräfte und Momente erst auf die Wirkvariablen projiziert werden. Dasselbe trifft aber auch auf ein System zu, formuliert in generalisierten Koordinaten, wenn von außen Kräfte oder Momente auf das System einwirken.

$$\mathbf{Q}_{ges} = \mathbf{Q}_0 + \sum_{i=1}^{n} \left[\left[\frac{\partial}{\partial \dot{q}} \omega \right]^T \mathbf{m} + \left[\frac{\partial}{\partial \dot{q}} \mathbf{v}^p \right]^T \mathbf{f} \right]_i \tag{2.22}$$

$\mathbf{Q}_0$: generalisierte Kräfte
$\mathbf{v}^P$: Geschwindigkeit des Kraftangriffspunktes
$\mathbf{f}$: Kraft
$\mathbf{m}$: Moment
n : Anzahl der Körper

2.8 Das Newton-Euler Verfahren

Das Verfahren zur Erstellung von Bewegungsgleichungen wurden von Euler um 1775 entwickelt. Es setzt sich aus den Newtonschen Gleichungen der Mechanik und den Kreiselgleichungen von Euler zusammen, mit zusätzlicher Projektion auf die Variablen, um

die Zwangskräfte zu eliminieren Das Verfahren ist nicht forminvariant unter Koordinatentransformationen, wie auch schon die Newtonschen Gleichungen, sodaß verschiedene Formulierungen existieren:

Newton-Euler Verfahren im körperfesten Koordinatensystem

$$\sum_{i=1}^{n} \left[\left[\left[\frac{\partial}{\partial \dot{q}} \mathbf{v}^s \right]^T [\dot{\mathbf{p}} + \ \boldsymbol{\Omega} * \mathbf{p} - \mathbf{f}] \right]_K + \left[\left[\frac{\partial}{\partial \dot{q}} \omega \right]^T \left[\dot{\mathbf{l}} + \boldsymbol{\Omega} * \mathbf{l} - \mathbf{m} \right] \right]_K \right]_i = 0 \quad (2.23)$$

Newton-Euler Verfahren im Inertialkoordinatensystem

$$\sum_{i=1}^{n} \left[\left[\left[\frac{\partial}{\partial \dot{q}} \mathbf{v}^s \right]^T [\dot{\mathbf{p}} - f] \right]_I + \left[\left[\frac{\partial}{\partial \dot{q}} \omega_{\ I} \right]^T \left[\dot{\mathbf{l}} - \mathbf{m} \right] \right]_I \right]_i = 0 \quad (2.24)$$

Möglich ist auch eine Mischung, in der beide Impulssätze in unterschiedlichen Systemen beschrieben sind. Den kleinsten Rechenaufwand erfordert es, wenn der Impuls im Inertialsystem und der Drehimpuls im körpereigenen System beschrieben ist.

- $\mathbf{p}$: mechanischer Impuls
- $\mathbf{l}$: Drehimpuls
- $\mathbf{f}$: die in das mechanische System eingeprägten Kräfte, jedoch keine Zwangskräfte, da diese implizit schon enthalten sind.
- $\mathbf{m}$: in das System eingebrachte Momente (Keine Zwangsmomente)
- n : Anzahl der Körper

2.9 Das Lagrange - Verfahren

Es wurde um 1788 von Lagrange entwickelt. Da die Bewegungsgleichung aus der kinetischen und potentiellen Energie abgeleitet wird, bleibt es bei Koorinatentransformationen forminvariant. Das Verfahren arbeiten mit generalisierten Koordinaten, sodaß eine Unterscheidung zwischen Längen- und Winkelvariablen unnötig ist. Aus diesem Grund wird hier auch nicht mehr zwischen eingeprägten Kräften und Momenten unterschieden, es handelt sich hierbei um generalisierte Kräfte.

$$\sum_{i=1}^{n} \left[\frac{d}{dt} \left[\frac{\partial}{\partial \dot{q}} T \right] - \left[\frac{\partial}{\partial q} T \right] + \left[\frac{\partial}{\partial q} V \right] - \mathbf{Q} \right]_i = 0 \tag{2.25}$$

T : kinetische Energie
V : Potential
$\mathbf{Q}$: eingeprägte generalisierte Kräfte
n : Anzahl der Freiheitsgrade

2.10 Lösen der Bewegungsgleichungen

Alle oben genannte Verfahren erzeugen f Gleichung folgenden Typs:

$$\alpha \ddot{q}_1 + ... + \mu \ddot{q}_f + g_1 [t, q_1, ... , q_f, \dot{q}_1, ... , \dot{q}_f] - \mathbf{Q} [t, q, \dot{q}] = 0$$

Diese Gleichungen werden zu einer f - dimensionalen Vektordifferenzialgleichung zusammengefaßt. Die Matrix M wird verallgemeinerte Massenmatrix genannt und ist f x f dimensional.

$$\mathbf{M} [\mathbf{q}] * \begin{bmatrix} \ddot{q}_1 \\ \cdot \\ \cdot \\ \ddot{q}_f \end{bmatrix} + \begin{bmatrix} g_1 [t, q, \dot{q}] \\ \cdot \\ \cdot \\ g_f \ [t, q, \dot{q}] \end{bmatrix} - \mathbf{Q} [\mathbf{t}, \mathbf{q}, \dot{\mathbf{q}}] = \mathbf{0}$$

Um die Gleichung berechenbar zu gestalten, wird sie nach den zweiten Ableitungen aufgelöst.

$$\ddot{q} = \mathbf{M} [\mathbf{q}]^{-1} * [\mathbf{Q} [\mathbf{t}, \mathbf{q}, \dot{\mathbf{q}}] - \mathbf{g} [\mathbf{t}, \mathbf{q}, \dot{\mathbf{q}}]]$$

Durch Variablensubstitution $\dot{q}_i \rightarrow v_i \quad i \in (1 \dots f)$ von wird die Differenzialgleichung 2. Ordnung in eine Dgl 1. Ordnung überführt. Dies ist notwendig, um die Dgl numerisch lösen zu können.

$$\frac{d}{dt}\begin{bmatrix} \mathbf{q} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{M}[\mathbf{q}]^{-1} * [\,\mathbf{Q}[\mathbf{t},\,\mathbf{q},\,\mathbf{v}] - \mathbf{g}[\mathbf{t},\mathbf{q},\,\mathbf{v}]\,] \end{bmatrix} \tag{2.26}$$

Anmerkungen zur Matrizeninversion:

- Singularität :
  Wird z.B. ein Kreisel in Eulerwinkeln beschrieben, so wird die Matrix singulär, falls der Elevationswinkel den Wert 0.0 erreicht. Diese Singularitäten treten immer nur bei bestimmten Wertepaarungen auf. Durch Auflösen des Gleichungssystems $\det[\mathbf{M}[\mathbf{q_1},...,\mathbf{q_f}]\,] = 0$ werden sie bestimmt. Werden aber statt der körpereigenen Koordinatensystem oder des Inertialsystems 2 Referenzkoordinatensystem verwendet, und anschließend, falls die verallgemeinerte Massenmatrix singulär wird, das andere Referenzsystem verwendet, so läßt sich diesem Problem, allerdings mittels beträchtlichen Mehraufwands, umgehen [BRE 88].

- Aufwand :
  Wird die Inversion symbolisch durchgeführt, so erniedrigt sich der numerische Rechenaufwand für das Lösen der Diffgleichung, da sonst bei jedem Integrationsschritt eine numerische Inversion durchgeführt werden muß. Die symbolisch Inversion hat aufwandsbedingt aber auch ihre Grenzen, die bei ca. 5 x 5 Matrizen liegen dürfte.

2.11 Einheiten im Simulationsprogramm

In erster Linie wird die Verwendung von Einheiten im Simulationsprogramm nicht unterstützt. Da aber alle Eingaben in ihrer eigentlichen Form erhalten bleiben (es werden nur Operationen wie ableiten, multiplizieren usw. verwendet) , verändern sich die Einheiten nur in vorhersagbarer Form. Die Verwendung von generalisierten Koordinaten definiert somit die Einheit der entsprechenden Wirkungsvariablen.

$Laenge$	$Winkel$	$Masse$	$Zeit$	$Einheitensystem$
$[m]$	[]	$[kg]$	$[s]$	MKS
$[cm]$	[]	$[g]$	$[s]$	CGS

Der Winkel ist immer in rad angegeben, und damit einheitenlos.

Somit ergibt sich folgendes für die abgeleiteten Einheiten

Winkelvariablen:

$Einheitensystem$	$Winkegeschwindigkeit$	$Winkelbeschleunigung$	$Drehmoment$
MKS	$\left[\frac{1}{s}\right]$	$\left[\frac{1}{s^2}\right]$	$[Nm]$
CGS	$\left[\frac{1}{s}\right]$	$\left[\frac{1}{s^2}\right]$	$[10^{-4}Nm]$

Längenvariablen:

$Einheitensystem$	$Geschwindigkeit$	$Beschleunigung$	$Kraft$
MKS	$\left[\frac{m}{s}\right]$	$\left[\frac{m}{s^2}\right]$	$[N]$
CGS	$\left[\frac{cm}{s}\right]$	$\left[\frac{cm}{s^2}\right]$	$[10^{-4}N]$

2.12 Beispiel einer Starrkörpersimulation

Das folgende Beispiel soll der Vorgehensweise, der Veranschaulichung und der Verifikation des Simulationsprogrammes dienen. Als Beispiel ausgesucht wurde das Kreiselparadoxon im 3 - dimensionalen Raum. Hierbei handelt es sich um einen schweren Kreisel mit allen 3 Freiheitsgraden der Rotation, parametrisiert durch die drei Eulerwinkel. Der Kreisel besitzt keine Freiheitsgrade bezüglich der Translation, somit ist genau ein Punkt des Starrkörpers, im weiteren soll dieser Punkt mit Aufpunkt bezeichnet werden, raumfest. Auf das System wirken keine expliziten Kräfte, es befindet sich aber im Gravitatiospotential. Da das System demnach als abgeschlossen betrachtet werden kann, handelt es sich um ein konservatives Problem, somit muß die Energie des Systems eine Erhaltungsgröße sein.

Die Syntax der Starrkörperbeschreibung des Simulationsprogrammes :

K[*name*] = “Euler - Kreisel” ;
Qq = {q1[t],q2[t],q3[t]};
K[1][*initschwerpunkt*] = 0,0,0.5 ;
K[1][*initlage*][1] = 0,0,1 ;
K[1][*potential*] = (- K[1][*masse*] * K[1][*schwerpunkt*] . g);
K[1][*aufpunkt*] = 0 , 0 , 0 ;
K[1][*drehung*][1] = Rot[z, q1[t]];
K[1][*drehung*][2] = Rot[x, q2[t]];
K[1][*drehung*][3] = Rot[z, q3[t]];
Evalbody;

K[1] : Die Daten beziehen sich auf den ersten Körper
Qq[[i]] : Der Variablennamen des i. ten Freiheitsgrade ist hier definiert
[*initschwerpunkt*] : Die Schwerpunktkoordinate im körpereigenen Koordinatensystem
[*initlage*][1] : Die 1. Referenzkoordinate im körpereigenen Koordinatensystem
[*aufpunkt*] : Aufpunkt, angegeben im inertialen Koordinatensystem
[*drehung*][1] -

$[drehung]$[n] : Die Rotationsfreiheitsgrade des Starrkörpers werden hierdurch definiert. Das Produkt von $[drehung]$[1] * ... * $[drehung]$[n] definiert den Tensor , der das körpereigene Koordinatensystem in das inertiale Koordinatensystem transformiert. Mit Hilfe von Gleichung 2.2 werden die Vektoren $[initschwerpunkt]$ und $[initlage]$[1] in das inertiale Koordinatensystem transformiert und sind fortan unter dem Namen $[schwerpunkt]$ und $[position]$[1] verfügbar.

g : Erdbeschleunigung = (0, 0, - 9.81)

Evalbody : Berechnet alle weiteren notwendigen Daten

Für eine vollständige Beschreibung des mechanischen Systems bedarf es noch einiger Konstanten:

$[masse]$: Masse des Starrkörpers

$[trägheitstensor]$: Trägheitstensor des Starrkörpers im körpereigenen Koordinatensystem, günstigerweise im Hauptachsensystem

KraftMoment : generalisierte Kräfte (entspricht Q in Gleichung 2.22)

tbegin : Anfangszeit der Integration

tend : Endzeit der Integration

qstart : Startwerte für die Integration der gesuchten Funktionen $q_i[t]$ in der Reihenfolge (q1 [t], q2 [t], ...)

qpstart : Startwerte für die Ableitungen der gesuchten Funktionen $q_i[t]$ in der Reihenfolge (q1' [t], q2' [t], ...)

Mittels dieser Daten ist die Mechanik des Starrkörper vollständig definiert und die Dgl kann berechnet werden. Anschließend wird die Dgl als C-Programm ausgegeben und integriert.

Die Anfangswerte für die Integration und die Konstanten des Kreisels :

tbegin = 0.0

tend = 2.0 bzw. 15.0

qstart = (0.0, Pi/2, 0.0)

qpstart = (0.0, 0.0, 30.0)

$[masse]$ = 8

$$[trägheitstensor] = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{pmatrix}$$

Im Anhang A wird das C-Programm aufgeführt

Die Integrationsdauer wurde mit Hinsicht auf die graphische Ausgabe einmal sehr kurz gewählt, um die kleinen Variationen in den Variablen sichtbar zu machen, und einmal sehr groß gewählt, um das langfristige zeitliche Verhalten zu dokumentieren.

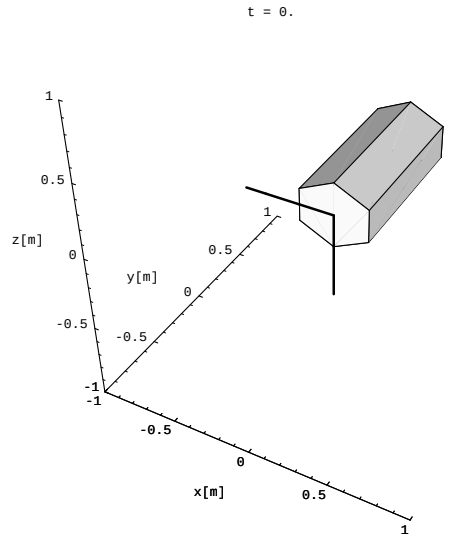

Abbildung 2.3: Simulierter Starrkörper

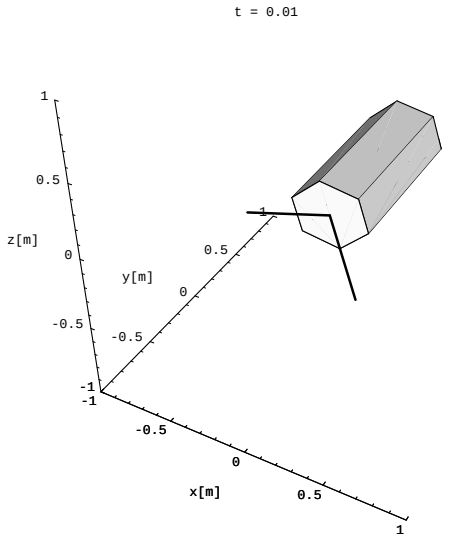

Abbildung 2.4: Simulierter Starrkörper

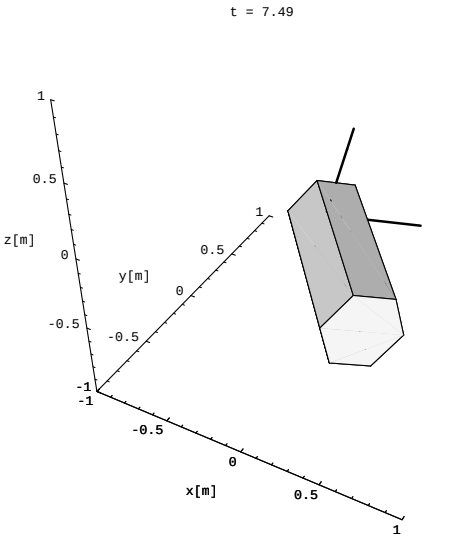

Abbildung 2.5: Simulierter Starrkörper

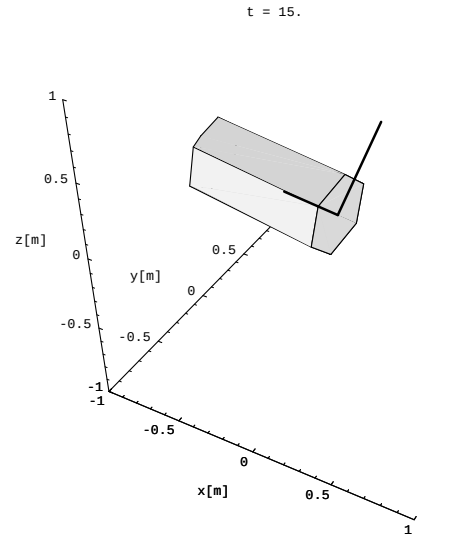

Abbildung 2.6: Simulierter Starrkörper

In Abb.2.3 bis 2.6 wird der Kreisel zu verschiedenen Zeitpunkten gezeigt. Der Punkt (0, 0, 0), markiert durch ein kleines Achsenkreuz, ist der raumfeste Aufpunkt des Kreisels. Für eine Darstellung des Problems sind diese Bilder gut geeignet, doch läßt sich die zeitliche Entwicklung des Systems nur durch eine Animation am Bildschirm verfolgen. Auf Papier läßt sich dies leider nicht realisieren, sodaß im folgenden die Trajektorien und Variablen den zeitlichen Verlauf klären sollen.

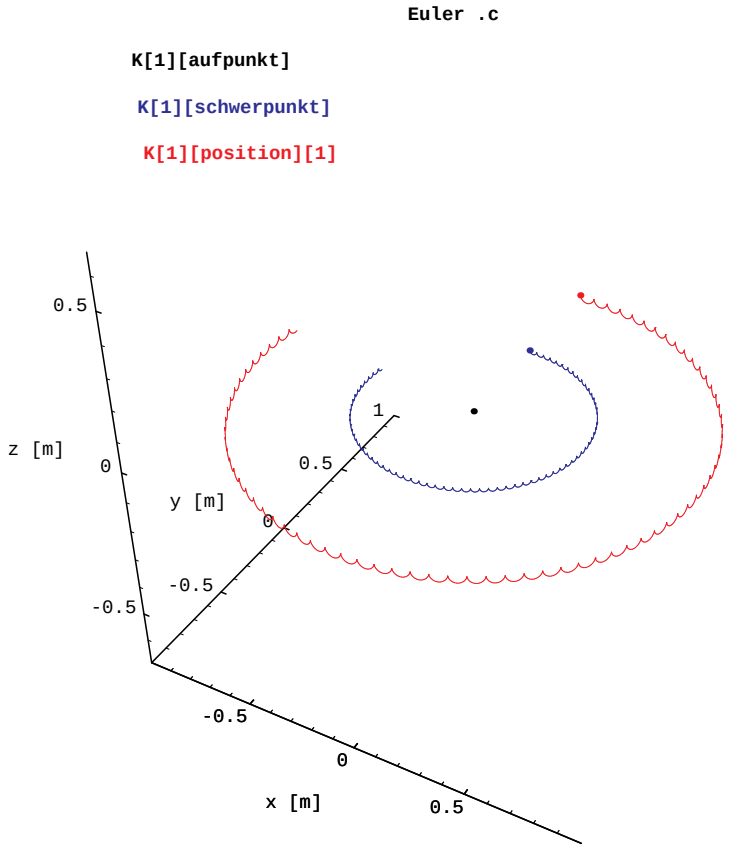

Abbildung 2.7: Die Trajektorie des Schwerpunkts und des 1. Referenzpunktes

Der 1. Referenzpunkt, dargestellt durch K[1][$position$][1], markiert den Endpunkt des Körpers und hat im körpereigenen Koordinatensystem den Wert (0, 0, 1). Der Schwerpunkt hat im körpereigenen Koordinatensystem den Wert (0, 0, 0.5). Der dicke Punkt am Anfang jeder Trajektorie markiert die Zeit t = 0, das Ende der gezeichneten Trajektorie liegt bei t = 15. Wie gefordert ist die Energie in Abb.2.12 während der Integrationszeit annähernd konstant. Die Schwankungen werden durch den Integrator verursacht, in diesem Fall von ODINT aus den Numerical Recipies [PTVF94]. Dabei handelt es sich um

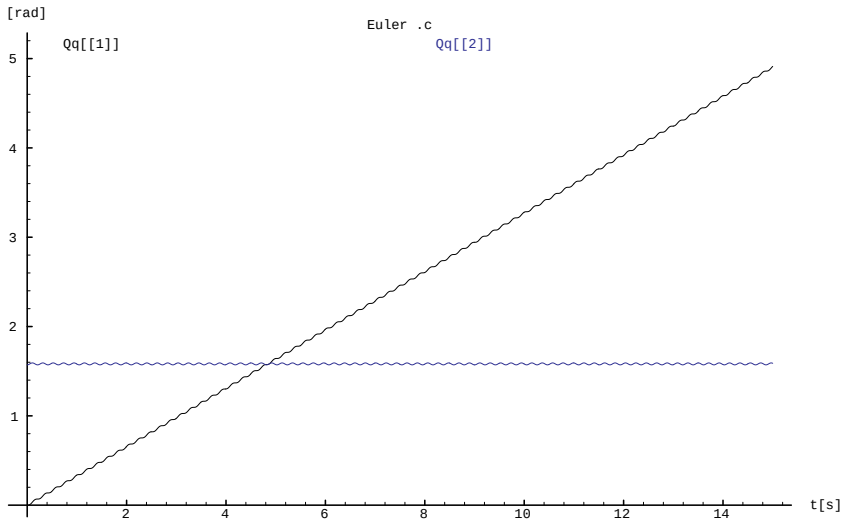

Abbildung 2.8: Rotation des Körpers um die inertiale z - Achse, und um die x' - Achse

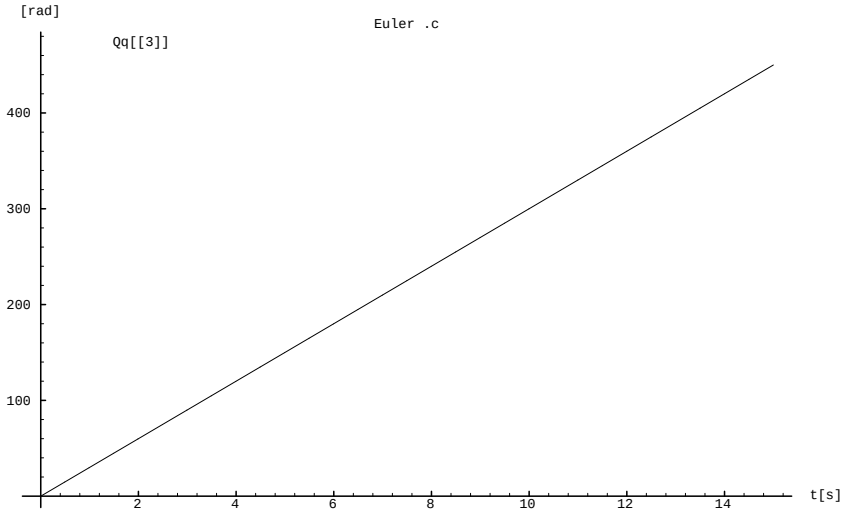

Abbildung 2.9: Rotation des Körpers um die z' - Achse, wobei die z' - Achse der korpereigenen z - Achse entspricht.

einen schrittweitengesteuerten Runge - Kutta Integrierer 4. Ordnung , dessen Genauigkeitsanforderung auf $10^{-}7$ eingestellt ist. Es bleibt noch anzumerken, daß die Forderung einer konstanten Energie nicht explizit in die Dgl eingebracht wird, sondern die Formulierung der Mechanik des Körpers beinhalte diese Forderung implizit. Werden Kräfte oder Momente in das System eingebracht, verändert sich demzufolge auch die Energie des Systems. Der Betrag des Drehimpulses, wie er in Abb.2.13 dargestellt ist, bleibt annähernd

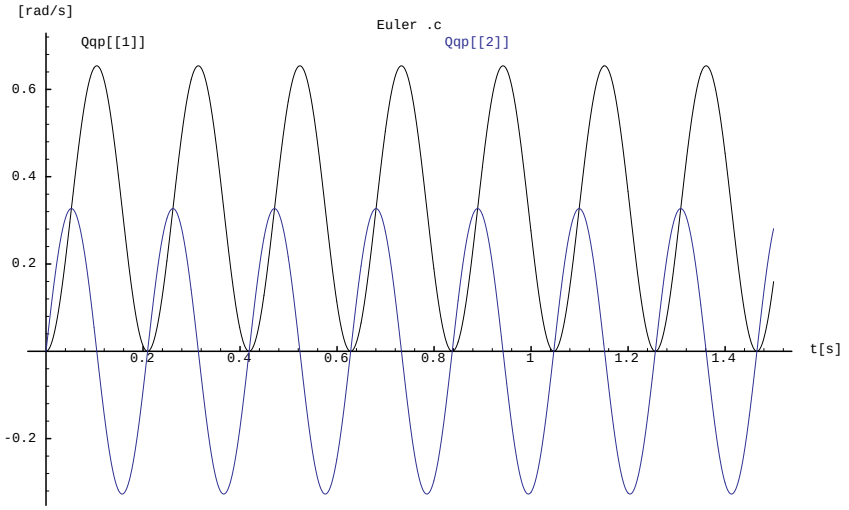

Abbildung 2.10: Die Ableitung der Rotation des Körpers um die z und x' - Achse

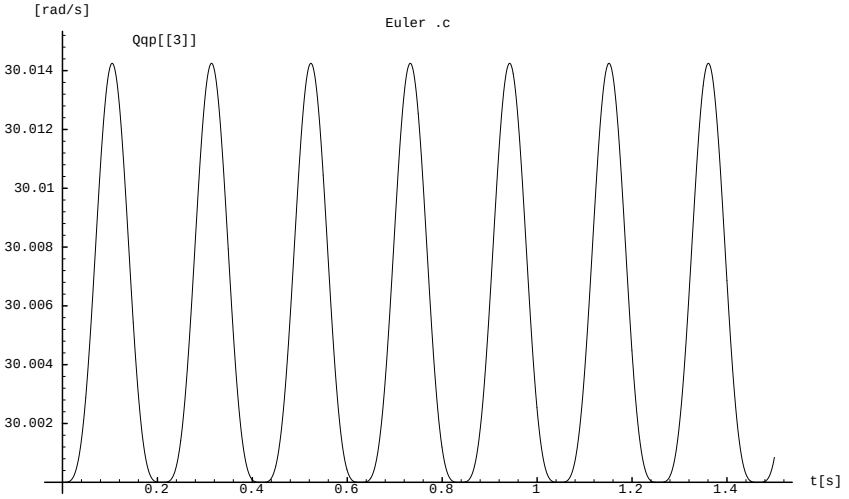

Abbildung 2.11: Die Ableitung der Rotation des Körpers um die z' - Achse

konstant, die Abweichung ist hier aber nicht numerischer Natur, sie beruht auf der minimalen Variation der Rotation der x' - Achse (Variable q2(t) , mit direktem Einfluß auf T_rot)). Da das System ganz ohne Reibung simuliert wird, bleiben diese kleinen Schwingungen während der ganzen Simulationszeit auch erhalten.

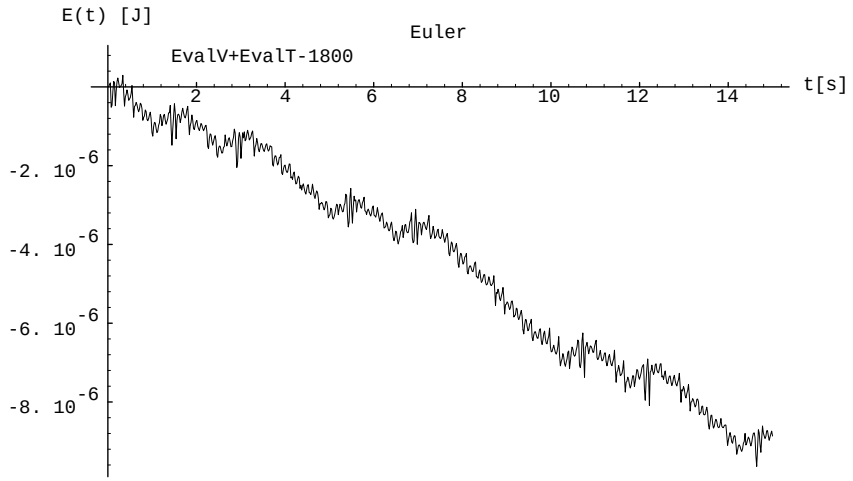

Abbildung 2.12: Die Gesamtenerie des Systems EvalV : Die potentielle Energie des Systems EvalT : Die kinetische Energie

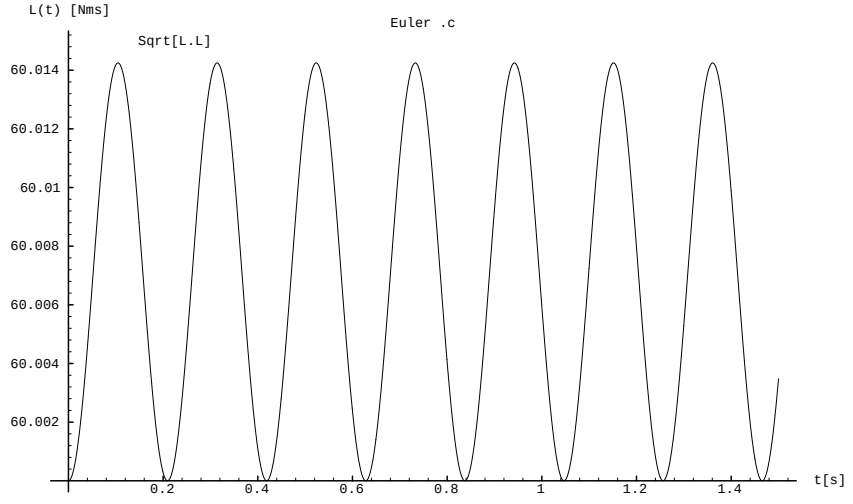

Abbildung 2.13: Der Betrag des DrehimpulsesL = $\omega_K \Theta_{KK}$: Drehimpuls des Systems

2.12.1 Dsikusion des Simulators

Das Simulationsprogramm ist in der Lage, den präzedierenden Kreisel zu simulieren. Das Ergebnis ist in den Grenzen einer Simulation korrekt, sogar der Drehsinn der Ausweich-

bewegung[2] des Kreisels wird korrekt vorhergesagt.

[2]Mit einem einfachen Rad, dessen Achse nur an einem Endpunkt unterstützt ist, läßt sich der Drehsinn einfach ermitteln

Kapitel 3

Neuronale Netze

Künstliche neuronale Netze sind Ausdruck des Versuches biologische Informationsverarbeitung nachzubilden. Die elementaren Einheiten von neuronalen Netzen sind Neuronen. Aus diesem Grund wurden zuerst biologische Neuronen untersucht, um aus diesen Daten künstliche Neuronen nachzubilden. Das Problem dabei ist die Vielfältigkeit der in der Natur auftretenden Modelle (...), sodaß sich künstliche Neuronenmodelle großteils auf das wesentliche beschränken. Da das “wesentliche” vom Standpunkt abhängig ist, haben sich grundsätzlich verschieden mathematische Modellierungen etabliert.

3.1 Neuronen

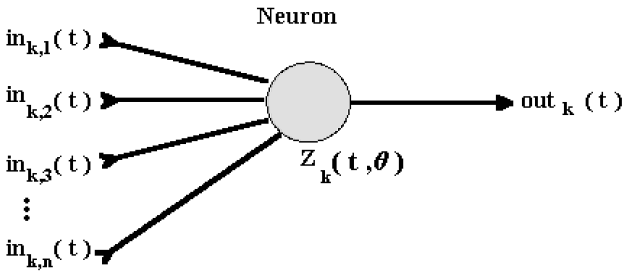

Abbildung 3.1: Modell des Neurons

t : Zeitindex

k : Nummer des Neurons
$\mathrm{in}_{k,i}(t)$: Eingang *i* des Neurons
$\mathrm{out}_k(t)$: Ausgang des Neurons
θ_k : Schwellwert des Neurons
$\mathrm{Z}_k(t)$: innerer Zustand (Aktivierung) des Neurons

Im Allgemeinen werden die Eingänge der Neuronen nicht durchindiziert, per Definition besitzt jedes Neuron soviel Eingänge wie benötigt werden.

Beispiele für die mathematische Modellierung von Neuronen

1. Standard - Neuronen - Modell
   Die Ein- und Aus-gänge sind amplidtudenmodulierte (reelle oder binäre) Werte kodiert. Hieraus resultiert eine sehr einfache Implementation :

$$\begin{aligned} f &= \textstyle\sum_i^n in_i(t)\alpha_i \\ Z_k(t+1) &= z\left[Z_k(t), \theta_k, f\left[in\right]\right] \\ out(t+1) &= g\left[Z_k(t+1)\right] \end{aligned} \tag{3.1}$$

f : Propagierungsfunktionen
z : Aktivierungsfunktionen
g : Ausgabefunktionen
f, z, g : beliebige, im allgemeinen jedoch stetige, monoton steigende Funktionen, Abb. 3.2 und 3.3 sind zwei typische Beispiele für f, z oder g

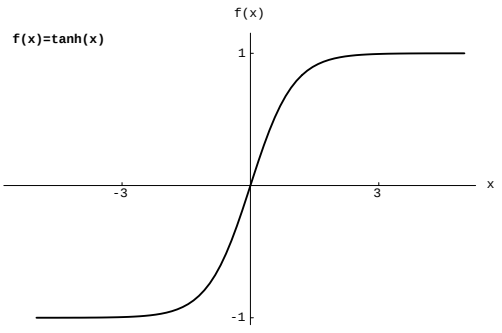

Abbildung 3.2: tangens hyberbolicus

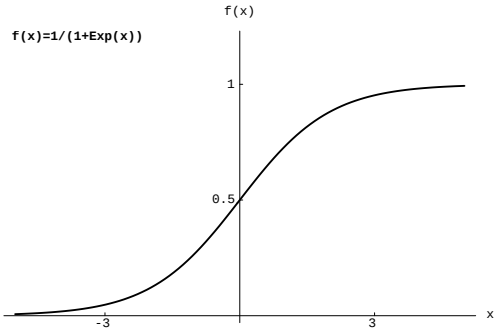

Abbildung 3.3: logistische Funktion

Ein sehr einfaches Modell ergibt sich bei der Verwendung von $f(x) = g(x) = x$. Diese Standard-Propagierungsfunktion erlaubt eine einfache und schnelle Implementierung auf Computern. Aus diesem Grund ist dieses Modell, in der Verbindung mit der logistischen Aktivierungsfunktionen, sehr verbreitet.

$$\mathrm{out}(t+1) = \frac{1}{1+\exp^{-\left[\sum_i^n \mathrm{in}_i(t)\alpha_i\right]-\theta}} \tag{3.2}$$

Eine wichtige Eigenschaft dieses einfachen Modells ist die Beschränkung der Ausgangsgröße auf den Bereich $[0 \ldots 1[$. Dadurch wird eine Aussteuerungsfestigkeit gegenüber den Eingabedaten erreicht.

2. Dgl - Neuronen - Modell
   Wie oben angeführt existieren zahllose mögliche Modelle, um das zeitliche Verhalten realer Neuronen mehr oder wenige exakt nachzubilden, deshalb sei hier nur ein Beispiel für ein Neuronenmodell mit Selbstadaption angeführt. Die Ein- und Aus-gänge werden hierbei häufig durch Frequenzmodulation kodiert.

$$\begin{aligned} \tau_0 \dot{u} &= -u - \textstyle\sum_i^n f\left[in_i(t)\right] - \beta v + u_0 \\ \tau_1 \dot{v} &= -v + g(u) \end{aligned}$$

   $\tau_0, \tau_1, \beta, u_0$ = $const$
   f , g : wie oben

   Beispiele für biologisch motivierte Modell verschiedener Nervenzellen sind z.B. in [unk95] beschrieben.

3.2 Aufbau Neuronaler Netze

Nimmt man mehrere Neuronen und verbindet sie anhand bestimmter Regeln erhält man Neuronale Netze mit spezifischen Eigenschaften. Die Verbindungsregeln werden am günstigsten in Matrixschreibweise formuliert. In der folgenden Abbildung wurden der Übersichtlichkeit halber nicht alle möglichen Kopplungen eingezeichnet. Die Stärke der Kopplung kann sowohl positiv (exhibitorisch) als auch negativ (inhibitorisch) sein, in den biologischen Vorbildern gibt es jedoch nur exhibitorische Kopplungen (dafür aber inhibitorische Eingänge).

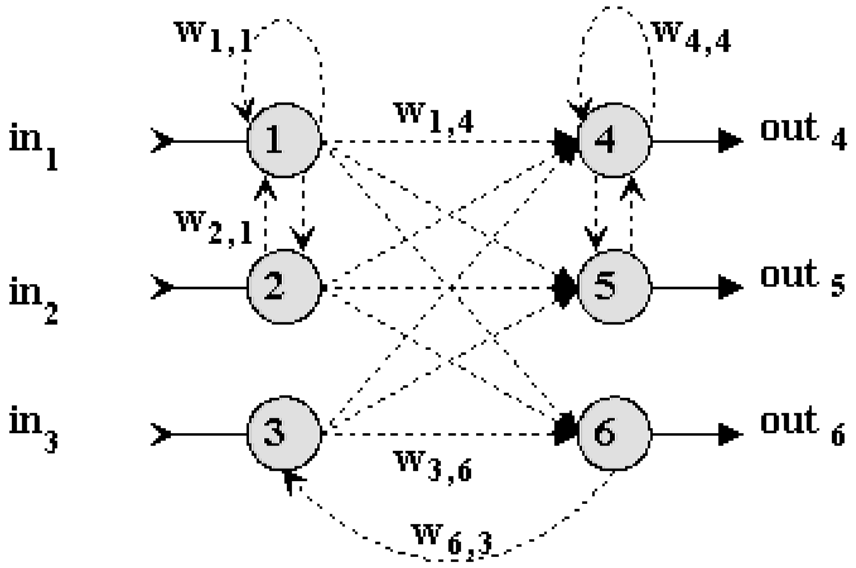

Abbildung 3.4: Neuronales Netz

$$\mathbf{Out}(t+1) = g\left[\left[\sum_{i}^{n+k} f\left[\mathbf{O}(t) * \begin{pmatrix} w_{1,1} & \ldots & w_{n+k,1} \\ \ldots & \ldots & \ldots \\ w_{1,n+k} & \ldots & w_{n+k,n+k} \end{pmatrix}\right]\right] - \theta_k\right] \tag{3.3}$$

$\mathbf{O}$ (t) = $[out_1, out_2, \ldots, out_n, in_1, \ldots, in_k]$
$\mathbf{Out}$ (t) = $[out_1, out_2, \ldots, out_n]$
$w_{k,l}$: Kopplung von Neuron k mit Neuron l
$\mathbf{W}$: Kopplungsmatrix

Die Kopplungsmatrix $\mathbf{W}$ (häufig auch als Gewichtsmatrix bezeichnet) definiert die Topologie des Neuronalen Netzes, umgekehrt ist dies natürlich auch zutreffend. Im allgemeinen werden die Neuronen mit den Eingängen durch spezielle Eingabeneuronen repräsentiert, um die Eingabe skaliert an die nächste Schicht weiterzureichen. Hierdurch wird $\mathbf{O}$ (t) überflüssig und geht in $\mathbf{Out}$ (t) über. Die Schicht, an der die Ausgabe des Netzes abgegriffen wird, heißt Ausgabeschicht, oder auch Ausgabe-Layer. Alle Schichten (Layers) zwischen Ein - und Ausgabe - Layer werden als verdeckte Schicht bzw. ” hidden layers ” bezeichnet. Da die eigentliche Informationsverarbeitung des Netzes in

die Gewichtsmatrix eingebettet ist, wird ein Netz mit n Layers als n - 1 stufiges Netz bezeichnet.

Beispiele für Netzwerktopologien :

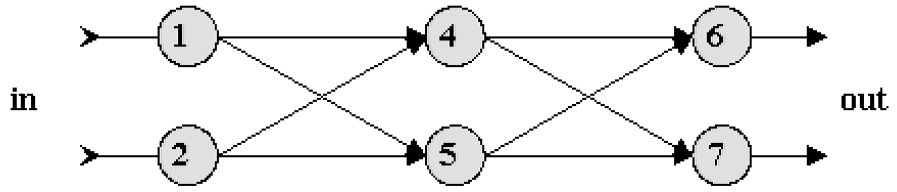

Abbildung 3.5: Feedforward - Netz

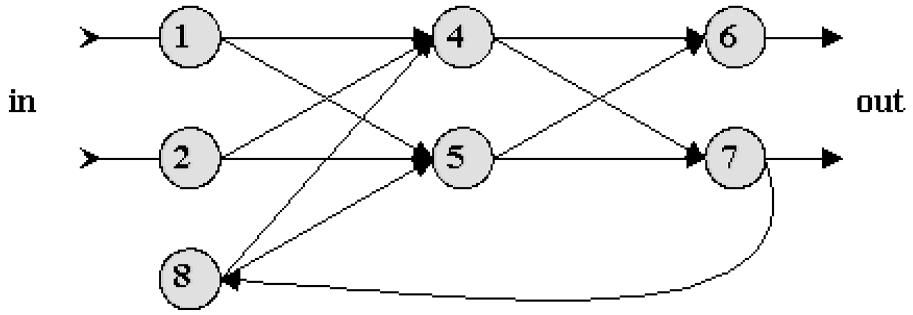

Abbildung 3.6: Jordan - Netz

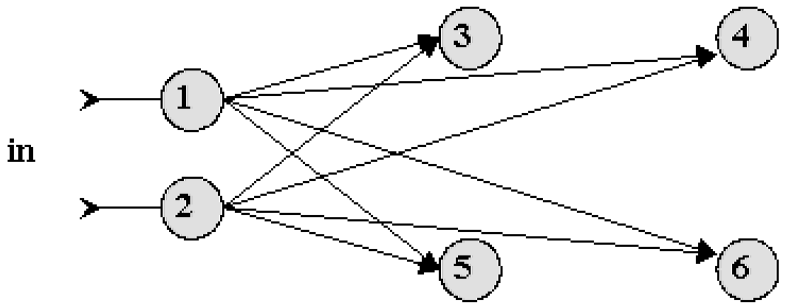

Abbildung 3.7: Kohonen - Map

Möglich sind auch alle anderen denkbaren Kombination, verbreitet sind vorallem solch einfache Implementierungen wie in Abb. 3.5-3.7, da hierfür zahllose Lernalgorithmen zur Verfügung stehen. Außerdem steigt der Lernaufwand ganz erheblich mit der Anzahl der Rückkopplungszweige.

3.3 Lernalgorithmen

Für einen sinnvollen Einsatz von Neuronalen Netzen, muß zuerst die gewünschten Informationen in das Netz eingebracht werden. Um dies zu ermöglichen, gibt es verschiedene Eingriffsmöglichkeiten in das Netz :

1. Änderung der Kopplung ($\mathbf{W}$) zwischen den Neuronen
2. Änderung der Schwellwerte (θ) der Neuronen
3. Modifikation der Aktivierungs -, Propagierungs - oder Ausgabefunktionen (f, z, g)
4. Erzeugung zusätzlicher Neuronen
5. Löschen von vorhandenen Neuronen

Überwiegend wird eine Änderung der Kopplung ($\mathbf{W}$) zwischen den Neuronen bevorzugt, da dies algorithmisch am einfachsten und effizientesten zu handhaben ist. Werden die Schwellwerte der Neuronen durch spezielle On-Neuronen repräsentiert, werden diese automatisch mit der Gewichtsmatrix variiert. Die Algorithmen 3, 4 und 5 werden selten angewandt, sodaß sie im folgenden nicht mehr erwähnt werden.

Die Lernmethoden, mit denen das repräsentierte Wissen in die Neuronale Netze eingebracht werden soll, lassen sich in folgende Kategorien einteilen :

- Überwachtes Lernen
- Bestärkendes Lernen
- Unüberwachtes Lernen

Beim *überwachten Lernen* wird dem Netz das Eingabe - ($\mathbf{i}_i$) und auch das Ausgabemuster ($\mathbf{t}_i$) präsentiert. Das Netz berechnet dann aus der Eingabe seine Ausgabe ($\mathbf{o}_i$) , die Differenz ($\mathbf{t}_i - \mathbf{o}_i$) zwischen diesem Ergebnis und dem Ausgabemuster wird dazu benutzt, das Netz mittels einer oben angeführten Möglichkeit zu verbessern. Der Mittelwert der Differenz ist zusätzlich ein Maß für die Güte des Gelernten. Wird ein neuronale Netz nach Abb. 3.5 durch Gl. 3.2 beschrieben, so lernt es eine vektorielle Funktion mit $f(\mathbf{i}) = \mathbf{t}$, wobei das Netz die Funktion nicht mathematisch exakt nachbildet, es verbleibt meist ein gewisser Restfehler.

Beim *bestärkenden Lernen* wird das Ausgabemuster nicht mit angegeben, es werden nur Aussagen über richtige oder falsche Repräsentation gegeben.

Beim *unüberwachten Lernen* werden dem Netz nur die Eingabemuster präsentiert. Das Netz muß sich anhand der Eingaben selber organisieren, ohne daß von außen eingegriffen werden kann.

3.3.1 Lernen für feedforward - Netze

Die allgemeine Hebbsche Lernregel bildet die Grundlage für viele anderen Lernregeln. Sie lautet : *Ist das Neuron i mit dem Neuron j durch das Gewicht* w_{ij} *verbunden und sind beide Neuronen gleichzeitig stark aktiviert, so wird das Gewicht folgendermaßen verändert :*

$$\Delta w_{ij} = \eta * out_i * Z_j \tag{3.4}$$

η : Lernrate ($= const$)
out_i : Ausgabe des Neurons
Z_i : Aktivierung des Neurons

Die wohl bekannteste, davon abgeleitete Lernregel ist die allgemeine Backpropagations - Regel

$$\Delta w_{ij} = \eta * out_i * \delta_j \tag{3.5}$$

$$\delta_j = \begin{cases} z'_j[\mathrm{Z}_j(\mathrm{t}), \theta_j, f_j[\mathrm{in}]] * (\mathrm{t}_j - \mathrm{out}_j) & \text{, falls j Ausgabezelle} \\ z'_j[\mathrm{Z}_j(\mathrm{t}), \theta_j, f_j[\mathrm{in}]] * \sum_k (\delta_k \mathrm{w}_{jk}) & \text{, falls j verdeckte Zelle} \end{cases}$$

$\mathbf{t}_j$: Teach Input
$\sum_k$: Summe über alle Nachfolgeneuronen

Die Formel für δ_j läßt eine Rekursion für die hidden layers erkennen, daher rührt auch der Name für diese Regel, die Aktität eines Neurones wird an die vorangenden Schichten "zurückgereicht". Der einzigste Unterschied zur Hebbschen Lernregel ist, daß die Aktivierung der Neuronen in den verdeckten Schichten erst berechnet werden muß.

Genaugenommen minimieren alle Lernverfahren die durch die Gewichtsmatrix $\mathbf{W}$ definierte Fehlerfläche $\sum_i \mathbf{t}_i - \mathbf{o}\left(\mathbf{W}\right)_i$.

3.3.2 Lernen für Kohonen-Karten

Kohonen-Karten sind Vertreter des unüberwachten Lernens, d. h. es werden nur Eingangsvektoren angelegt und der Lernalgorithmis sorgt dann für die Selbsorganisation des Netzwerks.

Die Idee der Kohonen-Karte (Kohonen-map) entstammt dem biologischen Vorbild Gehirn. Dort wurden sensorische Bereiche entdeckt, in denen benachbarte Sensoren auch benachbart abgebildet sind, nicht aber maßstabgetreu. Die Dimension der Karte richtet sich nach der Dimension des Problems. Handelt es sich z.B. um ein 3. dimensionales Problem, welches durch n Parameter charakterisiert ist, hat die Kohonen - Karte n Eingänge, die an ein 3. dimensionales Netzwerk angeschlossen sind. Es ist aber auch möglich, Netzwerke mit niedrigerer Dimension zu verwenden, wobei die Nachbarschaft der Eigabevektoren nicht unbedingt erhalten bleibs. Die spezielle Topologie des verwendeten Netzwerks ist von untergeordneter Bedeutung, sodaß aufgrund der einfacheren Implementation häufig ein einfaches quaderförmiges Netzwerk verwendet wird.

Für den Lernalgorithmus benötigt man eine beliebige Nachbarschaftfunktion und eine beliebige Norm, wobei es sich aber gezeigt hat, daß eine stetige Nachbarschaftfunktion zu bevorzugen ist.

n : Anzahl der Eingänge
k : Anzahl der Neuronen in der Karte
$\| \ \|$: beliebige Norm, meist euklidische Norm
$\eta\left(\mathrm{t}\right)$: Lernrate mit $0 \leq \eta\left(\mathrm{t}\right) \leq 1$
$\mathrm{h}_{cj}\left(\mathrm{t}\right)$: normiert Nachbarschaftsfunktion für Neuron j mit
$0 \leq h_{cj}(t) \leq 1$ und $h_{cj}\left(\mathrm{t}\right) = h\left(t, \|\mathbf{r}_c - \mathbf{r}_j\|\right)$
$\mathbf{in} = \left(\mathrm{in}_1, \ldots, \mathrm{in}_n\right)$: n dimensionaler Eingabevektor

$\mathbf{w}_j = (\mathbf{w}_{1,j}, \ldots, \mathbf{w}_{n,j})$: j-ter Spaltenvektor von $\mathbf{W}$

$$||\mathbf{in} - \mathbf{w}_c|| = \min_{j=1}^{k} ||\mathbf{in} - \mathbf{w}_j||$$

Damit ist c das Neuron in der Karte, dessen Gewichtsvektor am nächsten am Eingangsvektor liegt und somit als Gewinnerneuron als einzigstes aktiviert wird.

Mit Hilfe von c, dem Gewinnerneuron, werden nun die Gewichtsvektoren aller Neuronen wie folgt verändert :

$$\mathbf{w}_j\,(\mathrm{t}+1) = \mathbf{w}_j\,(\mathrm{t}) + \mathrm{h}_{c,j}\,(\mathrm{t}) * [\mathbf{in}\,(\mathrm{t}) - \mathbf{w}_j\,(\mathrm{t})] \qquad \text{mit j} \in \{1 \ldots \mathrm{k}\}$$

Abb. 3.8 ist ein Beispiel für eine typische Nachbarschaftsfunktion für eine 2 dimensionale Kohonen - Karten, bei der das Gewinnerneuron mit $\mathrm{c} = (1,1)$ im Koordinatensystem (x, y) ausgewählt wurde

$$h_{(1,1),j=(x,y)}(t) = \frac{1}{\sigma(t)\sqrt{2\pi}} \exp\left(-\frac{\left((1-x)^2 + (1-y)^2\right)}{2\sigma\,(t)^2}\right)$$

$\sigma(t)$: Größe des Nachbarschaftsradiuses

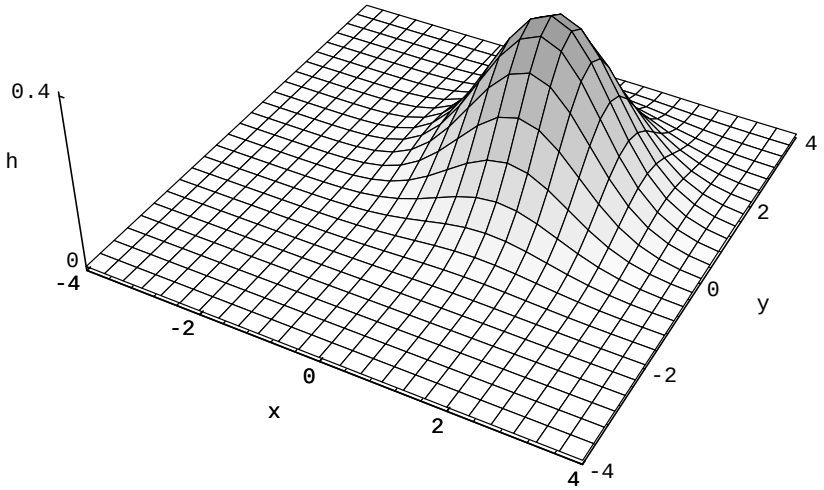

Abbildung 3.8: Nachbarschaftsfunktion für den Lernalgorithmus einer Kohonen-Karte

Bei genauer Kenntnis der Nachbarschaftsfunktion sollte der Algorithmus es vermeiden, alle Gewichtsvektoren zu verändern, da die Nachbarschaftsfunktion im weiteren Umfeld zu 0 werden muß, und der Rechenaufwand für größere Kohonen - Karten in mehreren Dimensionen uferlos wird. Damit obige Gleichung von Anfang an funktioniert, muß die Gewichtsmatrix mit zufälligen Werten gefüllt werden, da ansonsten alle Gewichte die selbe Distanz zu beliebigen Eingabevektoren haben.

Desweitern muß noch darauf hingewiesen werden, daß die Kohonen - Karte sich nicht immer so wie gewünscht strukturiert, es treten z.B. topologische Defekte auf, oder der Algorithmus divergiert aufgrund zu hoher Symmetrie im neuronalen Netzwerk [Zel94].

Kapitel 4

Neuronale Steuerung zielgerichteter Bewegungen

Um eine Steuerung für komplexe Bewegungen zu realisieren, wird die gestellte Aufgabe in überschaubare Teilaufgaben zerlegt. Die Zerlegung orientiert sich wieder am biologischen Vorbild, das Gehirn gibt einen “Sollwert” vor (z.B. ein Punkt im 3 dimensionalen Raum, zu dem die Bewegung hingerichtet sein soll, oder einen Bewegungsablauf), das Kleinhirn übernimmt diese Vorgaben und regelt über die entsprechenden Muskeln den Ablauf. Über sensorische Rezeptoren erhält das Kleinhirn Informationen über den aktuellen Zustand, sodaß es die Bewegung tatsächlich regeln kann.

In der Simulation werden die Aufgaben entsprechend Abb.4.1 verteilt.

- mechanische Simulation :

  Eine wie in Kapitel 2 eingeführte Simulation soll die physikalischen Sachverhalte wiederspieglen.

- Regler :

  Der Regler minimiert die Differenz zwischen dem Zustand des mechanischen Systems und seiner Sollwertvorgabe (Regeldifferenz). Aus der Regeldifferenz werden die generalisierten Kräfte für die mechanische Simulation errechnet.

- Koordinatentransformation :

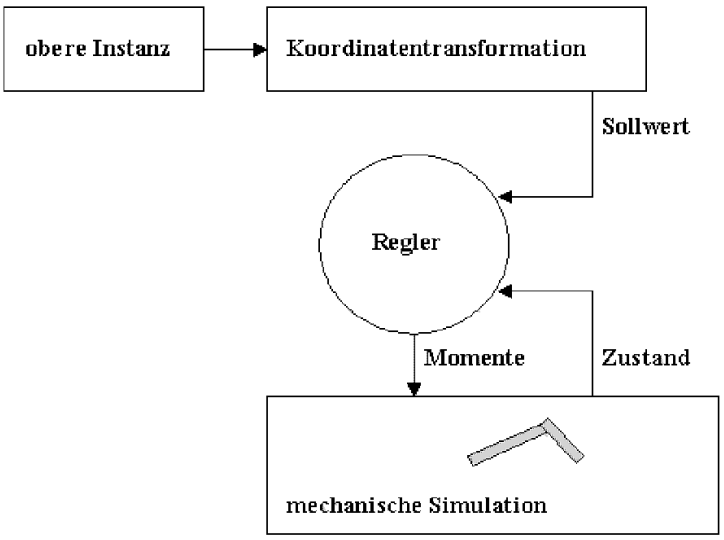

Abbildung 4.1: Block - Plan

Die Sollwertevorgaben werden hierdurch in das Koordinatensystem des Starrkörpermodells transformiert. Für jeden Freiheitsgrad des mechanischen Modells muß durch die Transformation eine Vorgabe vorhanden sein. Ist dies nicht der Fall, muß durch einen anderen Algorithmus eine Vorgabe geschaffen werden (z.B. beibehalten der alten Vorgabe, oder Vorgabe so ändern, daß die Energie des Systems minimal bleibt).

- obere Instanz :

  Sie hat nur die Aufgabe, Sollwerte zu liefern. Ein Sollwert definiert einen Punkt in einem 'gut zugänglichen' Bezugssystem wie z.B. dem kartesischen- oder dem polaren- Koordinatensystem. Die Anzahl der durch den Sollwert definierte Freiheitsgrade kann kleiner sein als die Anzahl der Freiheitsgrade des mechanischen Systems (siehe Kapitel 4.2).

4.1 Mechanische Simulation

Als Modell wird eine 2-gliederige Starrkörperkette verwendet, deren Aufpunkt raumfest ist. Der erste Starrkörper hat 3 Freiheitsgrad, der zweite Körper hat 1 Freiheitsgrad, alle

Freiheitsgrade sind rotatorischer Natur.

Die Starrkörperbeschreibung mit der in Kap. 2.5 eingefürten Syntax :

K[1][$initlage$][1] = (0.3, 0, 0);
K[1][$initschwerpunkt$] = (0.15, 0, 0);
K[1][$potential$] = (- K[1][$masse$] * K[1][$schwerpunkt$] . g);
K[1][$aufpunkt$] = (1.0, 1.0, 1.0);
K[1][$drehung$][1] = Rot[x, q1[t]];
K[1][$drehung$][2] = Rot[z, q2[t]];
K[1][$drehung$][3] = Rot[y, q3[t]];
K[1][$masse$] = 3.0
K[1][$trägheitstensor$] = Zylinder[m, r][1]

K[2][$initlage$][1] = (0.3, 0, 0);
K[2][$initschwerpunkt$] = (0.15, 0, 0);
K[2][$potential$] = (- K[2][$masse$] * K[2][$schwerpunkt$] . g);
K[2][$aufpunkt$] = K[1][$position$][1];
K[2][$drehung$][1] = K[1][$drehung$][1];
K[2][$drehung$][2] = K[1][$drehung$][2];
K[2][$drehung$][3] = K[1][$drehung$][3];
K[2][$drehung$][4] = Rot[y, q4[t]];
K[2][$masse$] = 2.5
K[2][$trägheitstensor$] = Zylinder[m, r]

tbegin = 0.0
tend = 2.0 bzw. 15.0
qstart = (Pi/2, 0.0, 0.0)
qpstart = (0.0, 0.0, 30.0)

Das Modell aus Abb. 4.2 ist dargestellt mit den Werten (q1[t],q2[t],q3[t],q4[t]) : (Pi/2, Pi , 0.0, Pi/3), der raumfeste Aufpunkt liegt bei (1, 1, 1)

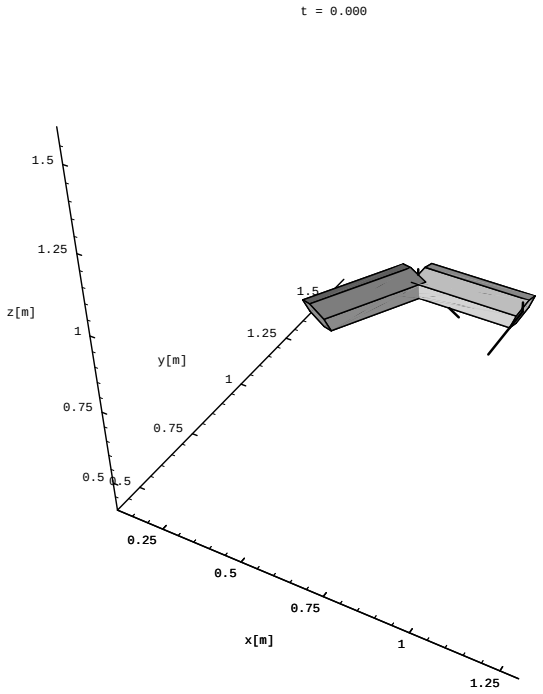

Abbildung 4.2: Verwendetes Armmodell

4.2 Neuronaler Regler

Die Regelung technischer Bewegungsabläufe wird im allgemeinen mit PID-Reglern bewerkstelligt. Da die Anwendung von Reglern in mechanischen Modellen gut durchführbar ist, wird die neuronale Steuerung einem PD-Regler nachempfunden. Anschließend wird ein Feedforward-Netz mit den Daten des PD-Reglers belernt.

Definition der Reglereigenschaften :

- P-Regler (Proportional) $\mathbf{Q} = f_{\mathrm{P}}\left(\mathbf{x}_{\mathrm{ist}}\left(\mathrm{t}\right), \mathbf{x}_{\mathrm{soll}}\left(\mathrm{t}\right)\right)$
  Er erzeugt eine Kraft in Abhängigkeit von der Abweichung zur Sollgröße. Der einfachste P-Regler wird durch eine lineare Feder realisiert mit $\mathbf{Q} = -\alpha\left(\mathbf{x}_{ist}\left(t\right) - \mathbf{x}_{soll}\right)$, daher rührt auch der Name.

- I-Regler (Integral) $\mathbf{Q} = \int_{\mathrm{t_0}}^{\mathrm{t}} f_{\mathrm{I}}\left(\mathbf{x}_{\mathrm{ist}}\left(\mathrm{t}\right), \mathbf{x}_{\mathrm{soll}}\left(\mathrm{t}\right)\right)$
  Der I-Regler hat die Aufgabe die Reglerdifferenz im zeitlichenen Mittel zu minimieren. Er ist vorallem dafür verantwortlich den Restfehler des P-Reglers zu kompensieren.

- D-Regler (Differential) $\mathbf{Q} = f_{\mathrm{D}}\left(\dot{\mathbf{x}}_{\mathrm{ist}}(\mathrm{t}), \mathbf{x}_{\mathrm{ist}}(\mathrm{t}), \mathbf{x}_{\mathrm{soll}}(\mathrm{t})\right)$
  Eventuell auftretende Systemschwingungen werden durch den D-Regler bedämpft. Die einfachste Implementierung eines D-Regler ist eine konstante Reibung .

$\mathbf{x}_{ist}$: aktueller Zusand der zu regelnden Variablen (ist-Wert)
$\dot{\mathbf{x}}_{ist}$: Ableitung des aktuellen Zusands
$\mathbf{x}_{soll}$: Zusand, den $\mathbf{x}_{ist}$ erreichen soll (soll-Wert)
$\mathbf{Q}$: Verallgemeinerte Kraft
α : Parameter des Proportionalanteils des Reglers
β : Parameter des Differentialanteils

Als Regler wurde ein einfacher PD-Regler gewählt, da der Integralanteil nur den Restfehler der P-Regelung verkleinert, die Regelung aber nicht auf besondere Präzision ausgelegt ist.

$$\mathbf{Q}\left(\mathbf{x}_{\mathrm{ist}}, \mathbf{x}_{\mathrm{soll}}, \dot{\mathbf{x}}_{\mathrm{ist}}\right) = -\alpha * \left(\mathbf{x}_{\mathrm{ist}} - \mathbf{x}_{\mathrm{soll}}\right) - \dot{\mathbf{x}}_{\mathrm{ist}} \left(\frac{1}{\beta + \left(\mathbf{x}_{\mathrm{ist}} - \mathbf{x}_{\mathrm{soll}}\right)^2}\right) \qquad (4.1)$$

Der D-Anteil des Reglers wurde so formuliert, daß die Dämpfung erst beim Annähern an den Sollwert nennenswert wird. Auf diese Art wird der Sollwert schneller erreicht.

$$\mathbf{Q}\left(\mathbf{x}, \dot{\mathbf{x}}\right) = -\alpha_1 * (\mathbf{x}) - \alpha_2 * \tanh\left(\alpha_3 \mathbf{x}\right) - \dot{\mathbf{x}} \left(\frac{1}{\beta_1 + \beta_2 \mathbf{x}^2}\right) - \frac{\sinh\left(\gamma_1 \dot{\mathbf{x}}\right)}{\gamma_2} \qquad (4.2)$$

$\mathbf{x}$ = $\mathbf{x}_{ist} - \mathbf{x}_{soll}$
$\dot{\mathbf{x}}$ = $\dot{\mathbf{x}}_{ist}$
γ_i : Parameter der Geschwindigkeitsbegrenzung

Die Gleichung 4.2 stellt die verbesserte Version von Gleichung 4.1 dar. Sie erlaubt es, die Maximaleschwindigkeit der Bewegung zu begrenzen. Diese Eigenschaft wurd in Hinsicht auf die menschliche Bewegung implementiert, da er, falls es nicht erforderlich ist, Bewegungen zu einer neuen Positionen hin, nicht besonders schnell ausführt.

Das Regelfeld aus Abb.4.3 wird dem neuronalen Netz aus Abb.4.4, mittels 6000 zufällig ausgewählter und skalierter Datensätze trainiert. Im Umfeld neuronaler Netze wird dieser

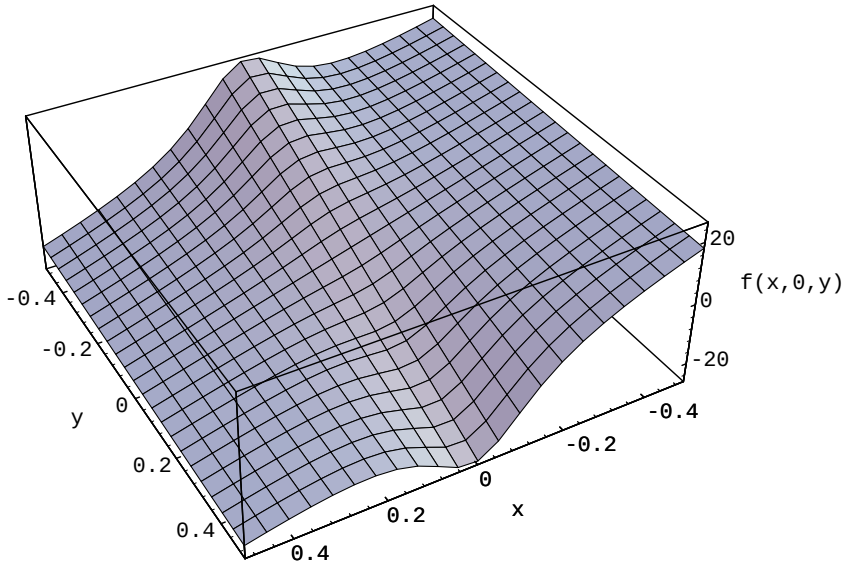

Abbildung 4.3: Das Kennfeld des Reglers von Gleichung 4.1 mit den Konstanten $\alpha = 40.0,\ \beta = 0.02$

Datensatz auch als Muster oder Pattern bezeichnet. Als Lernalorithmus wird Standard-Back-Propagation gewählt, alle Muster werden dann innerhalb von 10000 Lernzyklen mit einem Restfehler von durchschnittlich kleiner 1 % reproduziert.

Die Eingabe des Feedforward-Netz setzt sich genau wie Eingabe von Gl. 4.1 zusammen. Da das neuronale Netz auf die logistische Aktivierungsfunktion von Gl. 3.2 aufbaut, ist es für das Netzwerk in erster Linie unmöglich, negative Momente auszugeben. Um dies zu ermöglichen, wird auf das Flexor-Extensor-Modell (Beuger - Strecker) zurückgegriffen. Somit ergibt sich für die ausgegebenen Momente des neuronalen Netzes:

$$\mathrm{Q}_i = \delta_i * (\mathrm{o}_1 - \mathrm{o}_2) \tag{4.3}$$

Q_i : generalisierte Kraft für die Funktionen $\mathrm{q}_i\ (t\,)$

δ_i : Maxinalmoment des jeweiligen Freiheitsgrades
Für die Positionierung wurde $\delta_{1\ldots4}$ auf $\{300, 280, 260, 260\}$ gesetzt. Dies bedeutet, daß das maximal auftretende Drehmoment bei 300 Nm liegt.

In Abb. 4.6 bis 4.8 wird dem Regler eine einfach Positinieraufgabe gestellt. Dabei wurde für jeden Freiheitsgrad jeweils der identische Regler verwendet, um das zeitliche Verhalten des Reglers besser zu dokumentieren. So ist z.B. in Abb. 4.6 ein Überschwingen zu erkennen, welches sich durch anpassen der Parameter verhindern ließe. Dabei gilt es

aber zu beachten, daß die “wirksame träge Masse” direkt von der Position der einzelnen Segmente abhängt.

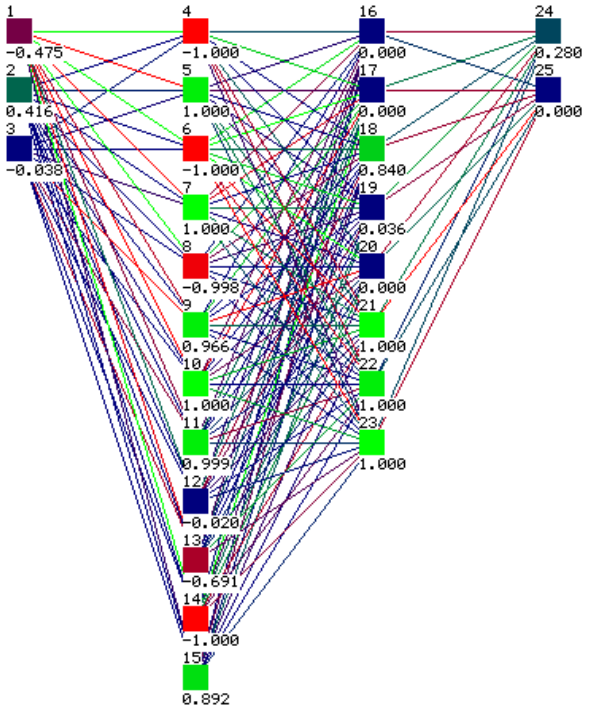

Abbildung 4.4: Feedforward-Netz des Reglers (Eingabe rechts, Ausgabe links)
In dieser Form stellt das Neuronale Netz, genau wie der Regler auch, einen Lageregler dar.

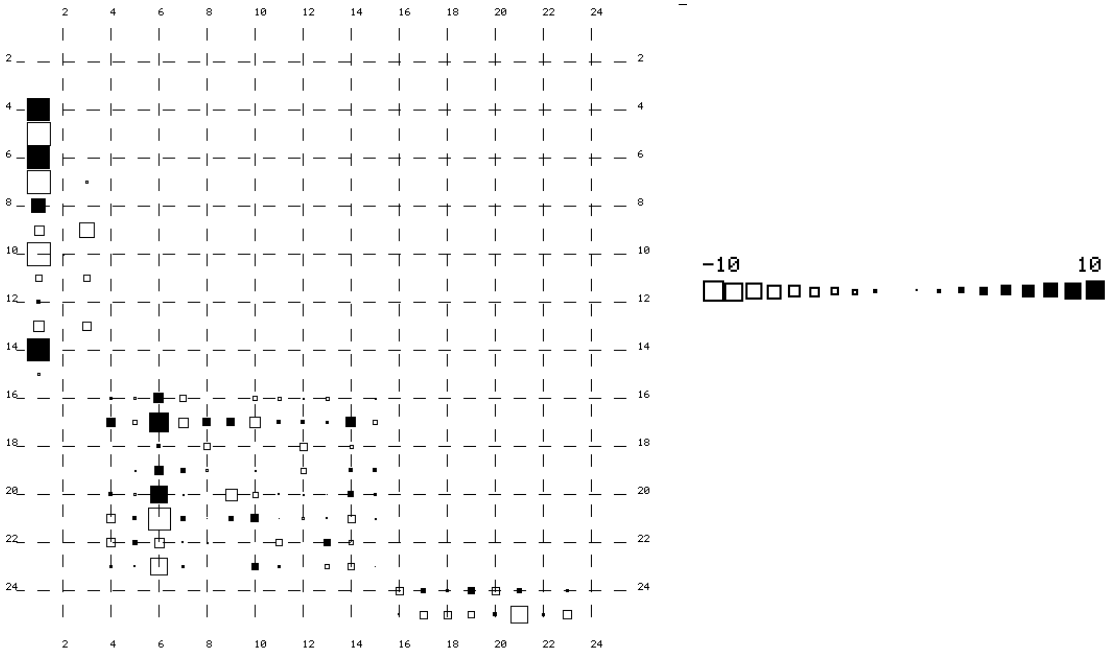

Abbildung 4.5: Die graphische Darstellung der Gewichtsmatrix des neuronalen Netzes aus Abb. 4.4

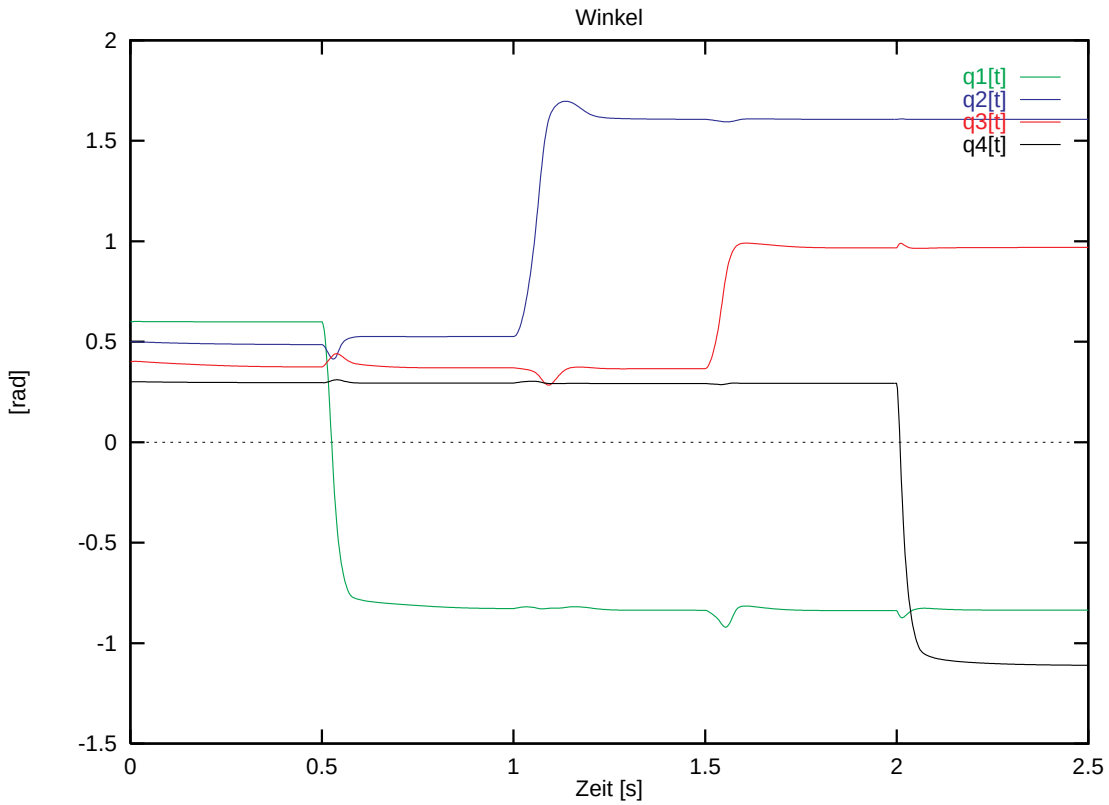

Abbildung 4.6: Neuronaler Regler: Die Winkelvariablen

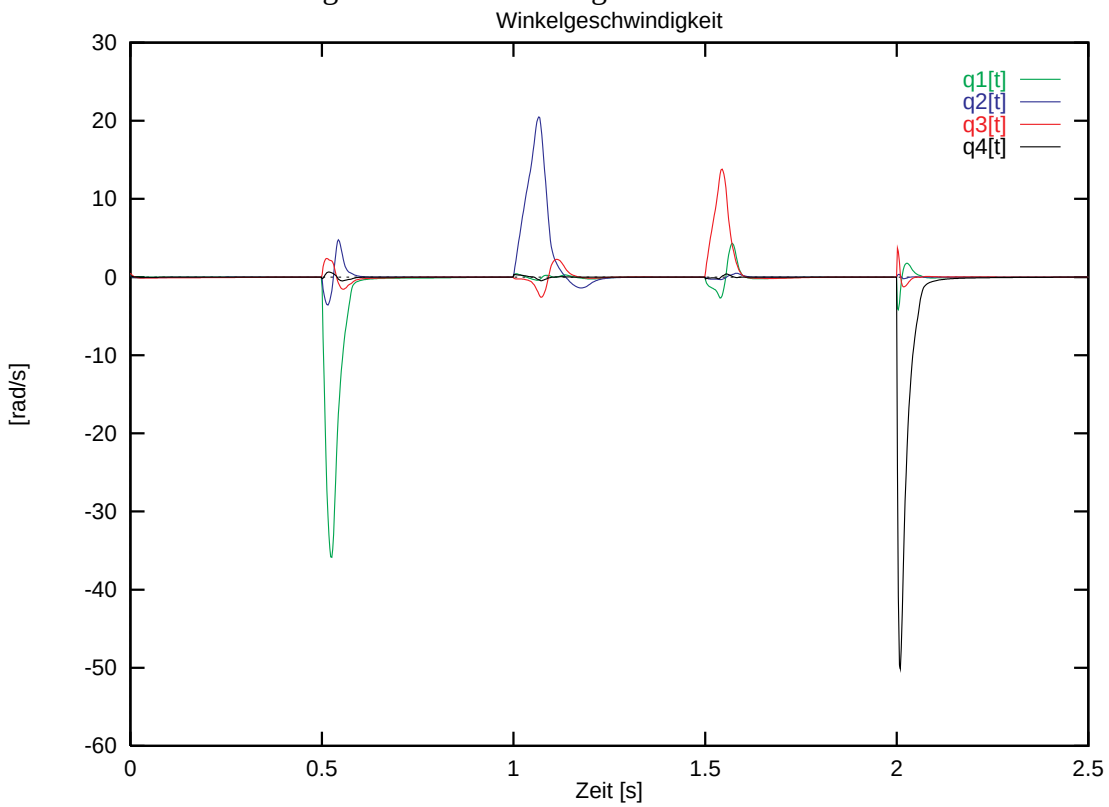

Abbildung 4.7: Neuronaler Regler: Die Ableitungen der Winkelvariablen

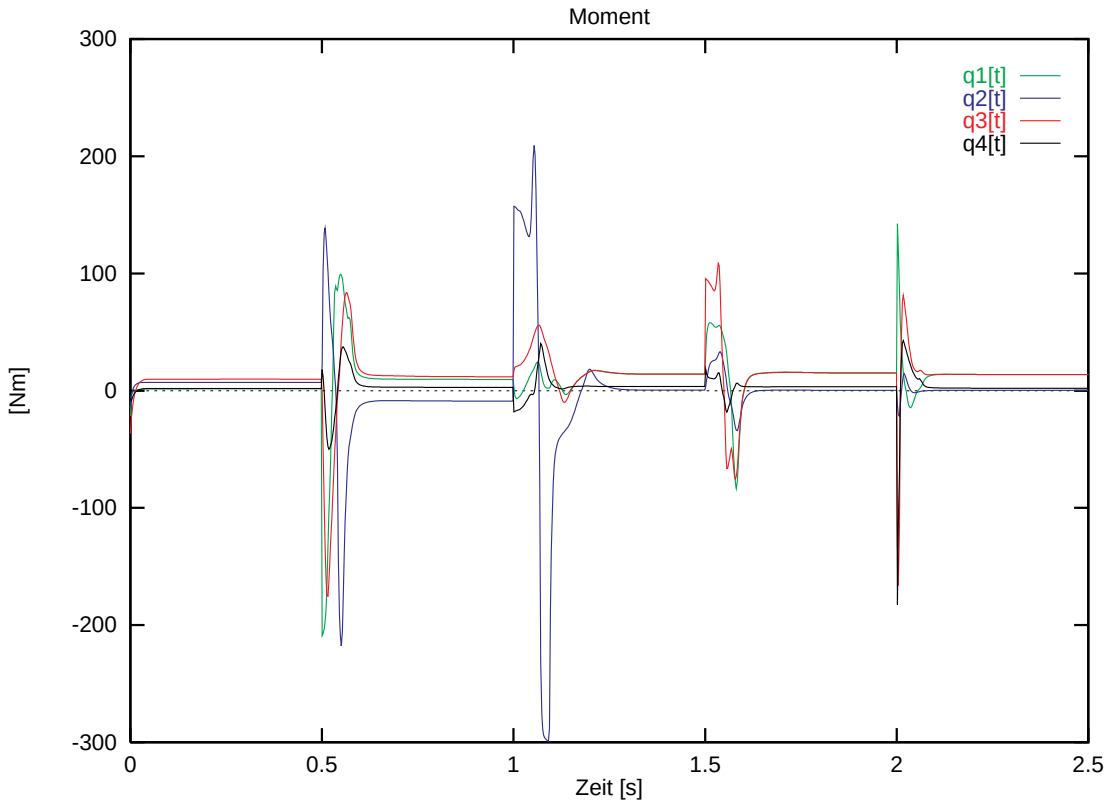

Abbildung 4.8: Neuronaler Regler: Das Drehmoment

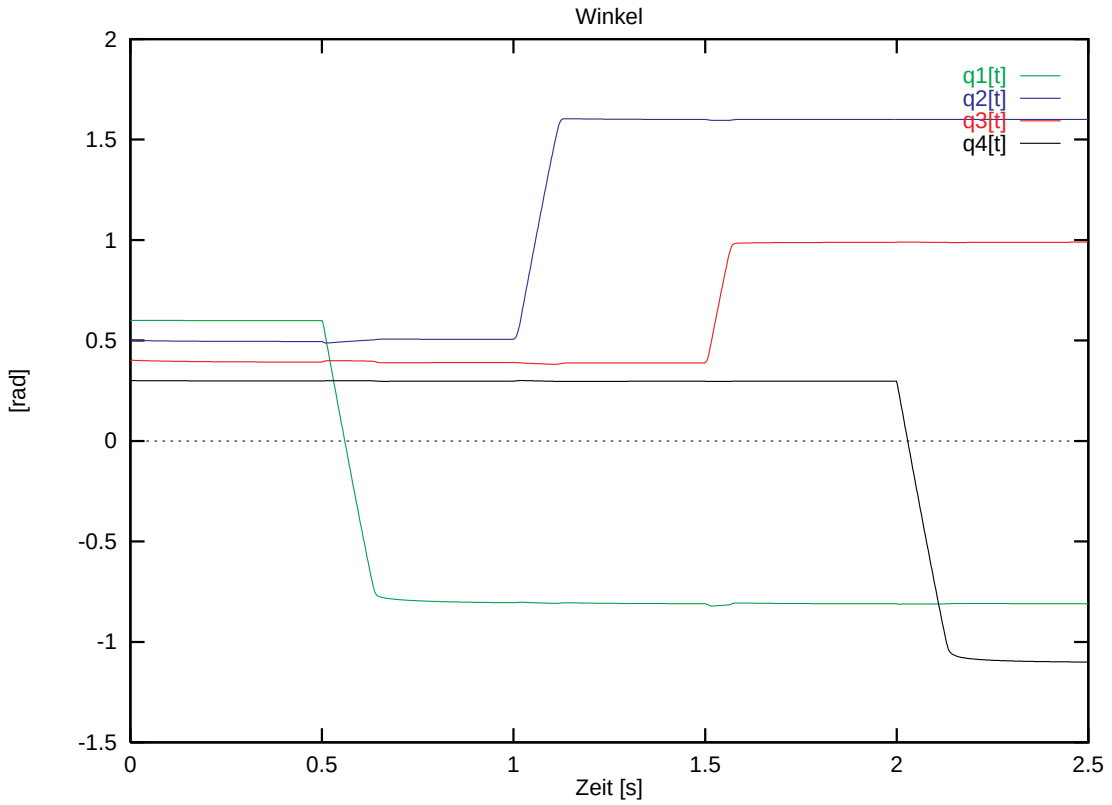

Abbildung 4.9: Geschwindigkeitsbegrenzter Neuronaler Regler: Die Winkelvariablen

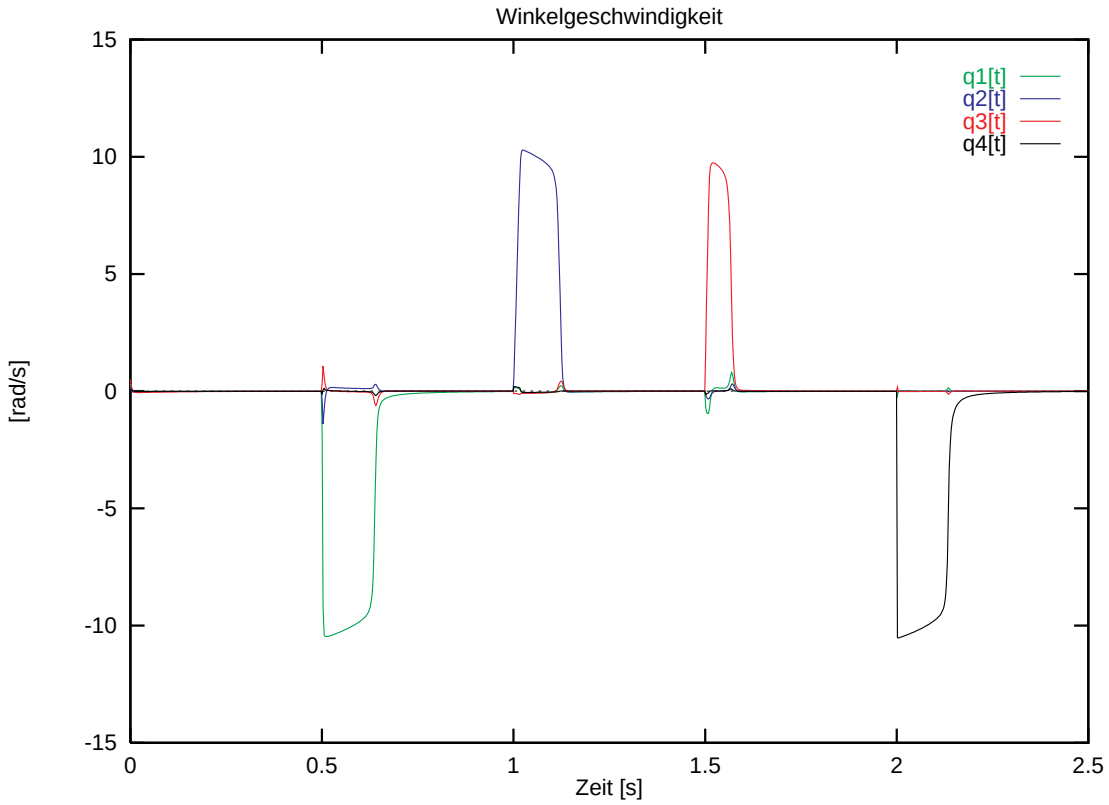

Abbildung 4.10: Geschwindigkeitsbegrenzter Neuronaler Regler: Die Ableitungen der Winkelvariablen

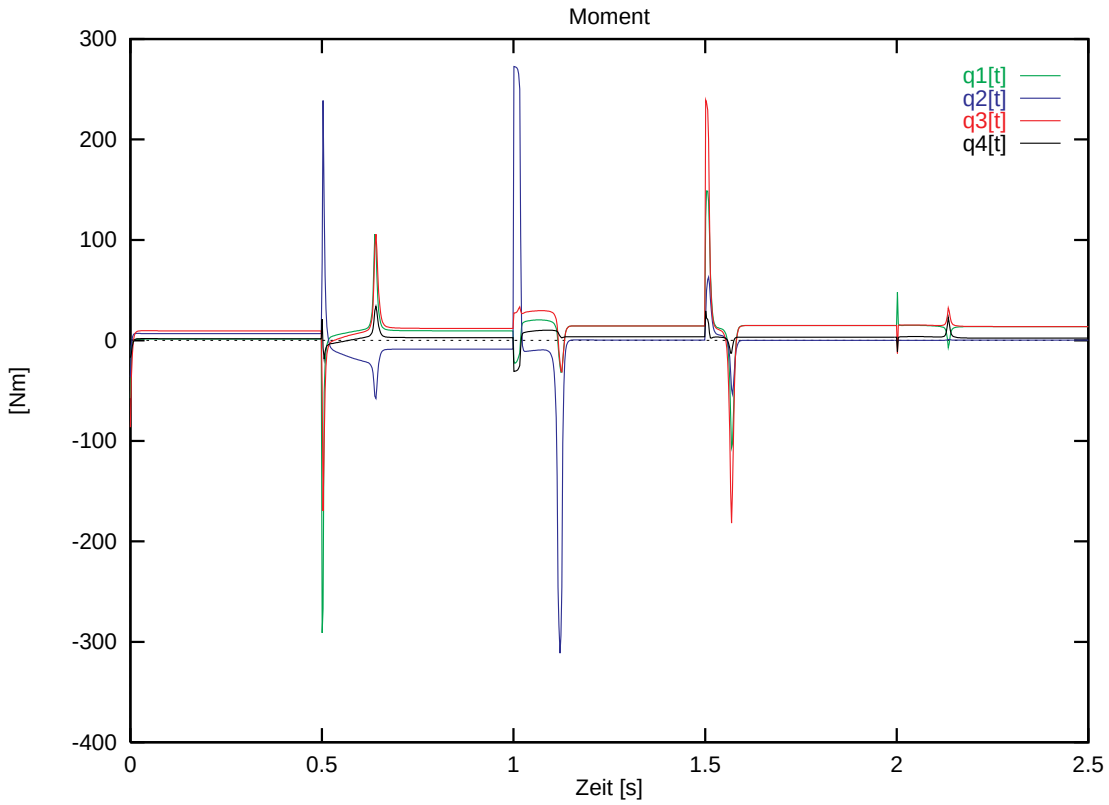

Abbildung 4.11: Geschwindigkeitsbegrenzter Neuronaler Regler: Das Drehmoment

4.3 Koordinatentransformation

Leider ist es völlig unbekannt, welche Strategien das Gehirn bei der Positionierung von Gliedmaßen, wie z.B. eines Armes, anwendet. Entsprechend grundsätzlich verschiedener Anforderungen (z.B. Greifen, Werfen, Stillhalten, Gehen usw.), die meist im laufe der Kindheit erst erlernt werden, treten bestimmte Eigenschaften in den Vordergrund, und andere in den Hintergrund. Beim Greifen zu einem bestimmten Punkt hin, ist die absolute Position der Zwischengelenke (z.B. des Ellenbogens) von untergeordneter Bedeutung. Beim Gehen indes wird der Kniewinkel durch die Vorderung einer effizienten und kraftsparenden Fortbewegung bestimmt.

Als evident gilt aber, daß im Gehirn ein topologisches Abbild der Außenwelt existiert [Koh84]. Um dieses Abbild in physikalische Gelekwinkel der Gliedmaßen umzurechnen, wird eine Koordinatentransformation eingesetzt. Bei der Transformation tritt das Problem auf, daß die Anzahl der mechanischen Freiheitsgrade der Gliedmaßen oft höher ist als die

für diese Aufgabe (z.B. Positionierung) benötigten Freiheitsgrade.

Die Überbestimmtheit der mechanischen Freiheitsgrade in Bezug auf die Positionierung des Endpunktes wird für die Simulation mittels folgender Möglichkeiten gelöst:

1. Vordefinierte Werte für die unbestimmten Freiheitsgrade

   Werden vordefinierte Werte verwendet, läßt sich Koordinatentransformation mit ein Feed-Forward-Netz realisieren. Dies kann aber nur zu einem richtigen Ergebnis führen, wenn die Koordinatentransformation eindeutig ist, also umkehrbar ist. Ist die Umkehrbarkeit nicht gegeben, wird der Fehler beim Lernen des Feed-Forward-Netz in Bezug auf das präsentierte Muster reduziert, d.h. das Netz stellt sich auf den Mittelwert der präsentierten Muster ein.

   Beispiel:

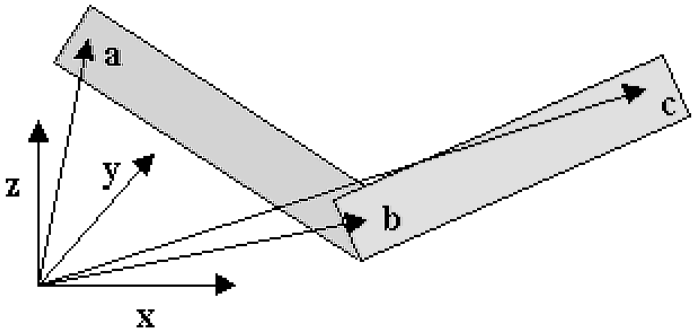

Abbildung 4.12: Freiheistgrade

$\mathbf{a}$: : const
$\mathbf{b}$, $\mathbf{c}$: : $f\,(\alpha, \beta, \gamma, \delta\,)$
α, β, γ, δ : : Freiheitsgrade der Körper

Dem Netz wird der Vektor $\mathbf{c}$ als Muster präsentiert, als Ausgabe wird α, β, γ, δ erwartet. Der Freiheitsgrad, der z.B. durch den Winkel der Ebene $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ beschrieben werden kann, bleibt unbestimmt, sodaß sich das neuronale Netz nicht korrekt anpassen kann.

2. Assoziativ-Speicher
   Ein Assoziativ-Speicher ist ein Speicher der beim Anlegen eines unvollständigen Eingabevektors trotzdem den vollständigen Ausgabevektor liefert. Eine mögliche Implementierung hiervon ist das Counterpropagation-Netz aus Abb. 4.13.

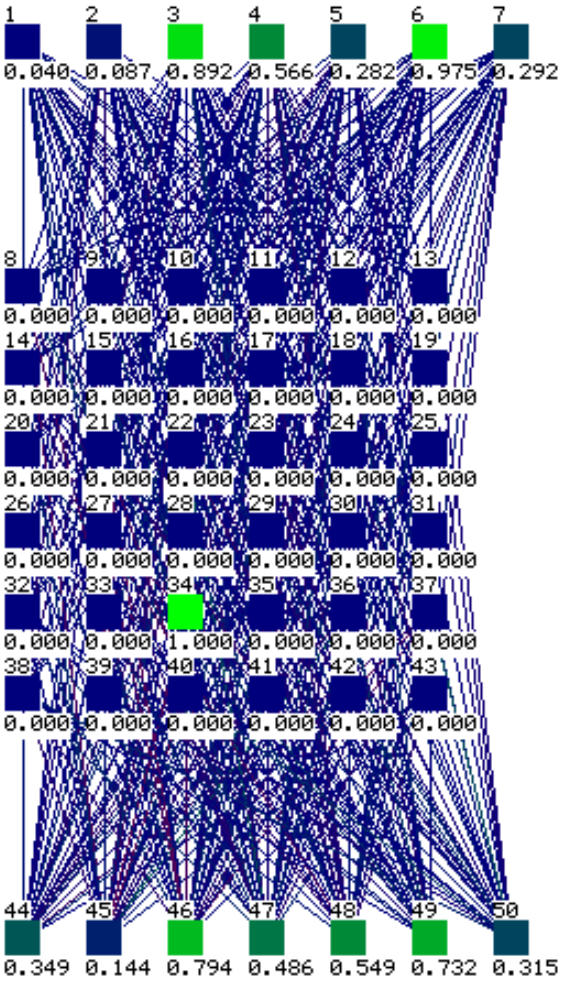

Abbildung 4.13: Counterpropagation-Netz

Das erste Layer des Counterpropagation-Netz entspricht einer Kohonen-Karte, das zweite Layer wird Grossberglayer genannt.

Wie in Kapitel 3.3.2 erläutert wurde, ist jedes Neuron der Kohonen-Karte für einen definierten Eingabebereich zuständig. Da gleichzeitig immer nur ein Neuron der

Karte aktiv ist, wird durch dieses eine Neuron die Eingabe vollständig repräsentiert. Diese Art der Abbildung wird als quantisierte Abbildung bezeichnet. Die Genauigkeit der Abbildung wird nur über die Anzahl der Neuronen einer Kohonen-Karte definiert. Für Abb. 4.13 bedeutet dies, das alle Eingabevektoren auf 36 quantisierte Raumsegmente" abgebildet werden.

Das Grossberglayer ist ein Feedforward-Netz, welches mittels z.B. Backpropagation so trainiert wird, daß es den ganzen Eingabevektor reproduziert.

Das Netz wird mit folgenden Ein- und Aus-gabevektoren trainiert:

$$\begin{aligned} \mathbf{in} &= (\mathbf{a}, \mathbf{b}) \qquad \text{mit } f(\mathbf{a}) = \mathbf{b} \\ \mathbf{out} &= (\mathbf{a}, \mathbf{b}) \end{aligned}$$

Da das Netz immer das Neuron der Kohonen-Karte aktiviert, welches die kleinste Distanz zum Eingabevektor beseitzt, sind auch unvollständige Eingaben möglich.

$$\begin{aligned} \mathbf{in} &= (\mathbf{a}, \mathbf{0}) \Rightarrow \mathbf{out} = (\mathbf{a}, \mathbf{b}) \\ \mathbf{in} &= (\mathbf{0}, \mathbf{b}) \Rightarrow \mathbf{out} = (\mathbf{a}, \mathbf{b}) \qquad \text{falls } \exists\, f^{-1} \end{aligned}$$

Dies bedeutet, daß **a** mit **b** assoziiert ist, und umgekehrt.

Damit ist es möglich, eine Assoziation zwischen dem Endpunkt des Armes, angegeben in Kugelkoordinaten und den 4 Freiheitsgraden zu schaffen. Mittels dieser 7 Eingabedaten wird ein Counterpropagation-Netz wie aus Abb. 4.13, bestehend aus 25 * 25 Kohonen-Neuronen, trainiert.

4.4 Simulation des Weitwurfes

Um die "Fähigkeiten" des simulierten Armes zu testen, wird an ihn die Aufgabe des Weitwurfes gestellt. Hierzu wird ein Minimierer eingesetzt, der die Wurfweite und die Richtung des Wurfes optimieren soll. Zur Variation stehen 4 verschieden Punkte als Sollwertvorgaben, von denen jeder die 4 Freiheitsgrade des Armes beinhaltet. Zusätzlich wird der Zeitpunkt dieser Sollwertvorgaben mitvariiert. Damit ist die Wurfweite- und -richtung eine Funktion mit 20 variablen Parametern. Versuche mit mehr als 4 verschiedenen Sollwertvorgaben zeigten keine deutlichen Verbesserungen bezüglich der Wurfweite, weshalb diese Anzahl beibehalten wurde.

Die Integrationszeit wurde, wegen des numerischen Aufwands, mit 0.5 s relativ kurz gewählt. Nach Ablauf der Integrationszeit gilt der 3. Körper (der zu werfende Körper) als frei, aus seiner, zu diesem Zeitpunkt erreichten, Schwerpunktsgeschwindigkeit wird die Wurfweite (w) und die Wurfrichtung (ϕ) berechnet. Der Nachteil, daß der Zeitpunkt des Abwurfes durch die Integrationsdauer festgelegt ist, wird durch die möglich Zeitverschiebung der Sollwertvorgaben wieder kompensiert.

$$\begin{aligned} \mathrm{w} &= \frac{\mathrm{r}^2 \sin(2\theta)}{\mathrm{g}} \\ \mathrm{r} &= \sqrt{{}_3\mathrm{v}_x^{s\,2} + {}_3\mathrm{v}_y^{s\,2} + {}_3\mathrm{v}_z^{s\,2}} \\ \phi &= \arctan\left(\frac{{}_3\mathrm{v}_y^s}{{}_3\mathrm{v}_x^s}\right) \\ \theta &= \arctan\left(\frac{\sqrt{{}_3\mathrm{v}_x^{s\,2} + {}_3\mathrm{v}_y^{s\,2}}}{{}_3\mathrm{v}_z^s}\right) \end{aligned} \tag{4.4}$$

$$\text{mit : } {}_3\mathbf{v}_I^s = \begin{pmatrix} {}_3\mathrm{v}_x^s \\ {}_3\mathrm{v}_y^s \\ {}_3\mathrm{v}_z^s \end{pmatrix}$$

Für das Erreichen eines "sinnvollen" biomechanischen Wurfes müssen die Parameter gewissen Einschränkungen unterworfen werden. Werden keine Einschränkungen gemacht, wird der Wurf auf extrem weites Ausholen, im Sinn von mehreren Umdrehungen um die z-Achse hin optimiert. Die Einschränkung einer begrenzten Bewegungsfreiheit wird durch die Verwendung eines Counterpropagation-Netz wie in Abb. 4.13 realisiert. Das

Netzwerk begrenzt nur die Sollwertvorgaben, bedingt durch die Dynamik des mechanischen Systems, können jedoch auch Werte außerhalb dieser Beschränkung erreicht werden Das Netzwerk wurde mit zufälligen Mustern aus dem folgenden Wertebereich trainiert:

$$\begin{array}{lcl} q_1\left(t_{\{1..4\}}\right) & \in & \left\{-\frac{\pi}{2}\ldots\pi\right\} \\ q_2\left(t_{\{1..4\}}\right) & \in & \left\{-\frac{\pi}{2}\ldots\pi\right\} \\ q_3\left(t_{\{1..4\}}\right) & \in & \left\{-\frac{\pi}{2}\ldots\pi\right\} \\ q_4\left(t_{\{1..4\}}\right) & \in & \{0\ldots\pi\} \end{array}$$

Die zweite Einschränkung begrenzt die maximale Distanz zwischen den Vorgabpunkten. Diese Vorderung soll verhindern, daß der Wurf nur durch extremes Ausholen seine Weite erzielt.

$$\mid q_j\left(t_i\right) - q_j\left(t_{i+1}\right) \mid < \frac{\pi}{4} \qquad \text{mit} \qquad i,j \in \{1\ldots 4\}$$

Die lezte Einschränkung betrifft die zeitlichen Vorgaben der Sollwerte. Diese sollen geordnet und innerhalb der vorgegebenen Integrationszeit liegen.

$$0.0 < t_1 < \ldots < t_4 < 0.5$$

Wird durch den Minimierer eine der Einschränkungen verletzt, werden Strafpunkte (penalties) verteilt, sodaß sich diese Einschränkungen auch durchsetzten. Die zu minimierende Funktion hat daher folgende Gestalt:

$$f = -\mathrm{w} + \left(\alpha\left(\phi - \phi_{\text{soll}}\right)\right)^2 + \text{penalty} \tag{4.5}$$

Mit $\phi_{\text{soll}} = 0$ ergibt sich ein Wurf in Richtung der negativen x-Achse.

Der Parameter α aus Gl. 4.5, bestimmend für die Wurfrichtung, wird bei Beginn der Minimierung auf 0 gesetzt und im Laufe der Optimierung auf Werte bis ca. 100 gesetzt. Dieses Vorgehen vermindert den Rechenaufwand beträchtlich, da zuerst eine beliebig orientierte Geschwindigkeit entwickelt wird, und diese mit wachsendem α in die richtige Richtung gedreht wird.

4.4.1 Ball-Weitwurf

Der erste Versuch besteht darin, einen Weitwurf mit einem Gewicht von 300 g durchzuführen. Der verwendete Regler entspricht dem aus Abb. 4.4, da der Einsatz einer Geschwindigkeitsbegrenzung beim Werfen nicht viel Sinn macht. Das maximal verfügbare Drehmoment nach Gl. 4.3 wurde auf 200 Nm begrenzt. Dies entspricht ungefähr der Leistungsfähigkeit eines sehr gut trainierten Menschen. Aufgrund des geringen zusätzlichen Gewichts folgt der Arm zügig den vorgegebenen Sollwerten.

Anhand von Abb. 4.15 läßt sich die Vorgehensweise des Minimierers in zwei Phansen einteilen. Dabei wird der Arm zuerst in eine günstige Ausgangsposition gebracht, um dann, in der zweiten, relativ kurzen Phase, beschleunigt zu werden.

Die Wurfweite ,berechnet nach Gl. 4.5, beträgt 59.03 $[\mathrm{m}]$.

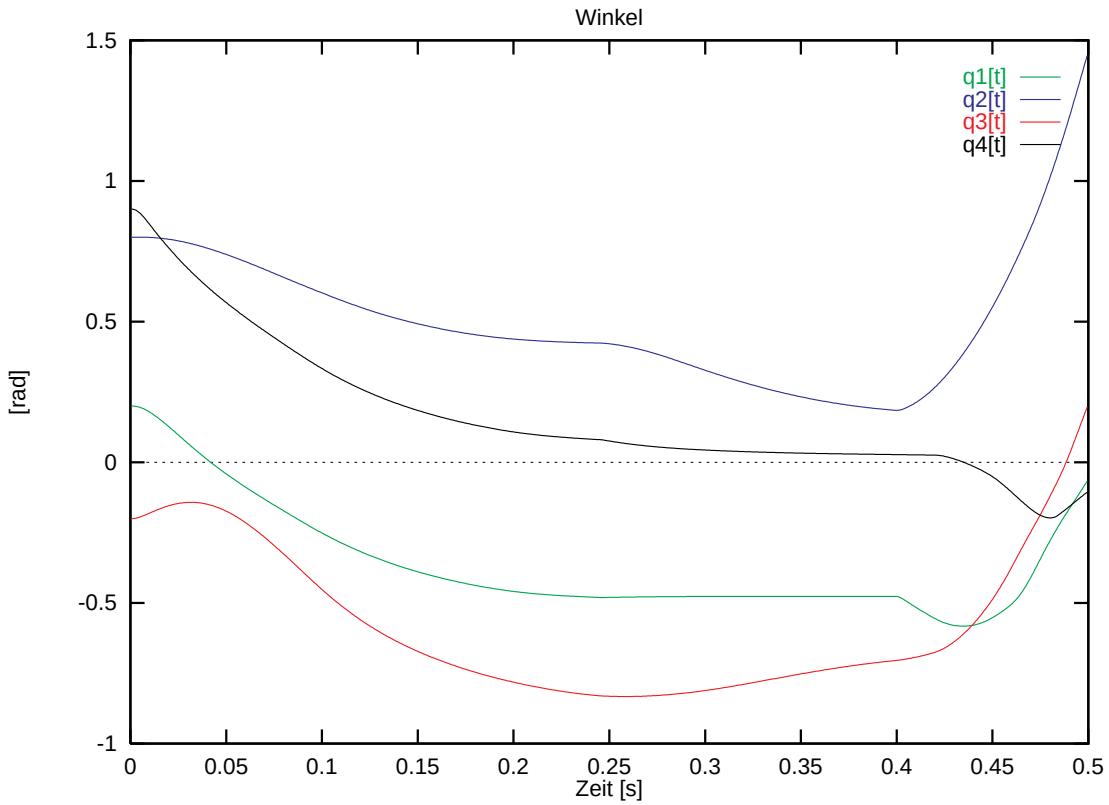

Abbildung 4.14: Weitwurf mit 0.3 Kg: Die Winkelvariablen

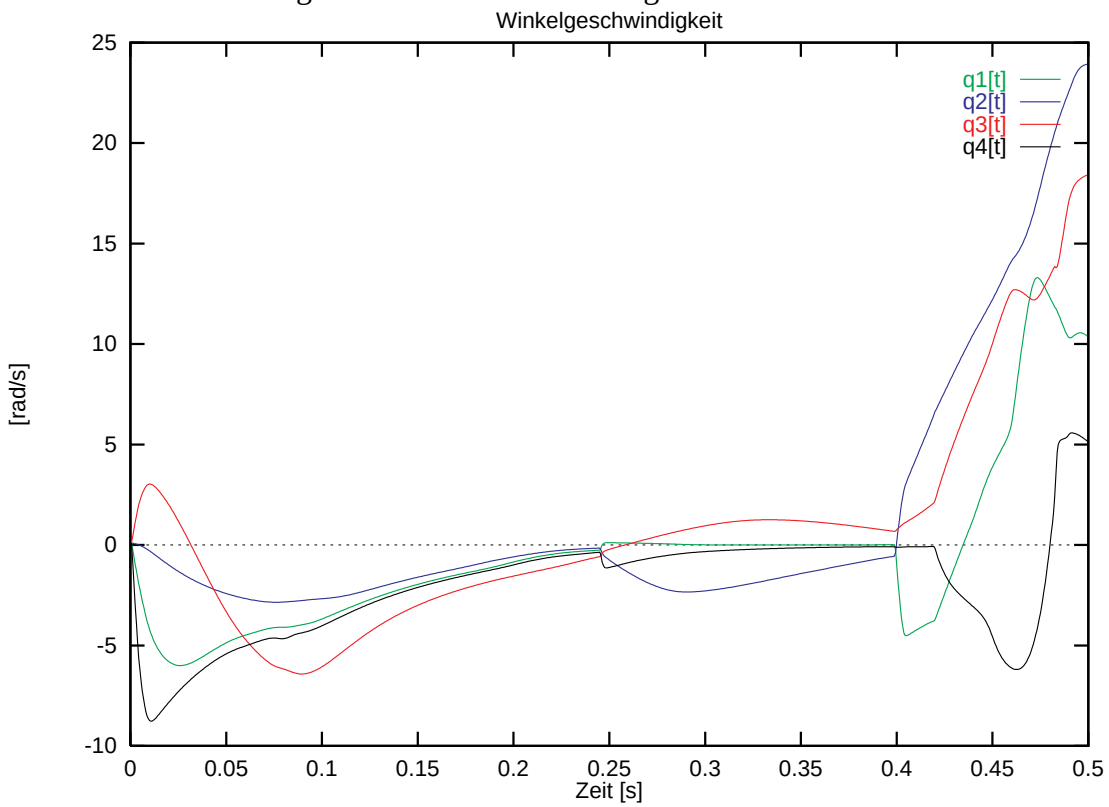

Abbildung 4.15: Weitwurf mit 0.3 Kg: Die Ableitungen der Winkelvariablen

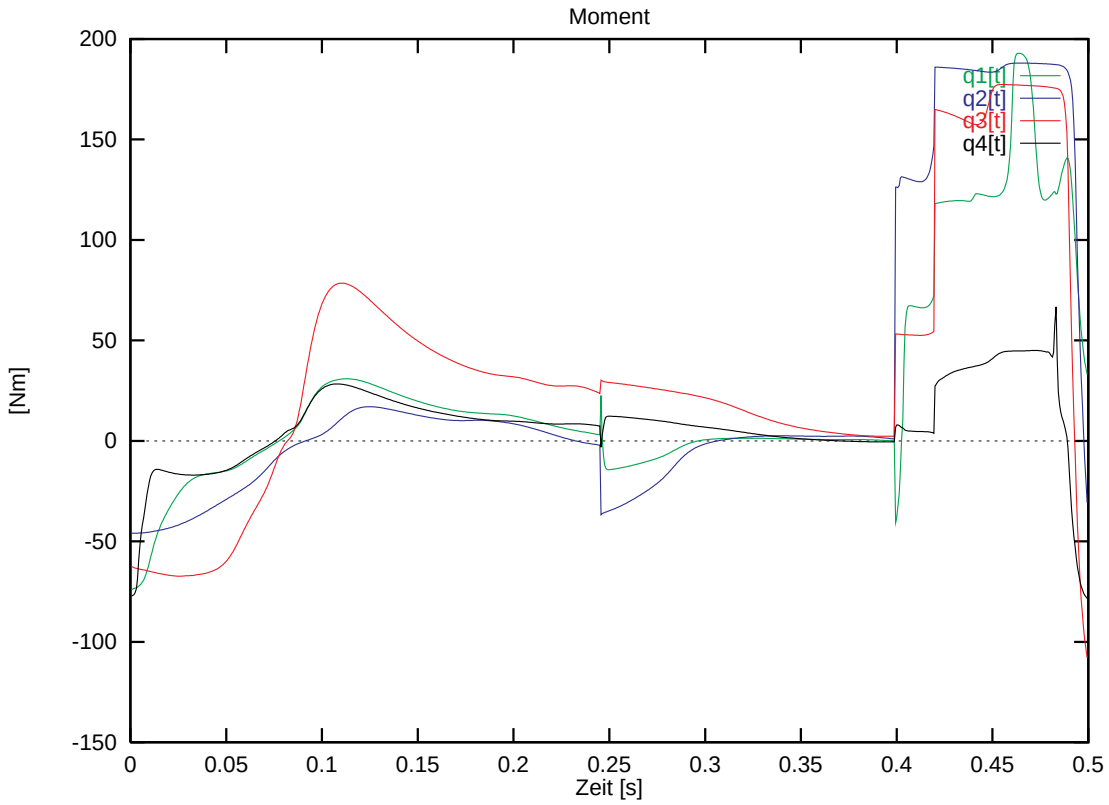

Abbildung 4.16: Weitwurf mit 0.3 Kg: Das Drehmoment

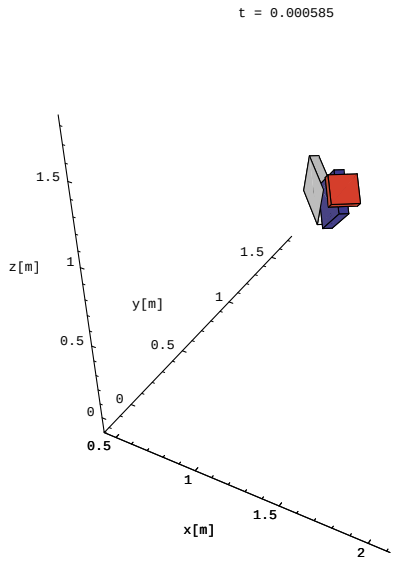

Abbildung 4.17: Weitwurf

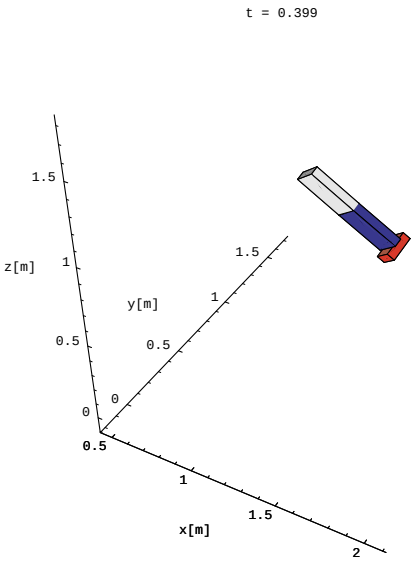

Abbildung 4.18: Weitwurf

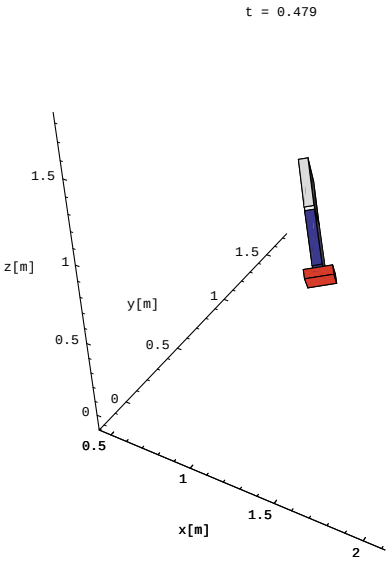

Abbildung 4.19: Weitwurf

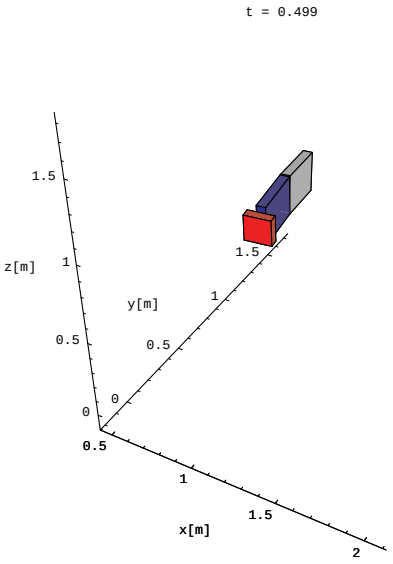

Abbildung 4.20: Weitwurf

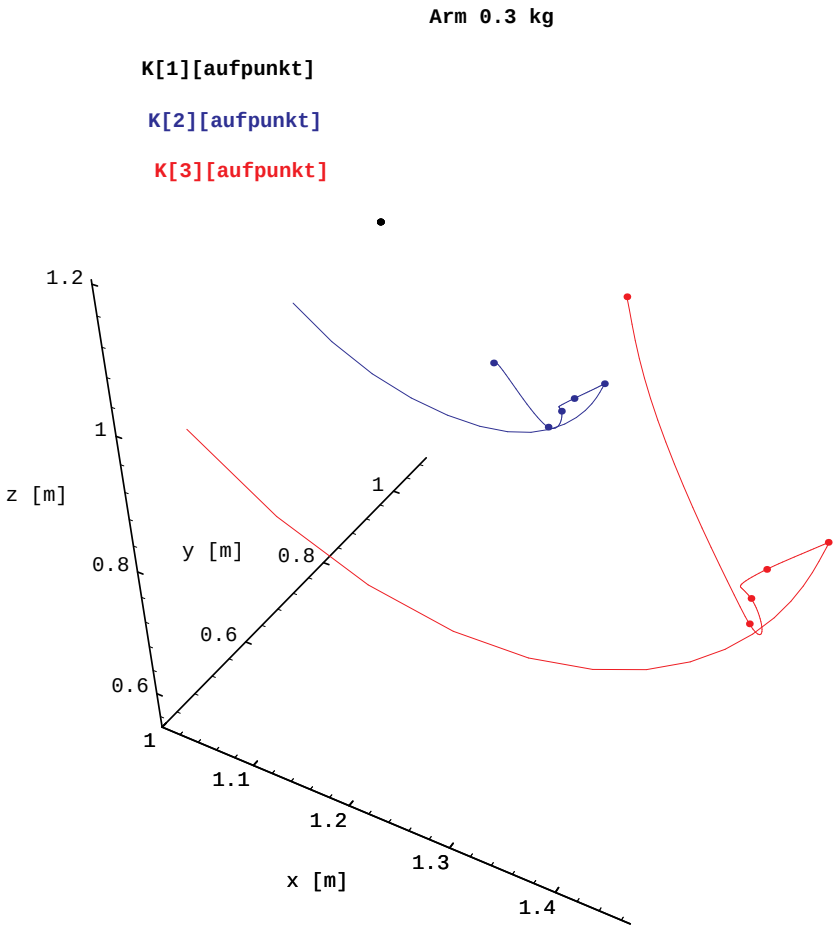

Abbildung 4.21: Weitwurf mit 0.3 Kg: Die Trajektorie

4.4.2 Kugelstoßen

Diese Simulation ist bis auf die mit 8 Kg größer Masse identisch mit der vorangegangenen.

Die Wurfweite beträgt 10.35 $[\mathrm{m}]$.

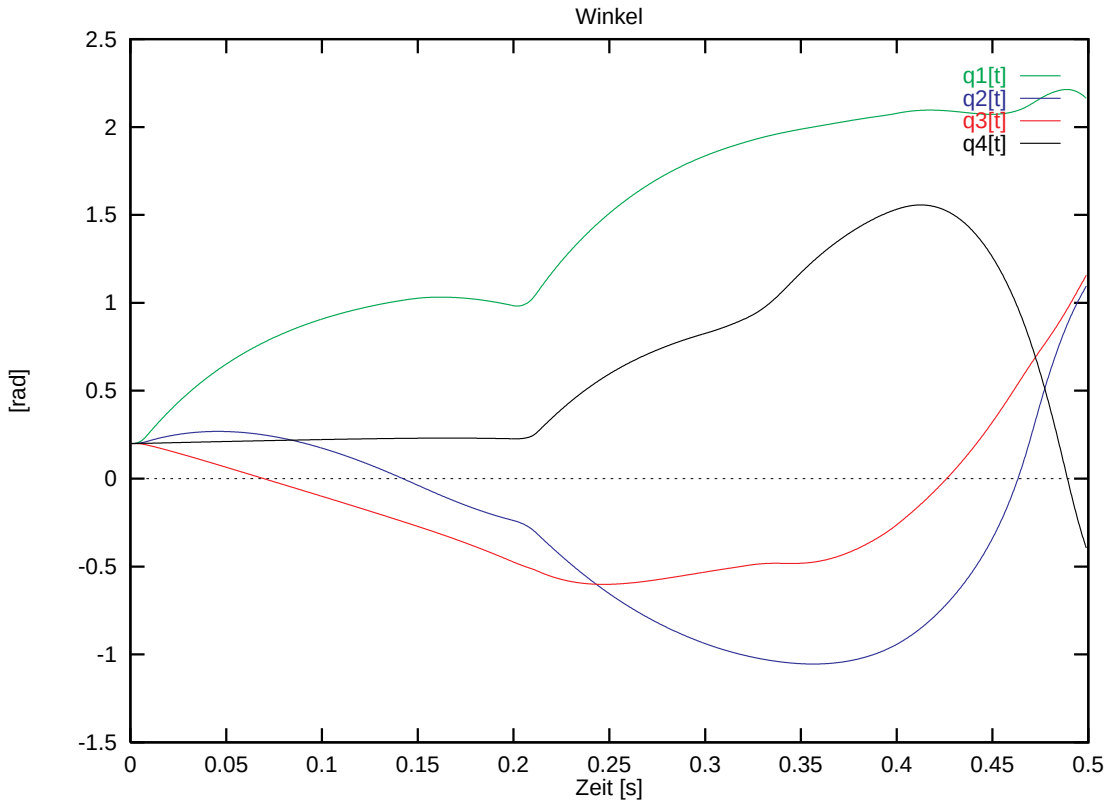

Abbildung 4.22: Weitwurf mit 8 Kg: Die Winkelvariablen

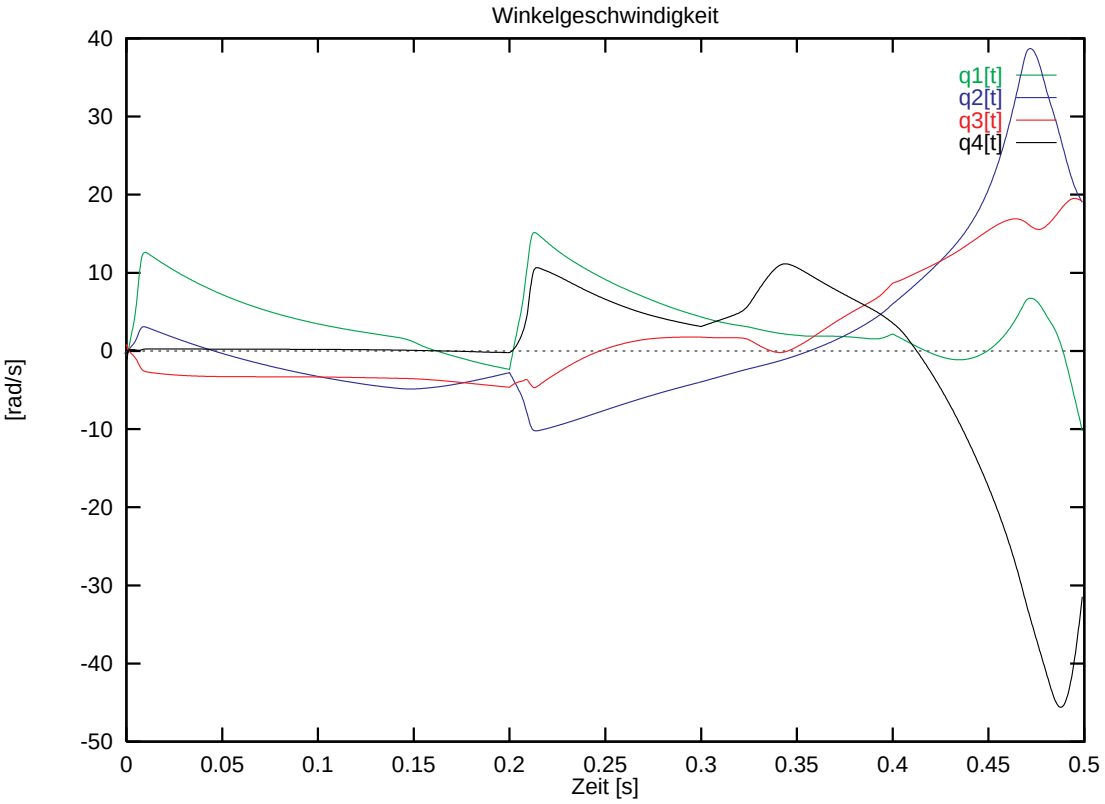

Abbildung 4.23: Weitwurf mit 8 Kg: Die Ableitungen der Winkelvariablen

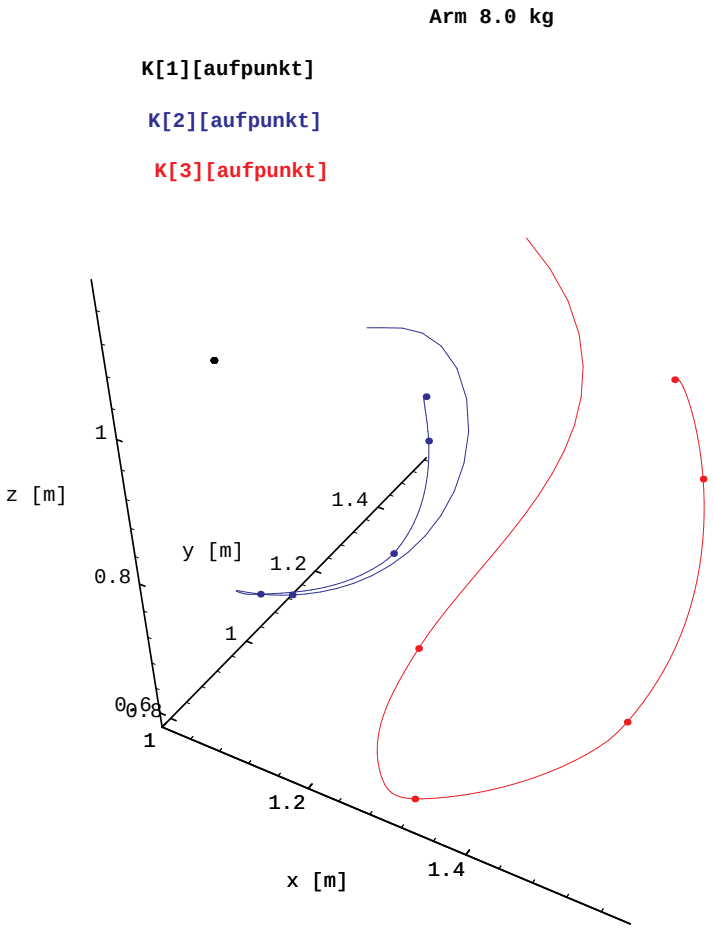

Abbildung 4.24: Weitwurf mit 8 Kg: Die Trajektorie

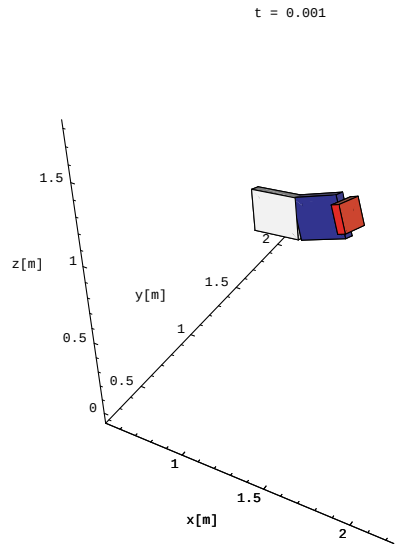

Abbildung 4.25: Kugelstoßen

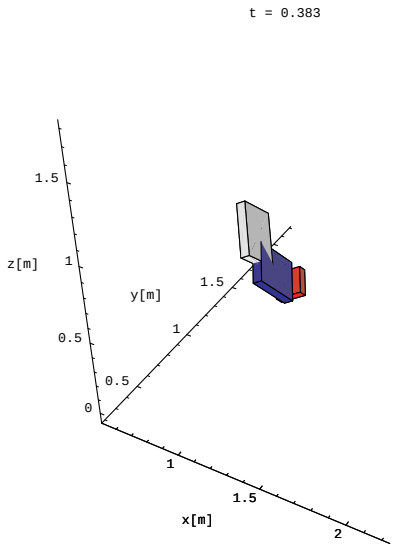

Abbildung 4.26: Kugelstoßen

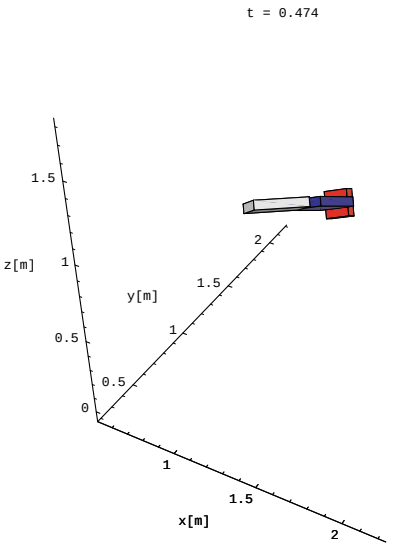

Abbildung 4.27: Kugelstoßen

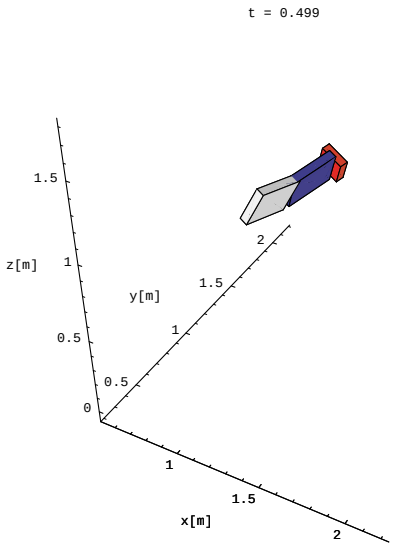

Abbildung 4.28: Kugelstoßen

Danksagung

An erster Stelle möchte ich mich bei Herrn Hanns Ruder bedanken, für seine offene gesprächsbereitschaft und für die Aufnahme an seinem Institut.

Desweiteren gilt mein Dank Herrn Thomas Rosenmaier für seine Betreuung und für alle ideenreiche Gespräche.

Bei Herrn Michael Günther bedanke ich mich für seine ständige diskussionsbereitschaft bezüglich diverser physikalischer Sachverhalte.

Frau Uta Keppler danke ich für ihre unendliche Geduld.

Herrn Valentin Keppler gilt besonderen Dank. Um diese spezielle Liste nicht ausarten zu lassen, bekanke ich mich einfach für alles bei ihm.

Herrn Stefan Kulla danke ich für sein Gehör beim Entwickeln diverser virtueller Betriebssysteme.

Herzlich bedanken möchte ich mich noch bei meinen Eltern.

Anhang A

Quellcode

```
/* C-Program, geschrieben von Mathematica, um die Differenzialgleichung
  von: Euler .c : zu lösen ,  geschrieben am {1997, 3, 14, 13, 57, 3}  */

 #include <stdio.h>
 #include <stddef.h>
 #include <stdlib.h>
 #include <math.h>
 #include <time.h>
  #define N 6              /* 2 * Anzahl Variablen */
  #define M 4              /* Anzahl Variablen + 1 */
  #define ZWANG 0          /* Anzahl Zwangsbedinungen */
  #define MORECONST 2
  #define DIM 3
  #define KANZ 1          /* Anzahl Körper */
                          /* Veändeliche Konstanten */
  #define USEDPOINTS 1500
  #define FLOAT double      /* entweder float oder double */
  #define TBEGIN 0.
  #define TEND 15.
  #define E 2.718281828459045
                          /* Ausgabe ob und wohin */
  #define MATHOUT
  #define OUTFILE1 "euler.dat"
```

```
  #define GNUPLOT
  #define OUTFILE2 "euler.gnu"
  #define NAME "Euler .c"
 #include "integrat.h"

/* Die Startwerte in der Reihenfolge { 0, y, y', ( y'') } */
FLOAT  dummy[ 7 ];
FLOAT y[ 10 ] = { 0, 0., 1.57079632, 0., 0., 0., 30., 0, 0, 0 };
/* Die Start- und Endwerte der Integration */
FLOAT  tbegin = TBEGIN,    tend = TEND;
FLOAT  tderivs = TBEGIN,   tdeltaderivs;
/* Die Genauigkeit der Integration (von 10e-2 bis 10e-12) */
FLOAT epsilon = 1.e-7;
/* Die Anzahl der Ausgabepunkte */
int usedpoints = USEDPOINTS;

/* Die Konstanten des Systes */
FLOAT Mas[ 2 ] =              { 0, 8. };
FLOAT Ss[ 4 ] =               { 0, 0.5, 1.1, 1.1 };
FLOAT Ll[ 7 ] =               { 0, 1., 1.2, 1.2, 1.2, 1.2, 1.2 };
FLOAT Kk[ 7 ] =               { 0, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3 };
FLOAT Ii[ 2 ][ 4 ] = {        { 0, 0, 0, 0 },
                              { 0, 2., 2., 4. }   };

 #define DETMZERO 1.0e-8
 FLOAT detm, rdgl[M], invm[M][M], mat[M][M], KfMo[M];
 int derivscount = 0;
void derivs(FLOAT t,FLOAT y[],FLOAT dydx[])
{
  derivscount++;
  tdeltaderivs = t - tderivs;
  tderivs = t;

KfMo[1] = 0.;
KfMo[2] = 0.;
KfMo[3] = 0.;
```

```
mat[1][1] = 4;
mat[1][2] = 0;
mat[1][3] = 4*cos(y[2]);
mat[2][1] = mat[1][2];
mat[2][2] = 4.;
mat[2][3] = 0;
mat[3][1] = mat[1][3];
mat[3][2] = mat[2][3];
mat[3][3] = (32. + 0.*cos(2*y[2]))/8;

detm = -(mat[1][3]*mat[1][3]*mat[2][2]) + mat[1][1]*mat[2][2]*mat[3][3];

invm[1][1] = mat[2][2]*mat[3][3];
invm[1][2] = 0;
invm[1][3] = -(mat[1][3]*mat[2][2]);
invm[2][1] = 0;
invm[2][2] = -(mat[1][3]*mat[1][3]) + mat[1][1]*mat[3][3];
invm[2][3] = 0;
invm[3][1] = -(mat[1][3]*mat[2][2]);
invm[3][2] = 0;
invm[3][3] = mat[1][1]*mat[2][2];

rdgl[1] = -4*y[5]*y[6]*sin(y[2]);
rdgl[2] = (-39.24 + 4*y[4]*y[6])*sin(y[2]) + 0.*(y[6]*y[6])*sin(2*y[2]);
rdgl[3] = -4*y[4]*y[5]*sin(y[2]) + 0.*y[5]*y[6]*sin(2*y[2]);

if(fabs(detm) <= DETMZERO )
  fprintf(stderr,"Die Determinante des Systems %f\n",detm);

dydx[ 1 ] = y[ 4 ];
dydx[ 2 ] = y[ 5 ];
dydx[ 3 ] = y[ 6 ];
dydx[ 4 ] = -(((-KfMo[1] + rdgl[1])*invm[1][1] +
(-KfMo[2] + rdgl[2])*invm[1][2] +
     (-KfMo[3] + rdgl[3])*invm[1][3])/detm);
```

```
dydx[ 5 ] = -(((-KfMo[1] + rdgl[1])*invm[2][1] +
(-KfMo[2] + rdgl[2])*invm[2][2] +
     (-KfMo[3] + rdgl[3])*invm[2][3])/detm);
dydx[ 6 ] = -(((-KfMo[1] + rdgl[1])*invm[3][1] +
(-KfMo[2] + rdgl[2])*invm[3][2] +
     (-KfMo[3] + rdgl[3])*invm[3][3])/detm);

ycopy(dydx);

}

FLOAT Ekin_dat;
void Ekin (FLOAT t)
{
Ekin_dat = (32*(y[4]*y[4]) + 32.*(y[5]*y[5]) + 32.*(y[6]*y[6])
 + 0.*cos(2*y[2])*(y[6]*y[6]) +
   64*cos(y[2])*y[4]*y[6])/16;
}

FLOAT Epot_dat;
void Epot (FLOAT t)
{
Epot_dat = 39.24*cos(y[2]);
}

FLOAT pos1_dat[ 4 ];
void pos1 (FLOAT t)
{
pos1_dat[1] = sin(y[1])*sin(y[2]);
pos1_dat[2] = cos(y[1])*sin(y[2]);
pos1_dat[3] = cos(y[2]);
}
```

Literaturverzeichnis

[Ber94] Karsten Berns. *Steuerungsansätze auf Basis Neuronaler Netze für sechsbeinige Laufmaschinen*. infix, 1994.

[BFM96] Heinrich Braun, Johannes Feulner, and Rainer Malaka. *Praktikum Neuronaler Netze*. Springer Verlag, 1996.

[Bre88] H. Bremer. *Dynamik und Regelung mechanischer Systeme*. B. G. Teubner Stuttgart, 1988.

[dad94] *DADS Users Manual*. CADSI, 1994.

[Koh84] T. Kohonen. *Self-Organization and Associative Memory*. Springer Series in Information Sciences 8, Heidelberg, 1984.

[KR90] Kernigan and Ritchie. *Programmieren in C*. Hanser, 1990.

[Kra90] Hans Peter Kratzer. *Neuronale Netze*. Hanser, 1990.

[Pla95] Réjean Plamandon. A kinematic theory of rapid human movements. In *Biologocal Cybernetics*, volume 72. Springer Verlag, 1995.

[PP96] Réjean Plamandon and Claudio M. Privitera. A neural model for generating and lerning a rapid movement sequence. In *Biologocal Cybernetics*, volume 74. Springer Verlag, 1996.

[PTVF94] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C The Art of Scientific Computing*. Cambridge University Press, version 2.04, 2nd edition, 1994.

[Sch86] W. Schiehlen. *Technische Dynamik*. Teubner, 1986.

[Sch93] W. Schiehlen. *Advanced Multibody System Dynamics*. Kluwer ACADEMIC Publishers Dordrecht/Boston/London, 1993.

[unk95] unknown. *GENESIS*. unknown, 1995.

[Zel94] A. Zell. *Simulation Neuronaler Netze*. ADDISON WESLEY, 1994.