Hausarbeit

Simulation

„Mathematische Modellbildung und Simulation eines physikalischen Pendels mit Biegung"

bei Herrn Dr. J.-T. Meyer

-Fachbereich Automatisierungstechnik-
-Sommersemester 2002-

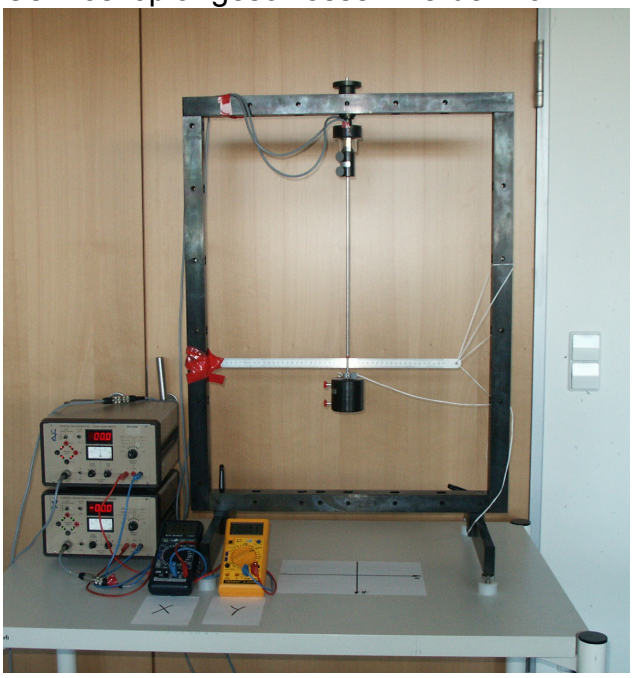
von

Matthias Hübner
Danny Schulz

Inhalt

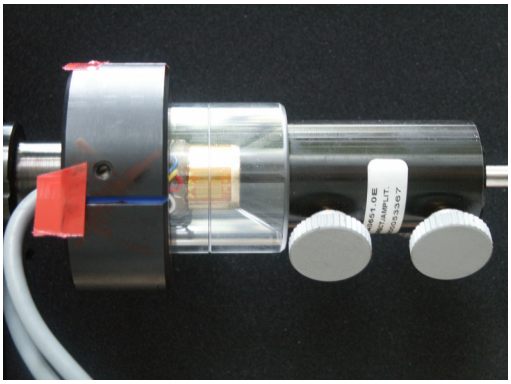
1. Beschreibung der Pendelvorrichtung

Die hier zu modellierende und zu simulierende Pendelvorrichtung der Firma Phywe ist, wie in Bild 1.1 zu erkennen, aus unterschiedlichen mechanischen Komponenten aufgebaut. Sie besteht zunächst aus einem massiven Trägerrahmen aus schwarz lackiertem Stahl, der für ausreichende Stabilität der Vorrichtung sorgt, einer Stange aus Stahl, die bei der Pendelbewegung verbogen wird und somit eine Rückstellkomponete erzeugt, einem schwarzen Metallzylinder, der an der Stange in variabler Höhe mittels zwei Schrauben zu befestigen ist und mit zwei Kilogramm die größte Masse im System darstellt. Zylinder und Stange sind im Rahmen selbst mittels der in Bild 1.2 gezeigten Vorrichtung befestigt. Diese erfüllt zugleich den Zweck, die Biegung der Stange zu messen, indem sie mit 8 Dehnungsmessstreifen (DMS), 4 pro Kraftrichtung (x,y), versehen ist, die jeweils in einer Vollbrücke pro Kraftrichtung verschaltet sind und durch die Biegung verformt werden, so dass sich deren Widerstände ändern. Die somit erzeugte Spannungsdifferenz pro Achse wird Messverstärkern der Firma Phywe zugeleitet, deren Ausgangsspannung an einen mitgelieferten x-y-Schreiber oder an ein Oszilloskop angeschlossen werden kann.



Aus den dieser Hausarbeit beiliegenden Messreihen lässt sich vermuten, dass es sich hier um ein schwach gedämpftes System handeln muss, das man mit einer linearen Differenzialgleichung zweiter Ordnung beschreiben könnte. Zudem ließ sich durch drei Messreihen für den stationären Fall ein lineares Verhältnis zwischen Auslenkung des Pendels in einem definierten Messpunkt, der durch die Höhe bei Auslenkung null definiert ist, und der von den DMS-Verstärkern erzeugten Spannung ermitteln.

Bild 1.1: Pendelvorrichtung



Der Korrelationsfaktor liegt bei 0.99, was diese Nährung statistisch untermauert. Diese Messreihen liegen dieser Hausarbeit in Form einer Excel-Tabelle (siehe Anhang) bei.

Bild 1.2: DMS-Stangenbefestigung

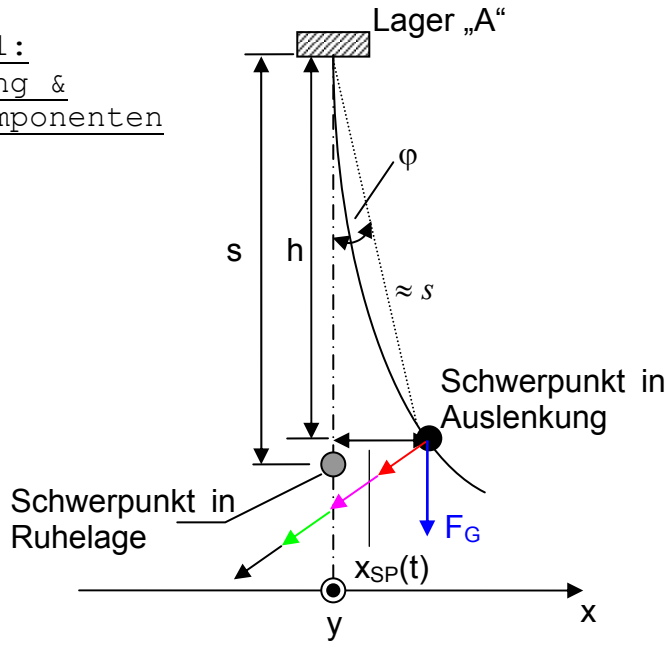
Es liegt daher für den Autor dieser Arbeit nahe, zunächst ein Modell für die Bewegung eines Punktes auf der Stange zu beschreiben, um anschließend über einen messtechnisch ermittelten Proportionalitätsfaktor daraus die Spannung der Messverstärker zu ermitteln.
Als Punkt zur Beschreibung wurde hier der Schwerpunkt gewählt und alle angreifenden Kräfte auf diesen bezogen.
Als wirkende Kräfte kann man hierbei im ausgelenkten Zustand des Pendels von einer durch die Gesamtmasse von Pendelstange und Zylinder bedingten gravitativen Rückstellkraft ausgehen. Zusätzlich zu dieser wirkt noch ein Rückstellmoment, das durch die Verbiegung der Stange erzeugt wird. Der Zusammenhang zwischen Auslenkung und diesen beiden Rückstellkomponenten wird in den folgenden Kapiteln hergeleitet. Zudem wirkt der Bewegungsrichtung der Auslenkung noch die Massenträgheit entgegen. Durch die Verbiegung der Stange selbst ist durch die Verschiebung der Gefügeschichten eine Reibungungskraft anzunehmen, die insbesondere für das Abnehmen der Amplitude verantwortlich ist. Mit diesen Ansätzen wird nun im folgenden Kapitel ein Modell für den Schwerpunkt des Systems aufgestellt.

2. Mathematische Modellbildung

2.1. Abstraktion und Definition

Im Folgenden sei das Pendel allein bestehend aus einem dreiwertigen Lager, Pendelstange und Zylinder beschrieben. Die Masse beider letztgenannten mechanischen Komponenten wird auf den gemeinsamen Schwerpunkt verschoben dargestellt (Bild 2.1). Sieht man frontal auf den Rahmen des Pendels, so ist hiernach in horizontaler Ebene die x-Koordinate definiert. Senkrecht zu dieser, in Bezug auf den Boden, sei die y-Koordinate. Da die Schwingungsgleichungen für beide Koordinaten wegen der Achsensymmetrie des Pendels gleich sein müssen, sei hiernach nur die x-Koordinate und damit die bezeichnete x-Auslenkung im Schwerpunkt x_{SP} betrachtet. Die hieraus ermittelte Schwingungsgleichung ist äquivalent für die y-Auslenkung, nur mit der Variablen y_{SP}, der Auslenkung in y im Schwerpunkt des Systems.
Aus den im Rahmen dieser Hausarbeit durchgeführten Messungen ging außerdem hervor, dass hierbei nur Auslenkungen im Schwerpunkt von maximal 3 cm betrachtet werden, da ansonsten das Material der Stange vom linearen Bereich in den Fließbereich kommen würde und die Verformung nicht mehr elastisch, sondern plastisch sein würde, was nicht Sinn dieser Versuchseinrichtung sein kann. Zudem scheint der Stahl der Stange recht spröde zu sein, was hieße, dass bei einer solchen gleichzeitig auch noch vorhandenen Wechselbeanspruchung sich bald ein Bruch des Materials einstellen würde. Wegen dieser maximalen Auslenkung von 3 cm werden hier keine größeren Winkel φ als 16° erzeugt.

Bild 2.1: Anordnung & Kraftkomponenten



Rot:	$F_G * \sin(\varphi)$	Anteil der Gravitation
Rosa:	F_{Bieg}	Biegekraft
Grün:	$F_{Träg}$	Trägheitsanteil
Schwarz:	F_{Reib}	innere Reibung

Dies hat zur Folge, dass man nach Taylor den Sinus des Winkels ersetzen kann, und zwar durch das erste Polynom der Reihenentwicklung für den Sinus.
Es gilt:

$$\sin(\varphi) = \frac{x_{SP}(t)}{h}$$ (1), wobei man h aufgrund der kleinen Winkel ersetzen kann:

$h \approx s$, so dass $\sin(\varphi) = \frac{x_{XP(t)}}{s}$ (2).

Nach Taylor-Polynom 1. Grades ist

$$\sin(\varphi) \approx \varphi$$ (3).

In Bild 2.1 der abstrahierten Pendelvorrichtung sind bereits die angenommenen Kräfte eingezeichnet, die der Bewegung des Schwerpunktes (hier in positive x-Richtung) entgegenwirken. Da es sich angenähert jedoch um eine rotatorische Bewegung um das dreiwertige Lager handelt, ist es schwer möglich, die Trägheitskraft zu berechnen. Vielmehr liegt es nahe, das Trägheitsmoment zu berechnen, das folgendermaßen definiert ist:

$$M_{Träg} = \theta_A \cdot \ddot{\varphi}$$ (4)

θ_A ist hier das Massenträgheitsmoment der Pendelstange und des Zylinders bezüglich des Lagers "A". Mit $\ddot{\varphi}$ ist die Winkelbeschleunigung bezeichnet (zweite Ableitung des Winkels).

Anstatt der Kräfte für Biegung, Reibung und Gravitationsanteil setze man nun die Momente.
So sei das erzeugte Moment durch die Gravitation definiert durch

$$M_{Grav} = s \cdot m \cdot g \cdot \sin(\varphi)$$ (5).

Hier wird also der Anteil der Gewichtskraft multipliziert mit dem Abstand Lager-Schwerpunkt (s). Aufgrund der recht geringen Biegung kann diese Nährung vorgenommen werden.
Für die Reibung, das heißt die Erwärmung durch die Biegung im Inneren des Stabes, wird an dieser Stelle zunächst M_{Reib} angenommen.
Für die Biegung sei hier M_{Bieg} anzunehmen.

Es kann nun also eine Momentengleichung folgender Form aufgestellt werden:

$$M_{Grav} + M_{Bieg} + M_{Träg} + M_{Reib} = 0$$ (6)

Eingesetzt ergibt sich dann:

$$s \cdot m \cdot g \cdot \sin(\varphi) + M_{Bieg} + \theta_A \cdot \ddot{\varphi} + M_{Reib} = 0 \qquad (7)$$

Für die Annährung des Winkels ergibt sich:

$$\sin(\varphi) \approx \varphi \approx \frac{x_{SP}(t)}{s} \qquad (8)$$

Damit ist die linearisierte Momentengleichung:

$$s \cdot m \cdot g \cdot \frac{x_{SP}(t)}{s} + M_{Bieg} + \theta_A \cdot \frac{\ddot{x}_{SP}(t)}{s} + M_{Reib} = 0 \qquad (9)$$

Es sind also noch zwei Größen dieser Gleichung herzuleiten. Das folgende Kapitel wird sich nun dem Biegemoment widmen.

2.2. Modellierung des Biegemomentes

Zentrales Axiom für die folgende Herleitung ist, dass die Biegung des Stabes für eine beliebige Auslenkung x_{SP} einen Kreisausschnitt darstellt. Dies bedeutet, dass für jedes $x_{SP}(t)$ auch ein für diesen Moment konstanter Biegeradius R=const. existiert. Es wird nun ausgehend vom statischen Fall einer konstanten Auslenkung das Biegemoment modelliert, was im Grunde die Herleitung der Differenzialgleichung der Biegelinie beinhaltet.
Zur Veranschaulichung der Biegung diene Bild 2.2.

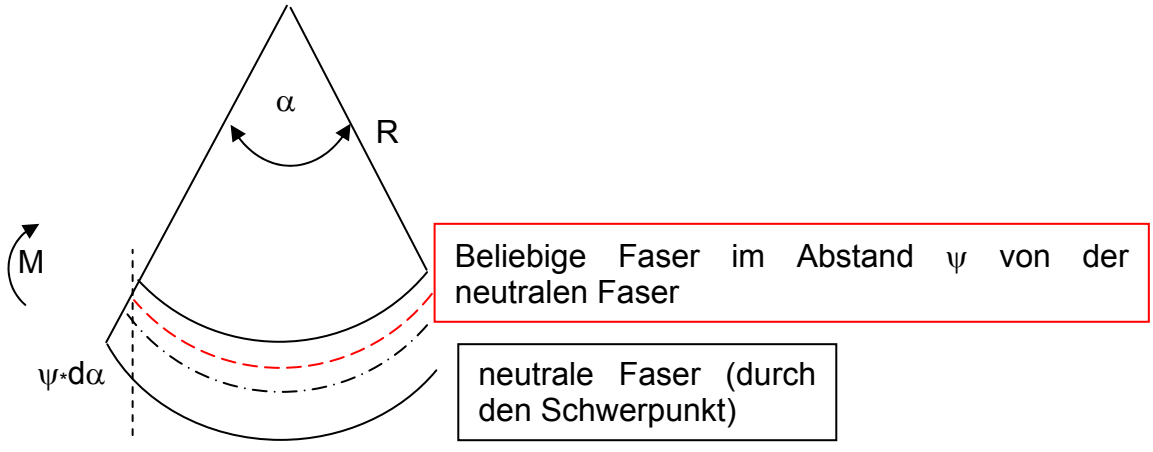


Bild 2.2

Offensichtlich wird die Stange durch die Biegung verlängert, wovon jede einzelne, infinitesimal kleine Faser der Stange beeinflusst (verlängert) wird. Die Fasern werden also gedehnt. Die Dehnung (ε) ist definiert als Längenänderung dividiert durch die Ausgangslänge.

Geht man davon aus, dass aber einem Winkel von null Grad alle Fasern dieselbe Länge haben, so muss die Längenänderung der Faserabstand (ψ) der zu betrachtenden Faser multipliziert mit dem (infinitesimal kleinen) Winkel sein. Die Ausgangslänge ist hierbei der Radius multipliziert mit dem Winkel. Für die Dehnung einer Faser ergibt sich also:

$$\varepsilon(\psi) = \frac{\psi \cdot d\alpha}{R \cdot d\alpha} \qquad (10)$$

hier kann also der Winkel herausgekürzt werden, so dass

$$\varepsilon(\psi) = \frac{\psi}{R} \qquad (11)$$

Dem Biegemoment, das zur Verformung aufgebracht werden muss stehen die Momente der Spannungen der einzelnen Fasern des Stangenquerschnitts entgegen. Nach Hooke verteilen sich diese Biegespannungen $\sigma(\psi)$ linear über den Querschnitt, mit dem E-Modul des Stangenwerkstoffs als Proportionalitätsfaktor:

$$\sigma(\psi) = E \cdot \varepsilon(\psi) = \frac{E}{R} \cdot \psi \qquad (12)$$

Da die Summe aller Momente um einem beliebigen Drehpunkt null sein müssen (d'Alembert) gilt verknüpft mit oben genanntem:

$$0 = M_{Bieg} - \int_A \sigma(\psi) dA \cdot \psi \qquad (13),$$

ψ ist mit dem Radius der Stange sozusagen der Hebelarm für die durch das Integral der Spannungen berechneten Kräfte.
Setzt man nun Formel (12) in Formel (13) ein, so ergibt sich mit einer Termumformung:

$$M_{Bieg} = \frac{E}{R} \cdot \int_A \psi^2 dA \qquad (14)$$

Da ψ hier dem Radius der Stange entspricht, entspricht das Integral dem aus der Mechanik bekannten Flächenträgheitsmoment zweiter Ordnung I_a.

$$I_a = \int_A \psi^2 dA = \int_A r^2 dA \qquad (15)$$

stellt das „FTM 2. Ordnung" dar, so dass in (XIV) der Integralausdruck ersetzt werden kann:

$$M_{Bieg} = \frac{E}{R} \cdot I_a \qquad (16)$$

Das Biegemoment ist also umgekehrt proportional zum Krümmungsradius R. Es stellt sich nun also die Frage, welche Funktion der Krümmungsradius von der Auslenkung x_{SP} hat.
Ausgangspunkt sei hier die Krümmung χ, die für einen definierten, konstanten Radius R auch konstant sein muss, weil folgender Zusammenhang gilt:

$$\chi = \frac{1}{R} \qquad (17)$$

Löst man nun Gleichung (16) nach 1/R auf so, kann man in obiger 1/R ersetzen zu:

$$\chi = \frac{M_{Bieg}}{E \cdot I_a} \qquad (18)$$

Aus der Mathematik sollte für die Krümmung einer Funktion y(x) an der Stelle x der folgende Zusammenhang bekannt sein, auf dessen Herleitung hier nicht eingegangen werden soll:

$$\chi = \frac{\ddot{y}(x)}{(1+\dot{y}^2)^{3/2}} \qquad (19)$$

Fasst man die Krümmung des Balkens als Funktion y(x) auf, so lässt sich diese Gleichung kombinieren mit (18).

$$\frac{\ddot{y}(x)}{(1+\dot{y}^2)^{3/2}} = \frac{M_{Bieg}}{E \cdot I_a} \qquad (20)$$

Es wird für die weitere Herleitung ein anderes Koordinatensystem eingeführt. Hierzu stelle man sich den eingespannten Balken waagerecht vor. Dies verdeutlich Bild 2.3:

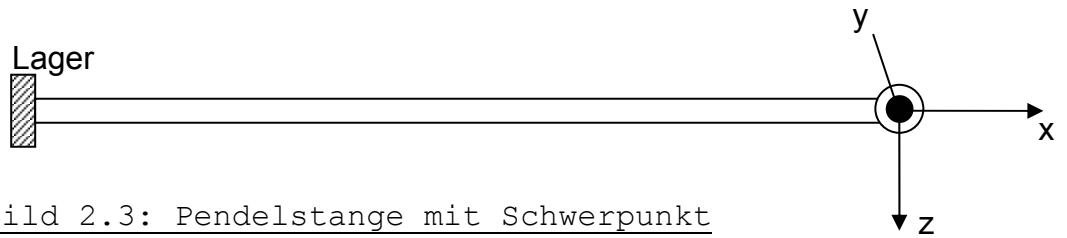


Bild 2.3: Pendelstange mit Schwerpunkt

Man beachte hierbei, dass die angreifende Kraft, die die Verbiegung des Balkens verursacht hiernach in z-Richtung wirke. Diese Kraft wirkt auf dem Hebel x (Abstand vom Lager), so dass das dadurch erzeugt Moment M_{Bieg} in die y-Richtung wirken muss, denn es gilt:

$$\vec{M}_{Bieg} = \vec{F} \times \vec{x} \qquad (21)$$

M_{Bieg} erzeugt also eine Krümmung die nach Formel (21) in y-Richtung wirken müsste. Dem Betrag der Krümmung in z-Richtung ist diese aber äquivalent, so dass folgt:

$$\frac{\ddot{z}(x)}{(1+\dot{z}(x)^2)^{3/2}} = \frac{M_{Bieg}}{E \cdot I_a} \qquad (22)$$

Da die Tangentenneigungen der Biegelinie des Balkens zumeist sehr klein sind, kann man für kleine Neigungen, d.h. auch kleine Auslenkungen, (20) annähern.
Es gilt:

$$(1+\dot{z}(x))^{3/2} \approx 1 \qquad (23)$$

Kombiniert mit (22) ergibt dies:

$$\ddot{z}(x) = \frac{M_{Bieg}}{E \cdot I_a} \qquad (24)$$

Da jedoch im gewählten Koordinatensystem einem positiven Biegemoment eine Abnahme des Neigungswinkels der Tangente an der Biegelinie mit fortschreitendem x entspricht, muss die Krümmung aber negativ sein, so dass die soeben hergeleitete Differenzialgleichung ersetzt wird durch:

$$\ddot{z}(x) = -\frac{M_{Bieg}}{E \cdot I_a} \qquad (25)$$

Nun wird diese Differenzialgleichung in Bezug auf unsere Stange gelöst. Die Kraft greife hierbei im Schwerpunkt unseres Stange-Zylinder-Systems an. Der Abstand von der Einspannung der Stange zum Schwerpunkt sei s. (Bild 2.4)

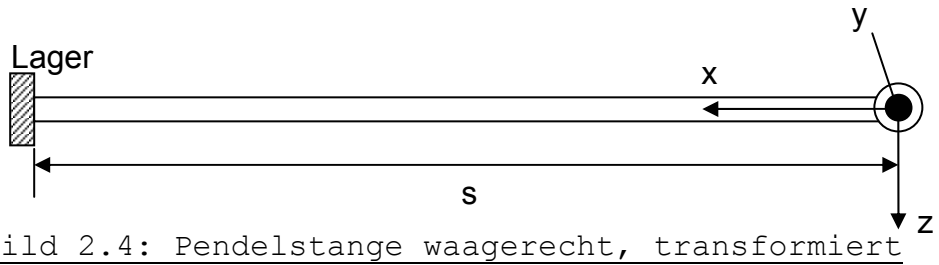


Bild 2.4: Pendelstange waagerecht, transformiert

Nach Bild 2.4 in Kombination mit Bild 2.3 gilt:

$$M_{Bieg} = -F \cdot x \qquad (26)$$

Dies kombiniert mit Gleichung (24) ergibt:

$$\ddot{z}(x) = \frac{F \cdot x}{E \cdot I_a} \qquad (27)$$

Um z(x) nun zu bestimmen müssen wir zweimal integrieren.
Somit ist

$$\dot{z}(x) = \int \frac{F \cdot x}{E \cdot I_a} dx = \frac{F}{E \cdot I_a} \cdot x^2 + C_1 \qquad (28)$$

Zur Bestimmung der Integrationskonstanten C_1 setze man die Randbedingung $\dot{z}(x = s) = 0$, da am Anfang aufgrund des Lagers die Steigung der Biegung null sein muss.
Es ergibt sich für $\dot{z}(x)$

$$\dot{z}(x) = \frac{F \cdot x^2}{2 \cdot E \cdot I_a} - \frac{F \cdot s^2}{2 \cdot E \cdot I_a} \qquad (29)$$

Es folgt für die zweite Integration:

$$z(x) = \int \dot{z}(x) dx = \frac{F \cdot x^3}{6 \cdot E \cdot I_a} - \frac{F \cdot s^2 \cdot x}{2 \cdot E \cdot I_a} + C_2 \qquad (30)$$

Man ermittelt hier C_2 analog zu C_1:

$$z(x = s) = 0 = \frac{F \cdot s^3}{E \cdot I_a} \cdot \left(1/6 - 1/2 \right) + C_2 \qquad (31)$$

$$C_2 = \frac{F \cdot s^3}{3 \cdot E \cdot I_a} \qquad (32)$$

Eingesetzt ergibt sich für z(x):

$$z(x) = \frac{F \cdot x^3}{6 \cdot E \cdot I_a} - \frac{F \cdot s^2 \cdot x}{2 \cdot E \cdot I_a} + \frac{F \cdot s^3}{3 \cdot E \cdot I_a} \qquad (33)$$

Bei unserem Pendelmodell liegt die angreifende Kraft F im Schwerpunkt, d.h. bei x=0, so dass sich die Gleichung (33) vereinfacht zu:

$$z(x = 0) = \frac{F \cdot s^3}{3 \cdot E \cdot I_a} \qquad (34)$$

Ist nun die Auslenkung im Schwerpunkt bekannt (hier z), und wir gehen davon aus, dass die Differenzialgleichung nach (0) uns einen Ausdruck hierfür liefert, so lässt sich aus (34) die wirkende Biegekraft ermitteln.

$$F = \frac{3 \cdot E \cdot I_a}{s^2} \cdot z \qquad (35)$$

Für das Biegemoment ist seit jeher bekannt:

$$M_{Bieg} = F \cdot s \qquad (36)$$

Dies gilt für die Kraft F, die in z-Richtung, d.h. in Richtung der Auslenkung wirkt.

Somit ergibt sich für das Biegemoment:

$$M_{Bieg} = \frac{3 \cdot E \cdot I_a}{s^2} \cdot z \qquad (37)$$

Wird nun z durch x als Variable für die Auslenkung nach dem nach Bild 1.1 definierten Koordinatensystem ersetzt, kann man die Momentengleichung (20) um das Biegemoment ergänzen:

$$s \cdot m \cdot g \cdot \frac{x_{SP}(t)}{s} + \frac{3 \cdot E \cdot I_a}{s^2} \cdot x_{XP}(t) + \theta_A \cdot \frac{\ddot{x}_{SP}(t)}{s} + M_{Reib} = 0 \qquad (38)$$

Analog gilt für die Auslenkung in y-Richtung nach Skizze 1:

$$s \cdot m \cdot g \cdot \frac{y_{SP}(t)}{s} + \frac{3 \cdot E \cdot I_a}{s^2} \cdot y_{SP}(t) + \theta_A \cdot \frac{\ddot{y}_{SP}(t)}{s} + M_{Reib} = 0 \qquad (39)$$

Nach Pythagoras ließe sich also nach der Lösung beider DGLs die Auslenkung insgesamt berechnen, die wie folgt nun definiert sei als q_{SP}:

$$q_{SP} = \sqrt{x_{SP}^2 + y_{SP}^2} \qquad (40)$$

2.3. Modellierungsansätze der Reibung

Zur Modellierung der inneren Reibung, die durch die Verschiebung der Gitterebenen im Stahl bei der Pendelbewegung erzeugt wird, ist es notwendig, ein abstraktes Modell für diese Transformation eines bestimmten Energiebetrages der Pendelbewegung in Wärme zu finden. Hierfür würde sich die Annahme einer elastischen Hysterese eignen.
Solange ein Körper noch nicht elastisch deformiert ist, wird der Zusammenhang zwischen Spannung und Dehnung durch die vom Nullpunkt nach A verlaufende Kurve beschrieben, die auch als „Neukurve“ oder jungfräuliche Kurve bezeichnet wird.

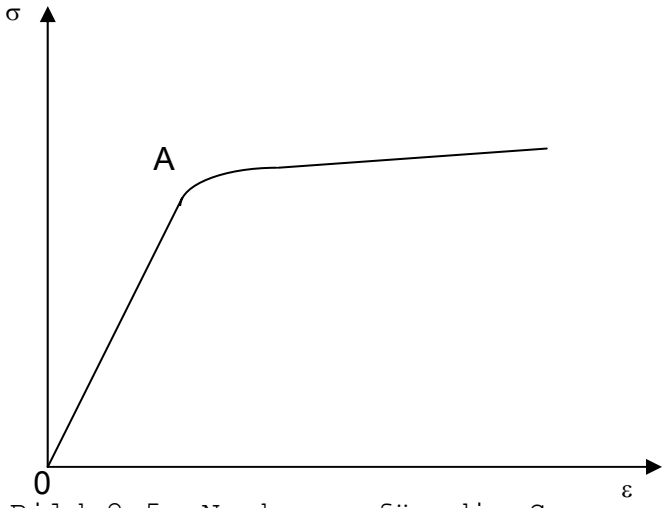


Bild 2.5: Neukurve für die Spannung in Abhängigkeit der Längenänderung

Wenn die Spannung bei der elastischen Deformation jedoch wieder verringert wird, durchläuft die Dehnung eine andere Kurve; der Spannung Null entspricht noch eine endliche Dehnung und es bedarf einer entgegengesetzten Spannung, um die Dehnung aufzuheben. Steigert man nun diesen Druck, so wird der Körper komprimiert, Wenn nach Erreichung des Zustandes B die Deformation rückgängig gemacht wird, bedarf es wiederum eines Zuges, um die Dehnung Null zu erreichen. Dies wird in dem folgenden Graph zugehöriger Hysteresekurve deutlich:

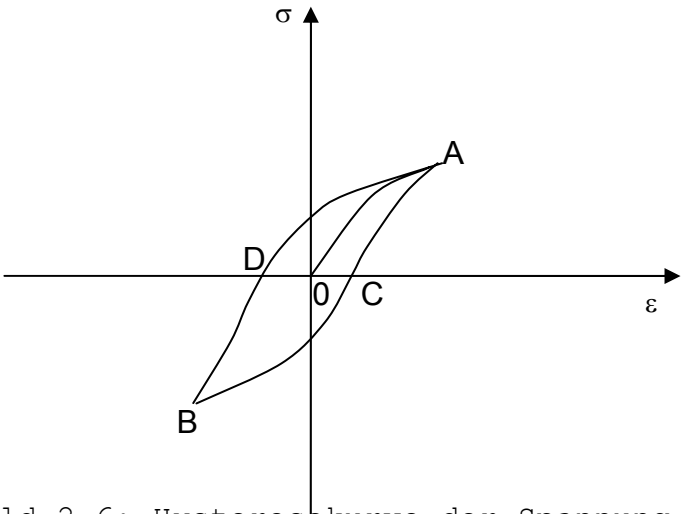


Bild 2.6: Hysteresekurve der Spannung in Abhängigkeit der Längenänderung

Da die Pendelstange durch die Bewegung gedehnt wird, liegt die Einführung der elastischen Energie nahe. Eine erste Feststellung kann daher sein:
Um im Allgemeinen ein Prisma vom Querschnitt A und der Länge l um Δl zu dehnen, muss eine gewisse Arbeit ΔW aufgewandt werden:

$$\Delta W = \int_0^{\Delta l} A \cdot \sigma \cdot dl \qquad (41)$$, wobei σ die Spannung des Primas bezeichnet.

Möchte man oben genannte Arbeit durch die Dehnung $\varepsilon = \Delta l / l$ aus, so ergibt sich:

$$\Delta W = \int_0^{\varepsilon} A \cdot \sigma \cdot l \cdot d\varepsilon = V \cdot \int_0^{\varepsilon} \sigma \cdot d\varepsilon \qquad (42)$$

Das Integral $\int_0^{\varepsilon} \sigma \cdot d\varepsilon$ gibt also die je Volumeneinheit zu leistende Verformungsarbeit, respektive die elastische Energie je Volumeneinheit an. Diese Beziehung gilt auch für andere Verformungstypen, z.B. die Scherung, wobei dann σ durch die Schubspannung τ und ε durch den Scherungswinkel α zu ersetzen sind.
Sofern das Hooke'sche Gesetz als gültig anzusehen ist, d.h. $\sigma = E \cdot \varepsilon$ gilt also:

$$\int_0^{\varepsilon} \sigma \cdot d\varepsilon = \tfrac{1}{2} \cdot E \cdot \varepsilon^2 \qquad (43)$$, d.h. bei Rückkehr in den spannungsfreien Zustand wird die gesamte aufgewandte Energie wieder freigesetzt. Dies gilt auch noch für eine nichtlineare Beziehung zwischen σ und ε, wobei die elastische Energie je Volumeneinheit durch den Inhalt der Fläche gegeben ist, die von der $\sigma = f(\varepsilon)$-Kurve und der Abszisse bis zum Endwert der Verformung ε eingeschlossen wird.
Anders ist dies im Fall der „elastischen Hysterese", wie sie bei dem Phywe-Pendel, d.h. bei der Verformung der Stange, vorzukommen scheint. Hier ist die beim Anlegen der Spannung aufzuwendende Arbeit (o bis A) größer als die bei Entspannung (A bis C) frei werdende. Je Volumeneinheit ist ein dem Flächenstück 0AC entsprechender Betrag an elastischer Energie verloren gegangen, d.h. irreversibel in Wärme umgewandelt worden. Bei periodischem Durchlaufen der ganzen Kurve ACBD (Wechselbeanspruchung) geht jedes Mal ein ihrem Flächeninhalt entsprechender Energieanteil je Volumeneinheit verloren. Dieses Verhalten entspricht laut Literatur einer gedämpften Schwingung, welches bei dem vorliegenden Pendel nach den Aufzeichnungen mit dem x-y-Schreiber vorliegen sollte.
Daher wird laut der normierten Form einer Differenzialgleichung einer freien, gedämpften Schwingung

$$\ddot{x} + 2 \cdot \delta \cdot \dot{x} + \omega_0^2 \cdot x = 0 \qquad (44)$$

für das angenommene Moment der Reibung $M_{Reib} = 2 \cdot \delta \cdot \dot{x} = c \cdot \dot{x}$ (45) eingesetzt. Später wird der geneigte Leser sehen, dass sich aus der Lösung unserer Momentengleichung, d.h. der Ermittlung einer Funktion $x_{SP}(t)$, und einem nachfolgenden Abgleich mit der Schwingdauer der Messungen der Dämpfungsfaktor δ ermitteln lässt. Nachteilig hierbei ist die Abhängigkeit des Faktors von Messungen, er lässt sich leider nicht ohne den hohen Aufwand der finiten Elemente- Methode nicht analytisch herleiten. Da innerhalb dieses Studienganges jedoch nicht diese in den Naturwissenschaften hochrelevante mathematische Methode behandelt wurde, wird auf solche Betrachtungen an dieser Stelle verzichtet. Auch die spätere Simulation des Pendels mittels dieser Methode konnte nicht erfolgen, da die Hochschule nicht über entsprechende Software verfügt, diskrete Matrizen für DGL-Systeme zu erstellen, die in das Mehrkörpersimulationsprogramm „Adams“ integriert werden können.

Daher wird nun die folgende Differenzialgleichung aus unserer Momentengleichung formuliert:

$$s \cdot m \cdot g \cdot \frac{x_{SP}(t)}{s} + \frac{3 \cdot E \cdot I_a}{s^2} \cdot x_{XP}(t) + \theta_A \cdot \frac{\ddot{x}_{SP}(t)}{s} + M_{Reib} = 0 \qquad (46)$$

kombiniert mit $M_{reib} = c \cdot \dot{x}$ ergibt sich die Gleichung zu:

$$\left(m \cdot g + \frac{3 \cdot E \cdot I_a}{s^2} \right) \cdot x_{SP}(t) + c \cdot \dot{x}_{SP}(t) + \frac{\theta_A}{s} \cdot \ddot{x}_{SP}(t) = 0 \qquad (47)$$

2.4. Lösung und Analyse der Differenzialgleichung

2.4.1. Normierung und Lösung

Die normierte, allgemeine Differenzialgleichung zweiter Ordnung einer gedämpften Schwingung lautet nach (44):

$$\ddot{x} + 2 \cdot \delta \cdot \dot{x} + \omega_0^2 \cdot x = 0$$

Daher ist unsere Gleichung (47) zwecks Koeffizientenvergleich umzuformen zu:

$$\left(m \cdot g + \frac{3 \cdot E \cdot I_a}{s^2} \right) \cdot \frac{s}{\theta_A} \cdot x_{SP}(t) + c \cdot \frac{s}{\theta_A} \cdot \dot{x}_{SP}(t) + \ddot{x}_{SP}(t) = 0 \qquad (48)$$

Der Koeffizientenvergleich liefert hierbei:

1. $$\omega_0 = \sqrt{\left(m \cdot g + \frac{3 \cdot E \cdot I_a}{s^2}\right) \cdot \frac{s}{\theta_A}} \qquad (49)$$

2. $$2 \cdot \delta = c \cdot \frac{s}{\theta_A} \qquad (50)$$

So kann für die Lösung der normierten Differenzialgleichung 2. Ordnung

$$x = x_0 \cdot e^{-\delta \cdot t} \cdot \cos(\omega_d \cdot t) \qquad (51)$$

berechnet werden.
Die benutzen Parameter werden wie folgt bezeichnet:

- δ bezeichnet den Dämpfungsfaktor oder die Abklingkonstante
- ω_0 bezeichnet die Eigenkreisfrequenz des *ungedämpften* Systems
- ω_d bezeichnet die Eigenkreisfrequenz der gedämpften Schwingung und ist definiert als:

$$\omega_d = \sqrt{\omega_0^2 - \delta^2} \qquad (52)$$

- x_0 bezeichnet die Anfangsauslenkung der Schwingung (d.h. die Anfangsauslenkung des Pendels)

Somit lässt sich die Lösung unserer Systemgleichung folgendermaßen formulieren:

$$x_{SP}(t) = x_0 \cdot e^{-\frac{c \cdot s}{2 \cdot \theta_A} \cdot t} \cdot \cos\left(\sqrt{\left(\left(m \cdot g + \frac{3 \cdot E \cdot I_a}{s^2}\right) \cdot \frac{s}{\theta_A}\right) - \left(\frac{c \cdot s}{2 \cdot \theta_A}\right)^2} \cdot t\right) \qquad (53)$$

Die Differenzialgleichung hätte also für eine Anfangsauslenkung von 3 Längeneinheiten, einer Dämpfung δ von 0.4 1/sec und einer Kreisfrequenz ω_d von 2 1/sec die in Bild 2.7 dargestellte folgende Form, die nur zur Veranschaulichung dient.

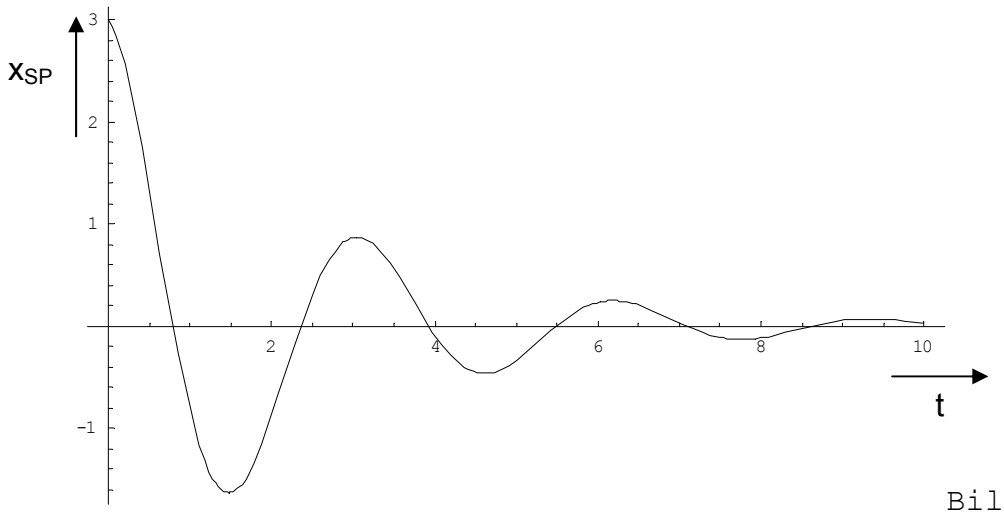


Bild 2.7

2.4.2. Numerische Berechnung eines beliebigen Stangenpunktes

Es lässt sich also die Auslenkung des Schwerpunktes analytisch bestimmen. Das Interessante hierbei ist zudem, dass aufgrund der angenommenen konstanten Krümmung der Stange (mit entsprechend großem Radius) es möglich ist, mittels der Parameterdarstellung des Kreises die Auslenkung der Stange jeden beliebigen Punktes zu ermitteln. Es ist dazu notwendig, ein Koordinatensystem der gebogenen Stange zu definieren. Zunächst soll jedoch der jeweilige Krümmungsradius in Abhängigkeit der Schwerpunktsauslenkung ermittelt werden, was im Anschluss bei der Parameterform der Kreisgleichung verwendet wird.
Es wurden bereits zwei Gleichungen für das Biegemoment ermittelt:

$$M_{Bieg} = \frac{E}{R} \cdot I_a \qquad \text{(Gleichung 16),}$$

sowie über die Herleitung und Lösung der Differenzialgleichung der Biegelinie:

$$M_{Bieg} = \frac{3 \cdot E \cdot I_a}{s^2} \cdot x_{SP} \qquad \text{(Gleichung 37)}$$

Es ist also erlaubt diese Gleichungen gleichzusetzen und darüber den Krümmungsradius R zu bestimmen:

$$\frac{E}{R} \cdot I_a = \frac{3 \cdot E \cdot I_a}{s^2} \cdot x_{SP} \qquad (54)$$

$$R = \frac{4 \cdot I_a \cdot s^2}{3 \cdot \pi \cdot r^4 \cdot x_{SP}} \qquad (55)$$

Zu beachten sei hierbei, dass x_{SP} nach wie vor eine Funktion der Zeit (t) ist. Zur Verdeutlichung einer späteren Berechnung sei sich vorzustellen, es seien im (begrenzten) Speicherbereich eines Rechners diskrete Werte x_{SP} gespeichert, woraus sukzessive die jeweils zugehörigen Werte R ermittelt werden.

Es wird im Folgenden das Koordinatensystem der Pendelstange definiert:

Man betrachte das Koordinatensystem (η,ξ) als kartesisches Koordinatensystem eines euklidschen Raumes, wie er auch für die Momentengleichung schon angenommen wurde:

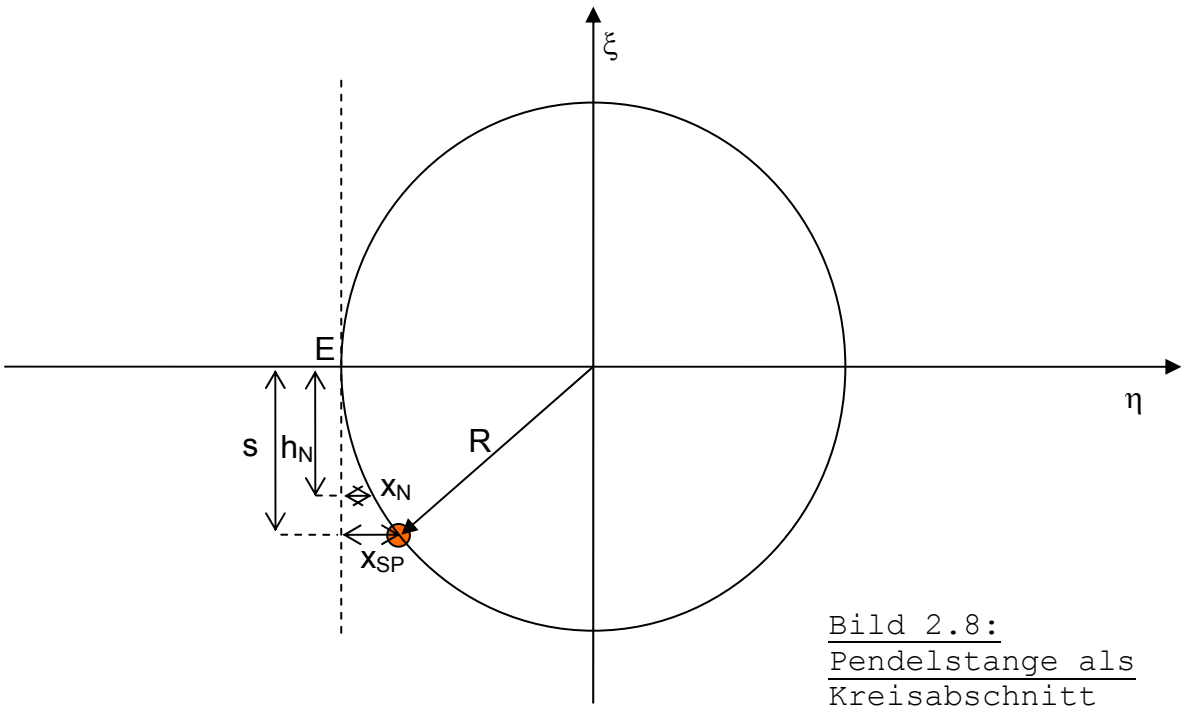


Bild 2.8: Pendelstange als Kreisabschnitt

Definition:
Der Kreisbogen im 3. Quadrant des Koordinatensystems repräsentiere hierbei die gebogene Pendelstange. So liegt die Einspannung im Punkt E. Der rote Punkt markiert den Schwerpunkt auf der ausgelenkten Stange. Die Höhe auf der gestrichelten Senkrechten bis zum Schwerpunkt wird als s bezeichnet. Hierbei wird angenommen, dass die normalerweise zu beachtende Verkürzung der Höhe auf der gestrichelten Projektionslinie eines Punktes x_N hier aufgrund der geringen Auslenkung und damit geringen Winkel vereinfacht wird zu einer Konstanz dieser Höhe (h_N). Das bedeutet für den Schwerpunkt, er sei im nicht verbogenen Zustand die Höhe (Senkrechte) von E ab definiert als s, sowie s auch als Höhe ab E für den *ausgelenkten* Zustand definiert ist. Analog gelte dies für alle Punkte x_N und den diesbezüglichen Höhen h_N. Dies ist konform mit der Vereinfachung des Sinus nach Kapitel 2.1.
Es gilt die Parameterform der Kreisgleichung:

$$\eta^2 + \xi^2 = R^2 \qquad (56)$$

Für einen beliebigen Punkt x_N sei daher:

$$\eta_N^2 + \xi_N^2 = R^2 \qquad (57)$$

Koordinatensubsitution an ξ:

$$\xi = -h_N \qquad (58)$$

Koordinatensubstitution an η:

$$\eta_N = -(R - x_N) = x_N - R \qquad (59)$$

$$h_N{}^2 + (x_N - R)^2 = R^2 \qquad (60)$$

Löst man diese quadratische Gleichung, so ergibt sich für die Auslenkung eines beliebigen Punktes N und dessen Auslenkung x_N als Element der natürlichen Zahlen, der definiert sei über die Höhe ab der Einspannung E ‚h_N':

$$x_N = R \pm \sqrt{R^2 - h_N{}^4} \qquad (61)$$

Diese Gleichung gilt für Punkte oberhalb und unterhalb des Schwerpunkts.
Angemerkt sei an dieser Stelle nochmals: h_N wird stets ab der Einspannung E gemessen.

Mit dem eingesetzten R ergibt sich die Gleichung:

$$x_N = \frac{4 \cdot I_a \cdot s^2}{3 \cdot \pi \cdot r^4 \cdot x_{SP}} \pm \sqrt{\left(\frac{4 \cdot I_a \cdot s^2}{3 \cdot \pi \cdot r^4 \cdot x_{SP}}\right)^2 - h_N{}^4} \qquad (62)$$

Zur Verdeutlichung der Zeitabhängigkeit kann man auch schreiben:

$$x_N(t) = \frac{4 \cdot I_a \cdot s^2}{3 \cdot \pi \cdot r^4 \cdot x_{SP}(t)} \pm \sqrt{\left(\frac{4 \cdot I_a \cdot s^2}{3 \cdot \pi \cdot r^4 \cdot x_{SP}(t)}\right)^2 - h_N{}^4} \qquad (63)$$

Somit ist nachgewiesen, dass sich für beliebige zeitabhängige Auslenkungen x_{SP} jeweils eine Auslenkung x_N pro Zeiteinheit eines gewählten Punktes n, der durch die Höhe h_N definiert ist, berechnen lässt.
Es sind jetzt nur noch N Punkte in beliebigen, jedoch möglichst äquidistanten Höhen h_N zu wählen, die in einem späteren Programm nur noch mittels Kreisinterpolation zu verknüpfen sind. Die Lage der Punkte pro Zeitschritt ist hierbei also eindeutig durch (x_N,h_N) definiert, wobei h_N für jeden Punkt als konstant angenähert wird.

3. Bestimmung der Pendelparameter

Aus den bisher hergeleiteten Gleichungen gehen diejenigen Parameter hervor, die von den Pendelkomponenten, wie Stange und Zylinder und von deren Ausrichtung zueinander abhängen:

3.1. Einzelparameter

3.1.1. Die Pendelstange

Bei der Pendelstange handelt es sich um einen axialsymmetrischen Metallstab mit Kreisquerschnitt.

Der Radius des Kreisquerschnittes wird mit r bezeichnet und beträgt bei dem benutzen Modell der Firma Phywe:

Radius: r = 0.003 m

Die Länge L der Pendelstange insgesamt beträgt:

Länge: L = 0.7 m

Die **Teillänge** ab der Einspannung wird mit **l** bezeichnet!

Die Masse m der Stange beträgt:

Masse: m = 0.1529 kg

Das ergibt eine Dichte ρ von:

Dichte: ρ =7725,3304 kg/m^3
Diese benötigt man, um die tatsächlich wirkende Masse der Stange im eingespannten Zustand zu berechnen.

Das Flächenträgheitsmoment zweiter Ordnung I_a (FTM2.O):

$I_a = \int_A r^2 dA$, wobei $dA = \pi \cdot r \cdot dr$,
wobei r ein beliebiger infinitesimal kleiner Radius der Pendelstange sei.

Daraus folgt:

$$I_a = \pi \int_A r^3 dr = \frac{r^4}{4}\pi \qquad (64)$$

Für diese Stange ergibt sich mit dem Radius r = 3mm

FTM2.O: I_a = 63,6173 (mm)^4

Das E-Modul E der Pendelstange wird aus den Messungen des Biegemomentes berechnet (siehe Kapitel 4):

E-Modul: E = 2,02716E+11 N/m²

Das Massenträgheitsmoment θ_{Stg} der Stange bezüglich ihres eigenen Schwerpunkts wird folgendermaßen berechnet:

MTM: $$\theta_{Stg} = \frac{1}{12} \cdot m \cdot l^2 \qquad (65)$$

Wie bereits erwähnt kann der in Kapitel 2.4 erwähnte Dämpfungsfaktor c aus den Messungen ermittelt werden, indem die dortige Kreisfrequenz ω_d abgelesen wird. Es gilt der schon ermittelte Zusammenhang

$$\omega_d = \sqrt{\omega_0^2 - \delta^2} \text{ , mit } \quad \delta = c \cdot \frac{s}{2 \cdot \theta_A} \quad \text{und} \quad \omega_0^2 = \left(m \cdot g + \frac{3 \cdot E \cdot I_a}{s^2} \right) \cdot \frac{s}{\theta_A}$$

3.1.2. Der Pendelzylinder

Der Radius R des Pendelzylinders beträgt:

Radius: R = 35 mm

Die Masse M des Pendelzylinders beträgt:

Masse: M = 2 kg

Die Höhe des Pendelzylinders ist:

Höhe: H = 68 mm

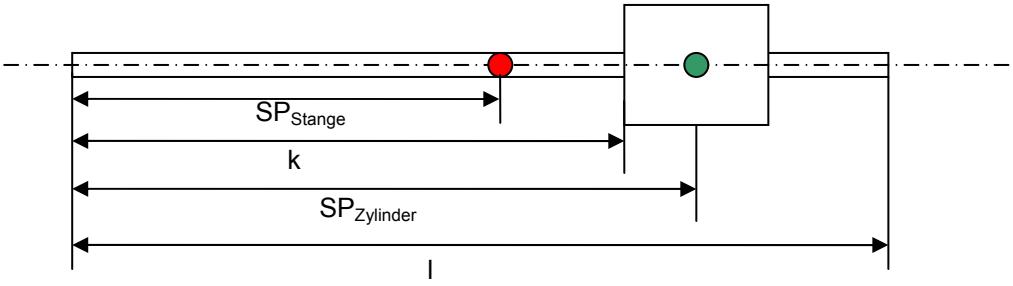
Das Massenträgheitsmoment des Zylinders wird folgendermaßen berechnet:

MTM: $$\theta_{Zylinder} = \frac{M \cdot R^2}{2} \qquad (66)$$

3.2. Gemeinsame Parameter

3.2.1. Schwerpunktsberechnung

Für die Schwerpunktsberechnung betrachte man Stange und montierten Zylinder im unausgelenkten Zustand von der Seite. Der rote Punkt in der folgenden Skizze markiert den Schwerpunkt der Stange, der grüne Punkt den Schwerpunkt des Zylinders (die Mitte des Kreises bezeichnet den jeweils gemeinten Punkt; der Kreis wurde nur zur besseren Visualisierung gewählt). Der hier neu eingeführte Parameter k bezeichnet den Abstand zwischen Einspannung und Zylinder. In der Skizze befindet sich die Einspannung auf der linken Seite.



3.2.1.1. Betrachtung des Zylinders

Schwerpunkt des Zylinders:

$$SP_{Zylinder} = \frac{H}{2} + k \qquad (67)$$

Fläche des Zylinders:

$$A_{Zylinder} = 2 \cdot H \cdot R \qquad (68)$$

3.2.1.2. Betrachtung der Stange

Schwerpunkt der Stange:

$$SP_{Stg} = \frac{l}{2} \qquad (69)$$

Fläche der Stange:

$$A_{Stg} = 2 \cdot l \cdot r \qquad (70)$$

3.2.1.3. Gemeinsamer Schwerpunkt

Die Flächensumme beider Komponenten ist:

$$A_{\Sigma} = A_{Gewicht} + A_{Stg} \qquad (71)$$

Der gemeinsame Schwerpunkt s (gemessen ab Einspannung) ist die Summe beider Schwerpunktsstrecken jeweils multipliziert mit der Komponentenfläche geteilt durch die Flächensumme.

$$s = \frac{SP_{Zylinder} \cdot A_{Zylinder} + SP_{Stg} \cdot A_{Stg}}{A_{\Sigma}} \qquad (72)$$

Für die angenommenen Komponentenparameter ergibt sich für s:

$$s = \frac{l^2 \cdot 3mm + (34mm + k) \cdot 0{,}00476m^2}{0{,}00476m^2 + l \cdot 6mm} \qquad (73)$$

3.2.2. Trägheitsmomentenberechnung

Nach dem Satz von Steiner ist das Trägheitsmoment bezüglich der Einspannung θ_A definiert als:

$\theta_A = \theta_S + m_{\Sigma} \cdot s^2$, wobei m_{Σ} hier die Gesamtmasse der beiden Komponenten darstellt. θ_S ist hier das Trägheitsmoment bezüglich des gemeinsamen Schwerpunktes, was als Summe des Stangenträgheitsmomentes und des Zylinderträgheitsmomentes definiert ist:

$$\theta_S = \frac{1}{12} \cdot m \cdot l^2 + \frac{M \cdot R^2}{2} \qquad (74)$$

So ergibt sich für das Trägheitsmoment θ_A:

$$\theta_A = \frac{1}{12} \cdot m \cdot l^2 + \frac{M \cdot R^2}{2} + m \cdot s^2 \qquad (75)$$

Dieses Massenträgsheitsmoment lässt sich auch aus den Messungen ermitteln. Dies lässt sich mittels des Zusammenhangs

$\omega_0^2 = \left(m \cdot g + \frac{3 \cdot E \cdot I_a}{s^2} \right) \cdot \frac{s}{\theta_A}$ bestimmen, nachdem ω_0 ermittelt wurde aus dem zu messenden ω_d und dem durch Messungen ermittelten Dämpfungsfaktor c.

4. Messwertanalyse

4.1. Untersuchung der DMS-Spannung im statischen Fall

Zunächst wurden die Größen Spannung und Auslenkung im Schwerpunkt im statischen Fall untersucht.
Die Auslenkungen wurden in 38 cm Abstand von der Einspannung gemessen.
Hierfür wurden jeweils für den Verstärkungsfaktor V=100 μm/m und V=300 μm/m die DMS-Spannung in x-Richtung, sowie in y-Richtung untersucht. Für die Messungen mit V=300 μm/m wurden jeweils sieben Auslenkungen von 0 bis 30 mm betrachtet (in 5 mm –Schritten), während für die Messungen mit V=100 μm/m eine geringere Maximalauslenkung von 15 mm genommen werden musste (d.h. 4 Messwerte), da die resultierende Ausgangsspannung von ca. 1 Volt nicht mehr auf dem x-y-Messschreiber darstellbar ist. Die Messreihen hierzu sind in der Excel-Datei „Messreihen_26022002.xls“ dargestellt.
Die Ergebnisse sind wie folgt zusammenzufassen:
Der Korrelationskoeffizient der Messreihen beträgt bei allen >0,999, was bedeutet, dass, wie für Dehnungsmessstreifen erwartet werden kann, die Messverstärkerspannung in Abhängigkeit der Auslenkung im Schwerpunkt linear angenähert werden kann. So sind die Proportionalitätsfaktoren wie folgt berechnet:

Bei V=100 μm/m : 6.61333E-2 V/mm
Bei V=300 μm/m : 2.30667E-2 V/mm

Tabelle 5.1 zeigt die Messreihen aus der Excel-Datei.

Tabelle 5.1: DMS-Spannungswerte im statischen Fall

X-Spannung				
300 µm / m				
Auslenkung X / mm	X-Spannung / V 1	X-Spannung / V 2	X-Spannung / V 3	Mittelwert 1,2,3 /V
0	0,000	0,000	0,000	0,000
5	0,108	0,108	0,106	0,107
10	0,222	0,221	0,223	0,222
15	0,343	0,347	0,337	0,342
20	0,457	0,455	0,454	0,455
25	0,569	0,572	0,572	0,571
30	0,696	0,690	0,689	0,692
Y-Spannung				
300 µm / m				
Auslenkung Y / mm	Y-Spannung / V 1	Y-Spannung / V 2	Y-Spannung / V 3	Mittelwert 1,2,3 /V
0	0,000	0,000	0,000	0,000
5	0,102	0,113	0,102	0,106
10	0,217	0,231	0,231	0,226
15	0,326	0,339	0,330	0,332
20	0,438	0,442	0,433	0,438
25	0,555	0,561	0,556	0,557
30	0,666	0,667	0,675	0,669
X-Spannung				
100 µm / m				
Auslenkung X / mm	X-Spannung / V 1	X-Spannung / V 2	X-Spannung / V 3	Mittelwert 1,2,3 /V
0	0,000	0,000	0,000	0,000
5	0,350	0,349	0,343	0,347
10	0,700	0,687	0,690	0,692
15	1,060	1,050	1,050	1,053
Y-Spannung				
100 µm / m				
Auslenkung Y / mm	Y-Spannung / V 1	Y-Spannung / V 2	Y-Spannung / V 3	Mittelwert 1,2,3 /V
0	0,000	0,000	0,000	0,000
5	0,320	0,323	0,317	0,320
10	0,670	0,655	0,640	0,655
15	0,992	0,998	0,987	0,992

4.2. Messung des statischen Biegemoments

Es wurden zunächst für 3 Auslenkungen im Schwerpunkt die Rückstellkraft mit Kraftmessfedern gemessen, was in Tabelle 4.2 dargestellt ist.

Auslenkung x im Schwerpunkt	Rückstellkraft im Schwerpunkt (F_{Bieg})
1 cm	5.8 N
2 cm	11.8 N
3 cm	18.0 N

Tabelle 4.2: Kraft in Abhängigkeit der Auslenkung

Aus dem Zusammenhang

$$M_{Bieg} = F_{Bieg} \cdot s = \frac{3 \cdot E \cdot I_a}{s^2} \cdot x$$

Mit den bereits ermittelten Parametern I_a und s lässt sich also das E-Modul berechnen. Es hat den Wert von

E = 202716090369 N/m²

An den Messungen erkennt man recht gut, dass die Rückstellkraft und somit auch das Biegemoment linear zunehmen. Vergleicht man den ermittelten Wert des E-Moduls mit gängigen Werten des E-Moduls für Stahl aus Tabellenwerken, die im Bereich 186 bis 216 kN/mm² liegen, so wird der Wert als wahr angenommen.
Das Modell des Biegemoments ist daher als korrekt zu bezeichnen.

4.3. Messung der Spannung und Schwingungsauswertung

Bild 4.1 zeigt den Verlauf der Spannung über der Zeit für die nachfolgenden Parameter, aufgenommen mit dem x-y-Schreiber.

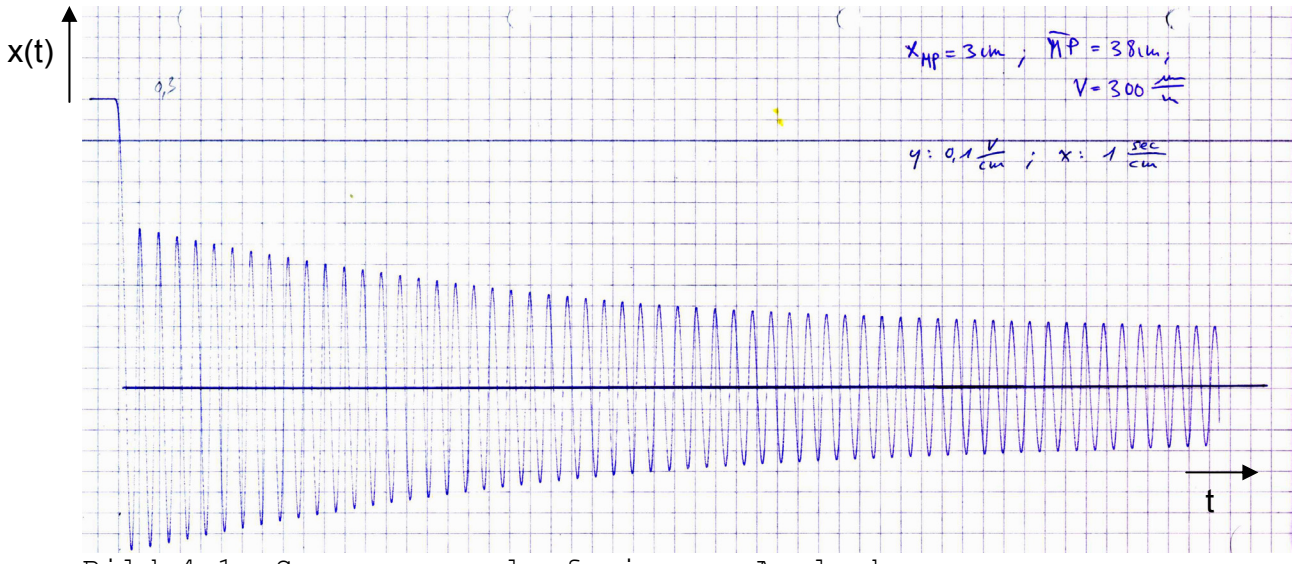


Bild 4.1: Spannungsverlauf einer x-Auslenkung

Die Parameter dieser Spannungsaufnahme sind auf diesem Messblatt angegeben mit:

$x_{SP}(t=0) = 3$ cm [Anfangsauslenkung]

$s = 0.377$ m [Schwerpunktlage]

(2 Kästchen entprechen 1cm):

Die Verstärkung des Messverstärkers liegt bei V=300 µm/m.

Ein Zentimeter auf der Zeitachse entspricht einer Sekunde.

Ein Zentimeter auf der Spannungsachse entspricht 0.1 Volt.

Die Messung des Bilds 4.1 dient exemplarisch der Auswertung und Veranschaulichung aller Messungen. Nach diesem speziellen Fall wird nochmals für einen zweiten Fall die Messauswertung in kurzer Form durchgeführt.

Auffällig an allen Messungen ist der sehr hohe Abfall der Spannung nach dem Durchlauf der ersten Periode, wie es auch in Bild 4.1 zu bemerken ist.
Zudem lässt sich feststellen, dass die Periodenzeit dieser ersten Periode länger ist, als die Periodenzeit der folgenden Perioden. Die Schwingung der ersten Periode wird nachfolgend als parasitäre Schwingung bezeichnet, die folgenden Schwingungen als Folgeschwingung.

Da die ermittelte Differenzialgleichung nach (48) jedoch von einer konstanten Dämpfung, d.h. konstanten Abnahme der Auslenkung und damit der Spannung (beide sind zueinander proportional), ausgeht, stellt sich die Frage, welche der hier offensichtlich vorhandenen Schwingungen (ausgehend von den Dämpfungen) von der Differenzialgleichung geliefert wird.

Aufgrund der unterschiedlichen Dämpfungskoeffizienten und Frequenzen beider vorhandenen Schwingungen wird nun ein Modell eingeführt, das rein mathematisch diese Funktion der Zeit insgesamt beschreibt, worauf basierend die Dämpfungskoeffizienten und Frequenzen aus den Messungen ermittelt werden können. Über die Kreisfrequenz der nicht-parasitären Schwingung lässt sich die Kreisfrequenz der nicht gedämpften Schwingung ω_0 ermitteln, woraus sich ein messtechnisch ermitteltes Massenträgheitsmoment θ_A berechnen lässt. Dies wird mit dem theoretisch ermittelten (gem. (6)) verglichen.

Die messtechnisch ermittelten Dämpfungsfaktoren und Periodenzeiten sind in der Excel-Datei ‚Systemberechnung.xls' in Abhängigkeit der Schwerpunktlage s eingegeben worden, woraus sich in Excel interpolierte Funktionen ergeben.

Da es sich bei den Messungen um zwei zusammengesetzte Schwingungen handelt, sei im Folgenden diese Funktion der Zeit als f(t) bezeichnet.

Definition:

Es sei die Funktion f(t) definiert als:

$$\begin{cases} f(t) = r_1 \cdot e^{-\frac{c_1 \cdot s}{2 \cdot \theta_A} \cdot t} \cdot \cos\left(\frac{2\pi}{T_1} \cdot t\right) \text{ für } t \le T_1 \\ f(t) = r_2 \cdot e^{-\frac{c_2 \cdot s}{2 \cdot \theta_A} \cdot (t - T_1)} \cdot \cos\left(\frac{2\pi}{T_2} \cdot (t - T_1)\right) \text{ für } t > T_1 \end{cases} \qquad (76)$$

wobei $t \ge 0$

r_1 bezeichnet die Anfangsauslenkung der Pendelstange im Schwerpunkt oder, wenn man die Spannung betrachtet, die Anfangsspannung. Da Auslenkung und Spannung zueinander proportional sind ändert sich gemäß der jeweils betrachteten Größe nur der Faktor r_1.

r_2 bezeichnet die Auslenkung oder die Spannung nach dem Durchlauf der parasitären Schwingung.

Es ergeben sich aus den Anfangswerten der verwendeten x- Aufzeichnung in Bild 4.1 für den Fall eines Schwerpunktes in s=0.3776 m und einer Anfangsauslenkung von x(t=0)=0.003 m = r_1 die folgenden Periodenzeiten:

T_1 = 0.6 s [Periodenzeit der parasitären Schwingung]
T_2 = 0.45 s [Periodenzeit der Folgeschwingung]

Es gilt für r_2:

$$r_2 = r_1 \cdot e^{\frac{-c_1 \cdot s}{2 \cdot \theta_A} T_1} \qquad (77)$$

Nach der normierten Differenzialgleichung zweiter Ordnung ist die Dämpfung wie folgt definiert.

$$\delta = \frac{cs}{2\theta_A}$$

Es können aus der Messung nach Bild 4.1 folgende δ-Werte ermittelt werden (aus dem Ansatz f(t=0) und f(t=T_1)):

δ_1 = 2,87933629 1/sec

δ_2 = 0,166550248 1/sec

Nach $\omega_d = \frac{2 \cdot \pi}{T}$ lassen sich nun die Kreisfrequenzen aus der Messung ermitteln:

Für T_1: ω_d = 10.4719 1/sec
Für T_2: ω_d = 13.9626 1/sec

Die Lösung der Differenzialgleichung liefert jedoch ein ω_d = 16.61 1/sec, weshalb die Lösung der DGL nur die Modellierung des zweiten Funktionsteils nach (76) darstellen kann.

Es wurden nun nach diesem Schema für unterschiedliche Schwerpunkte, d.h. Lagen k des Zylinders die Dämpfungskoeffizienten δ_1 und δ_2 ermittelt, wofür 8 Messungen ausgewertet wurden. Durch diese Messwerte lassen sich nun durch Interpolation Funktionen $\delta_1(s)$ und $\delta_2(s)$ ermitteln. Dies wurde hier mit Excel durchgeführt, was in Bild 4.2 und Bild 4.3 gezeigt ist:

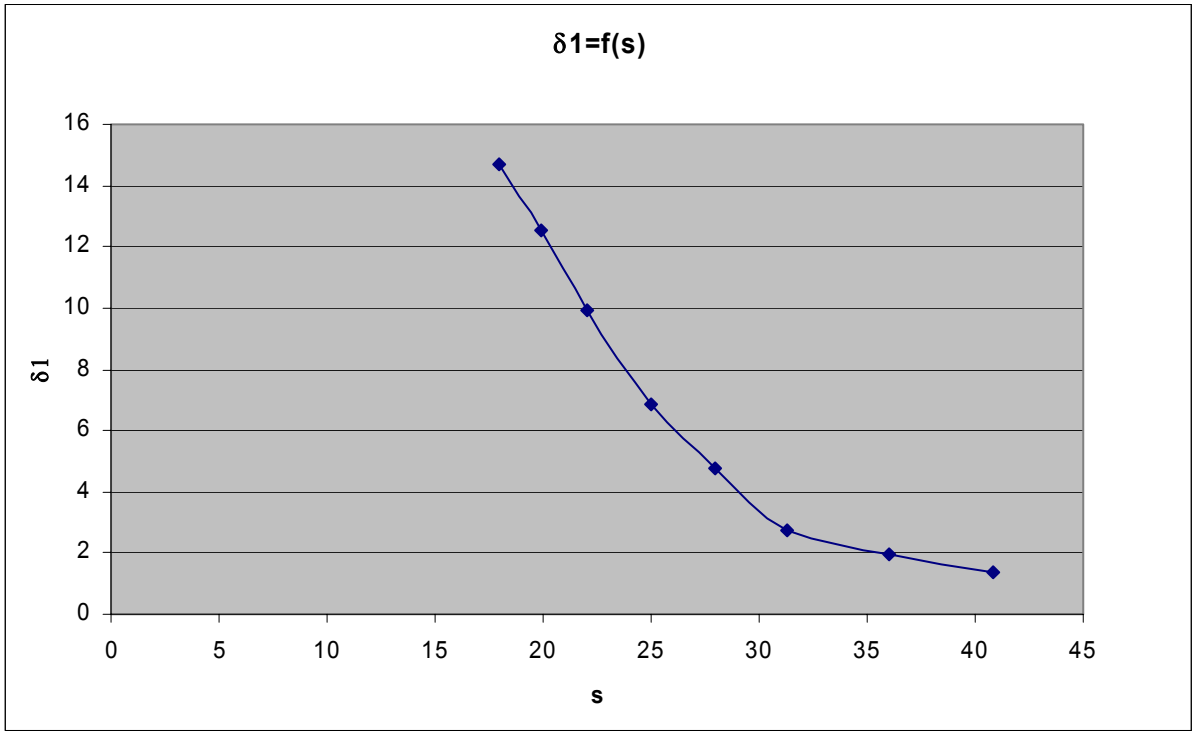


Bild 4.2: $\delta_1(s)$

Es lässt sich feststellen, dass es sich in beiden Fällen um einen umgekehrt proportionalen Verlauf handelt. Um in Excel die Werte für beliebige Schwerpunkte zu ermitteln, wurde auf die Excel-Funktion „VARIATION“ zurückgegriffen, welche die Regressionsfunktion für exponentielle Verläufe ermittelt. Optimaler wäre es direkt ein Polynom nach der Newton-Interpolation zu entwickeln, da dort eine noch geringere Abweichung vorhanden wäre. Die exponentielle Regression hat den Nachteil, dass man Abweichungen an den Stützstellen selbst hat.

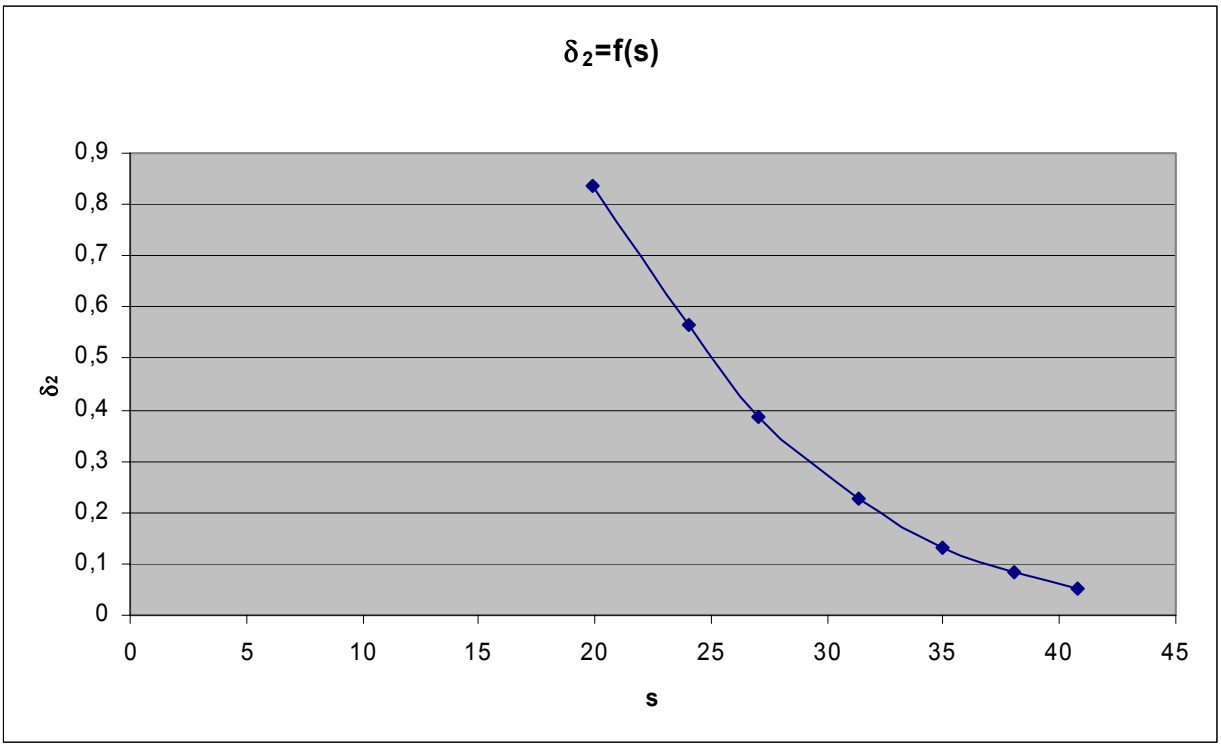


Bild 4.3: $\delta_1(s)$

Daher wurden die Koeffizienten der Newton-Interpolation für die Polynome bereits in Excel entwickelt. Nur eine Implementierung der Polynome war in Excel leider nicht möglich, da es keine direkte Funktionseingabe unterstützt.

Im Folgenden wird nur noch der zweite Funktionsteil nach (74) und somit die Lösung der Differentialgleichung betrachtet.

Wie festzustellen ist, liegt eine Differenz zwischen gemessenem und theoretisch ermitteltem ω_d vor. Der Dämpfungskoeffizient δ_2 wird aus der Messung entnommen und ist für beide Lösungen, f(t) und die der DGL, per definitionem identisch.

Es lässt sich jedoch aus den Messungen mittels der Gleichungen

$$\omega_0^2 = \left(m \cdot g + \frac{3 \cdot E \cdot I_a}{s^2} \right) \cdot \frac{s}{\theta_A}$$

$$\omega_d = \sqrt{\omega_0^2 - \delta_2^2}$$

zuerst das tatsächliche ω_0 ermitteln. Da die Korrektheit der Parameter m, g, E, I_a und s durch Messungen gewährleistet ist, bleibt nur noch eine Überprüfung auf den Parameter θ_A.

Aus den Messungen ergibt sich:
θ_A = 0,399754511 kg(m)²

Das theoretische θ_A ergibt sich aus Gleichung (6) und beträgt für den hier betrachteten Fall:

θ_A = 0,30385553 kg(m)²

Offensichtlich sind die in der Realität mitschwingenden Massen höher, da der messtechnisch ermittelte Wert für das Massenträgheitsmoment höher ist.

Interpretation:
Dies lässt sich damit begründen, dass in diesem theoretischen Modell (der Ermittlung der Differenzialgleichung) nur die Pendelstange und der Zylinder betrachtet wurden, während in der Realität zudem noch der Metallrahmen der Einspannung mitschwingt.
Es lässt sich mittels Parametervariation von l oder k, die in Excel durchgeführt wurde, feststellen, dass mit abnehmendem Schwerpunkt s die Differenz beider Werte θ_A abnimmt. Auch dies lässt sich physikalisch begründen:
Hohe Massen, wie die Rahmenmasse, dämpfen hohe Schwingfrequenzen, weshalb sich diese bei hohen Frequenzen auch nur gering an der Schwingung beteiligen. Daher ist das Massenträgheitsmoment für hohe Frequenzen, d.h. geringe Schwerpunktlagen, geringer, als bei hohen Schwerpunktlagen.

So liegt der relative Fehler, der auch in der Excel-Datei berechnet wird, bei unseren gewählten Simulationsparametern bei

F_r = 0,239894677

während er bei einem Schwerpunkt von s = 28.4 cm bei

F_r = 0,022929287

liegt.

Dieser Effekt des Mitschwingens des Rahmens kann nur mit der Methode der finiten Elemente mit in das theoretische Modell einbezogen werden.

Dasselbe gilt für den Effekt der hohen Energievernichtung (hohen Dämpfung), den die parasitäre Schwingung nach (76) darstellt. Aufgrund der hohen Eigenspannung des Stabs und der damit messbar geringeren Anfangsfrequenz schwingt der Metallrahmen des Pendelaufbaus stärker mit.
Diese Frequenz lässt sich aus derselben Excel-Datei entnehmen, da dort neben den Dämpfungskurven auch die Kurven der Periodenzeiten in Abhängigkeit des Schwerpunktes (s) gegeben sind.

Diese Kurven zeigen die Bilder 4.4 und 4.5
T_1 ist die Periodenzeit der parasitären Schwingung.
T_2 ist die Periodenzeit der Folgeschwingung.

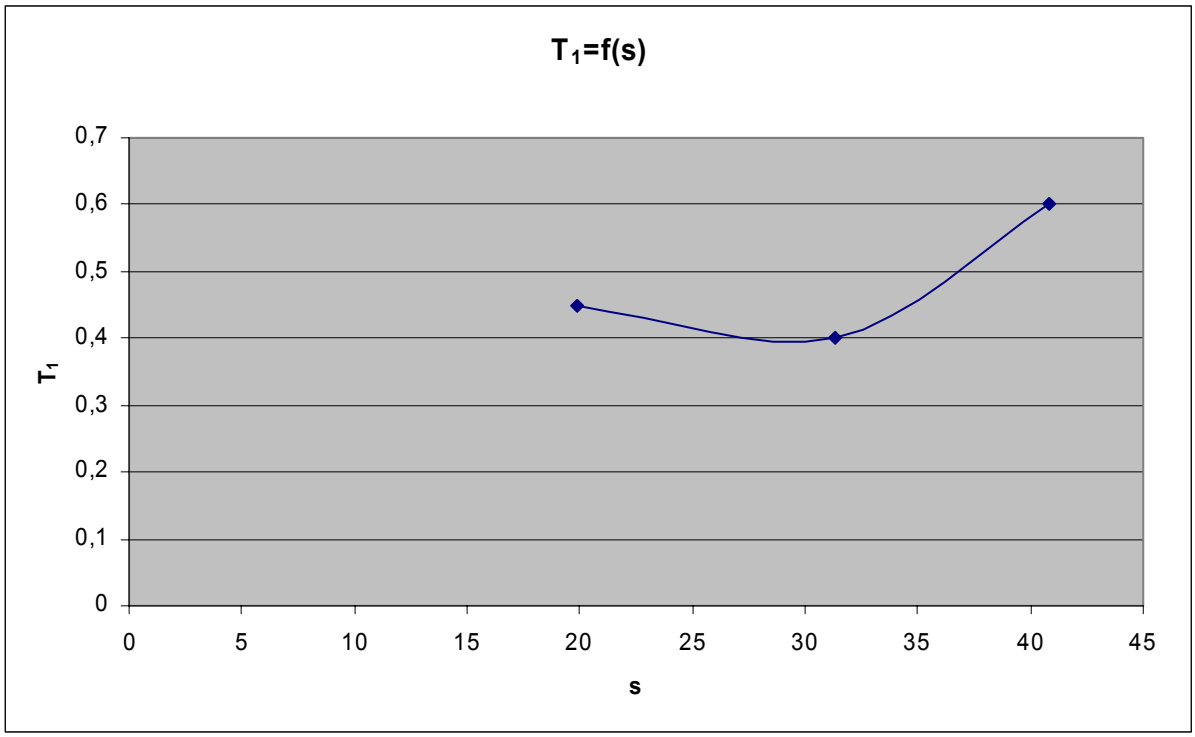


Bild 4.4: Periodenzeit für den ersten Funktionsteil nach (76)

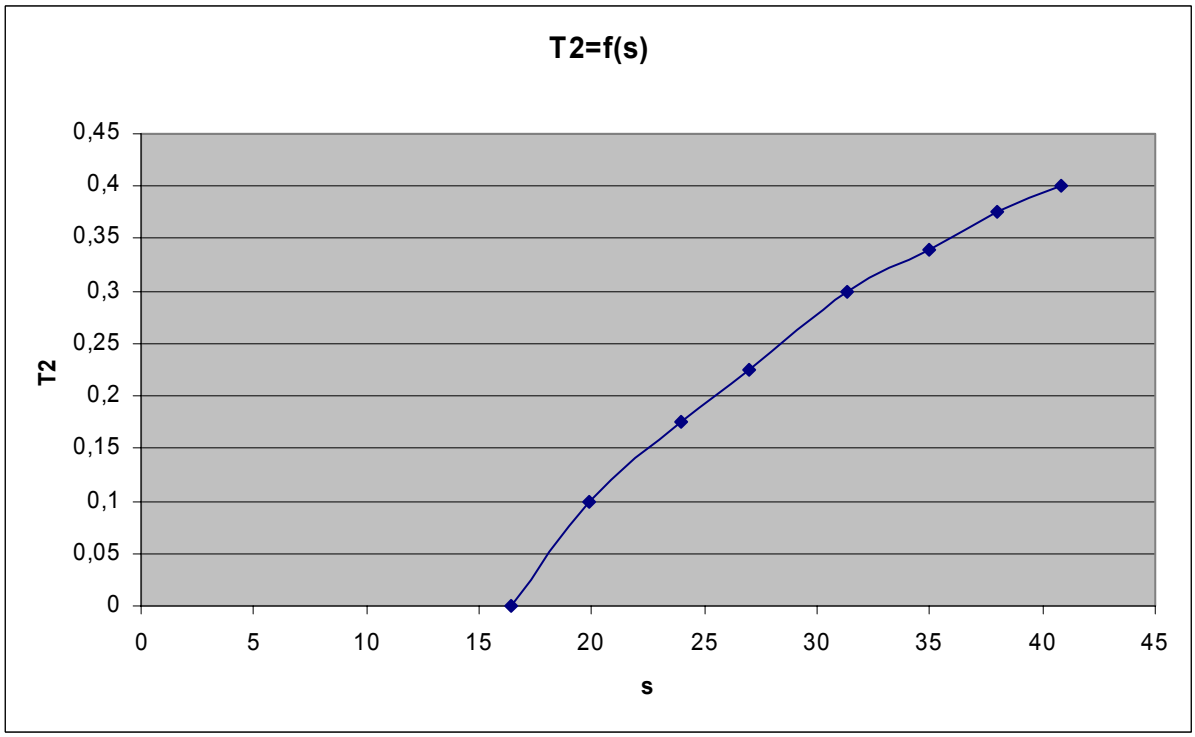


Bild 4.5: Periodenzeit für den zweiten Funktionsteil nach (76)

Mit Boris oder Mathematica lässt sich nun die Differenzialgleichung zweiter Ordnung simulieren, nachdem die Koeffizienten in Excel nach der Parameterwahl in der Folie „Systemkonstanten“ berechnet wurden.
Für die Simulation der Zusammengesetzen Schwingung ist Mathematica zu empfehlen, wozu sich das Notebook eignet, welches diese Hausarbeit ergänzt.

Dort wird der Funktionsteil 1 (die parasitäre Schwingung) nach (76) als Funktion p(x) bezeichnet, während Funktionsteil 2 nach (76) als Funktion e(x) definiert ist. x stellt hier die Zeitvariable dar. p und e jeweils die Auslenkungen oder die Spannung, die sich aus der Multiplikation der Auslenkung mit dem berechneten Proportionalitätsfaktor ergibt .
Für den hier dargestellten Fall liefert Mathematica folgende Funktionen:

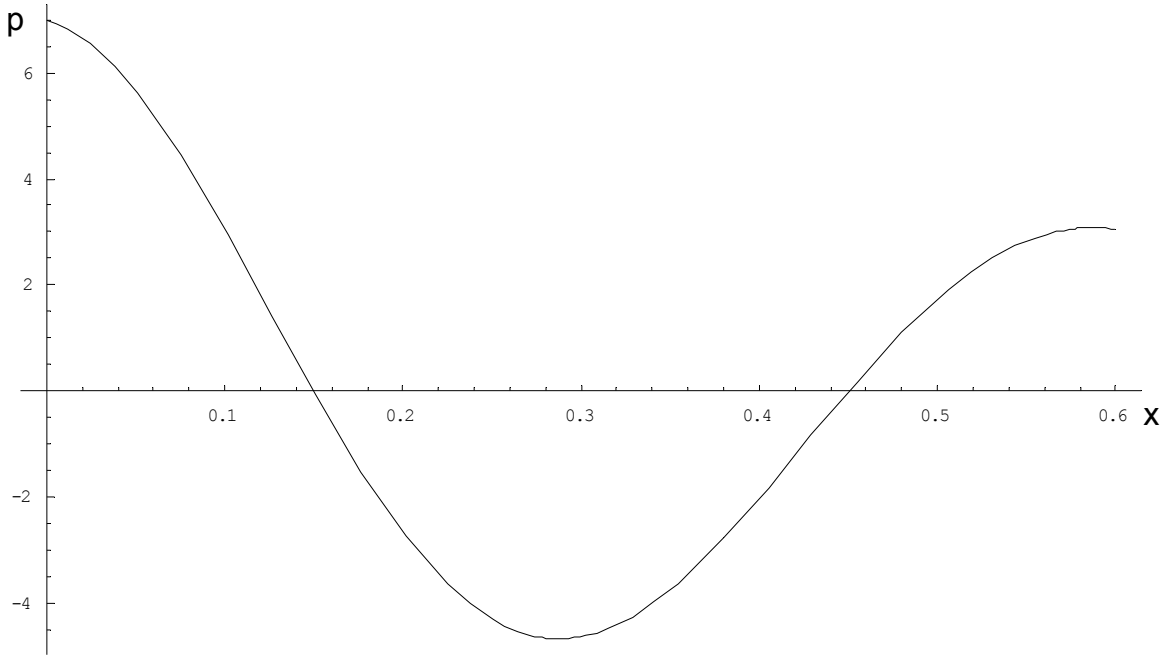


Bild 4.6: Funktionsteil 1 nach (76) in Mathematica

Bild 4.6 zeigt somit die Funktion f(t) nach (76) von t bis T_1. Die starke Dämpfung ist hier besonders gut zu sehen. Die Funktion startet bei 7 [cm], was jedoch nicht die Spannung, sondern die direkte Anzeige des x-Schreibers für V=300µm/m mit einer Skalierung von 0.1V/cm liefert. Da die Zusammenhänge zwischen Skalierung, Messverstärkerspannung und Auslenkung im Schwerpunkt proportional sind, lässt sich der jeweils gewünschte Kurvenverlauf der Größen mittels Ersetzen des Startwertes ermitteln.
Die Berechnung des Endpunktes von Funktionsteil 1 und damit des Anfangspunktes von Funktionsteil 2 ist auch in Excel implementiert. In der Folie „Auswertung“ unter dem Punkt „Berechnung des Anfangswertes der Zweitfunktion“ wird in Abhängigkeit von der einzugebenden Anfangsauslenkung dieser Wert berechnet.
Dieser Wert ist dann in die Funktion 2 von Mathematica oder in die Borissimulation einzusetzen.

```
p[x_] = {b * e^(-k * s * x / (2 * t)) * Cos[(2 * π * x / T1)]}
e[x_] = {d * e^(-u * s * (x - T1) / (2 * t)) * Cos[(2 * π * (x - T1) / T2)]}
```

Bild 4.7 Funktionen in Mathematica nach (76)

In Bild 4.7 sind der Vollständigkeit halber die in Mathematica definierten Funktionen gezeigt. Zur Definition der Variablen sei hier auf das Mathematica-Notebook , lsg_dgl_var_par1.nb’ verwiesen.

Die Funktion e[x] bzw. der Funktionsteil 2 nach Gleichung (76) in Mathematica ist in Bild 4.8 abgebildet:

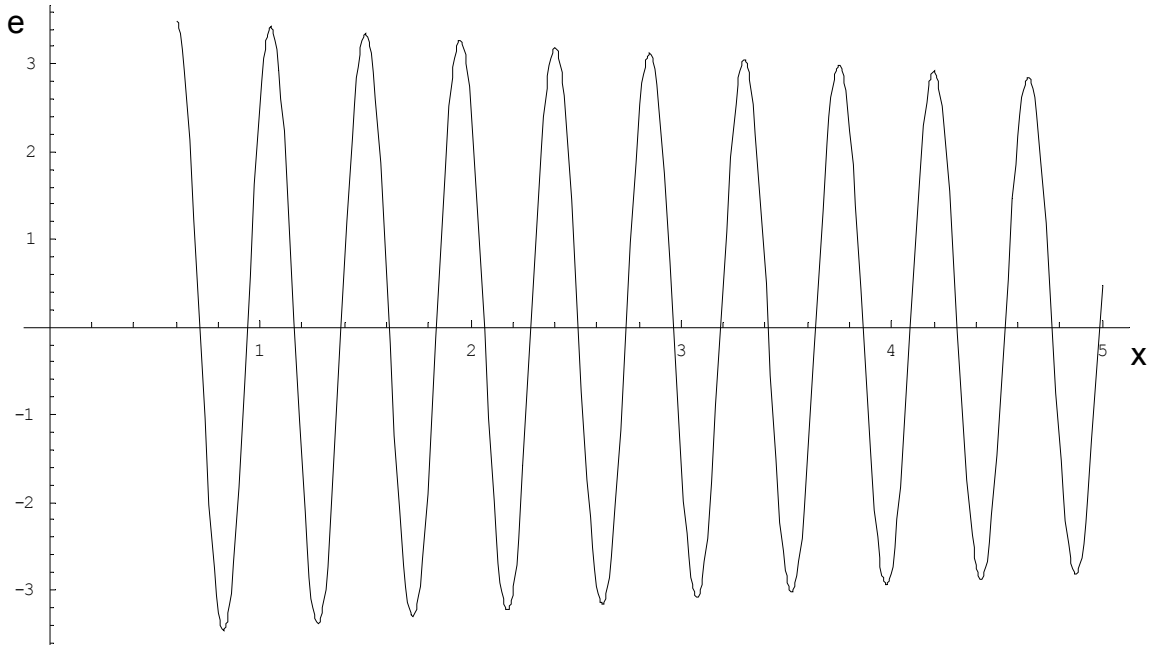


Bild 4.8: Funktionsteil 2 nach (76) in Mathematica

Zweitmessung:
Zur Verdeutlichung wird hier noch eine zweite Messung für den Schwerpunkt von s = 31.32 cm angeführt (Bild 4.9). Die Anfangsauslenkung $x_{SP}(t=0)$ beträgt 2 cm.

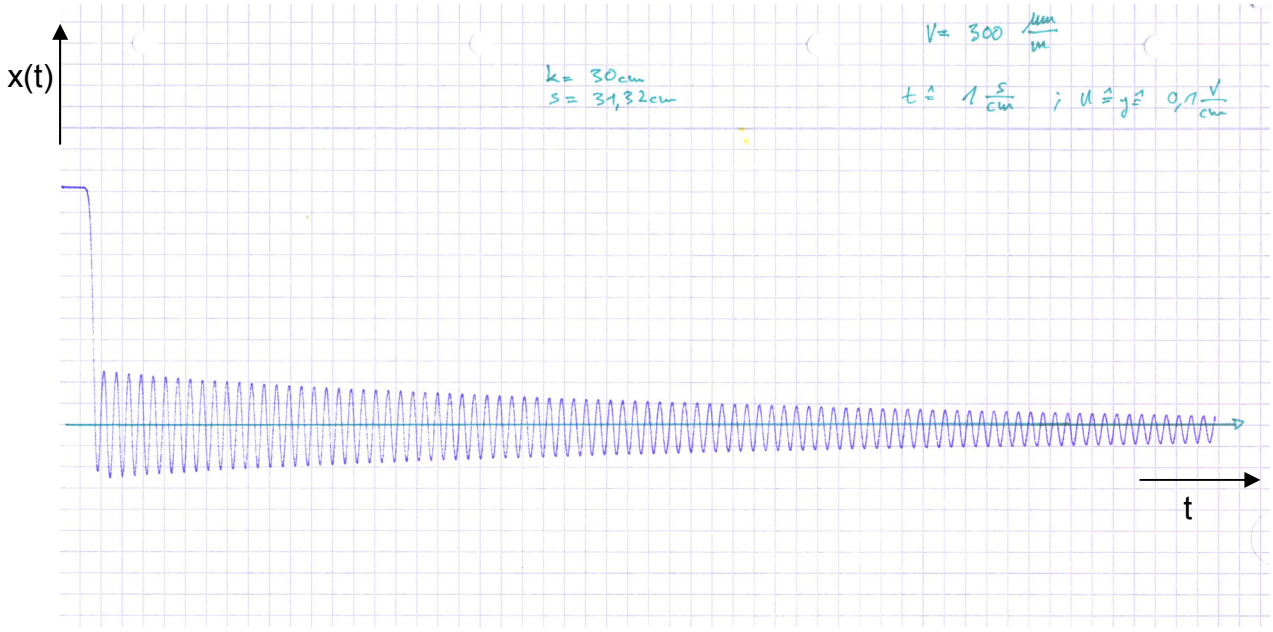


Bild 4.9: Messung mit s = 30, cm und $x_{SP}(t=0)=2$ cm

Aus der Vermessung des Graphen in Bild 4.9 geht der Anfangswert von

r_1 = 2 cm, und der Fortsetzungswert r_2 = 0,459 cm, jeweils im Schwerpunkt gemessen hervor.

Für die Periodendauern der Einzelfunktionen (gem. (76)) ermittelt man hier:

$T_1 = 0.5$ sec

$T_2 = 0.3$ sec

Es werden folgende Dämpfungswerte ermittelt:

$\delta_1 = 2.7701$/sec

$\delta_2 = 0.242$ 1/sec

Diese Werte werden von der Excel-Tabelle mit einer Schwerpunktabhängigen Abweichung ermittelt, was an der suboptimalen Interpolationsfunktion „VARIATION“ liegt, die hierfür benutzt wurde.

Es ergibt sich ein tatsächliches Massenträgheitsmoment von

$\theta_A = 0.22278$ kg(m)²

Das theoretische liegt bei

$\theta_A = 0.1937$ kg(m)²

Daher ist auch ein Unterschied in der gemessenen Kreisfrequenz ω_d der Folgeschwingung bemerkbar.

Das tatsächliche liegt hier bei

$\omega_d = 20.94$ 1/sec

Die theoretische Kreisfrequenz liegt bei

$\omega_d = 19.61$ 1/sec

Es würde also in der Simulation der Folgeschwingung nur ein geringer relativer Fehler von 7% ergeben. Dieser ist auf den Spannungsgraphen des x-y-Schreibers kaum messbar und daher annehmbar.

Die Interpolation der Messwerte für die Dämpfungen in Abhängigkeit des Schwerpunktes ist noch zu optimieren (Newton-Interpolation), da die verwendete exponentielle Regressionsfunktion gerade für hohe Schwerpunktlagen s falsche (höhere) Dämpfungswerte ermittelt.

Weitere Messungen befinden sich in gescannter Form in der Anlage dieser Ausarbeitung (Kapitel 7).

5. Simulation mit Mathematica und Boris

Zunächst sollte die Simulation auf Basis des Mehrkörpersimulationsprogramm „Adams“ durchgeführt werden, jedoch stellte sich heraus, dass man für die Simulation der Biegung ein zusätzliches Programm benötigt, das zu Adams kompatible finite Element-Matrizen erzeugt, die in die Mehrkörpersimulationssoftware geladen werden. Jedoch besitzt die Hochschule solch ein Programm nicht, weshalb die Simulation mit Adams als nicht sinnvoll zu erachten ist. Zwar könnte man die Möglichkeit in Erwägung ziehen, den zu verbiegenden Stab in diskrete Abschnitte zu zerlegen, die mittels „Joints“, also definierter Verknüpfungen, miteinander verbunden werden. Um die Reibung zu simulieren, könnten zwischen den Flächen „Friction“-Werte angenommen werden. Aufgrund fehlendem mathematischen Modell für diesen Fall verbliebe die restliche Arbeit im Schätzen und Probieren von passenden „Friction“-Werten, was nicht Sinn dieser Simulationsarbeit sein kann. Außerdem würde sich die Parametervariation als schwierig gestalten, da das gesamte Modell beispielsweise für einen neuen k-Wert (Abstand Einspannung bis Zylinder) komplett neu entwickelt werden müsste. Eine Simulation mit der blockorientierten Simulation „Boris“ erscheint daher wesentlich angebrachter, da diese eine direkte Implementierung der Differenzialgleichung inklusive der Parameterberechnung gestattet, was jedoch das Modell zu unübersichtlich gestalten würde. Für die Parametervariation und Dämpfungsberechnung wird Excel verwendet. Die Datei, auf die hierfür referenziert wird nennt sich „systemberechnung.xls“. Zudem ist es möglich die ermittelten Werte für die Auslenkung über die DDE-Schnittstelle (Dynamic Data Exchange) an ein anderes Programm zu übergeben. Somit ließe sich mittels des in Kapitel 2.4.2 dargestellten Verfahrens die jeweilige Auslenkung für N gewählte Stangenpunkte ermitteln und über eine Interpolation derselben die sich verbiegende Stange simulieren.
Mathematica stellt eine sinnvolle Ergänzung für die Simulation dar, da sich hiermit die später hergeleitete, zusammengesetzte Funktion der Auslenkung darstellen lässt.
Außerdem stellt Mathematica den korrekten Kurvenverlauf dar, während Boris bei unterschiedlichen Parametern zur numerischen Integration sehr stark abweichende Ergebnisse liefert. Mit dem „Runge-Kutta“-Verfahren zur Integration des Signals und einer gewählten Schrittweite von 0.001 kann jedoch auch hier der korrekte Kurvenverlauf ermittelt werden.

5.1. Simulationsparameter

Folgende Parameter der Differenzialgleichung wurden im Vorfeld der Simulation messtechnisch bestimmt beziehungsweise im benannten Excel-Dokument berechnet: c und θ_A.
Hierbei ist zu beachten, dass hier der Fall für eine Stangeneinspannlänge von l=0,493 m und eine Zylinderbefestigung von k = l – H =0.425 m betrachtet wird.

Somit lassen sich aus den hergeleiteten Formeln folgende Parameter ermitteln:

Schwerpunkt: **s = 0.3776 m**

E-Modul: **E = 202716090369 N/m²**

FTM 2. O.: **I_a = 6.36172512E-11 m^4**

Gesamtmasse: **m = 2,107685286 kg**

MTM bezgl. A: **θ_A = 0.388983 kg m²**

Erdbeschleunigung: **g = 9.81 m/s²**

Dämpfungskonstante: **c = 0.18165755 kgm/s**

Die Dämpfungskonstante c wird hier aus der Messung der Spannung und einem anschließenden Koeffizientenvergleich der Differenzialgleichung, wie es in der Excel-Datei „Systemberechnung.xls“ implementiert wurde, ermittelt

Das Massenträgheitsmoment (MTM) wurde als via Koeffizientenvergleich den Messwerten entnommen (siehe vorheriges Kapitel und o.g. Excel-Datei).

Die Anfangsauslenkung x(t=0) beträgt 3 cm. Excel berechnet daraus den Verlauf der Folgeschwingung (definiert nach Kapitel 4.3) Gleichung (77) mit dem Startwert von:

$x(t=T_1)$=1.25 cm

5.2. Simulation in Boris

5.2.1. Modellierung

Bild 5.1 stellt das erstellt Modell, d.h. die abstrahierte DGL (48) in Boris dar

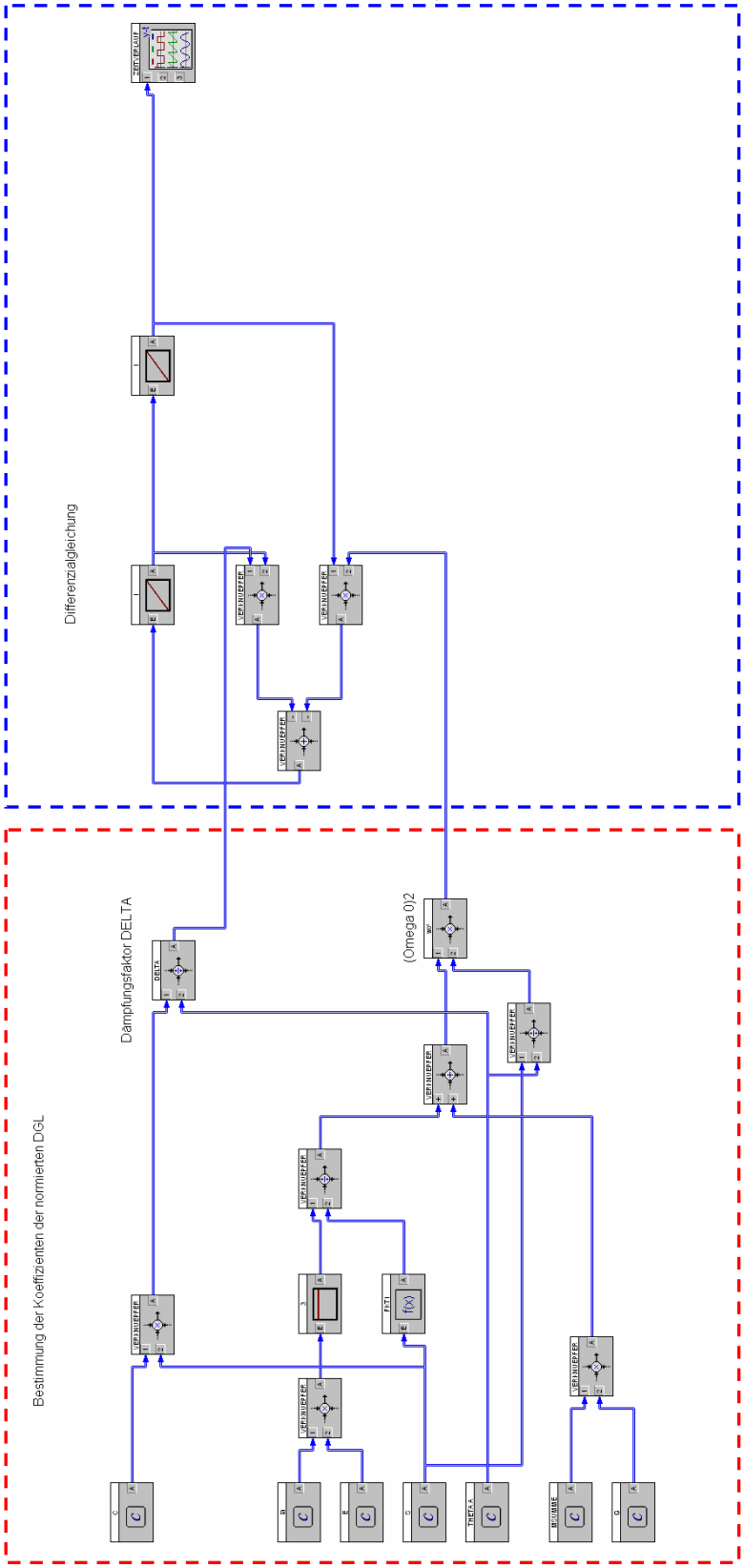


Bild 5.1 Modell in Boris

Der rot markierte Bereich stellt hier die Berechnung der Koeffizienten der normierten Differenzialgleichung dar. Die Eingangsgrößen sind hier:
c, l_a, E, s, θ_A, m_Σ, g, $x_{SP}(t=0)$.
Diese werden direkt der Excel-Tabelle („Systemberechnung.xls“) entnommen. So ist es möglich die Schwingung für das theoretisch ermittelte Massenträgheitsmoment θ_A zu ermitteln, als auch für das experimentell-reale. Der Dämpfungsfaktor c ist der Excel-Tabelle aus c_2 zu entnehmen, da nur der zweite Funktionsanteil gemäß Gleichung (76) simuliert wird.
Diese Größen werden entsprechend der hergeleiteten Gleichungen verknüpft und letztlich dem Teil der Berechnung der Differenzialgleichung zugeführt, der in Bild 5.1 dargestellt ist. Die Lösung Differenzialgleichung beruht hierbei gemäß der numerischen Integration auf einem iterativen Verfahren, wobei das Signal der zweiten Ableitung der Auslenkung zwei mal integriert wird. Die DGL sagt nun aus, das nach der ersten Integration erhaltene Signal mit dem Dämpfungsfaktor (δ) zu multiplizieren und das Signal nach der zweiten Integration mit dem Quadrat der Eigenkreisfrequenz der nicht gedämpften Schwingung (ω_0^2) zu multiplizieren. Diese beiden Ergebnisse werden addiert und negativ dem Eingang des ersten Integrierers zugeführt, da das so erzeugte Signal wiederum die zweite Ableitung der Auslenkung darstellen muss.
Zwecks Vollständigkeit sind in Bild 5.2 die Koeffizientenberechnung und Bild 6.3 die Lösung der DGL nochmals genauer dargestellt.

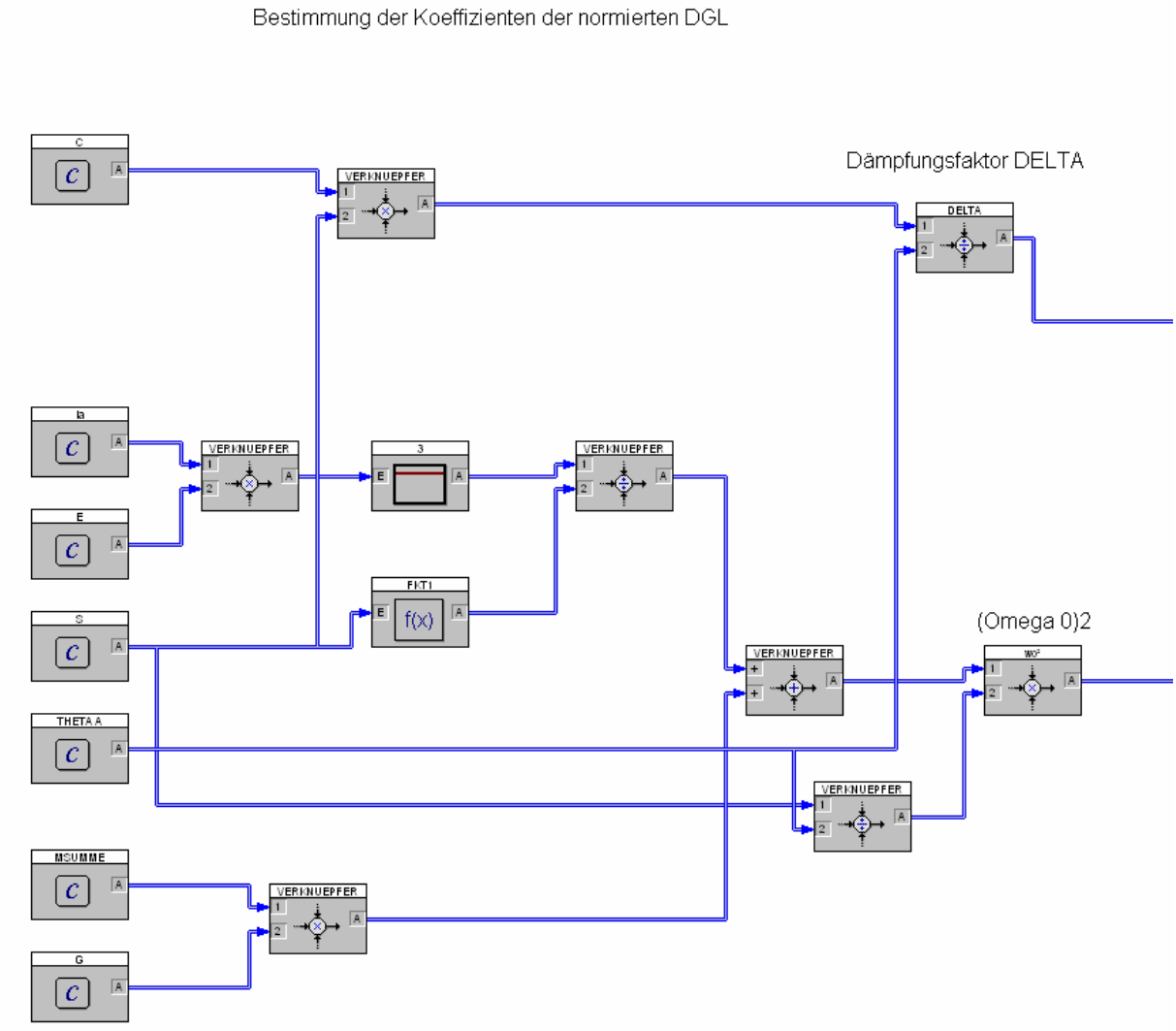


Bild 5.2: Boris: Koeffizientenberechnung der DGL

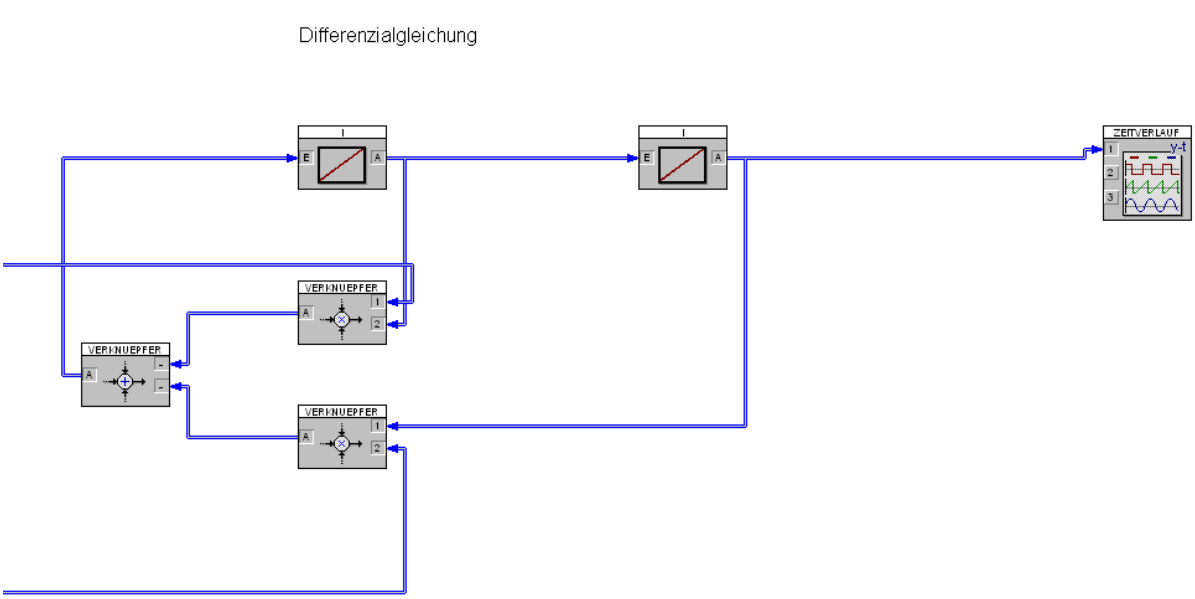


Bild 5.3: Boris: Lösung der Differenzialgleichung

Der Anfangswert $x_{SP}(t=0)$ der Auslenkung wird den zweiten Integrierer eingegeben. Hierbei ist der von Excel berechnete Anfangswert des zweiten Funktionsteils nach Gleichung (76) zu benutzen, da nur dieser hiermit simuliert werden kann.
Mit den Kapitel 5.1 definierten Simulationsparametern ergibt sich folgender Kurvenverlauf des Bilds 5.4:

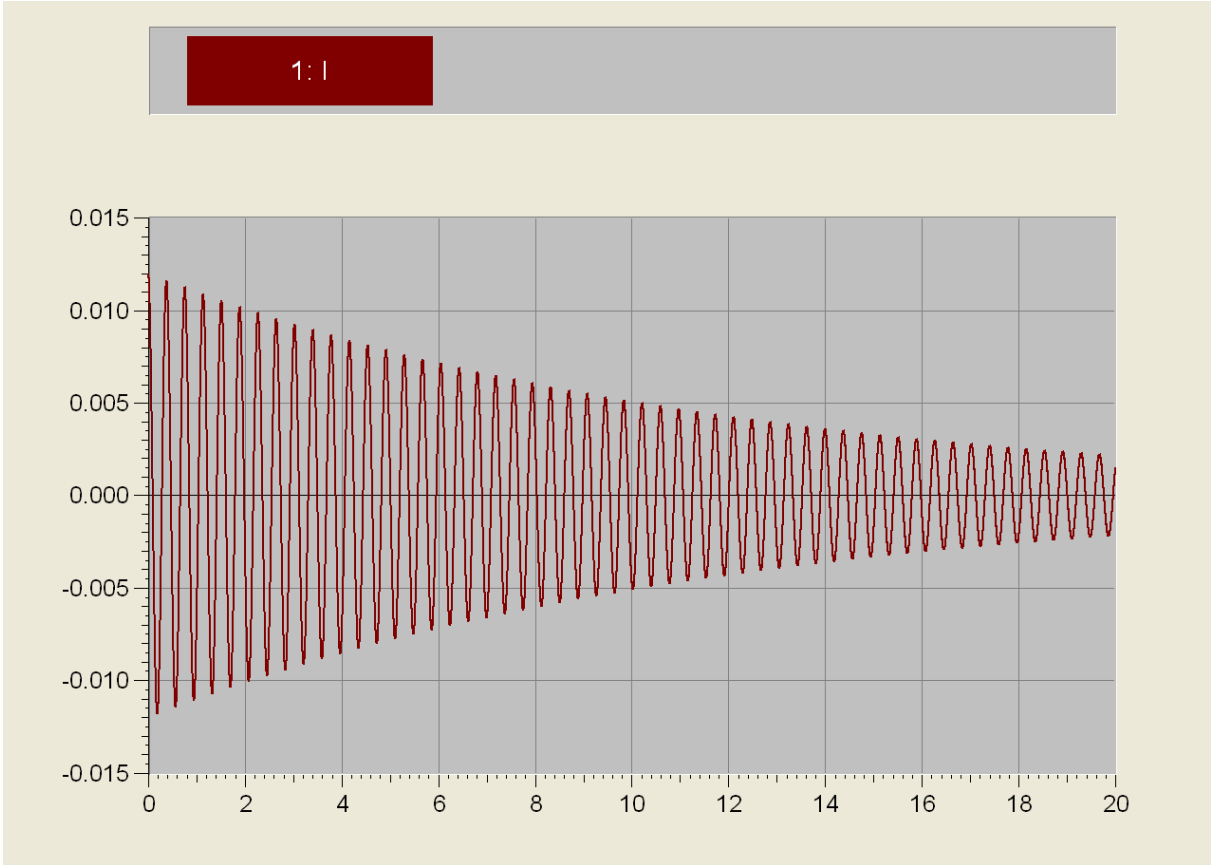


Bild 5.4: Schwingungssimulation in Boris nach Kapitel 5.1

Dieser Verlauf ist per definitionem identisch mit dem aus Mathematica (aus Kapitel 4.3), weil dieselben Parameter und die normierte Differenzialgleichung benutzt werden.
Ein Vergleich mit der Spannungsmessung aus Bild 5.1 bestätigt die gemeinsame Frequenz und den Startwert der Folgeschwingung für die Auslenkung. Für die Parametervariation im Nachhinein ist Boris jedoch wesentlich bequemer, weil Anwenderfreundlicher. Zudem wird die Differenzialgleichung hier anschaulich und plausibel dargestellt.
Die Ausgangsgröße x_{SP}, die hier in einer Senke zugeführt wird, kann nun auch einem DDE-Server-Block zugeführt werden, wodurch ein anderes Programm diese Daten direkt übertragen bekäme und somit auch eine Visualisierung der Pendelstange möglich wäre.

6. Pendelvisualisierung

Nachdem das mathematische Pendelmodell entwickelt und anhand von Messungen verifiziert wurde, wollten wir eine Stufe weiter gehen. Idee war nun, ein Programm zu entwickeln, welches das Pendel in graphischer Form animiert auf dem PC darstellt. So kann man sich einen viel besseren Eindruck von der Biegung der Pendelstange und der Physik des Pendels machen.
Die gesamte Zeichenfunktionalität inklusive Berechnung der Stangenbiegung erfolgt im Programm in der Funktion `DrawPenduleum`. Die wichtigen Teile dieser Funktion werden ab Kapitel 6.2 beschrieben.
Jede einzelne Zeile des Quellcodes zu kommentieren, würde den Rahmen dieses Dokuments sprengen, so dass wir uns auf die wichtigen Teile beschränken wollen. Der Quellcode des Programms wurde ausreichend kommentiert, um auch für fremde Personen nachvollziehbar zu sein.

6.1. Boris als Werteserver

In der ersten Stufe wird per DDE die von Boris berechnete Auslenkung übernommen. Hieraus wird nun mittels der in 2.4.2 beschriebenen Biegungsgleichung (Gl. (64)) die Stangenbiegung an ausgewählten Punkten berechnet und graphisch dargestellt.
Von Anfang an war klar, dass nicht für jedes Zielpixel ein Biegungswert berechnet werden kann. Immerhin soll das Pedel in Echtzeit dargestellt werden. Deshalb sollten nur eine handvoll Werte (wie sich später zeigte, sind zehn Werte ein guter Kompromiss zwischen Rechenzeit und Genauigkeit) errechnet werden und diese mittels einfacher und schneller linearer Interpolation zum Zielbild verbunden werden. Um die Geschwindigkeit noch weiter zu erhöhen, führt das Programm die lineare Interpolation nicht selbst aus, sondern zeichnet einfach Verbindungslinien zwischen den berechneten Stangenpunkten. Die in dem Windows-GDI (Graphical Device Interface) implementierten Zeichenfunktionen machen dies schnell und effektiv.
Von Anfang an eingeplant wurde eine Art Doublebuffering. Das Programm zeichnet also immer in einen unsichtbaren internen Buffer und kopiert das Bild anschließend auf den Bildschirm. Dadurch wird hässliches Flackern beim Bildaufbau vermieden (das Bild wird erst gelöscht und danach wird dann gezeichnet).

6.1.1. Boris DDE-Anbindung

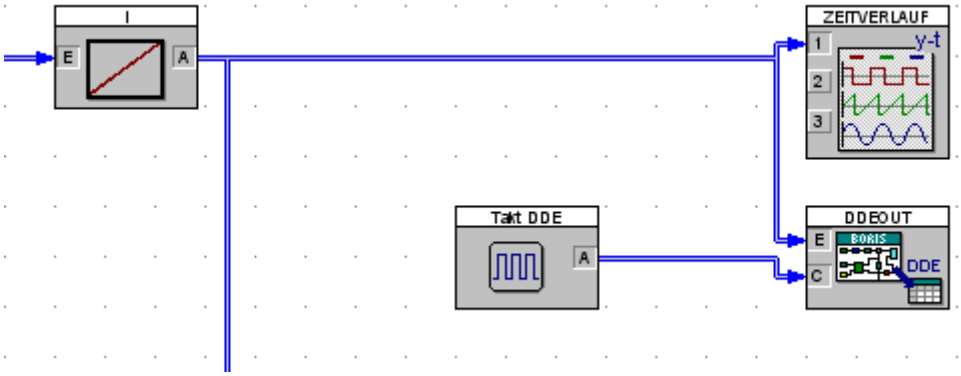


Bild 6.1: Boris DDE-Blöcke

In die bestehende Boris-Systemdatei wurde ein DDEOUT-Block eingefügt. Dieser hat zwei Eingänge. Ein Eingang nimmt den zu übertragenen Wert auf, der andere optionale Eingang ist der Trigger-Eingang. Hier kann man einen Takt anlegen, der in festen Abständen den Wert per DDE an die Zielapplikation überträgt. Ohne den Trigger würde der Wert mit jedem Rechenschritt übertragen werden, was die Zielapplikation überfordert. Der optimale Takt wurde experimentell ermittelt und entspricht einer Pulsweite 0,02 und einer Periodendauer 0,04.

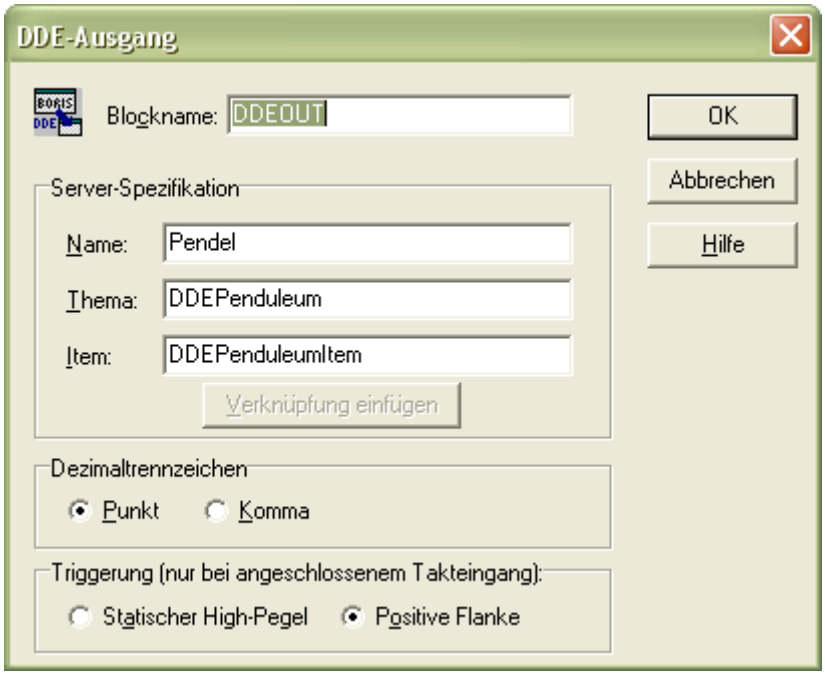


Bild 6.2: Boris DDE-Einstellungen

Für die DDE-Session werden ein Name, ein Thema und ein Item benötigt. Dies muss letztlich sowohl am DDE-Server als auch am DDE-Client gleich eingestellt sein.
In der C++-Applikation werden für die DDE-Kommunikation die beiden Controls `TDdeServerConv` und `TDdeServerItem` benutzt. Dem `TDdeServerItem` muss man als Namen den Itemnamen geben. Ebenso

stellt man unter den Eigenschaften noch den Themennamen ein (Eigenschaft `ServerConv`).
Der Name im DDE-Ausgang bei Boris muss mit dem Namen der Applikation übereinstimmen (in unserem Fall `Pendel`).
Nun wird bei jedem Wert, den Boris per DDE überträgt, das Ereignis `Change` des `TDdeServerItem`-Controls angesprochen. Hier kann dann der Wert geholt und weiterverarbeitet werden.

```
void __fastcall TfrmMain::DDEPenduleumItemChange(TObject *Sender)
// Dieses Ereignis erhält per DDE die Auslenkung im Schwerpunkt des Pendels von Boris
{
  char sTemp[32];
  int iFSAA;
  double dblScaleX, dblScaleY;

  // DDE-Text in Fließkommazahl wandeln (Einheit: Meter!)
  sscanf(DDEPenduleumItem->Text.c_str(), "%le", &dblAuslenkung);
  // Zahl für Anzeige vernünftig formatieren (keine Exponentialschreibweise!)
  sprintf(sTemp, "%+.06lf", dblAuslenkung);
  // Anzeigen...
  lblAuslenkung->Caption = sTemp;
  lblAuslenkung->Refresh();

  iFSAA = pow(2, rgFSAA->ItemIndex); // FSAA-Faktor berechnen
  // Skalierung berechnen
  dblScaleX = (608.52l / 300.0l) * double(imgPendel->Width);
  dblScaleY = (608.52l / 300.0l) * double(imgPendel->Height);
  // Pendel zeichnen
  DrawPenduleum(dblAuslenkung, imgPendel->Width, imgPendel->Height,
                iFSAA, dblScaleX, dblScaleY,
                atof(edtXFaktor->Text.c_str()), imgPendel->Picture->Bitmap);
  // Bitmap refreshen, damit es angezeigt wird
  imgPendel->Repaint();
}
```
Code 6.1: DDEPenduleumItemChange-Ereignis

Die hier erhaltene Auslenkung kann dann von der `DrawPenduleum`-Funktion verwendet werden, um das Pendel zu zeichnen.

6.1.2. Probleme beim Einsatz mit Boris

Wie sich im späteren Verlauf der Entwicklung herausstellte, gab es Probleme im Zusammenhang mit Boris. So verwendet Boris eine hohe Prozesspriorität, so dass unser Pendelprogramm nicht mehr genug Rechenzeit für die Darstellung erhalten hat. Speziell im Modus „Echtzeit“ (einzustellen im Menü Einstellungen→Parameter) von Boris schaltet sich Boris sogar auf die Priorität „system critical“, so dass kein anderes Programm auch nur einen Taktzyklus vom Scheduler abbekommt.
Die Lösung war, dass auch unser Programm sich „system critical“-Priorität geben muss. Damit war die Darstellung nun kein Problem mehr. Leider brachte dies zunächst Abstürze beim Beenden der Anwendung, die nach umfangreichem Debuggen gelöst werden konnten. Es ist wichtig, dass die Anwendung vor dem Schließen sich wieder eine normale Priorität gibt. Nur so konnten die Absturzprobleme behoben werden.

```
  // Prozeßpriorität erhöhen
  SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS);
```
Code 6.2: Prozeßpriorität erhöhen

6.2. Berechnung der Stangenbiegung

Wie sich nach ersten Tests herauskristallisierte, war der Rechenaufwand der Biegungsformel (vgl. Kap. 2.4.2, Gl. (63)) relativ hoch. Dies führte zu einer ruckeligen Darstellung des Pendels. Deshalb mussten Überlegungen angestellt werden, wie man die Rechengeschwindigkeit verbessern kann.
Eine Analyse der Formel zeigte, dass ein Term innerhalb eines Rechendurchlaufs immer konstant bleibt und außerdem noch in der Formel zweimal verwendet wird.
Die Strategie war also, möglichst viele Werte nur einmal am Anfang zu berechnen und nur die variablen Teile für jeden Stangenpunkt einzeln zu berechnen.

```
  // Vorberechnete Terme, die pro xsp-Wert konstant bleiben (Performance!)
  double fltTerm = ( (4.0l * Ia * s_pow2) / (3.0l * M_PI * r_pow4 * xsp) );
  double fltTerm_pow2 = fltTerm * fltTerm;
…
  // Stange berechnen
  icnt = 0;
  for(dblcnt = 0.0l; dblcnt < l; dblcnt += (l / 10.0l))
  {
    if(xsp == 0.0l)
      dblStangenWerte[icnt] = 0.0l;
    else
    {
      if(xsp >= 0)
      {
        dblStangenWerte[icnt] = fltTerm -
                                sqrt( fltTerm_pow2 - hoch4(dblcnt) );
      }
      else
      {
        dblStangenWerte[icnt] = fltTerm +
                                sqrt( fltTerm_pow2 - hoch4(dblcnt) );
      }
    }

    icnt++;
  }
```

Code 6.3: Berechnung der Pendelstange

Außerdem wurde eine kleine Funktion geschrieben, die n^4 berechnet, da anzunehmen ist, dass die in C++ integrierte pow-Funktion nicht besonders schnell ist. Die Funktion wurde als inline deklariert, damit möglichst keine langsamen Aufrufe über den Stack stattfinden müssen.

```
double __inline hoch4(double Wert)
// Eine hoffentlich schnelle Ersatzfunktion für pow(x, 4)
{
  double Temp = Wert * Wert;
  return Temp * Temp;
}
```

Code 6.4: Funktion für n^4

Erfolg dieser Optimierungen war eine wesentlich gesteigerte Rechengeschwindigkeit, welches der Bildrate sehr zu gute kommt.

6.2.1. Längenkontraktion der Stange

Es zeigte sich, dass die fehlende „Verkürzung“ der Stange beim Biegen Darstellungsfehler verursacht. Leider berücksichtigt unser mathematisches Modell nicht, dass die Stange durch die Biegung kürzer dargestellt werden muss. Dadurch ragte die Stange bei großen Winkeln unter dem Gewicht heraus. Dies ist besonders der Fall, wenn man die Darstellung in X-Richtung verzerrt darstellt, um die Darstellung der Biegung zu verstärken.
Um den Aufwand in Grenzen zu halten, wurde nun eine Näherungsformel hergeleitet, die bei den hier auftretenden Winkeln gute Resultate erzielt. Die Zeichenposition in y-Richtung wird hierbei mit dem Faktor $1-(\mathrm{iXFaktor})\cdot 0.012$ multipliziert, so dass die Stange in Abhängigkeit von der eingestellten X-Verzerrung verkürzt dargestellt wird. Der Faktor wurde experimentell ermittelt. Für die in unserem Modell vorkommenden Winkel erzielt diese Näherung sehr gute Resultate.

6.2.2. Zeichnen der Stange

Wie schon eingangs erwähnt wurde, werden zehn Stangenpositionen berechnet, die linear miteinander verbunden werden. Bei den kleinen Winkeln in unserem Modell reicht die Darstellungsqualität vollkommen aus.
Benutzt werden zum Zeichnen die MoveTo und LineTo-Methoden des TCanvas-Objekts (TCanvas ist die Zeichenfläche eines Bildes).

```
  // Stange zeichnen
  bmpBackbuffer->Canvas->Pen->Style = psSolid;
  bmpBackbuffer->Canvas->Pen->Color = clWhite;
  bmpBackbuffer->Canvas->Pen->Width = iFSAA * cStangenDicke * PdblScaleX;
  for(icnt = 0; icnt < 9; icnt++)
  {
    bmpBackbuffer->Canvas->MoveTo(iCenter + dblStangenWerte[icnt] * dblScaleX,
                                  icnt * (l / 10.0l) * dblScaleY * (1.0l - (iXFactor *
0.012l)));
    bmpBackbuffer->Canvas->LineTo(iCenter + dblStangenWerte[icnt + 1] * dblScaleX,
                                  (icnt + 1) * (l / 10.0l) * dblScaleY * (1.0l -
(iXFactor * 0.012l)));
  }
```

Code 6.5: Zeichnen der Pendelstange

Wie man sieht, ist der Code für das Zeichnen sehr einfach. Die Linie wird einfach vom Mittelpunkt des Bildes iCenter ausgehend um den berechneten Stangenwert verschoben und mit dem Skalierungsfaktor multiplizierten Wert gezeichnet.
Die Y-Position wird in festen Schritten hoch gezählt (Stangenlänge/10), mit dem Skalierungsfaktor multipliziert und noch mit der genäherten Längenkontraktion korrigiert

6.3. Zeichnen des Gewichts

Nun sollte natürlich auch das Gewicht korrekt gedreht dargestellt werden. Das Problem war, dass die Rechteck-Zeichenfunktion von Windows bzw. der VCL (Visual Component Library) keine gedrehten Rechtecke darstellen kann. Ein nicht korrekt gedrehtes Rechteck sieht aber ziemlich unrealistisch aus. Deshalb musste das Rechteck über die Polygon-Funktion gezeichnet werden. Hiermit können gefüllte Polygone auf einfache Art gezeichnet werden, indem man einfach eine Werteliste erstellt, welche die einzelnen Punkte enthält.
Anschließend wurde daher eine Formel modelliert, die die entsprechenden Punkte des gedrehten Rechtecks berechnet. Hierfür bot sich der Einsatz einer Koordinatentransformation an. Letztlich kann man hier annehmen, dass sich das Gewicht in einem verschobenen und gedrehten Koordinatensystem befindet.
In Code 6.6 ist die Funtkion für die Koordinatentransformation dargestellt.

```
void trans(double x, double y, double phi, double xx, double yy, double *xs, double
*ys)
// Koordinatentransformation des Punktes (xx,yy) in das um x und y verschobene und um
phi
// (im Bogenmaß) gedrehte System
{
  *xs = xx * cos(phi) - yy * sin(phi) + x;
  *ys = xx * sin(phi) + yy * cos(phi) + y;
  return;
}
```

Code 6.6: Koordinatentransformation

Die berechneten, transformierten Werte werden in die Zeigerparameter xs und ys geschrieben.
Die verwendete Formel für die Koordinatentransformation lässt sich dem Quellcode leicht entnehmen und lautet wie folgt:

$$\begin{aligned} x_s &= x_x \cdot \cos(\phi) - y_y \cdot \sin(\phi) + x \\ y_s &= x_x \cdot \sin(\phi) + y_y \cdot \cos(\phi) + y \end{aligned} \qquad (78)$$

Formel 6.1: Koordinatentransformation

Leider fehlt noch der Drehwinkel ϕ für unsere Transformation. Hierfür bietet sich der Winkel der letzten berechneten Stangenposition an.

$$\alpha = \frac{\arctan(x_{n-1} - x_n)}{l/10} \qquad (79),$$

wobei x die Werte der Stangenposition seien, l die Länge der Stange und n der Index der letzten berechneten Stangenposition.
Der Programmcode für die Winkelberechnung lässt sich Code 6.7 entnehmen.

```
  // Winkel im Endpunkt berechnen und anzeigen
  dblWinkel = atan((dblStangenWerte[icnt - 2] - dblStangenWerte[icnt - 1]) / (1 /
                   10.0l));
  dblWinkelGrad = (dblWinkel / M_PI) * 180.0l;
  sprintf(sTemp, "%+.02lf °", dblWinkelGrad);
  lblWinkel->Caption = sTemp;
  lblWinkel->Refresh();
  dblWinkel *= dblScaleX / dblScaleY;
```

Code 6.7: Winkelberechnung

Da der Winkel nun bekannt ist, ebenso die Funktion der Koordinatentransformation, kann nun das Gewicht gezeichnet werden. Wie schon erwähnt, wird das Gewicht als Polygon gezeichnet, da gedrehte Rechtecke nicht von der VCL unterstützt werden.

```
  // Gewicht zeichnen
  bmpBackbuffer->Canvas->Brush->Color = clWhite;
  x = iCenter; y = 0.4321961480311 * dblScaleY; phi = dblWinkel;
  // Punkt 1 transformieren ol
  xx = dblStangenWerte[icnt] * dblScaleX - (iFSAA * cGewichtGroesse * PdblScaleX);
  yy = iFSAA * -cGewichtGroesse * PdblScaleY;
  trans(x, y, phi, xx, yy, &xs1, &ys1);
  // Punkt 2 transformieren or
  xx = dblStangenWerte[icnt] * dblScaleX + (iFSAA * cGewichtGroesse * PdblScaleX);
  yy = iFSAA * -cGewichtGroesse * PdblScaleY;
  trans(x, y, phi, xx, yy, &xs2, &ys2);
  // Punkt 3 transformieren ur
  xx = dblStangenWerte[icnt] * dblScaleX + (iFSAA * cGewichtGroesse * PdblScaleX);
  yy = iFSAA * cGewichtGroesse * PdblScaleY;
  trans(x, y, phi, xx, yy, &xs3, &ys3);
  // Punkt 4 transformieren ul
  xx = dblStangenWerte[icnt] * dblScaleX - (iFSAA * cGewichtGroesse * PdblScaleX);
  yy = iFSAA * cGewichtGroesse * PdblScaleY;
  trans(x, y, phi, xx, yy, &xs4, &ys4);
  // Gewicht zeichnen
  GewichtPunkte[0] = Point(xs1, ys1);
  GewichtPunkte[1] = Point(xs2, ys2);
  GewichtPunkte[2] = Point(xs3, ys3);
  GewichtPunkte[3] = Point(xs4, ys4);
  GewichtPunkte[4] = Point(xs1, ys1);
  bmpBackbuffer->Canvas->Pen->Style = psClear;
  bmpBackbuffer->Canvas->Polygon(GewichtPunkte, 4);
```

Code 6.8: Zeichenfunktion für Gewicht

Wie man sieht, wird zuerst die Koordinatentransformation durchgeführt. Hier werden nun die vier Punkte des Rechtecks berechnet, und zwar aus den Stangenwerten (Stangenposition), dem Skalierungsfaktor und der gewünschten Größe des Rechtecks. Dann wird jeder Punkt einzeln transformiert und anschließend in das TPoint-Array `GewichtPunkte[]` geschrieben. Anschließend wird die Methode Polygon des TCanvas-Objektes mit dem Array und der (Anzahl der Punkte – 1) als Parameter aufgerufen. Zu beachten sei hierbei, dass wir fünf Punkte brauchen, um das Polygon schließen zu können (Punkt 4 wird zu Punkt 1 verbunden).

6.4. Aliasing-Problematik

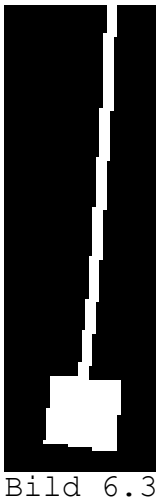


Bild 6.3

Nach den ersten Tests des Programms zeigte sich ein Darstellungsproblem, welches typisch für pixelorientierte Ausgabegeräte ist: Aliasing. Hierbei handelt es sich vereinfacht beschrieben um Verzerrungen und Treppenstufen, die durch fehlende Subpixelgenauigkeit entstehen. Bild 6.3 soll die Problematik erläutern.
Man kann deutlich die auftretenden Verzerrungen und Treppenstufen erkennen. Auch beim Rechteck macht sich die fehlende Subpixelgenauigkeit negativ bemerkbar.
Die Darstellungsqualität lässt also sehr zu wünschen übrig. Deshalb musste ein Algorithmus zum Vermeiden des Aliasings her. Prinzipiell gibt es mehrere Möglichkeiten, Aliasing zu vermeiden. Diese sollen im Folgenden mit ihren Vor- und Nachteilen dargestellt werden.

6.4.1. Tiefpassfilter

Eine Möglichkeit wäre der Einsatz eines Tiefpassfilters. Dieser würde die hohen Frequenzen im Bild wegfiltern, wodurch das ganze Bild unscharf und weich gezeichnet würde. Letztlich würden zwar Treppenstufen abgeschwächt werden, jedoch würde das eigentliche Aliasing trotzdem bestehen bleiben. Außerdem ist dieser Algorithmus aufwändig zu implementieren, da hierfür eine FFT-Funktion benutzt werden müsste. Auch der Rechenaufwand wäre einfach zu hoch, soll das Programm doch in Echtzeit abgearbeitet werden. Deshalb ist das Tiefpassfilter für unsere Zwecke ungeeignet.

6.4.2. Matrix-Operator

Eine weitere Möglichkeit wäre der Einsatz eines Matrixoperators. Dieser würde ein Zielpixel aus einem n*m-Pixelblock des Quellbildes berechnen. Hier kann man die Gewichtung im n*m-Pixelblock für jedes Pixel einzeln festlegen. Unter anderem ist so ein schneller und einfacher Unschärfefilter realisierbar.
Leider wird auch hier, genauso wie beim Tiefpassfilter, nur eine Unschärfe erzeugt, nicht jedoch das eigentliche Aliasing behoben.
Deshalb wäre auch dies keine optimale Lösung, trotz der gegenüber dem Tiefpassfilter viel höheren Geschwindigkeit.

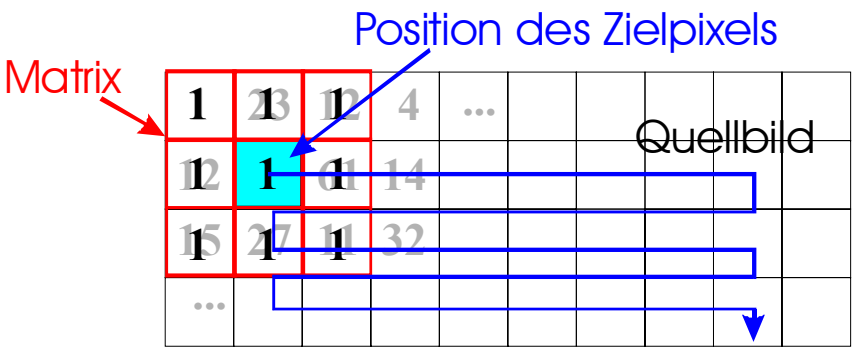


Bild 6.4: Zeichnung des Matrix-Operators

6.4.3. Full Scene Anti Aliasing (FSAA) mittels Multisampling

Eine ebenfalls mögliche und gebräuchliche Form des Anti-Aliasings ist das Multisampling. Hierfür wird das Bild mehrfach gerendert (bspw. viermal) und in jedem Bild werden die Objekte im Subpixelbereich unterschiedlich verschoben gerendert. Anschließend werden die Bilder zu einem Gesamtbild verschmolzen.
Dieser Algorithmus wird von einigen 3D-Grafikkarten eingesetzt, ist sehr effektiv gegen Aliasing und hat eine annehmbare Geschwindigkeit, sodass er auch wegen des relativ geringen Implementierungsaufwandes interessant für das Visualisierungsprogramm ist.

6.4.4. Full Scene Anti Aliasing (FSAA) mittels Supersampling

Das letztlich in diesem Programm eingesetzte Anti-Aliasing Verfahren ist das so genannte Supersampling. Hier wird das Bild intern einfach in n*n-facher Auflösung gerendert und anschließend auf die ursprüngliche Größe bilinear herunterskaliert. Auch dieses Verfahren wird von vielen 3D-Grafikkarten eingesetzt, da es sehr leicht zu implementieren ist, sehr effektiv gegen Aliasing wirkt und außerdem eine akzeptable Geschwindigkeit erzielt.
Die Einfachheit der Implementierung rührt daher, dass man einfach nur ein größeres Bild rendert. Der Aufwand hierfür geht fast gegen Null, vor allem, wenn man ohnehin schon ein flexibles internes Koordinatensystem hat, welches unabhängig vom Bitmap ist.
Nur das Herunterskalieren muss extra implementiert werden. Hierfür darf man aber nicht einfach nur jedes zweiten Pixel und jede zweite Zeile in das Zielbild übernehmen, da dies den Algorithmus unwirksam machen würde. Vielmehr muss man unbedingt bilinear herunterskalieren. Hierfür nimmt man einen n*n-Pixelblock vom Quellbild, bildet den arithmetischen Mittelwert und schreibt diesen in ein Zielpixel hinein.
Bild 6.5 soll dieses erläutern.

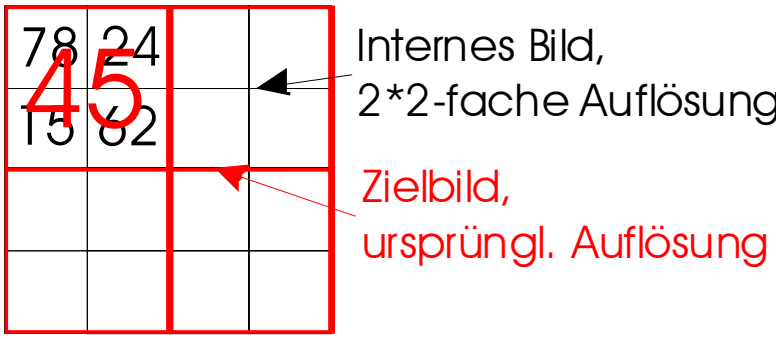


Bild 6.5: Bilineares Herunterskalieren 2*2

Es folgt der resultierende Programmcode für den Fall des 2-fach FSAA (Code 6.9):

```
  if(iFSAA == 2)
  {
    // Wenn 2xFull Scene Anti-Aliasing gewählt, dann das doppelt so groß gerenderte
    // Bitmap auf ein Viertel der Fläche mit hoher Qualität herunterskalieren
    for(iy = 0; iy < iHeight; iy+=2)
    {
      // Zeilenpuffer setzen
      src = (unsigned char *)bmpBackbuffer->ScanLine[iy];
      src2 = (unsigned char *)bmpBackbuffer->ScanLine[iy + 1];
      dest = (unsigned char *)Bitmap->ScanLine[iy >> 1];

      for(ix = 0; ix < iWidth; ix+=2)
      {
        dest[ix >> 1] = (src[ix] + src[ix + 1] + src2[ix] + src2[ix + 1]) >> 2;
      }
    }
  }
```

Code 6.9: Bilineares Herunterskalieren 2-fach FSAA

Die feste Skalierung auf ein Viertel der Fläche geht recht schnell, da man statt zeitraubender Divisionen sehr schnelle Bitschiebeoperationen benutzen kann. Man beachte: eine Bitschiebeoperation kann gewöhnlich innerhalb eines CPU-Taktes ausgeführt werden, eine Division braucht in den meisten Fällen 11 Takte.
Übliche FSAA-Stufen sind bei diesem Verfahren 2-fach und 4-fach, was bedeutet, dass intern in 2*2-facher bzw. in 4*4-facher Auflösung gezeichnet wird.
Der Rechenaufwand steigt mit der Erhöhung der FSAA-Stufe also quadratisch an, da bei doppelter FSAA-Stufe schon die vierfache Pixelzahl berechnet werden muss. Trotzdem ist dieses Verfahren noch vergleichsweise schnell.
Die Wirksamkeit soll anhand der weiter unten stehenden Bilder eindrucksvoll demonstriert werden. Links ist das Pendel ohne FSAA zu sehen, in der Mitte mit 2-fach FSAA, rechts mit 4-fach FSAA. Schon das 2-fach FSAA verbessert die Bildqualität deutlich. Und ein heutiger PC mit mehr als einem Gigahertz Taktfrequenz kann bei einer Auflösung von 600*600 Pixel und 2-fach FSAA eine flüssige Darstellung erzielen.

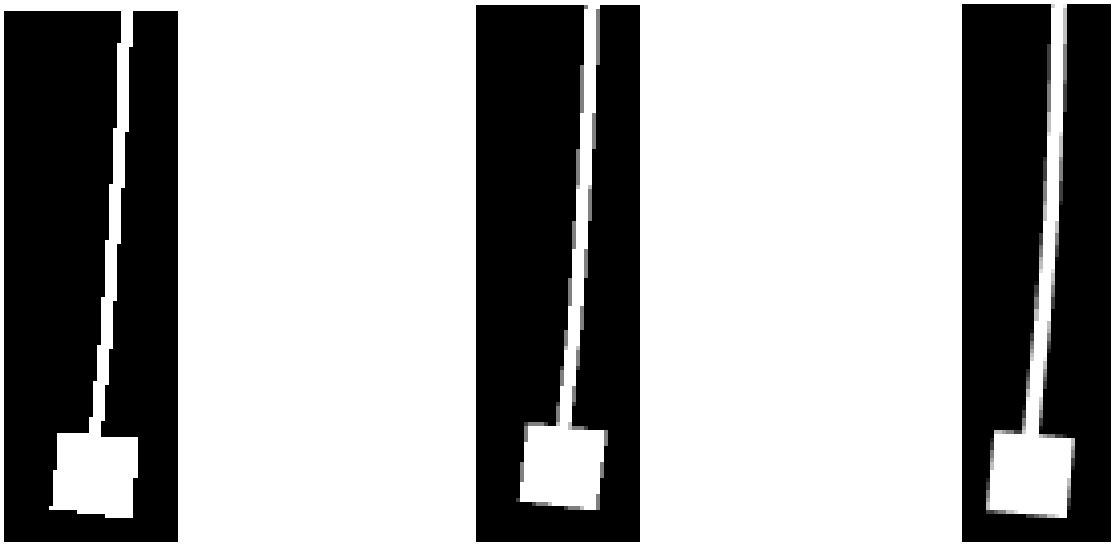


Bild 6.6: FSAA (Links: ohne, Mitte: 2x, Rechts: 4x)

6.5. Autarker Simulationsmodus

Wo nun das Programm in Verbindung mit Boris einwandfrei funktioniert, stellte sich die Frage, wie man das Programm noch weiter verbessern könnte. Da es sinnvoll erschien, das Pendel auch ohne die Blockorientierte Simulation 'Boris' simulieren zu können, war es nun Ziel, das mathematische Modell direkt in das C++-Programm zu implementieren. So würde das Programm optional auch ohne Hilfe von Boris das Pendel simulieren können.
Grundsätzlich soll im autarken Simulationsmodus immer in Echtzeit simuliert werden. Dies stellt aber auch kein größeres Problem dar: Es wird bei Simulationsstart einfach der Wert des Performance-Counters (ein sehr schneller CPU-interner Zähler für sehr genaue Zeitmessungen) gespeichert und dann kann simpel vor jeder Berechnung die Zeit aus dem aktuellen Zählerwert und der Frequenz des Performance-Counters berechnet werden. Somit kann immer das reale t in die Gleichung eingesetzt werden. Die Simulationsgeschwindigkeit ist also unabhängig von der Rechnergeschwindigkeit. Die Animation wird einfach flüssiger oder ruckeliger dargestellt, je nach Anti-Aliasing-Parameter, Fenstergröße und Rechnergeschwindigkeit.
Außerdem ermöglicht die autarke Simulation eine Fallunterscheidung bei der Berechnung, so dass nun der Dämpfungsfall (siehe Kap. 4.3, Gl. (76)) am Anfang des Pendelvorgangs ebenfalls korrekt simuliert werden kann. Unser Modell in Boris kann dies leider nicht und fängt erst ab Zeitpunkt T1 an.

6.5.1. Mathematisches Pendelmodell

Nun musste das das Pendelmodell implementiert werden. Interessant ist hierbei, dass die Schwingung des Pendels zwei zusammengesetzten Kosinusfunktionen entspricht. Im Fall t<=T1 direkt nach dem Anstoßen des Pendels haben wir eine sehr große Dämpfung, wobei T1 in unserem Modell dem Wert 0,52 sec. entspricht. Im Fall t>T1 hingegen ist die Dämpfung sehr viel geringer. Außerdem ist die Frequenzänderung zu beachten, die zwischen diesen beiden Fällen auftritt.

$$x_s = r_{anfang} \cdot e^{-\delta_1 \cdot t} \cdot \cos\left(2 \cdot \pi / T_{1\_angepasst} \cdot t \right) \quad \text{(80), Fall t<=T1}$$

$$x_s = r_{danach} \cdot e^{-\delta_2 \cdot t} \cdot \cos\left(2 \cdot \pi / T_2 \cdot t \right) \quad \text{(81), Fall t>T2}$$

Dieses sehr einfache Modell hat den Vorteil, dass es ohne aufwändige Differentialgleichungen sehr einfach in C++ zu implementieren ist. Außerdem ist der Rechenaufwand sehr gering, so dass das Programm problemlos in Echtzeit mit ausreichender Bildrate ablaufen kann.
Die genaue Herleitung ist in Kap. 4.3, Gl. (76) beschrieben.

6.5.2. Sättigungsfunktion

Wie sich herausstellte, gab es hässliche Sprünge in der Pendelanimation aufgrund des abrupten Wechsels von T1 zu T2. Deshalb musste eine Sättigungsfunktion implementiert werden, die $T_{1\_angepasst}$ von t=0 bis t<=T1 langsam an den Wert von T2 annähert.

$T_{1\_angepasst} = T_2 - T_2 \cdot e^{-\Xi \cdot t}$ (82), Sättigungsfunktion

Ξ sei hierbei der Annäherungskoeffizient, der frei eingestellt werden kann. Experimentell hat sich eine Wahl zwischen 10 und 20 1/sec als sinnvoll erwiesen.

Mit Hilfe dieser Sättigungsfunktion konnte der Frequenzübergang zwischen den beiden Fällen gut angeglichen werden.

6.5.3. Implementierung

Wie Code 6.10 zeigt, konnte somit das mathematische Pendelmodell implementiert werden.

```
  QueryPerformanceFrequency(&freq); // Zur exakten Messung der Zeit benutzt
  QueryPerformanceCounter(&TickAlt);
  Screen->Cursor = crHourGlass;
  do
  {
    QueryPerformanceCounter(&TickNeu);
    t = double(TickNeu.QuadPart - TickAlt.QuadPart) / freq.QuadPart;
    // den Wert für delta1 hier einstellbar machen  t1_annährungskoeffizient?
    T1_angepasst = T2 - T2 * exp(-atof(edtT1_Annaeher->Text.c_str()) * t);
    if(t <= T1)
      dblAuslenkung = r_anfang * exp(-delta1 * t) * cos((2 * M_PI / T1_angepasst) * t);
    else
      dblAuslenkung = r_danach * exp(-delta2 * t) * cos((2 * M_PI / T2) * t);

    // Zahl für Anzeige vernünftig formatieren (keine Exponentialschreibweise!)
    sprintf(sTemp, "%+.06lf", dblAuslenkung);
    // Anzeigen...
    lblAuslenkung->Caption = sTemp;
    lblAuslenkung->Refresh();

    iFSAA = pow(2, rgFSAA->ItemIndex); // FSAA-Faktor berechnen
    // Skalierung berechnen
    dblScaleX = (608.52l / 300.0l) * double(imgPendel->Width);
    dblScaleY = (608.52l / 300.0l) * double(imgPendel->Height);
    // Pendel zeichnen
    DrawPenduleum(dblAuslenkung, imgPendel->Width, imgPendel->Height,
                  iFSAA, dblScaleX, dblScaleY,
                  atof(edtXFaktor->Text.c_str()), imgPendel->Picture->Bitmap);
    // Bitmap refreshen, damit es angezeigt wird
    imgPendel->Repaint();
  } while(t <= 1.0l);
```

Code 6.10: Mathematisches Pendelmodell

Wie man sieht, ist die Implementierung recht einfach, da nur die entsprechenden Formeln programmiert werden mussten, um die Auslenkung zu ermitteln. Der schon fertigen Funktion `DrawPenduleum` wird dieser Wert anschließend übergeben, der Rest funktioniert analog zum DDE-Betrieb mit Boris.

6.6. Ergebnis

Ergebnis ist nun ein Programm, welches sowohl im Zusammenhang mit Boris per DDE als auch autark ein simuliertes Stangenpendel mit Biegung graphisch simulieren kann. Für die Darstellung kann auch ein einstellbares Anti-Aliasing benutzt werden, um die Qualität erheblich zu verbessern.
Die Oberfläche des Programms wurde absichtlich schlank gehalten, da das Programm im Zusammenhang mit Boris immer im Vordergrund dargestellt werden muss und somit nicht unnötig viel Platz auf dem Bildschirm verbrauchen darf.
Die Fenstergröße des Programms ist variabel, wobei die Größe in einem Bereich skaliert werden kann, wo das das Bild eine Auflösung zwischen 300*300 und 600*600 Pixel hat.
Die Parameter für die mathematischen Modelle sind in einem extra Konfigurationsdialog (über Button `Konfig.` Erreichbar) veränderbar. Jedoch wird die Benutzung der Standardwerte empfohlen, da beliebiges Ändern der Werte zu unvorhersehbaren Ergebnissen führen kann (bspw. fehlerhafte Darstellung).

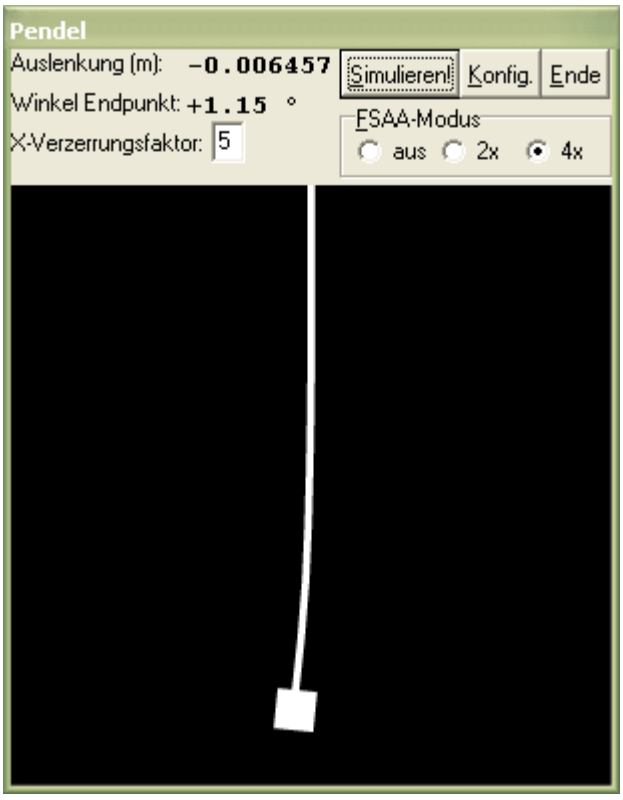


Bild 6.7: Das Programm

Im Hauptformular kann der FSAA-Modus gewählt werden. Außerdem kann man zur Verbesserung der Darstellung den X-Vezerrungsfaktor einstellen. Angezeigt werden während der Simulation die Auslenkung und der Winkel im Endpunkt.
Der Button `Simulieren!` lässt die autarke Simulation einmal in Echtzeit durchlaufen (Dauer in Konfigurationsdialog einstellbar, s.u.). Eine laufende Simulation kann jederzeit mittels Druck auf die `ESC`-Taste abgebrochen werden.

Falls der Einsatz mit Boris gewünscht ist, so muss nichts weiter unternommen werden. Es muss einfach in Boris die Simulation gestartet werden. Das Pendelprogramm wird automatisch die Werte von Boris erhalten und das Pendel darstellen.
Vom Hauptfenster aus ist auch der Konfigurationsdialog aufrufbar. Hier können die Parameter des Pendelmodells und die Simulationsdauer eingestellt werden.
Im oberen Bereich findet man die Parameter, die für die Berechnung der Stangenbiegung notwendig sind.
Im unteren Bereich sind die Parameter für das mathematische Pendelmodell zur Berechnung der Auslenkung. Außerdem ist hier auch die Simulationsdauer einstellbar.

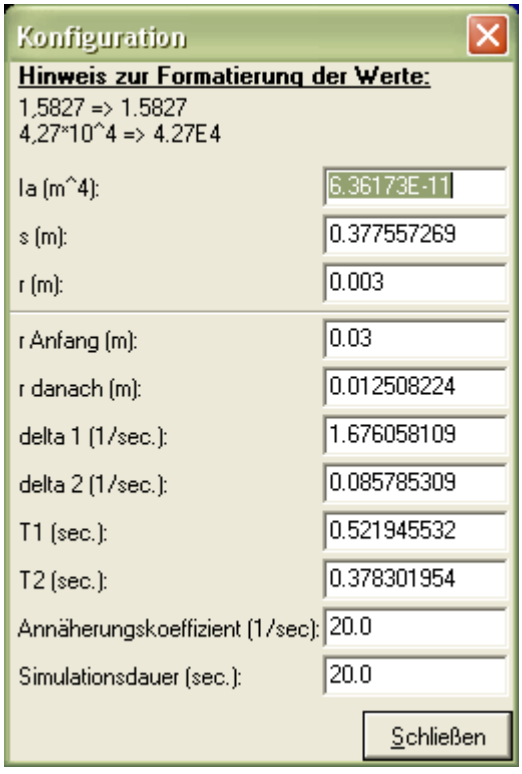


Bild 6.8: Der Konfigurationsdialog

6.6.1. Performance

Das Programm läuft in der Standardauflösung von 300*300 Pixel bei 4xFSAA auf einem Athlon 1,4 GHz mit einer GeForce 3-basierten Grafikkarte ruckelfrei. Wenn das Fenster bei dieser Hardware auf 600*600 skaliert wird, so ist ein leichtes Ruckeln bei 4xFSAA festzustellen, jedoch ist die Anzahl der Animationsphasen immer noch ausreichend für Anschauungszwecke.
Bei 2xFSAA ist das Bild in jeder Auflösung ruckelfrei.
Falls die Darstellungsgeschwindigkeit nicht ausreichen sollte, so sollte probehalber die Farbtiefe des Desktops von 24/32 Bit auf 16 Bit zurückgestellt werden. Speziell Grafikchips in Notebooks sind im 24/32 Bit-Modus meistens sehr langsam.
Ansonsten bleibt nur, die FSAA-Stufe herunterzustellen oder das Fenster nicht zu vergrößern.
Aber ab PCs der 600 MHz-Klasse sollte das Programm brauchbare Ergebnisse liefern.

7. Abschließende Bemerkungen

Mittels der mechanischen Grundprinzipien nach Newton, d'Alembert und Hooke ließ sich zunächst ein mathematisches Modell mit hohem Abstraktionsgrad für das Pendel der Firma Phywe ermitteln. Ausgehend von der Lösung einer Differentialgleichung für die Auslenkung im Schwerpunkt wurde die Krümmung der Pendelstange genauer betrachtet und dafür eine Funktion zur Beschreibung der Auslenkung eines beliebigen Punktes auf derselben hergeleitet.
Da ein Modell für die analytische Berechnung der Dämpfung fehlt, wurden insgesamt 14 Messungen ausgewertet.
Bei genauerer Betrachtung der Messungen stellt sich heraus, dass die Dämpfung nicht konstant ist. In der ersten Periode verliert die Schwingung ein hohes Maß Bewegungsenergie. Aufgrund der höheren Frequenz der auf diese Einschwingperiode folgenden Schwingung wird angenommen, dass es sich bei der Modellbildung nur um diese Folgeschwingung handeln kann. Rein mathematisch wird daher die Pendelschwingung bestehend aus zwei zusammengesetzten Schwingungsfunktionen beschrieben.
Die Dämpfungsfaktoren sind zudem abhängig von der Schwerpunktlage, weshalb für jede der Teilfunktionen eine Funktion Dämpfungskoeffizient in Abhängigkeit zur Schwerpunktlage aus Messwerten über Interpolation ermittelt wurde.
Es folgt darauf der Vergleich der Koeffizienten der Differenzialgleichung und des zweiten Funktionsteils basierend auf den Messwerten. Aufgrund des Unterschieds der Kreisfrequenz wird festgestellt, dass das tatsächlich wirkende Massenträgheitsmoment höher ist, als das theoretisch ermittelte. Dies wird damit begründet, dass der Metallrahmen, der eine zusätzliche Masse im Pendelsystem darstellt, nicht in der Modellbildung berücksichtigt wurde. Die Masse des Rahmens ist recht hoch und dämpft daher hohe Schwingungen, weshalb die Differenz zwischen theoretischem Massenträgheitsmoment und messtechnisch ermitteltem mit abnehmender Schwerpunktlage (von der Einspannung betrachtet) abnimmt, da auch die Kreisfrequenz damit steigt.
Aufgrund der Berechnungsmöglichkeiten des Excel-Dokuments „systemberechnung.xls" lassen sich die Fehler zwischen theoretischem und messtechnischem Modell einsehen, wobei jedoch zu beachten ist, dass durch die Interpolation der Dämpfungswerte auch ein systematischer Fehler in die Ergebnisse eingeht.
Insgesamt lässt sich beurteilen, dass das theoretische Modell bis zu Schwerpunktlagen von s = 28 cm sehr gut benutzbar ist, da der relative Fehler dort nur bei 9% liegt, was außerhalb der Messbarkeit Periodenzeiten liegt.
Zudem wäre es mit Excel kein Problem, die realen Werte aus den interpolierten Messdaten mit noch geringerer Abweichung zu entnehmen, nachdem die Interpolationsfunktion nach Newton von Hand implementiert würde. Dies ist im Rahmen des Projektes noch durchzuführen.
Die Entwicklung eines Visualisierungsprogramms hat die Überprüfung des theoretischen Modells mit der Praxis sehr erleichtert. So kann man visuell das simulierte Pendel schwingen sehen und kann die Bewegungen mit dem realen Pendel vergleichen.
Sehr nützlich ist bei diesem Programm die Möglichkeit, das Programm zusammen mit Boris zu verwenden. Dies ist bei der Modellentwicklung,

Modellbearbeitung und letztlich auch bei der Parametervariation sehr hilfreich. So kann man die Auswirkungen von Änderungen am Modell sehr eindrucksvoll in der Visualisierung betrachten, weil die Animation des sich biegenden Pendels viel anschaulicher ist, als nur der bloße Auslenkungsgraph.
Die Implementierung des mathematischen Modells direkt in die Visualisierung ermöglicht den Einsatz des Programms auch ohne Besitz der Software WinFACT. Außerdem funktioniert der Echtzeitbetrieb zusammen mit Boris nicht immer zufrieden stellend. Die integrierte Berechnung hingegen funktioniert immer stabil und zuverlässig in Echtzeit, dürfte außerdem eine etwas höhere Rechengeschwindigkeit haben als das Systemmodell in Boris.
Dank des Einsatzes eines Full Scene Anti Aliasing-Algorithmus erzielt die Visualisierung eine hervorragend Darstellungsqualität ohne dass nervende Aliasing-Effekte die Beurteilbarkeit beeinträchtigen würden.
Diese zwei Sätze ersetzen durch:
Abschließend ist die Visualisierung eine wichtige Erweiterung und Implementierung des mathematischen Modells, da man ansonsten auf den wenig anschaulichen Graphen von Boris angewiesen wäre, der zwar zur Spannung der DMS proportional ist, jedoch die Pendelbewegung nur gering zur Schau stellt.
Außerdem gilt zu beachten, dass ohne die Visualisierung mit ihrer internen Berechnung Lizenzierungsprobleme bei der Weitergabe des Arbeitsergebnisse auftreten würden. Letztlich müsste die Person, welche die Simulation durchführen will, immer eine Lizenz von WinFACT besitzen. Dank des Visualisierungsprogramms kann nun eine Simulation auch ohne WinFACT durchgeführt werden, wenn das integrierte Modell benutzt wird. Änderungen des Modells sind jedoch nur eingeschränkt möglich (nur Parametervariation).

8. Anlage

Excel-Messdaten „statische Spannung“:	‚Messreihen_26022002.xls’
Excel-Systemberechnung:	‚systemberechnung.xls’
Mathematica4.1-Notebook:	‚lsg_dgl_var_par1.nb’
Boris-Simulation:	‚pendel.bsy’

8.1. Quellcode der Pendelvisualisierung (Borland C++Builder 5.0 Projekt)

8.1.1. Modul `main.cpp`

```
/*************************************************************************
 * Pendelvisualisierung Version 1.1.0                                    *
 * Zeigt in graphischer Form ein simuliertes Stabpendel an               *
 * Die Lösung der DGL erfolt mittels Boris und wird per DDE an die       *
 * Visualisierung übermittelt oder wird selbst berechnet                 *
 *                                                                       *
 * Autor: Danny Schulz                                                   *
 * Versionshistorie: 1.0.0 - Erstversion                                 *
 *                   1.1.0 - Implementierung eines FSAA-Algorithmus      *
 *                           Einbau einer cos-Funktion zur Simulation der*
 *                            Auslenkung ohne Boris                      *
 *************************************************************************/
//---------------------------------------------------------------------------

#include <vcl.h>
#include <stdio.h>
#include <math.h>
#pragma hdrstop

#include "Main.h"
#include "Config.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmMain *frmMain;
//---------------------------------------------------------------------------
__fastcall TfrmMain::TfrmMain(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::SimPenduleum(void)
{
  char sTemp[32];
  int iFSAA;
  double dblScaleX, dblScaleY;

  double dblAuslenkung;
  LARGE_INTEGER TickAlt, TickNeu, freq;
  double t;
  double dblSimDauer = atof(frmConfig->edtSimDauer->Text.c_str());
  double T1_angepasst;
  const double r_anfang = atof(frmConfig->edtr_Anfang->Text.c_str());
  const double r_danach = atof(frmConfig->edtr_danach->Text.c_str());
  const double delta1 = atof(frmConfig->edtdelta1->Text.c_str());
  const double delta2 = atof(frmConfig->edtdelta2->Text.c_str());
  const double T1 = atof(frmConfig->edtT1->Text.c_str());
  const double T2 = atof(frmConfig->edtT2->Text.c_str());
  const double Xi = atof(frmConfig->edtT1_Annaeher->Text.c_str());

  // Prozeßpriorität absenken
  SetPriorityClass(GetCurrentProcess(), HIGH_PRIORITY_CLASS);

  QueryPerformanceFrequency(&freq); // Zur exakten Messung der Zeit benutzt
  QueryPerformanceCounter(&TickAlt);
  Screen->Cursor = crHourGlass;
  do
  {
    QueryPerformanceCounter(&TickNeu);
    t = double(TickNeu.QuadPart - TickAlt.QuadPart) / freq.QuadPart;
    // den Wert für delta1 hier einstellbar machen  t1_annährungskoeffizient?
    T1_angepasst = T2 - T2 * exp(-Xi * t);
    if(t <= T1)
      dblAuslenkung = r_anfang * exp(-delta1 * t) * cos((2 * M_PI / T1_angepasst) * t);
    else
      dblAuslenkung = r_danach * exp(-delta2 * t) * cos((2 * M_PI / T2) * t);

    // Zahl für Anzeige vernünftig formatieren (keine Exponentialschreibweise!)
    sprintf(sTemp, "%+.06lf", dblAuslenkung);
    // Anzeigen...
    lblAuslenkung->Caption = sTemp;
    lblAuslenkung->Refresh();
```

```
    iFSAA = pow(2, rgFSAA->ItemIndex); // FSAA-Faktor berechnen
    // Skalierung berechnen
    dblScaleX = (608.52l / 300.0l) * double(imgPendel->Width);
    dblScaleY = (608.52l / 300.0l) * double(imgPendel->Height);
    // Pendel zeichnen
    DrawPenduleum(dblAuslenkung, imgPendel->Width, imgPendel->Height,
                  iFSAA, dblScaleX, dblScaleY,
                  atof(edtXFaktor->Text.c_str()), imgPendel->Picture->Bitmap);
    // Bitmap refreshen, damit es angezeigt wird
    imgPendel->Repaint();
    if(int(t * 10) % 10 == 0)
      Application->ProcessMessages();
  } while(t <= dblSimDauer && !bBreak);
  bBreak = false;

  // Prozeßpriorität wieder erhöhen
  SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS);
  Screen->Cursor = crDefault;
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::DDEPenduleumItemChange(TObject *Sender)
// Dieses Ereignis erhält per DDE die Auslenkung im Schwerpunkt des Pendels von Boris
{
  char sTemp[32];
  int iFSAA;
  double dblScaleX, dblScaleY;

  // DDE-Text in Fließkommazahl wandeln (Einheit: Meter!)
  sscanf(DDEPenduleumItem->Text.c_str(), "%le", &dblAuslenkung);
  // Zahl für Anzeige vernünftig formatieren (keine Exponentialschreibweise!)
  sprintf(sTemp, "%+.06lf", dblAuslenkung);
  // Anzeigen...
  lblAuslenkung->Caption = sTemp;
  lblAuslenkung->Refresh();

  iFSAA = pow(2, rgFSAA->ItemIndex); // FSAA-Faktor berechnen
  // Skalierung berechnen
  dblScaleX = (608.52l / 300.0l) * double(imgPendel->Width);
  dblScaleY = (608.52l / 300.0l) * double(imgPendel->Height);
  // Pendel zeichnen
  DrawPenduleum(dblAuslenkung, imgPendel->Width, imgPendel->Height,
                iFSAA, dblScaleX, dblScaleY,
                atof(edtXFaktor->Text.c_str()), imgPendel->Picture->Bitmap);
  // Bitmap refreshen, damit es angezeigt wird
  imgPendel->Repaint();
}
//---------------------------------------------------------------------------
double __inline hoch4(double Wert)
// Eine hoffentlich schnelle Ersatzfunktion für pow(x, 4)
{
  double Temp = Wert * Wert;
  return Temp * Temp;
}
//---------------------------------------------------------------------------
void trans(double x, double y, double phi, double xx, double yy, double *xs, double
*ys)
// Koordinatentransformation des Punktes (xx,yy) in das um x und y verschobene und um
phi
// (im Bogenmaß) gedrehte System
{
  *xs = xx * cos(phi) - yy * sin(phi) + x;
  *ys = xx * sin(phi) + yy * cos(phi) + y;
  return;
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::DrawPenduleum(double dblAuslenkung, int iHeight, int iWidth,
                                        int    iFSAA,    double    PdblScaleX,    double
PdblScaleY,
                                        int iXFactor, Graphics::TBitmap *Bitmap)
// Diese Funktion zeichnet das Pendel für einen bestimmten Wert
// Parameter: dblAuslenkung - Pendelauslenkung in Meter
//            iHeight - Höhe des Zeichenfensters in Pixel
//            iWidth - Höhe des Zeichenfensters in Pixel
//            iFSAA - FSAA-Modus: 1=kein FSAA, 2=2-fach FSAA, 4=4-fach FSAA...
//                        PdblScaleX,  PdblScaleY:  Skalierung  Meter-Koordinaten  nach
Pixelkoordinaten
//                  iXFactor: Verzerrungsfaktor  für  X-Richtung  (um  Biegung  verstärkt
anzuzeigen)
//            Bitmap: Zeichen-Bitmap für FSAA-Operation
{
```

```
  double dblScaleX = iFSAA * PdblScaleX * iXFactor;
  double dblScaleY = iFSAA * PdblScaleY;
  iWidth *= iFSAA; iHeight *= iFSAA;

  int iCenter = iWidth / 2;
  double dblStangenWerte[10];
  double dblcnt;
  double dblWinkel, dblWinkelGrad; // Winkel im Endpunkt
  int icnt;
  char sTemp[32];
  double x, y, phi, xx, yy, xs1, ys1, xs2, ys2, xs3, ys3, xs4, ys4; // Für Berechnung
der Gewichtskoordinaten
  int ix, iy;
  Windows::TPoint GewichtPunkte[5]; // Für Berechnung der Gewichtskoordinaten
  unsigned char *src, *src2, *src3, *src4, *dest; // Zeiger auf Bildbuffer für FSAA-
Skalierung

  // Konstanten für unsere Stange
  const double Ia = atof(frmConfig->edtIa->Text.c_str());
  const double s_pow2 = pow(atof(frmConfig->edts->Text.c_str()), 2);
  const double r_pow4 = pow(atof(frmConfig->edtr->Text.c_str()), 4);
  const double xsp = dblAuslenkung;
  const double l = atof(frmConfig->edtl->Text.c_str());
  const double cStangenDicke = 0.006573325445341;

  // Konstante für Gewicht
  const double cGewichtGroesse = 0.01643331361341;

  // Vorberechnete Terme, die pro xsp-Wert konstant bleiben (Performance!)
  double fltTerm = ( (4.0l * Ia * s_pow2) / (3.0l * M_PI * r_pow4 * xsp) );
  double fltTerm_pow2 = fltTerm * fltTerm;

  // Backbuffer löschen
  bmpBackbuffer->Canvas->Brush->Color = clBlack;
  bmpBackbuffer->Canvas->FillRect(Rect(0, 0, iWidth - 1, iHeight - 1));

  // Stange berechnen
  icnt = 0;
  for(dblcnt = 0.0l; dblcnt < l; dblcnt += (l / 10.0l))
  {
    if(xsp == 0.0l)
      dblStangenWerte[icnt] = 0.0l;
    else
    {
      if(xsp >= 0)
      {
        dblStangenWerte[icnt] = fltTerm -
                                sqrt( fltTerm_pow2 - hoch4(dblcnt) );
      }
      else
      {
        dblStangenWerte[icnt] = fltTerm +
                                sqrt( fltTerm_pow2 - hoch4(dblcnt) );
      }
    }

    icnt++;
  }

  // Winkel im Endpunkt berechnen und anzeigen
  dblWinkel = atan((dblStangenWerte[icnt - 2] - dblStangenWerte[icnt - 1]) / (l /
10.0l));
  dblWinkelGrad = (dblWinkel / M_PI) * 180.0l;
  sprintf(sTemp, "%+.02lf °", dblWinkelGrad);
  lblWinkel->Caption = sTemp;
  lblWinkel->Refresh();
  dblWinkel *= dblScaleX / dblScaleY;

  // Stange zeichnen
  bmpBackbuffer->Canvas->Pen->Style = psSolid;
  bmpBackbuffer->Canvas->Pen->Color = clWhite;
  bmpBackbuffer->Canvas->Pen->Width = iFSAA * cStangenDicke * PdblScaleX;
  for(icnt = 0; icnt < 9; icnt++)
  {
    bmpBackbuffer->Canvas->MoveTo(iCenter + dblStangenWerte[icnt] * dblScaleX,
                                  icnt * (l / 10.0l) * dblScaleY * (1.0l - (iXFactor *
0.012l)));
    bmpBackbuffer->Canvas->LineTo(iCenter + dblStangenWerte[icnt + 1] * dblScaleX,
                                  (icnt + 1) * (l / 10.0l) * dblScaleY * (1.0l -
(iXFactor * 0.012l)));
  }
```

```
  // Gewicht zeichnen
  bmpBackbuffer->Canvas->Brush->Color = clWhite;
  x = iCenter; y = 0.4321961480311 * dblScaleY; phi = dblWinkel;
  // Punkt 1 transformieren ol
  xx = dblStangenWerte[icnt] * dblScaleX - (iFSAA * cGewichtGroesse * PdblScaleX);
  yy = iFSAA * -cGewichtGroesse * PdblScaleY;
  trans(x, y, phi, xx, yy, &xs1, &ys1);
  // Punkt 2 transformieren or
  xx = dblStangenWerte[icnt] * dblScaleX + (iFSAA * cGewichtGroesse * PdblScaleX);
  yy = iFSAA * -cGewichtGroesse * PdblScaleY;
  trans(x, y, phi, xx, yy, &xs2, &ys2);
  // Punkt 3 transformieren ur
  xx = dblStangenWerte[icnt] * dblScaleX + (iFSAA * cGewichtGroesse * PdblScaleX);
  yy = iFSAA * cGewichtGroesse * PdblScaleY;
  trans(x, y, phi, xx, yy, &xs3, &ys3);
  // Punkt 4 transformieren ul
  xx = dblStangenWerte[icnt] * dblScaleX - (iFSAA * cGewichtGroesse * PdblScaleX);
  yy = iFSAA * cGewichtGroesse * PdblScaleY;
  trans(x, y, phi, xx, yy, &xs4, &ys4);
  // Gewicht zeichnen
  GewichtPunkte[0] = Point(xs1, ys1);
  GewichtPunkte[1] = Point(xs2, ys2);
  GewichtPunkte[2] = Point(xs3, ys3);
  GewichtPunkte[3] = Point(xs4, ys4);
  GewichtPunkte[4] = Point(xs1, ys1);
  bmpBackbuffer->Canvas->Pen->Style = psClear;
  bmpBackbuffer->Canvas->Polygon(GewichtPunkte, 4);

  // Backbuffer in Frontbuffer kopieren
  if(iFSAA == 2)
  {
    // Wenn 2xFull Scene Anti-Aliasing gewählt, dann das doppelt so groß gerenderte
    // Bitmap auf ein Viertel der Fläche mit hoher Qualität herunterskalieren
    for(iy = 0; iy < iHeight; iy+=2)
    {
      // Zeilenpuffer setzen
      src = (unsigned char *)bmpBackbuffer->ScanLine[iy];
      src2 = (unsigned char *)bmpBackbuffer->ScanLine[iy + 1];
      dest = (unsigned char *)Bitmap->ScanLine[iy >> 1];

      for(ix = 0; ix < iWidth; ix+=2)
      {
        dest[ix >> 1] = (src[ix] + src[ix + 1] + src2[ix] + src2[ix + 1]) >> 2;
      }
    }
  }
  else if(iFSAA == 4)
  {
    // Wenn 4xFull Scene Anti-Aliasing gewählt, dann das viermal so groß gerenderte
    // Bitmap auf ein Sechzehntel der Fläche mit hoher Qualität herunterskalieren
    for(iy = 0; iy < iHeight; iy+=4)
    {
      // Zeilenpuffer setzen
      src = (unsigned char *)bmpBackbuffer->ScanLine[iy];
      src2 = (unsigned char *)bmpBackbuffer->ScanLine[iy + 1];
      src3 = (unsigned char *)bmpBackbuffer->ScanLine[iy + 2];
      src4 = (unsigned char *)bmpBackbuffer->ScanLine[iy + 3];
      dest = (unsigned char *)Bitmap->ScanLine[iy >> 2];

      for(ix = 0; ix < iWidth; ix+=4)
      {
        dest[ix >> 2] = (src[ix] + src[ix + 1] + src[ix + 2] + src[ix + 3] +
                         src2[ix] + src2[ix + 1] + src2[ix + 2] + src2[ix + 3] +
                         src3[ix] + src3[ix + 1] + src3[ix + 2] + src3[ix + 3] +
                         src4[ix] + src4[ix + 1] + src4[ix + 2] + src4[ix + 3]) >> 4;
      }
    }
  }
  else
  {
    // sonst einfach direkt zeichnen
    Bitmap->Canvas->Draw(0, 0, bmpBackbuffer);
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::FormCreate(TObject *Sender)
{
  // Prozeßpriorität erhöhen
  SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS);
```

```
  //  Formulargröße  korrigieren  (stimmt  nicht  mit  Entwicklungseinstellungen  genau
überein)
  frmMain->ClientHeight -= 8;

  // Kontrollbereichhöhe berechnen
  iControlAreaHeight = frmMain->ClientHeight - imgPendel->Height;

  // FSAA-Bitmap gleich auf richtige Größe bringen
  imgPendel->Picture->Bitmap->Width = imgPendel->Width;
  imgPendel->Picture->Bitmap->Height = imgPendel->Height;

  // Wir verwenden Doublebuffering, um Flackern zu vermeiden
  bmpBackbuffer = new Graphics::TBitmap;
  bmpBackbuffer->PixelFormat = pf8bit;
  bmpBackbuffer->Palette = imgPendel->Picture->Bitmap->Palette;

  // Variablen initialisieren
  bBreak = false;
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::FormDestroy(TObject *Sender)
{
  // Objekte löschen
  delete bmpBackbuffer;
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::edtXFaktorChange(TObject *Sender)
{
  if(atof(edtXFaktor->Text.c_str()) == 0)
    edtXFaktor->Text = "1";
  if(atof(edtXFaktor->Text.c_str()) > 5)
    edtXFaktor->Text = "5";
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::FormClose(TObject *Sender, TCloseAction &Action)
{
  // Prozeßpriorität senken
  SetPriorityClass(GetCurrentProcess(), NORMAL_PRIORITY_CLASS);
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::btnEndeClick(TObject *Sender)
{
  // Prozeßpriorität senken
  SetPriorityClass(GetCurrentProcess(), NORMAL_PRIORITY_CLASS);

  this->Close();
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::timeStartupTimer(TObject *Sender)
{
  timeStartup->Enabled = false;

  // Bild löschen (über cbxFSAAClick-Ereignis)
  rgFSAAClick(this);
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::rgFSAAClick(TObject *Sender)
{
  int iFSAA = pow(2, rgFSAA->ItemIndex); // FSAA-Faktor berechnen

  // Bild löschen
  imgPendel->Picture->Bitmap->Canvas->Brush->Color = clBlack;
  imgPendel->Picture->Bitmap->Canvas->FillRect(Rect(0,   0,   (imgPendel->Width)   -   1,
(imgPendel->Height) - 1));

  // Backbuffer initialisieren
  bmpBackbuffer->Width = imgPendel->Width * iFSAA;
  bmpBackbuffer->Height = imgPendel->Height * iFSAA;
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::FormResize(TObject *Sender)
{
  if(!timeStartup->Enabled) // schon fertig initialisiert?
  {
    // Seitenverhältnis korrigieren
    if(frmMain->ClientHeight - iControlAreaHeight > frmMain->ClientWidth)
```

```
      frmMain->ClientWidth = frmMain->ClientHeight - iControlAreaHeight;
    else if (frmMain->ClientHeight - iControlAreaHeight < frmMain->ClientWidth)
      frmMain->ClientHeight = frmMain->ClientWidth + iControlAreaHeight;
    imgPendel->Width = frmMain->ClientWidth;
    imgPendel->Picture->Bitmap->Width = frmMain->ClientWidth;
    imgPendel->Height = frmMain->ClientHeight - iControlAreaHeight;
    imgPendel->Picture->Bitmap->Height = frmMain->ClientHeight - iControlAreaHeight;
    rgFSAAClick(this);

    // Ein paar Controls verschieben
    btnEnde->Left = frmMain->ClientWidth - btnEnde->Width;
    rgFSAA->Left = frmMain->ClientWidth - rgFSAA->Width;
  }
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::btnSimulationClick(TObject *Sender)
{
  SimPenduleum();
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::btnKonfigClick(TObject *Sender)
{
  frmConfig->ShowModal();
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::FormKeyPress(TObject *Sender, char &Key)
{
  // Key = 27 => ESC gedrückt
  if(Key == 27)
    bBreak = true;
}
//---------------------------------------------------------------------------
```

8.1.2. Modul `main.h`

```
//---------------------------------------------------------------------------

#ifndef MainH
#define MainH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <DdeMan.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
#include <Mask.hpp>
#include <Buttons.hpp>
//---------------------------------------------------------------------------
class TfrmMain : public TForm
{
__published:   // Von der IDE verwaltete Komponenten
        TDdeServerConv *DDEPenduleum;
        TDdeServerItem *DDEPenduleumItem;
        TLabel *lblAuslenkung;
        TLabel *lblAuslenkungCap;
        TLabel *Label1;
        TLabel *lblWinkel;
        TLabel *Label2;
        TMaskEdit *edtXFaktor;
        TBitBtn *btnEnde;
        TImage *imgPendel;
        TTimer *timeStartup;
        TRadioGroup *rgFSAA;
        TBitBtn *btnSimulation;
        TBitBtn *btnKonfig;
        void __fastcall DDEPenduleumItemChange(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall FormDestroy(TObject *Sender);
        void __fastcall edtXFaktorChange(TObject *Sender);
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall btnEndeClick(TObject *Sender);
        void __fastcall timeStartupTimer(TObject *Sender);
        void __fastcall rgFSAAClick(TObject *Sender);
        void __fastcall FormResize(TObject *Sender);
        void __fastcall btnSimulationClick(TObject *Sender);
        void __fastcall btnKonfigClick(TObject *Sender);
        void __fastcall FormKeyPress(TObject *Sender, char &Key);
private:       // Anwender-Deklarationen
        void __fastcall TfrmMain::SimPenduleum(void);
        void __fastcall TfrmMain::DrawPenduleum(double dblAuslenkung, int iHeight, int
iWidth,
                                                int iFSAA, double PdblScaleX,
                                                double   PdblScaleY,   int   iXFactor,
Graphics::TBitmap *Bitmap);

        double dblAuslenkung;
        Graphics::TBitmap *bmpBackbuffer;
        int iControlAreaHeight; // Höhe des Kontrollbereichs über dem Bild
        bool bBreak; // ESC-Taste gedrückt
public:        // Anwender-Deklarationen
        __fastcall TfrmMain(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TfrmMain *frmMain;
//---------------------------------------------------------------------------
#endif
```

8.1.3. Modul `config.cpp`

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Config.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmConfig *frmConfig;
//---------------------------------------------------------------------------
__fastcall TfrmConfig::TfrmConfig(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TfrmConfig::btnCloseClick(TObject *Sender)
{
  frmConfig->Close();
}
//---------------------------------------------------------------------------
```

8.1.4. Modul `config.h`

```
//---------------------------------------------------------------------------

#ifndef ConfigH
#define ConfigH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TfrmConfig : public TForm
{
__published:   // Von der IDE verwaltete Komponenten
        TBitBtn *btnClose;
        TEdit *edtT1_Annaeher;
        TLabel *lblT1_Annaeher;
        TLabel *Label2;
        TLabel *Label3;
        TEdit *edtSimDauer;
        TLabel *Label1;
        TLabel *lblr_anfang;
        TEdit *edtr_Anfang;
        TLabel *lblr_danach;
        TEdit *edtr_danach;
        TLabel *lbldelta1;
        TEdit *edtdelta1;
        TLabel *lbldelta2;
        TEdit *edtdelta2;
        TLabel *lblT1;
        TEdit *edtT1;
        TLabel *lblT2;
        TEdit *edtT2;
        TEdit *edtIa;
        TLabel *lblIa;
        TEdit *edts;
        TLabel *Label4;
        TEdit *edtr;
        TLabel *lblr;
        TEdit *edtl;
        TLabel *lbll;
        TBevel *Bevel1;
        void __fastcall btnCloseClick(TObject *Sender);
private:       // Anwender-Deklarationen
public:        // Anwender-Deklarationen
        __fastcall TfrmConfig(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TfrmConfig *frmConfig;
//---------------------------------------------------------------------------
#endif
```