BACHELORARBEIT

Vergleich von WinFACT und MATLAB/Simulink anhand der Portierung von Beispielen

durchgeführt am Bachelorstudiengang
Informationstechnik und System–Management
Fachhochschule Salzburg GmbH

vorgelegt von:
Edis Hodzic, Daniel Rohrmoser, Wolfgang Wasmeier

Studiengangsleiter:	FH-Prof. DI Dr. Gerhard Jöchtl
Betreuer:	DI Dr. Andreas Magauer

Salzburg, Februar 2010

Eidesstattliche Erklärung

Ich/Wir versichere(n) an Eides statt, dass ich/wir die vorliegende Bachelorarbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und alle aus ungedruckten Quellen, gedruckter Literatur oder aus dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte gemäß den Richtlinien wissenschaftlicher Arbeiten zitiert bzw. mit genauer Quellenangabe kenntlich gemacht habe(n). Diese Arbeit wurde in gleicher oder ähnlicher Form weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt und stimmt mit der durch die BegutachterInnen beurteilten Arbeit überein.

Ort, Datum — Name und Unterschrift der/s Studierenden

Kurzzusammenfassung

Anhand der Portierung verschiedener in WinFACT vorliegender Beispiele nach Simulink werden die Möglichkeiten beider Anwendungssysteme geprüft und hinsichtlich der Verwendbarkeit im Regelungstechnik-Unterricht evaluiert. Die Beispiele umfassen Themen aus den Bereichen: Dynamische Modelle elektrischer/mechanischer/elektromechanischer Systeme, Reglereinstellverfahren, Integralkriterien, Pulsweitenmodulation, Digitale und Fuzzy-Regelungen.

Abstract

The capabilities of both software systems – especially regarding the use in education of control system engineers – are evaluated by means of porting WinFACT examples to Simulink. The examples cover problems of various topics, including: dynamic models of electronic/mechanical/electromechanic systems, tuning rules, integral criteria, pulse width modulation, digital and fuzzy control systems.

1 Einleitung

Aufgrund der rapiden Leistungssteigerung, die PC-Systeme in den letzten Jahren erfahren haben, ist es nun möglich, selbst komplexe und rechenintensive Simulationen aus dem Bereich der Regelungstechnik in kurzer Zeit durchzuführen. Dem Regelungstechniker werden Werkzeuge an die Hand gegeben, die eine unmittelbare Überprüfung struktureller und rechnerischer Überlegungen erlauben. Der Engineering-Prozess wird dadurch signifikant beschleunigt und, bei entsprechender – überlegter – Anwendung, die Fehlerwahrscheinlichkeit bei der schlussendlich notwendigen Realisierung gesenkt.

Auch in der Lehre tun sich durch das enorme Angebot an Werkzeugen Möglichkeiten auf, den Unterricht eingängiger und interessanter zu gestalten. Das Gelernte kann sofort durch Simulation belegt, diffizile Vorgänge durch Animationen in anschaulicher Art präsentiert und Ergebnisse reproduzierbar abgelegt werden. Die Notwendigkeit aus früheren Zeiten, einen praktischen Bezug durch Anschaffung teurer Hardware herzustellen, besteht nicht mehr – vielfach werden jene Problemstellungen auf Software abgebildet.

Diese Arbeit beschäftigt sich mit zwei auf dem Markt verfügbaren Software-Paketen, die sowohl in Lehre und Forschung als auch in der Industrie breite Anwendung finden. Zum Einen ist dies das Produkt *WinFACT* (Version 7, Block-Simulationsprogramm *BORIS* in Version 7.1.1.307) aus dem Hause *Kahlert* [8], zum Andern das Paket *MATLAB/Simulink* (Student Version 7.8.0, R2009a) von *MathWorks* [12].

Anhand der Portierung von für WinFACT vorliegenden Beispielen nach MATLAB/Simulink soll das Potenzial und die Handhabbarkeit beider Software-Systeme für den Einsatz in der Lehre evaluiert und eine Basissammlung von Simulink-Beispielen für die Lehrveranstaltung *System- und Regelungstechnik* an der Fachhochschule Salzburg erarbeitet werden.

Zu Beginn werden im theoretischen Teil die technischen Hintergründe beider Systeme verglichen, wobei der Fokus auf die Ausstattungsvielfalt der Funktionsblöcke und die Numerik-Algorithmen gelegt wird.

In weiterer Folge werden eine Vielzahl von Beispielen in praktischer Umsetzung dargestellt – sowohl in WinFACT, als auch in Simulink – wobei der praktische Teil nach den Autoren dieser Arbeit unterteilt ist. Im ersten praktischen Kapitel erfolgt die Erörterung grundlegender Strukturen (dynamische Modelle am Beispiel des Gleichstrommotors, empirische Reglereinstellverfahren und Fuzzy-Regelungen). Daran anschließend werden erweiterte Steuermechanismen für die Simulation (am Beispiel der Regleroptimierung mittels Integralkriterien) und mit der PWM ein weiteres Thema aus dem Gebiet der dynamischen Modelle vorgestellt. Den Abschluss bilden erweiterte Regelkreisstrukturen (PLL), Visualisierungsmöglichkeiten (am Beispiel von Füllstands- und Verladekranregelung) sowie die Behandlung von Abtastsystemen.

2 Theoretischer Teil

In diesem Kapitel werden die theoretischen Aspekte für den Vergleich der beiden Anwendungen erläutert. Beginnend mit einer Darstellung des Aufbaus der zu vergleichenden Softwarepakete und einer ersten Gegenüberstellung der für die Portierungen nötigen Modellierungsmöglichkeiten (hinsichtlich Blockbibliothek) wird eine Evaluierung der Möglichkeiten (speziell in Hinblick auf die Verwendung in der Lehre) möglich. Es folgt ein Überblick auf die zu Verfügung stehenden Algorithmen zur numerischen Integration und damit der Lösung von Differentialgleichungen, welche den physikalischen Zusammenhängen zugrunde liegen.

Zur Wahrung des Kontexts werden die den Simulationen zugrunde liegenden physikalischen Modelle im praktischen Teil im Zuge der Beschreibung der jeweiligen Umsetzungen näher erläutert und sind daher nicht Teil dieses Kapitels.

2.1 Aufbau der Softwarepakete

2.1.1 WinFACT

WinFACT ist eine Software-Umgebung, die auf das Gebiet der Regelungstechnik spezialisiert ist. Das Paket umfasst mehrere Programme, die in modularer Art und Weise bei der Lösung von Problemen der konventionellen linearen Regelungstechnik, aber auch den Einsatz „moderner“ Verfahren (wie des Fuzzy Controls) unterstützt [8, 9]. Die wichtigsten Anwendungen:

BORIS Die BORIS-Umgebung (Blockorientierte Simulation) ist das für diese Arbeit zentrale bzw. hier eingesetzte Tool. Mit ihm können Regelkreise aller Arten unter Verwendung der geläufigen Darstellung in Blockdiagrammen simuliert, analysiert und optimiert werden. Neben den zu Verfügung stehenden Blockelementen können vom Benutzer eigene Blöcke durch sog. *Superblöcke* (in einen Block verpackte Subsysteme) oder *User-DLLs* hinzugefügt werden.

LISA Stellt ein Werkzeug zur linearen Systemanalyse dar, das die geläufigsten Darstellungsformen (Sprungantwort, Bode-Diagramme/-Knickzüge, Ortskurven, Wurzelortskurven, Pol-/Nullstellendiagramm) eines linearen Systems ermitteln und anzeigen kann.

IDA Mit IDA kann aus einer Messreihe mit Eingangs- und Ausgangsdaten eines System heraus eine Übertragunsfunktion (rationale Form mit Totzeit) identifiziert werden.

RESY Bietet Möglichkeiten zur Synthese linearer Regler für bereits bekannte Regelstrecken. Die Reglerparameter können dabei unter Verwendung von Sprungantworten, Bode-Diagrammen und Ortskurven optimiert werden.

SUSY Um Systeme mittels der Zustandsraumdarstellung behandeln zu können enthält WinFACT das Programmm SUSY. Hiermit können Zustandsmodelle durch Anwendung der gängigen Analyse- und Synthesemethoden (bspw. Trajektorienfelder oder Synthese mittels Polplatzierung) bearbeitet werden.

FLOP Die Fuzzy-Shell FLOP wird beim Entwurf von Fuzzy-Reglern benötigt. Sie bietet alle gängigen Methoden für die Fuzzyfizierung, die Bildung der Inferenzmaschine und der Defuzzyfizierung. Die erstellten Regler können dann z.B. in BORIS eingesetzt werden.

INGO Wird zur grafischen Aufbereitung von Simulationsergebnissen verwendet.

Zudem existieren weitere, spezialisierte Programme, die zusätzliche Problemstellungen im Bereich der Sytem- und Regelungstechnik abdecken (wie z.B. Tools zur experimentellen Frequenzgangermittlung oder zur Generierung von C-Sourcecode für Fuzzy-Regler). Weiters sind speziell

für den Unterricht ausgelegte Anwendungen (z.B. BODE- oder SIM-Trainer) erhältlich, die je nach Bedarf in der Ausbildung eingesetzt werden können. Für diese Arbeit von besonderer Bedeutung ist das Programm *BORIS*, da es im Wesentlichen das Gegenstück zu Simulink darstellt. Ebenfalls zur Anwendung kommt *FLOP* im Zuge der Erstellung und Konfiguration von Fuzzy-Systemen.

Trotz der (besonders für den Unterricht) guten Abdeckung der relevanten Themengebiete wird WinFACT in einer Preisklasse angeboten, die auch für den Privatanwender (und in weiterer Folge auch für Ausbildungsstätten) von Interesse ist. Neben der akademischen Anwendung findet WinFACT aber auch in einer Vielzahl von industriellen Betrieben eine breite Anwendung, was die Leistungsfähigkeit des Paketes unterstreicht.

2.1.2 MATLAB/Simulink

MATLAB ist eine Software-Umgebung, die darauf ausgerichtet ist, technische und wissenschaftliche Probleme numerisch zu lösen. Die Anwendung bietet dazu eine Kommandozeilenoberfläche, über die der numerische Kernel bedient wird. Die höhere Sprache bietet Scripting-Fähigkeiten und greift auf effiziente Numerik-Algorithmen zur Vektor-/Matrizenrechnung zurück, die bereits von der Kernanwendung bereitgestellt werden.

Neben dem mathematischen Kern wird MATLAB (auf Bedarf) mit einer Vielzahl von sog. *Toolboxen* ausgeliefert, die weitere Algorithmen für spezielle Probleme anbieten. Eine für die Regelungstechnik interessante Auswahl dieser Toolboxen könnte sein:

- Control System Toolbox
- System Identification Toolbox
- Fuzzy Logic Toolbox
- Robust Control Toolbox
- Neural Network Toolbox
- Signal Processing Toolbox

Darüber hinaus ist es möglich, MATLAB eigenständig (entweder mit der enthaltenen Skript-Hochsprache oder binärkompilierten Modulen) zu erweitern. Viele technische Probleme wurden bereits gelöst und werden im Internet frei oder auf Anfrage angeboten.

In der Regelungstechnik zum Zwecke der Simulation häufiger verwendet wird allerdings *Simulink*, eine Erweiterung von MATLAB, die eine grafische Oberfläche zur Behandlung von Systemen mittels Blockdiagrammen bereitstellt. Einzelne Blöcke werden in sog. *Blocksets* zusammengefasst, die, ähnlich zur konventionellen MATLAB-Umgebung, das Basisangebot erweitern. Simulink und die zur Verfügung stehenden Blocksets nutzen dabei den mathematischen „Unterbau“, den MATLAB zusammen mit den Toolboxen anbietet.

Die hier wichtigsten Blocksets sind folgende:

- Simulink (Basisbibliothek; bietet viele grundlegende Blöcke)
- Simulink Extras (Erweiterungsbibliothek mit zusätzlichen Grundbausteinen)
- Control System Toolbox
- System Identification Toolbox
- Fuzzy Logic Toolbox
- Signal Processing Blockset
- Communicationss Blockset
- Simulink 3D Animation

Zusätzlich dazu sind noch viele weitere Bibliotheken (sowohl für MATLAB, als auch für Simulink) erhältlich, die das Anwendungspotential und damit verbunden die Komplexität wesentlich vergrößen können. Beispiele wären die Bildverarbeitung, die Simulation elektronischer/mechanischer/elektromechanischer Systeme, HF-Anwendungen oder die Behandlung von Zustandsautomaten.

Aus dieser Auflistung ist zu ersehen, dass MATLAB/Simulink eine hochgradig modulare Numerik-Software-Umgebung darstellt, die ein breites Themenspektrum abdeckt. Aufgrund der Mächtigkeit und Qualität des Algorithmensystems ist MATLAB nicht nur für das akademische Umfeld, sondern auch für die Industrie von Interesse. Die Preise liegen dabei wesentlich höher als es bspw. bei WinFACT der Fall ist, es bieten sich dafür aber mehr Möglichkeiten, das Tool auch in Unterrichtsgegenständen einzusetzen, die nicht ausschließlich das Gebiet der Regelungstechnik behandeln.

2.2 Vergleich der Blockausstattungen

Für die hier angewendete Methode der Signalflusspläne bzw. Blockschaltbilder stellen beide Pakete eine reichliche Anzahl vordefinierter Blöcke bereit. Sowohl für den konventionellen linearen Fall, wie auch für nichtlineare oder diskrete Anwendungen sind die wesentlichen Grundblöcke verfügbar, wobei die „Verfeinerung“ in die Basisfunktionen unterschiedlich gewichtet wird: So stellt BORIS beispielsweise eine Vielzahl von Elementargliedern (P, I, D, PT_1, PT_1, DT_1, ...) bereit, wohingegen sich MATLAB/Simulink auf wenige, allgemeine Glieder (wesentlich: *Integrator*, *Derivative* und eine allgemeine *Transfer Function* mit Angabe von Zähler- und Nennerpolynom) beschränkt. Ein anderes Beispiel wären mathematische Funktionen, die in Simulink breit gefächert vorhanden sind (Multiplikation, Division, Absolutwert, trigonometrische Funktionen etc.), während in BORIS lediglich ein Block mit mathematischem Parser für eine oder mehrere Veränderliche angeboten wird.

Grundsätzlich verfügen beide Probanden über eine ausreichende Blockauswahl, die für die Modellierung und Simulation der gängigen dynamischen Systeme zur Genüge reicht.

Beachtenswert ist die Tatsache, dass WinFACT deutlicher auf Benutzerinteraktion während der Simulation ausgerichtet ist als Simulink, bei dem der wissenschaftliche Aspekt im Vordergrund steht. BORIS bietet eine Reihe von Controls an, mit denen der User das Verhalten des Regelkreises grafisch beeinflussen kann (Buttons, Slider Controls, grafische PID-Regler, diverse grafische Displays etc.), schränkt sich dafür aber bei Implementationen komplexerer Algorithmen ein. Die Vielfalt der mit MATLAB mitgelieferten Funktionalitäten hingegen ermöglicht es, Simulink auch mit anderen, der Regelungstechnik nicht zwangsweise nahen Wissenschaften zu verbinden, wie es an den Beispielen der Bildverarbeitung oder der Entwicklung von Embedded Software zu sehen ist.

Besonders im letztgenannten Punkt der Interoperabilität zeigt sich bereits ein wesentlicher Unterschied in der Ausrichtung der Software-Lösungen: Simulink bietet ein breites Anwendungsspektrum, das im Wesentlichen auf die Prüfung vorher berechneter (und durchaus komplexer) Aufgabenstellungen durch Simulation abzielt, währen bei WinFACT/BORIS eher die Bedienung während der Simulation (bzw. der Steuerung realer Systeme) im Vordergrund gehalten wird und damit eine intuitivere Herangehensweise ermöglicht.

2.3 Algorithmen zur numerischen Integration

Zur Lösung der Differentialgleichungen, durch die dynamische Systeme beschrieben werden, können in Simulationen eine Reihe von numerischen Integrationsverfahren zur Anwendung kommen. Zu den bekanntesten zählen hierbei das Streckenzugverfahren nach EULER oder die Berechnung nach RUNGE-KUTTA (Details siehe [14]). Der EULER-Ansatz ist dabei der einfachere, verliert jedoch mit steigender Simulationsschrittweite schnell an Genauigkeit. RUNGE-KUTTA weist ein bessere Präzision auf, was allerdings mit einem leicht erhöhten Rechenaufwand erkauft wird [10].

BORIS implementiert drei Verfahren [10]:

- EULER'sches Streckenzugverfahren
- RUNGE-KUTTA-Verfahren 4. Ordnung
- Matrixexponentialverfahren (theoretisch unendlich genau, aber nur für lineare Systeme geeignet)

Alle drei Verfahren werden im *Fixed-Step*-Betrieb angewandt, was bedeutet, dass für die Simulation eine fixe Schrittweite festgelegt wird.

In MATLAB/Simulink sind folgende *Fixed-Step-Solver* – also Algorithmen mit konstanter Simulationsschrittweite – verfügbar [11]:

- *ode5:* RUNGE-KUTTA-Verfahren 4./5. Ordnung mit konstanter Schrittweite nach DORMAND-PRINCE
- *ode4:* RUNGE-KUTTA-Verfahren 4. Ordnung
- *ode3:* RUNGA-KUTTA-Verfahren 2./3. Ordnung mit konstanter Schrittweite nach BOGACKI-SHAMPINE
- *ode2:* RUNGE-KUTTA-Verfahren 2. Ordnung nach HEUN
- *ode1:* EULER'sches Streckenzugverfahren

Zusätzlich dazu bietet MATLAB noch sog. *Variable-Step-Solver* an, die die Schrittweite während der Simulation verändern, um den Rechenzeiteinsatz zu optimieren. Gesteuert wird die Schrittweitenregelung durch Fehlertoleranzgrenzen. In MATLAB ist zwischen nachstehenden Lösungsalgorithmen mit variabler Schrittweite auszuwählen [11]:

- *ode45:* RUNGE-KUTTA-Verfahren 4./5. Ordnung mit variabler Schrittweite nach DORMAND-PRINCE
- *ode23:* RUNGE-KUTTA-Verfahren 2./3. Ordnung mit variabler Schrittweite nach BOGACKI-SHAMPINE
- *ode113:* ADAMS-BASHFORTH-MOULTON-Verfahren
- *ode15s:* Mehrschrittverfahren für steife Systeme
- *ode23s:* ROSENBROCK-Verfahren 2. Ordnung
- *ode23t:* Einschrittverfahren nach der Trapezregel
- *ode23tb:* Implizites RUNGE-KUTTA-Verfahren

Nach [11, 13] eignen sich die ersten drei genannten für nicht-steife Systeme, während *ode15s* bzw. *ode23s* bei steifen Systemen zur Anwendung kommen. Der *ode23t*-Algorithmus wird für mittlere Steifigkeit empfohlen, *ode23tb* eignet sich für große Toleranzen. „Steif" ist ein System gem. [10] dann, wenn es stark schwingfähig ist oder sehr unterschiedliche Zeitkonstanten aufweist.

Um grobe Ungenauigkeiten zu vermeiden, können die Variable-Step-Solver parametriert werden. Neben *Start time* und *Stop time* können mit der *Max step size* bzw. der *Min step size* die

maximale resp. minimale Schrittweite festgelegt werden. Ein Verkleinerung der max. Schrittweite führt zur Steigerung der Genauigkeit (auf Kosten der Rechenzeit), eine Erhöhung der Minimalschrittweite reduziert die Rechenzeit, kann allerdings zu Präzisionsverlusten oder Fehlern bei Überschreiten der Fehlertoleranzen führen. Standardmäßig wird die *Min step size* dynamisch ermittelt, die *Max step size* wird nach Gl. (1) kalkuliert.

$$\text{Max step size} = \frac{\text{Stop time} - \text{Start time}}{50} \tag{1}$$

Für rein diskrete Systeme stehen zudem je ein Fixed-Step- und ein Variable-Step-Solver zur Verfügung (*Discrete*). Diese können aber nur in Modellen zur Anwendung kommen, in denen keine kontinuierlichen Zustände (*Continous states*) vorhanden sind.

Für die Standardanwendungen liefert der in Simulink voreingestellte Variable-Step-Solver *ode45* meist gute Ergebnisse. In [1] wird für die Auswahl des passenden Integrationsalgorithmus empfohlen, zuerst einen der drei Algorithmen für nicht-steife System einzusetzen. Bei Schwierigkeiten (bspw. zu hoher Rechenzeit) können bei Bedarf die alternativen Auswahlmöglichkeiten ausprobiert werden. Eine detaillierte Beschreibung der Anwendungsfälle bietet hierbei die MATLAB/Simulink-Hilfe [13].

Anhand der Darstellung in diesem Abschnitt ist zu erkennen, dass MATLAB/Simulink eine wesentlich größere Bandbreite an Konfigurationsoptionen hinsichtlich der Simulationsparameter aufweist als es bei WinFACT/BORIS der Fall ist. Um allerdings die für die jeweilige Aufgabenstellung optimale Einstellung zu finden, ist ein tiefes Verständnis der Algorithmen vonnöten, wobei für die Standardanwendungen bei beiden Programmen meist das Runge-Kutta-Verfahren 4. Ordnung ein ausreichende Genauigkeit bei geringer Rechenzeit liefert.

3 Praktischer Teil – Grundlegende Strukturen und Fuzzy-Control

In diesem Kapitel der Bachelorarbeit werden folgende Themengebiete behandelt:

- Einstellregeln für Regelkreise
- Gleichstrommotor
- Fuzzy-Regelung

3.1 Einstellregeln für Regelkreise

Bei den praktischen Einstellregeln wird von experimentell ermittelten Kenngrößen des Regelkreises oder der Regelstrecke ausgegangen, ein mathematisch erheblicher Aufwand, der bspw. für die Integralkriterien benötigt wird, wird vermieden. Die Einstellregeln gelten nur für bestimmte Regelstrecken und können deshalb nur begrenzt eingesetzt werden. Die Reglereinstellung ermittelt sich aus Näherungsformeln von durchgeführten Messungen [11].

Grundsätzlich stehen zwei Verfahren für die Reglereinstellung zur Verfügung [11]:

- Einstellung nach Ziegler und Nichols
- Einstellung nach Chien, Hrones und Reswick

3.1.1 Einstellregel nach Ziegler und Nichols

Wenn für eine Regelstrecke kein mathematisches Modell vorliegt, diese jedoch ein Verhalten wie eine Reihenschaltung eines Verzögerungsgliedes 1. Ordnung (PT_1-Element) und eines Totzeitgliedes (T_t) oder wie eines PT_n-Gliedes aufweist, so wird die Einstellregel nach ZIEGLER und NICHOLS angewendet [15].

Es wird zunächst am Regelkreis experimentiert, wobei wie nachfolgend vorgegangen wird [15]:

1. Der Regler wird als reiner P-Regler eingestellt.
2. Die Verstärkung k_R wird solange vergrößert bis der Regelkreis an die Stabilitätsgrenze ($k_R = k_{krit}$) gelangt, d. h. eine Dauerschwingung ausführt.
3. Die kritische Periodendauer T_{krit} der Dauerschwingung wird gemessen.
4. Die empfohlenen Einstellwerte für die verschiedenen Reglertypen folgen aus Tabelle 1.

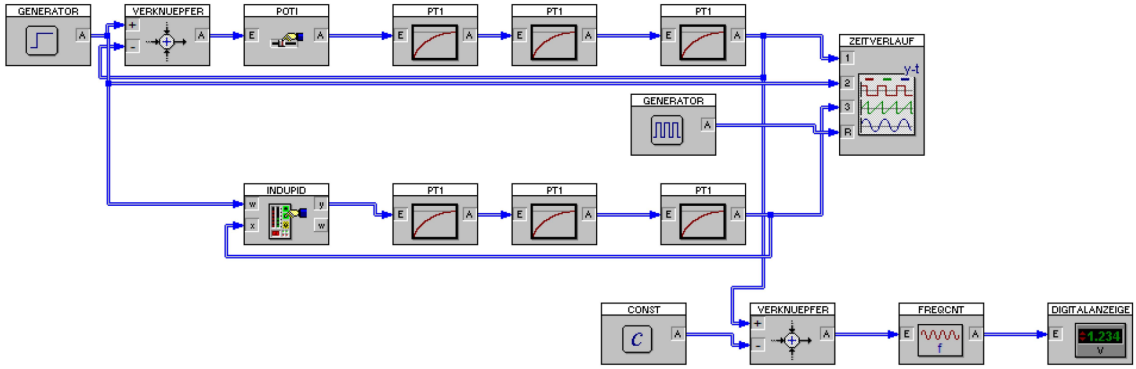
	Reglerparameter		
Reglertyp	k_R	T_N	T_V
P-Regler	$0.5\,k_{krit}$	–	–
PI-Regler	$0.45\,k_{krit}$	$0.85\,T_{krit}$	–
PID-Regler	$0.6\,k_{krit}$	$0.5\,T_{krit}$	$0.125\,T_{krit}$

Tabelle 1: Optimierung nach Ziegler und Nichols (nach [15, S. 224])

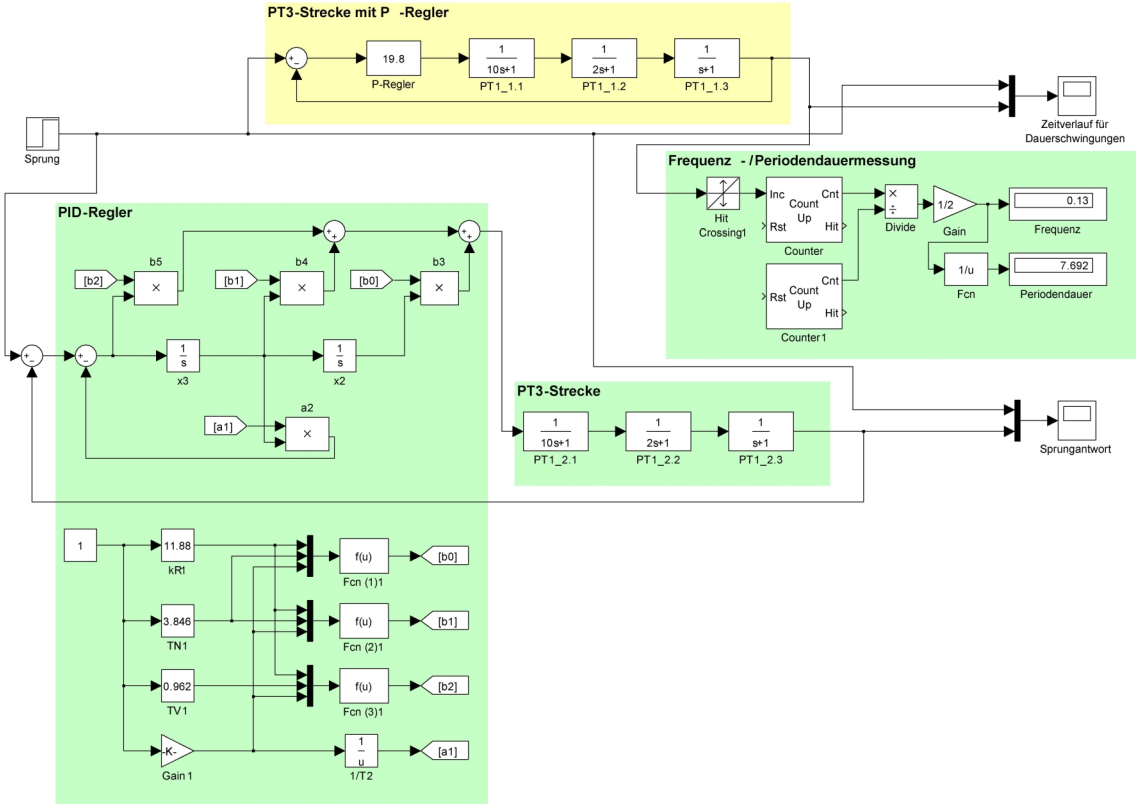
Das Einstellverfahren beschränkt sich auf Systeme, die Dauerschwingungen zulassen, ohne einen Schaden zu verursachen. Wenn dies jedoch unausweichlich sein sollte, kann mit dem Bode-Diagramm die Periodendauer der Schwingung $T_{krit} = \frac{2\,\pi}{\omega_{krit}}$ berechnet werden [15].

3.1.2 Beispiel zu Ziegler und Nichols

Anhand eines praktischen Simulationsbeispiels wird das Verfahren nach Ziegler und Nichols demonstriert.



(a) WinFACT/BORIS



(b) Simulink

Abbildung 1: Models zur Simulation der Einstellung nach Ziegler und Nichols

Die PT_3-Strecke wird mit einem P-Regler betrieben (Abbildung 1(b), gelb hinterlegt) um die Stabilitätsgrenze festzustellen. Die Reglerverstärkung wird solange erhöht, bis der kritische Zustand erreicht wird. Bei einer Verstärkung von $k_{\text{krit}} = 19.8$ bzw. bei einer Periodendauer von $T_{\text{krit}} = 7.692$ entstehen Dauerschwingungen im Regelkreis (Abb. 2).

Für die optimale Einstellung des PID-Reglers (in Abb. 1 grün hinterlegt) errechnen sich nach Tabelle 1 somit folgende Werte:

$$\begin{aligned} k_{\text{krit}} &= 0.6\, k_{\text{krit}} = 11.880 \\ T_{\text{N}} &= 0.5\, T_{\text{krit}} = 3.846 \\ T_{\text{V}} &= 0.125\, T_{\text{krit}} = 0.962 \end{aligned}$$

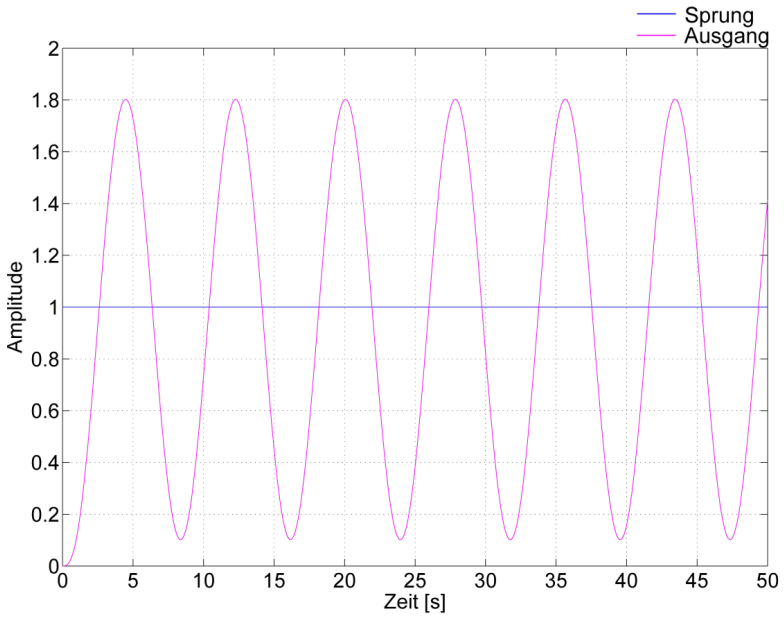


Abbildung 2: Dauerschwingung im kritischen Zustand

Die resultierende Sprungantwort der PT_3-Regelstrecke mit PID-Regler zeigt Bild 3.

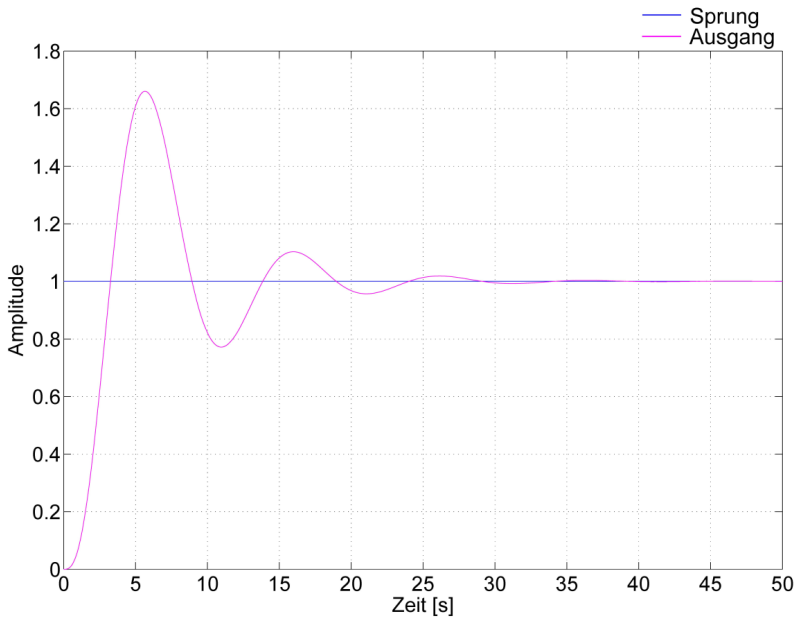


Abbildung 3: Sprungantwort einer PT_3-Strecke mit PID-Regler nach Ziegler-Nichols

3.1.3 Ziegler-Nichols – Vergleich WinFACT/BORIS und MATLAB/Simulink

Die Aufgabenstellung ist in beiden Software-Technologien problemlos lösbar. Ein etwas höheres Maß an Funktionalität bietet jedoch WinFACT/BORIS, da einige Funktionen bzw. Blöcke schon implementiert sind. Als vorteilhaft hat sich vor allem der PID-Regler erwiesen.

BORIS enthält einen Industrie-PID-Regler, der über eine Steuer- und Visualisierungsoberfläche verfügt. Somit lassen sich die Reglerparameter leicht verändern und einstellen.

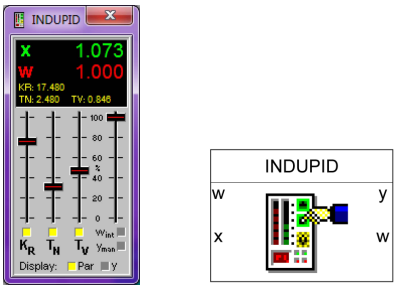


Abbildung 4: WinFACT – Industrie-PID-Regler

Simulink hingegen verfügt über keinen solchen PID-Regler, was in Hinblick auf die Benutzerfreundlichkeit einen Nachteil darstellt. Mittels eines manuell realisierten PID-Reglers wurde eine WinFACT-ähnliche Oberfläche erzeugt. Für die praktische Realisierung wird die Regelungsnormalfom (RNF) herangezogen.

Ein LTI-System ohne Totzeit kann in folgender Form dargestellt werden [20]:

$$G(s) = \frac{b_0 + b_1 s + \cdots + b_{n-1} s^{n-1} + b_n s^n}{a_0 + a_1 s + \cdots + a_{n-1} s^{n-1} + s^n} \tag{2}$$

Die Übertragungsfunktion muss also so beschaffen sein, dass die höchste Potenz des Nennerpolynoms den Koeffizienten $a_n = 1$ besitzt. Ist dies der Fall, so kann das System durch das in Abb. 5 gezeigte Blockschaltbild dargestellt werden.

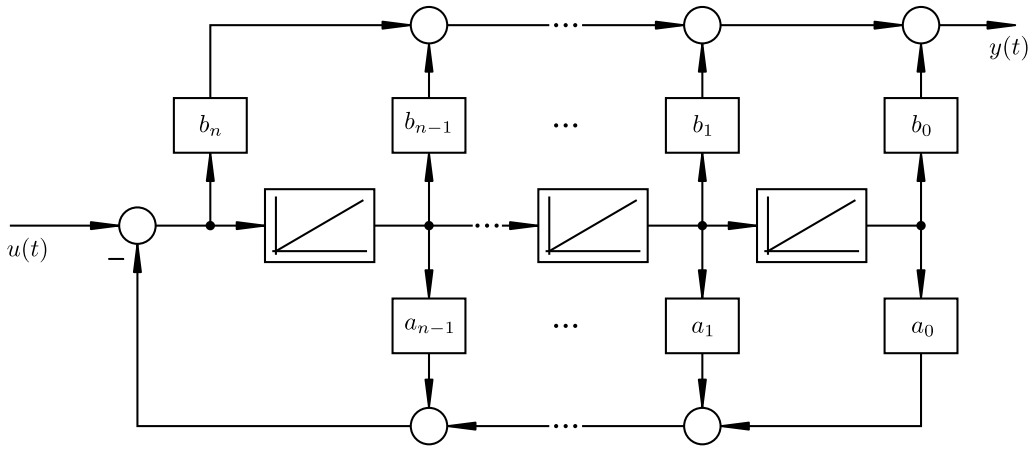


Abbildung 5: Blockschaltbild zur Regelungsnormalform

Die Übertragungsfunktion des PIDT$_1$-Reglers lautet

$$G(s) = k_\mathrm{R} \left(1 + \frac{1}{sT_\mathrm{N}} + \frac{sT_\mathrm{V}}{1 + sT_1}\right) \tag{3}$$

Es folgen einige Umformungen:

$$\begin{aligned} G(s) &= \frac{sk_\mathrm{R}T_\mathrm{N} + k_\mathrm{R}}{sT_\mathrm{N}} + \frac{sk_\mathrm{R}T_\mathrm{V}}{1+sT_1} = \frac{(sk_\mathrm{R}T_\mathrm{N} + k_\mathrm{R})(1+sT_1) + s^2k_\mathrm{R}T_\mathrm{V}T_\mathrm{N}}{sT_\mathrm{N} + s^2T_\mathrm{N}T_1} \\ &= \frac{k_\mathrm{R} + s(k_\mathrm{R}T_\mathrm{N} + k_\mathrm{R}T_1) + s^2(k_\mathrm{R}T_\mathrm{N}T_1 + k_\mathrm{R}T_\mathrm{V}T_\mathrm{N})}{sT_\mathrm{N} + s^2T_\mathrm{N}T_1} \end{aligned} \tag{4}$$

Der Berechnung zufolge ergibt sich für das System die Ordnung $n = 2$. Da die höchste Potenz im Nennerpolynom einen Koeffizienten $a_n = 1$ besitzen muss, wird der Bruch dementsprechend umgestellt:

$$G(s) = \frac{\frac{k_\mathrm{R}}{T_\mathrm{N}T_1} + sk_\mathrm{R}\left(\frac{1}{T_1} + \frac{1}{T_\mathrm{N}}\right) + s^2\left(k_\mathrm{R} + \frac{k_\mathrm{R}T_\mathrm{V}}{T_1}\right)}{s\frac{1}{T_1} + s^2} \tag{5}$$

Aus dieser Gleichung können schließlich die Koeffizienten für die RNF abgelesen werden:

$$\begin{aligned} a_0 &= 0 & b_0 &= \tfrac{k_\mathrm{R}}{T_\mathrm{N}T_1} \\ a_1 &= \tfrac{1}{T_1} & b_1 &= k_\mathrm{R}\left(\tfrac{1}{T_1} + \tfrac{1}{T_\mathrm{N}}\right) \\ a_2 &= 1 & b_2 &= k_\mathrm{R} + \tfrac{k_\mathrm{R}T_\mathrm{V}}{T_1} \end{aligned}$$

Die Umsetzung in ein Simulink-Blockschaltbild zeigt Abb. 6.

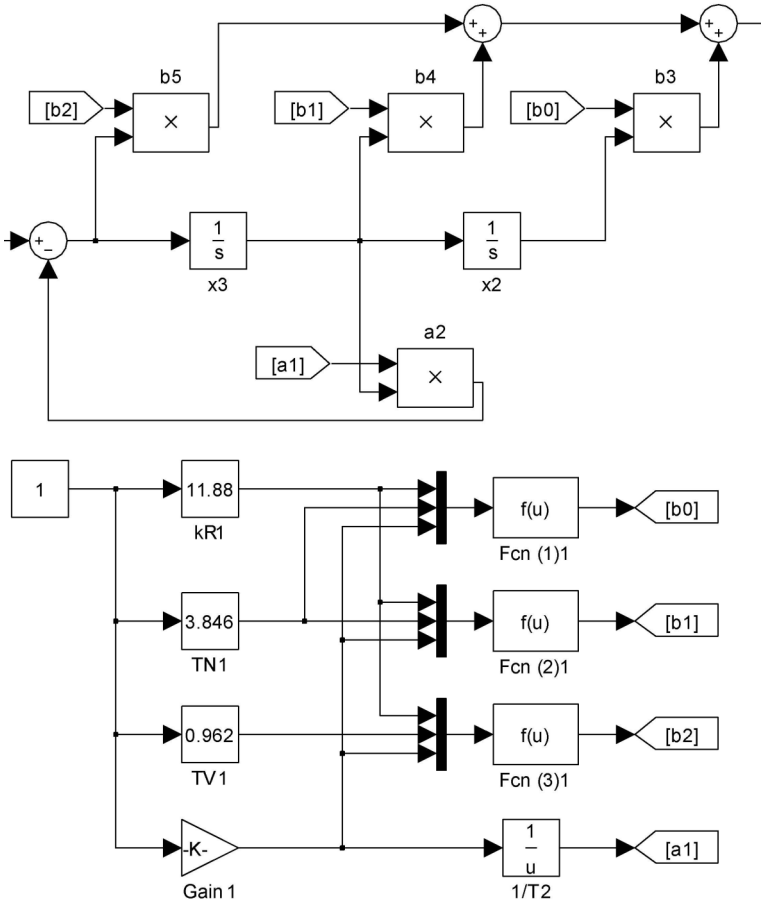


Abbildung 6: PID-Regler Regelungsnormalform – Simulink-Schaltung

3.1.4 Einstellregeln nach Chien, Hrones und Reswick

Ist die Sprungantwort einer Regelstrecke mit Verzögerung und ohne Überschwingen bekannt, kann mit diesem Verfahren die Verzugszeit T_u, die Ausgleichszeit T_g und die Streckenverstärkung k_S bestimmt werden.

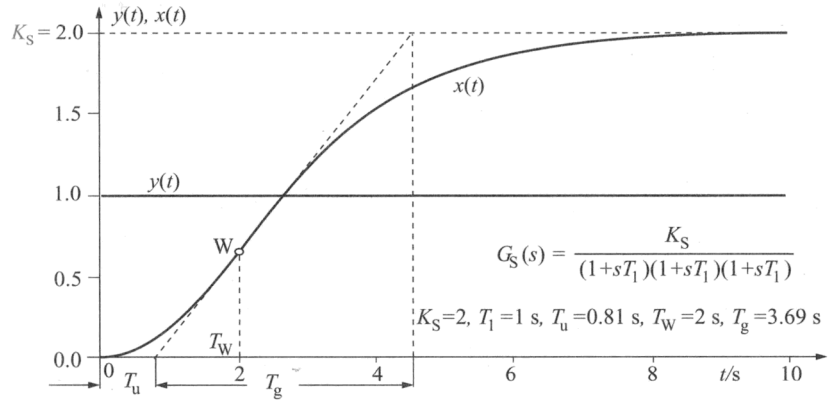


Abbildung 7: Sprungantwort einer Regelstrecke 3. Ordnung [11, S. 437]

Die Einstellregeln von Chien, Hrones und Reswick erlauben die Berechnung der Reglerparameter für ein günstiges Stör- bzw. Führungsverhalten und sind anwendbar für einen aperiodischen Regelverlauf kürzester Dauer ($\ddot{u} = 0\%$) oder für einen periodischen Verlauf kleinster Schwingungsdauer ($\ddot{u} = 20\%$) bei einem Überschwingen. Für die Anwendung der Einstellregel muss die Bedingung $\frac{T_g}{T_u} > 3$ erfüllt sein [11].

Die in Abbildung 7 dargestellte Kurve ist ein PT_n-Glied 3. Ordnung. Die Strecke besitzt jeweils einen Wendezeitpunkt T_w, eine Verzugszeit T_u und eine Ausgleichszeit T_g [11]. Ist die Voraussetzung für das Verfahren $\frac{T_g}{T_u} > 3$ erfüllt, so lassen sich die Reglerparameter (je nach Art des eingesetzten Reglers) mittels Tabelle 3 (falls gutes Störungsverhalten gewünscht ist) bzw. Tabelle 2 (für günstiges Führungsverhalten) errechnen.

Definition Führungsverhalten

> „Das Führungsverhalten eines Regelkreises beschreibt die dynamische Auswirkung einer Veränderung der Führungsgröße auf die Aufgabengröße.“ [18, S. 187]

Durch Vergleich von Soll- und Regelgröße kann jederzeit eine Regelabweichung festgestellt werden, die dem Regler zugeführt wird, um möglichst schnell eine Stellgröße zu erzeugen. Ziel des Führungsverhaltens ist, die Regelgröße durch dauernde Kontrolle auf dem Wert der Führungsgröße zu halten [18].

Reglertyp		Aperiodischer Regelverlauf ($\ddot{u} = 0\%$) bei Störungssprung	Regelverlauf mit 20% Überschwingen bei Störungssprung
P-Regler	k_R	$0.3\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$	$0.7\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$
PI-Regler	k_R	$0.35\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$	$0.6\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$
	T_N	$1.2\,T_\mathrm{g}$	$1.0\,T_\mathrm{g}$
PID-Regler	k_R	$0.6\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$	$0.95\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$
	T_N	$1.0\,T_\mathrm{g}$	$1.35\,T_\mathrm{g}$
	T_V	$0.5\,T_\mathrm{u}$	$0.47\,T_\mathrm{u}$

Tabelle 2: Chien, Hrones und Reswick mit gutem Führungsverhalten (nach [11, S. 437])

Definition Störverhalten

> „Das Störverhalten eines Regelkreises gibt die dynamischen Auswirkungen von Versorgungs- und Laststörungen auf die Regelgröße an."

Die externen Störeinflüsse, die auf die Regelstrecke einwirken und die Regelgröße verändern können, führen zu Differenzen zwischen Soll- und Istgröße. Die Aufgabe des Reglers besteht darin, auf die Strecke einzuwirken und die Störgrößen zu kompensieren [18].

Reglertyp		Aperiodischer Regelverlauf ($\ddot{u} = 0\%$) bei Störungssprung	Regelverlauf mit 20% Überschwingen bei Störungssprung
P-Regler	k_R	$0.3\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$	$0.7\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$
PI-Regler	k_R	$0.6\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$	$0.7\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$
	T_N	$4.0\,T_\mathrm{u}$	$2.3\,T_\mathrm{u}$
PID-Regler	k_R	$0.95\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$	$1.2\,\frac{T_\mathrm{g}}{T_\mathrm{u}\,k_\mathrm{S}}$
	T_N	$2.4\,T_\mathrm{u}$	$2.0\,T_\mathrm{u}$
	T_V	$0.42\,T_\mathrm{u}$	$0.42\,T_\mathrm{u}$

Tabelle 3: Chien, Hrones und Reswick mit gutem Störverhalten (nach [11, S. 437])

3.1.5 Beispiel zu Chien, Hrones und Reswick

Anhand eines praktischen Simulationsbeispiels wird das Verfahren veranschaulicht.

Von der PT_3-Strecke wird vorerst nur die Sprungantwort betrachtet, um die Kennwerte zu ermitteln. Nachdem die Kennwerte der Regelstrecke bekannt sind, sind mittels Tabelle 2 oder Tabelle 3 die entsprechenden Regelparameter für den eingesetzten PID-Regler zu berechnen.

Die Kennwerte (T_w, T_u, T_g) der Strecke sowie die Regelparameter (k_R, T_N, T_V) werden in MATLAB in einem externen m-File berechnet und gezeichnet. Das Ergebnis zeigt Bild 8.

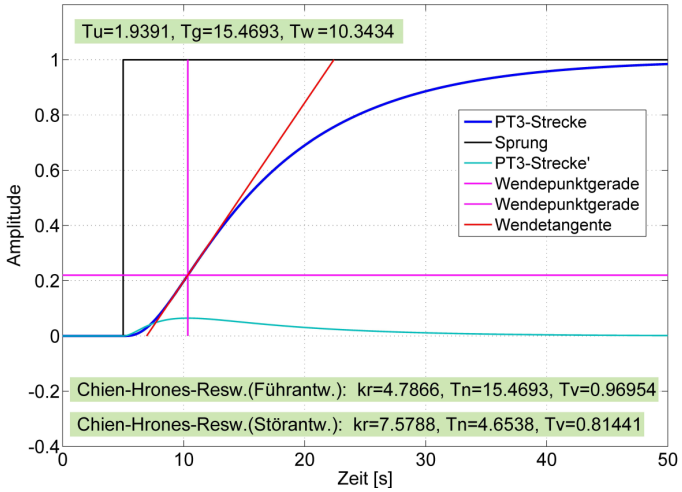
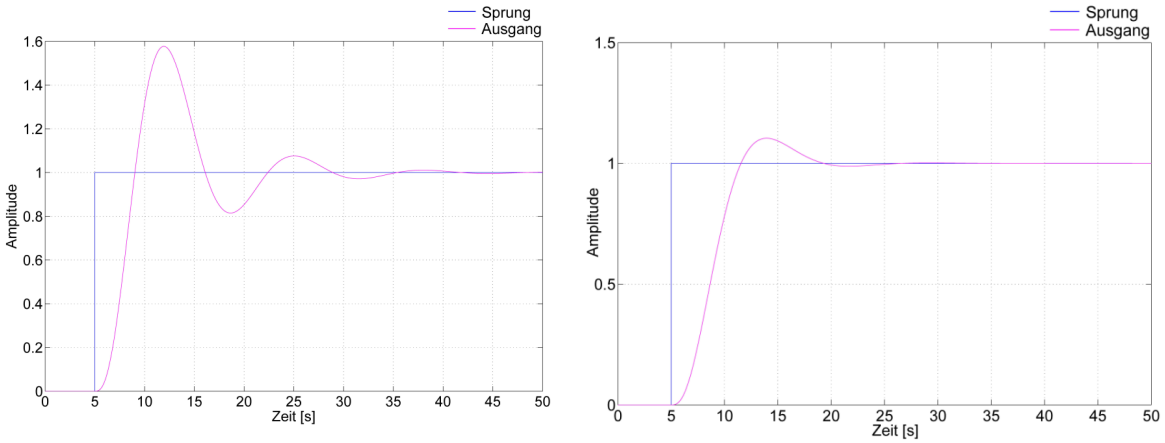


Abbildung 8: Sprungantwort mit berechneten Streckenkennwerten und Reglerparametern

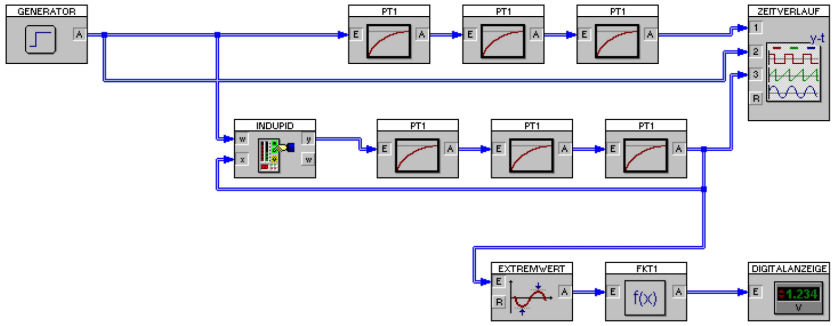
Die Voraussetzung für das Verfahren $\frac{T_g}{T_u} = \frac{15.469}{1.939} = 7.978 > 3$ ist erfüllt. Es lassen sich somit die Parameter für den PID-Regler mittels Tabelle 3 bei Störverhalten oder Tabelle 2 bei Führungsverhalten errechnen.



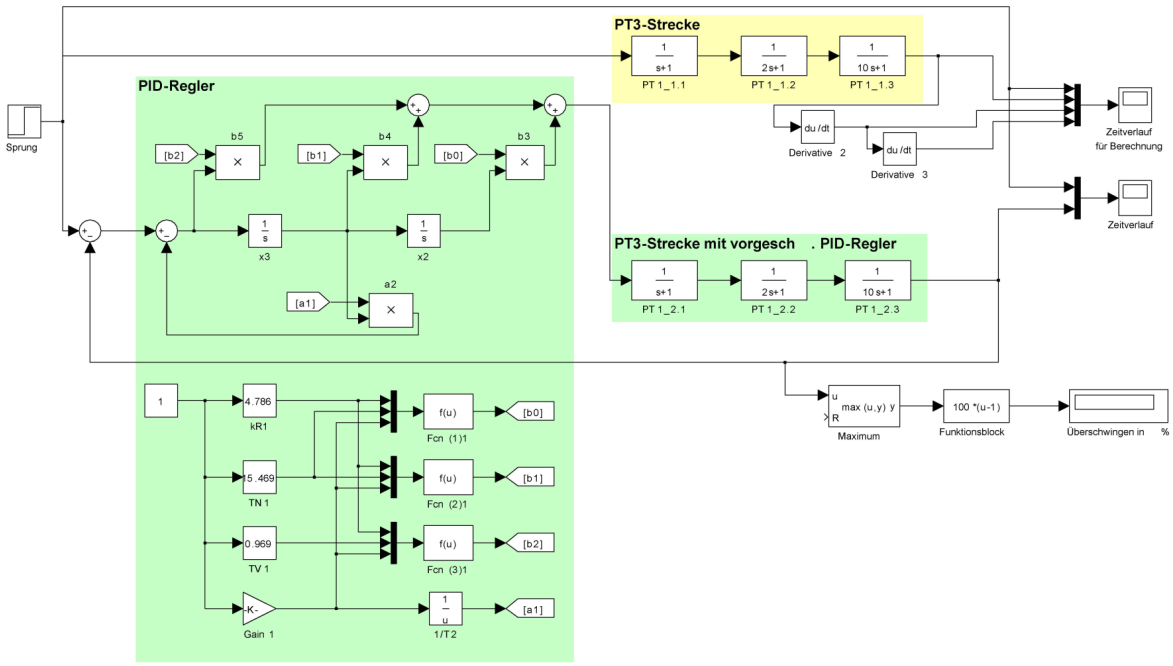
(a) Störverhalten (b) Führungsverhalten

Abbildung 9: Sprungantworten mit Einstellung nach Chien, Hrones und Reswick

Nach der Berechnung und dem Einsetzen der Regelparameter wird die Simulation gestartet. Die Sprungantworten des Regelkreises – mit eingestelltem PID-Regler – jeweils für Stör- und Führungsverhalten sind in Abb. 9 dargestellt.



(a) WinFACT/BORIS



(b) Simulink

Abbildung 10: Implementationsbeispiel zu Chien, Hrones und Reswick

3.1.6 Chien-Hrones-Reswick – Vergleich WinFACT/BORIS mit MATLAB/Simulink

Die Ermittlung der Streckenkennwerte sowie die Berechnung der Reglerparameter erfolgt in MATLAB/Simulink auf manueller Basis. WinFACT/BORIS hingegen zeigt sich hier sehr vorteilhaft, da die Ermittlung bzw. Berechnung der Kenngrößen implementiert ist bzw. automatisch erfolgen kann.

Mithilfe der Funktion *PID-Entwurf* lassen sich die Streckenparameter T_u, T_g und k_S ermitteln. Der nächste Schritt ist die Wahl der Optimierungskriterien sowie des Reglertyps und schließlich die Berechnung der Reglerparameter.

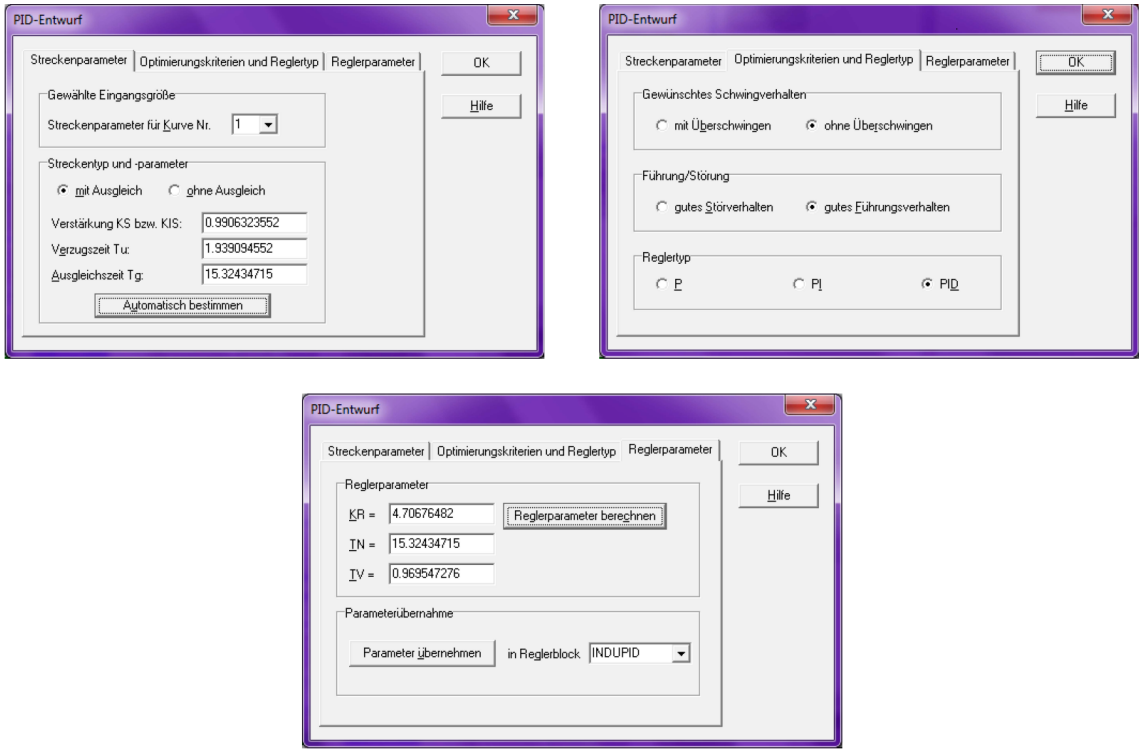


Abbildung 11: Automatischer PID-Entwurf in WinFACT/BORIS

3.2 Gleichstrommotor

3.2.1 Modell des physikalischen Systems

Als Beispiel für ein theoretisches Modell wird ein Gleichstrommotor betrachtet. Die Drehzahl wird wird über die Spannungsversorgung gesteuert und von der Belastung beeinflusst.

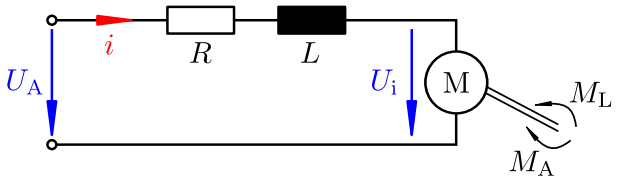


Abbildung 12: Gleichstrommotor – Ersatzschaltbild

Die physikalischen Zusammenhänge beschreiben folgende Gleichungen [5]:

$$U_A(t) - U_i(t) = R\,i(t) + L\frac{di(t)}{dt} \tag{6}$$

$$U_i(t) = k_2\,\omega(t) \tag{7}$$

$$M_A(t) = k_1\,i(t) \tag{8}$$

$$J\,\dot{\omega}(t) = M_A(t) - M_L(t) - M_R(t) \tag{9}$$

U_A bezeichnet hierbei dei Ankerspannung, i den Ankerstrom, U_i die induzierte Spannung sowie R und L den Widerstand bzw. die Induktivität im Ankerkreis. Die in diesem Kreis wirkenden Momente sind das Antriebsmoment M_A, das Lastmoment M_L, das Reibungsmoment M_R sowie das Rotorträgheitsmoment J. Die Ausgangsgröße ist die Winkelgeschwindigkeit ω bzw. deren

Ableitng, die Winkelbeschleunigung $\dot{\omega}$. Weiters werden die Ankerträgheitskonstante $T_\mathrm{A} = \frac{L_\mathrm{A}}{R_\mathrm{A}}$ und zwei Proportionalitätskonstanten k_1, k_2 benötigt [5].

Da die Linearitätsbedingung erfüllt ist, können die Gleichungen in den Laplace-Bereich transformiert werden.

$$\begin{aligned} U_\mathrm{A}(s) - U_\mathrm{i}(s) &= R\,I(s) + s\,L\,I(s) && (10)\\ U_\mathrm{i}(s) &= k_2\,\omega(s) && (11)\\ M_\mathrm{A}(s) &= k_1\,I(s) && (12)\\ s\,J\,\omega(s) &= M_\mathrm{A}(s) - M_\mathrm{L}(s) - M_\mathrm{R}(s) && (13) \end{aligned}$$

Mit $T_\mathrm{A} = \frac{L}{R}$ gilt

$$I(s) = \frac{\frac{1}{R}}{1 + sT_\mathrm{A}}\,(U_\mathrm{A}(s) - U_\mathrm{i}(s)) \qquad (14)$$

Der Ankerkreis entspricht demnach einem PT_1-Element mit der Zeitkonstante T_A. Die Winkelgeschwindigkeit ω folgt ebenfalls aus dem Gleichungssystem:

$$\omega(s) = \frac{1}{s\,J}\,(k_1\,I(s) - M_\mathrm{L}(s) - M_\mathrm{R}(s)) \qquad (15)$$

Aus diesen Beziehungen lässt sich ein Blockschaltbild bilden, wie es in Abb. 13 dargestellt ist.

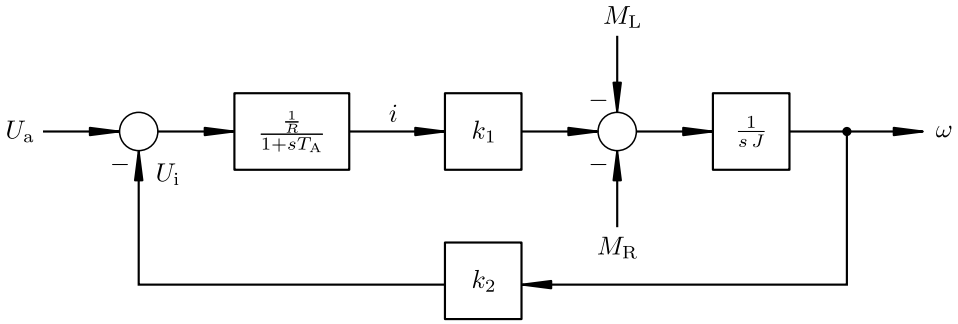


Abbildung 13: Gleichstrommotor – Blockschaltbild

3.2.2 Simulation

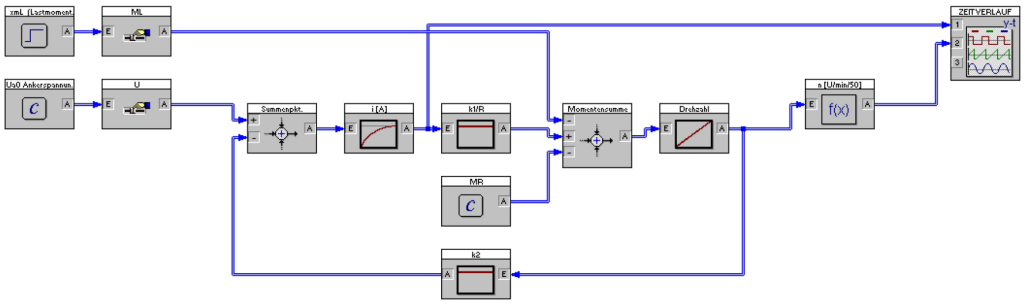
Der Motor wird mit einer Eingangspannung U_A versorgt, nach einer beliebig definierten Zeit wird dann die Last M_L dazugeschalten.

Anhand der zu portierenden Beispiele wurden folgende Kennwerte aus zusätzlichen Messungen herangezogen: $R = 0.3\Omega$, $L = 10\mathrm{mH}$, $M_\mathrm{L} = 0.971\mathrm{Nm}$, $J = 0.01\mathrm{kgm}$, $\omega_0 = 2\pi\frac{n_0}{60} = 282.7\mathrm{s}^{-1}$ und $i_0 = 5.8\mathrm{A}$. ω_0 bezeichnet die Winkelgeschwindigkeit im Leerlauf, n_0 die Leerlaufdrehzahl und i_0 den Leerlaufstrom.

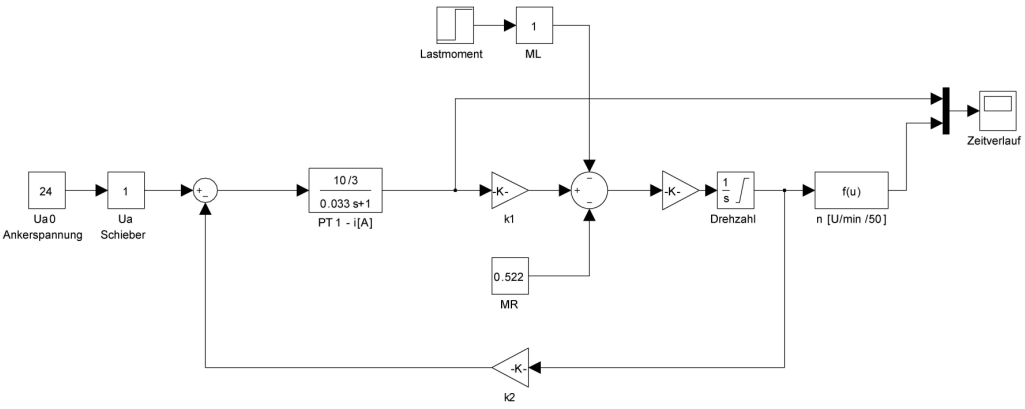
Folglich lassen sich die für die Simulation notwendigen Größen berechnen:

$$\begin{aligned} k_1 &= \frac{M_\mathrm{L}}{i_\mathrm{L} - i_0} = 0.09\frac{\mathrm{Nm}}{\mathrm{A}}\\ k_2 &= \frac{U_\mathrm{A} - R\,i_0}{\omega_0} = 0.079\mathrm{Vs}\\ M_\mathrm{R} &= k_1\,i_0 = 0.522\mathrm{Nm}\\ T_\mathrm{A} &= \frac{L}{R} = 33\mathrm{ms} \end{aligned}$$

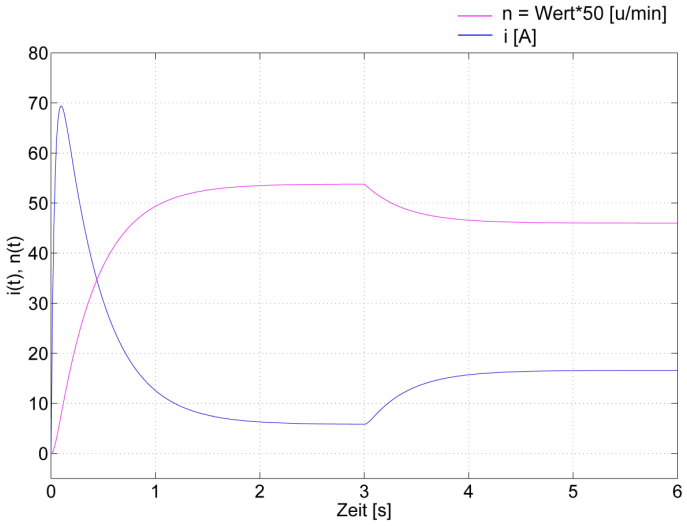
Der Gleichstrommotor wird mit einer Versorgungsspannung von $U_A = 24V$ gespeist. Nach einer einstellbaren Zeit (hier $t = 3s$) wird die Maschine mit dem Lastmoment $M_L = 0.971Nm$ belastet.



(a) WinFACT/BORIS



(b) Simulink



(c) Zeitverlauf von Strom und Drehzahl

Abbildung 14: Simulation des Gleichstrommotors

Die Umsetzung in die Simulationsmodelle, wie sie Abb. 14 wiedergibt, folgt direkt der Struktur aus dem Blockschaltbild (Bild 13). Die Führungsgröße und das Ausmaß der Lastmomentänderung kann über *Slider Gains* eingestellt werden. Der Lastmomentsprung bei $t = 3s$ erwirkt ein Absinken der Drehzahl bei gleichzeitigem Anstieg des Ankerstroms, wie es Abb. 14(c) darlegt. Am Beginn ist zudem die für Motoren typische Spitze im Einschaltstrom zu erkennen, die benötigt wird, um die Trägheit des Systems zu überwinden und den Motor auf die gewünschte Drehzahl hochzufahren.

3.2.3 Vergleich WinFACT/BORIS mit MATLAB/Simulink

Die Implementation der Aufgabe ließ sich in beiden Programmen ohne Probleme realiseren. Unterschiede bzw. Vor-/Nachteile konnten ebenfalls nicht festgestellt werden, da diese Aufgabe hauptsächlich mit Standardblöcken zu lösen ist.

3.3 Fuzzy-Regelungen

> „Zur Lösung von Problemstellungen mit unscharfen, ungenauen Aussagen kann die unscharfe Logik (Fuzzy-Logik) verwendet werden. Die Methoden der Fuzzy-Logik sind exakt. Die klassische zweiwertige (scharfe) Logik ist in der unscharfen Logik enthalten“ [11, S. 901].

Die Verfahren der Fuzzy-Regelung sind ein wichtiges Anwendungsgebiet der Automatisierungstechnik. Den Fuzzy-Reglern werden unscharfe (linguistische) Werte zugeführt. Aus einen oder mehreren Eingangsgrößen wird die Stellgröße ermittelt, die das Regelverhalten der Strecke bestimmt [11].

Es wird grundsätzlich zwischen zwei Arten von Fuzzy-Reglern unterschieden [11]:

- Relationale Fuzzy-Regler (Reglerkonzept nach Mamdani)
- Funktionale Fuzzy-Regler (Reglerkonzept nach Sugeno)

Bei relationalen Fuzzy-Reglern werden die unscharfen Schlussfolgerungen mittels Defuzzifizierungsverfahren in eine scharfe Stellgröße umgesetzt. Die funktionalen Fuzzy-Regler bilden mit scharfen Schlussfolgerungen Stellgrößenwerte als Funktionen der Eingangsgrößen. Die Funktionswerte werden mit den Erfüllungsgraden der Regeln gewichtet, das Ergebnis ist die scharfe Stellgröße [11].

3.3.1 Prinzipieller Aufbau eines relationalen Fuzzy-Reglers

Ein relationaler Fuzzy-Regler hat im Wesentlichen folgenden Aufbau [15] (s. Abb. 15):

1. **Fuzzifizierung**

   Fuzzifizierung beschreibt die Übersetzung der scharfen Eingangssignale in Wertigkeiten sprachlicher Aussagen (linguistische Wahrheitsgrade) mittels Zugehörigkeitsfunktionen, die für jede sprachliche Aussage definiert werden.

2. **Inferenz**

   Die Eingangs- und Ausgangsgrößen werden mit gewissen Regeln verbunden, die aus einem WENN-Teil (Bedingung) und einem DANN-TEIL (Schlussfolgerung) bestehen. Die Bedingungen bestehen im Allgemeinen aus zwei unscharfen Eingangswerten, die durch mathematische Operatoren (UND-, ODER-Verknüpfungen) einen bestimmten Erfüllungsgrad ergeben.

3. **Defuzzifizierung**

   Die Aufgabe der Defuzzifizierung besteht darin, aus dem Ergebnis (unscharfe Menge) der Fuzzy-Inferenz einen exakten (scharfen) Wert zu bilden. Für die Umwandlung können verschiedene Verfahren eingesetzt werden, z. B. Maximum- oder Schwerpunktsmethode.

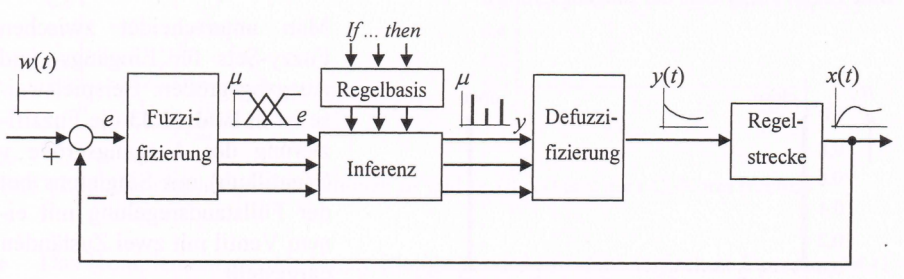


Abbildung 15: Struktur relationaler Fuzzy-Regler [15, S. 399]

3.3.2 Beispiel zur Fuzzy-Regelung

Anhand eines praktischen Simulationsbeispiels wird in Simulink ein Fuzzy-PI-Regler nachgebildet.

Um einen Vergleich zu ziehen, sind in den Abbildung 16 bzw. 17 ein Regelkreis mit Fuzzy-PI-Regler und PT_3-Strecke sowie ein Regelkreis mit PI-Regler und PT_3-Strecke realisiert worden. Mit der Fuzzy-Logic-Toolbox „Fuzzy Inference System (FIS)“ in MATLAB können für den Fuzzy-Regler Eingangs- und Ausgangswerte (Fuzzy Sets) sowie Regeln definiert werden.

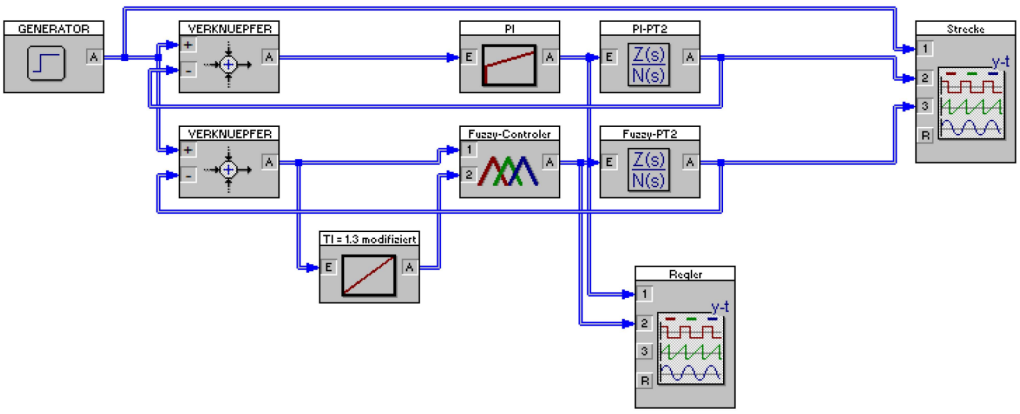


Abbildung 16: Aufbau eines Regelkreises mit Fuzzy-PI- bzw. PI-Regler (WinFACT)

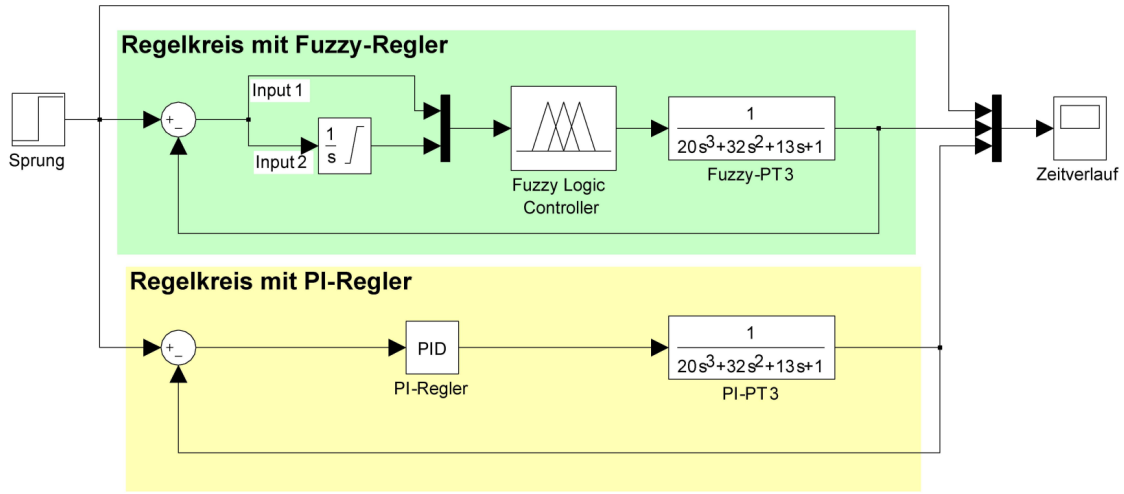


Abbildung 17: Aufbau eines Regelkreises mit Fuzzy-PI- bzw. PI-Regler (Simulink)

Die Fuzzy Sets für die Eingänge werden in Abb. 18 wiedergegeben. das Ausgangs-Fuzzy-Set ist in Abbildung 19 dargestellt.

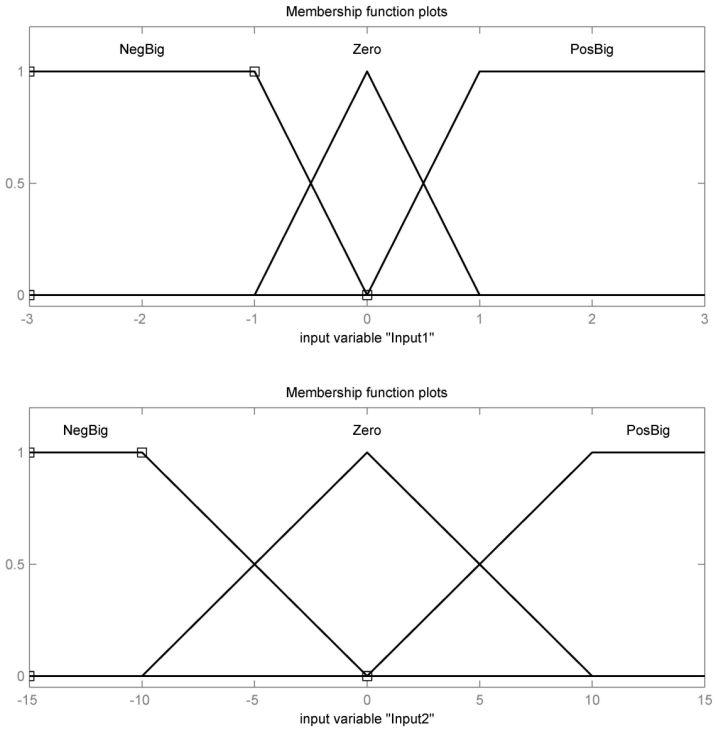


Abbildung 18: Input-Fuzzy-Sets

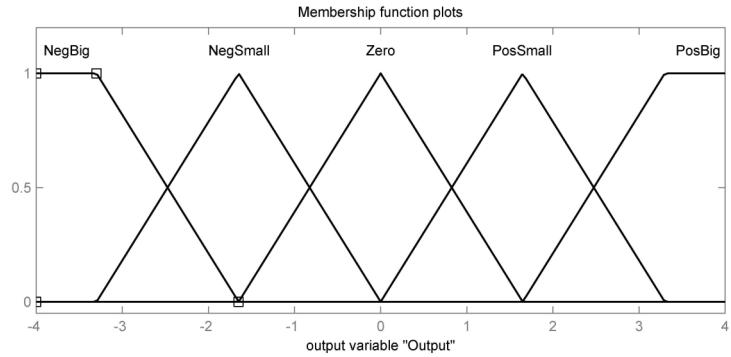


Abbildung 19: Output-Fuzzy-Set

Die Regelbasis für den relationalen Fuzzy-Regler wird folgendermaßen festgelegt:

1.	WENN	(Input1 = NegBig)	&	(Input2 = NegBig)	DANN (Output = NegBig)
2.	WENN	(Input1 = NegBig)	&	(Input2 = Zero)	DANN (Output = NegSmall)
3.	WENN	(Input1 = NegBig)	&	(Input2 = PosBig)	DANN (Output = Zero)
4.	WENN	(Input1 = Zero)	&	(Input2 = NegBig)	DANN (Output = NegSmall)
5.	WENN	(Input1 = Zero)	&	(Input2 = Zero)	DANN (Output = Zero)
6.	WENN	(Input1 = Zero)	&	(Input2 = PosBig)	DANN (Output = PosSmall)
7.	WENN	(Input1 = PosBig)	&	(Input2 = NegBig)	DANN (Output = Zero)
8.	WENN	(Input1 = PosBig)	&	(Input2 = Zero)	DANN (Output = PosSmall)
9.	WENN	(Input1 = PosBig)	&	(Input2 = PosBig)	DANN (Output = PosBig)

Die *FIS Toolbox* verfügt des weiteren über grafische Tools, die die Entwicklung vereinfachen. Sie ist eine Funktion *Surface* enthalten, die ein Fuzzy-System mit zwei Eingängen und einem Ausgang in einem 3D-Kennfeld (Abb. 20) zu visualisieren.

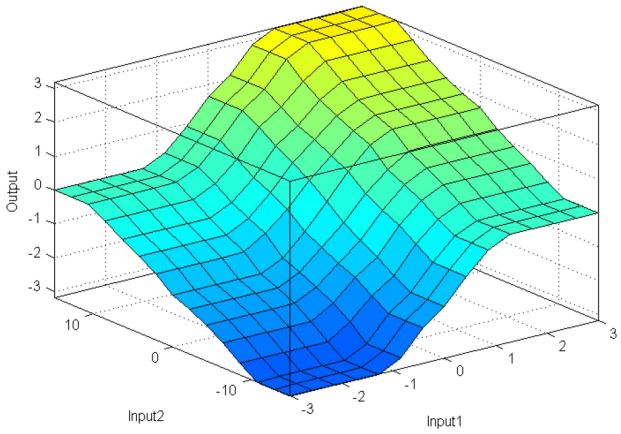


Abbildung 20: Surface

Zudem existiert eine GUI (Abb. 21), die es ermöglicht, verschieden Szenarien der Ausgangsgrößen bei Veränderung der Eingangsgrößen darzustellen (*Rule Viewer*).

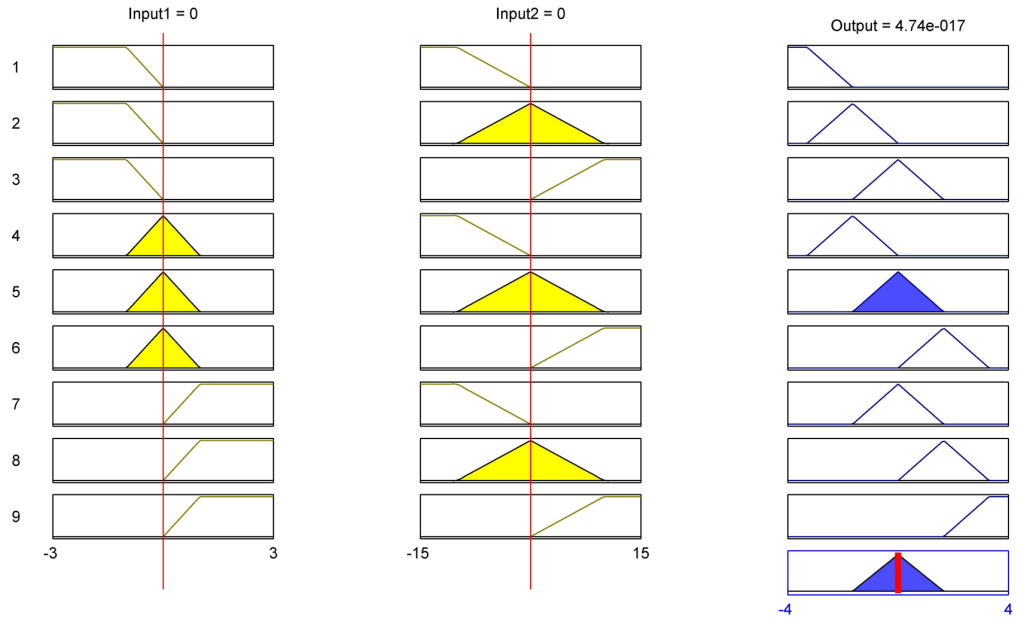


Abbildung 21: Rule Viewer

Nach Abschluss der Einstellung des Fuzzy-Reglers, ist die Simulation startbereit. In Abbildung 22 sind die Sprungantworten der beiden Regelkreise ersichtlich.

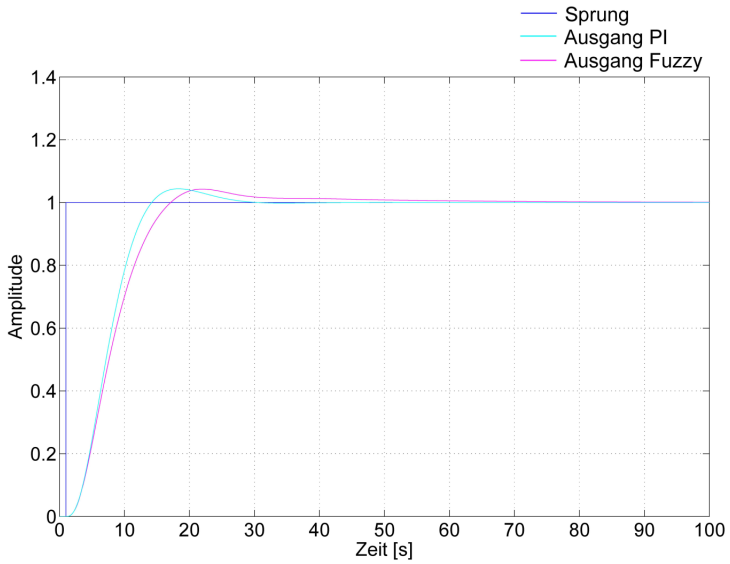


Abbildung 22: Sprungantwort PT_3-Strecke mit PI- bzw. Fuzzy-PI-Regler

3.3.3 Vergleich WinFACT/BORIS mit MATLAB/Simulink

Zwischen WinFACT und Simulink gibt es – bezogen auf das Beispiel einer Fuzzy-Regelung – außer den optischen keine weiteren Unterschiede. In beiden Programmen ist die Verwaltung der Fuzzy-Sets und der Regelbasis in ähnlicher Weise handhabbar. WinFACT/BORIS verwendet das Unterprogramm FLOP, MATLAB/Simulink die Toolbox „FIS“ für die Erstellung der Fuzzy-Sets und -Regeln.

4 Praktischer Teil – Gütemaße, PWM

4.1 Stabilitätsgüte – Integralkriterien

4.1.1 Ziel

Es soll ein Simulationsmodell erstellt werden, das diverse Integralkriterien auf einen Regelkreis anwendet. Weiters soll es in diesem Modell möglich sein, Reglerparameter systematisch zu verändern um festellen, bei welchen Werten ein Minimum für ein Gütemaß erreicht wird. Die Durchführung der Simulation mit den unterschiedlichen Parametern soll dabei automatisch erfolgen.

4.1.2 Gütemaße

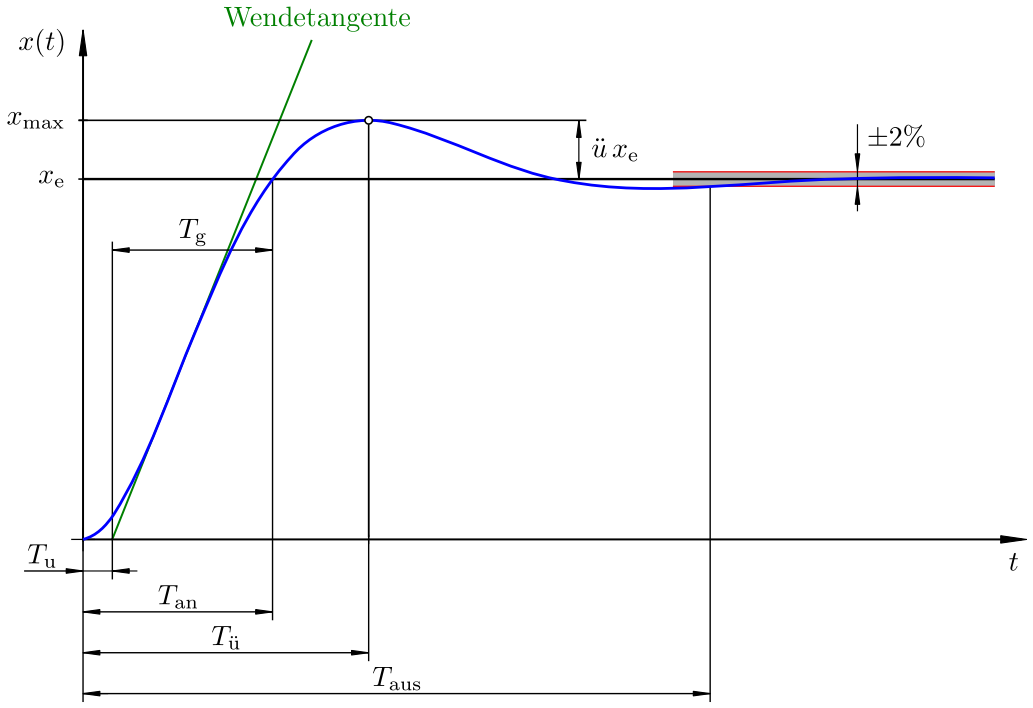


Abbildung 23: Antwort eines Regelkreises auf einen Einheitssprung (nach [5])

Um die Güte einer Regelung im Zeitbereich zu beurteilen ist es zweckmäßig, den zeitlichen Verlauf der Regelgröße $y(t)$ bzw. der Regelabweichung $e(t)$ unter Zuführung einer bekannten Eingangsgröße zu untersuchen, wobei meist der Einheitssprung $\sigma(t)$ als Testfunktion dient. Die resultierende Sprungantwort ist Grundlage zur Ermittlung diverser Kenngrößen, die Aufschluss über die Regelungsqualität geben [5]:

- Anregelzeit T_{an}: Zeit, bei der die Regelgröße $x(t)$ zum ersten Mal den Endwert x_e erreicht
- Ausregelzeit T_{aus}: Zeit, bei der der Endwert x_e endgültig innerhalb einer Abweichung $\pm 2\%$ verbleibt
- Überschwingzeit $T_{\ddot{u}}$: Zeitpunkt des Erreichens des ersten Maximums der Sprungantwort
- Überschwingweite $\ddot{u}$: Relative Abweichung der Sprungantwort vom Stationärwert beim Maximum (Gl. 16)

$$\ddot{u} = \frac{x_{max} - x_e}{x_e} \tag{16}$$

- Verzugszeit T_u: Zeit, bei der die Wendetangente des ersten Anstieges die Zeitachse schneidet
- Anstiegszeit T_g: Zeitliche Differenz zwischen den Schnittpunkten der Wendetangente mit der Zeitachse und dem Endwert x_e

Da diese Vielzahl an Kenngrößen den Vergleich zwischen unterschiedlichen Sprungantworten schwierig macht, wird es bevorzugt, die Qualität der Sprungantwort auf ein einziges Gütemaß zu beschränken.

Dafür werden Integralkriterien verwendet, die auf den zeitlichen Verlauf der Regelabweichung $e(t)$ angewendet werden. Im allgemeinen Fall lautet die Formel für ein Integralkriterium [19]:

$$I_k = \int_0^\infty f_k\left[e(t)\right]\,dt \tag{17}$$

wobei für $f_k\left[e(t)\right]$ u. a. Funktionen aus Tab. 4 zum Einsatz kommen können.

Bezeichnung	Gütemaß
IAE (Integral of Absolute value of Error)	$\int_0^\infty \lvert e(t)\rvert\,dt$
ISE (Integral of Squared value of Error)	$\int_0^\infty e^2(t)\,dt$
ITAE (Integral of Time multiplied Absolute value of Error)	$\int_0^\infty \lvert e(t)\rvert\,t\,dt$
ITSE (Integral of Time multiplied Squared value of Error)	$\int_0^\infty e^2(t)t\,dt$
ISTAE (Integral of Squared Time multiplied Absolute value of Error)	$\int_0^\infty \lvert e(t)\rvert\,t^2\,dt$
ISTSE (Integral of Squared Time multiplied Squared value of Error)	$\int_0^\infty e^2(t)t^2\,dt$

Tabelle 4: Integralkriterien

Die Qualität einer Regelung ist umso besser, je kleiner der Wert vom jeweiligen Kriterium I_k ist. In nachfolgendem Simulationsbeispiel werden diverse Integralkriterien auf ein Modell einer Regelung in MATLAB/Simulink angewendet.

4.1.3 Erstellung des Models

In dieser Simulation in MATLAB/Simulink wird die Regelung einer Strecke $F_\mathrm{S}(s)$ (Gl. 18) mit einem PID-Regler (Gl. 19) betrachtet.

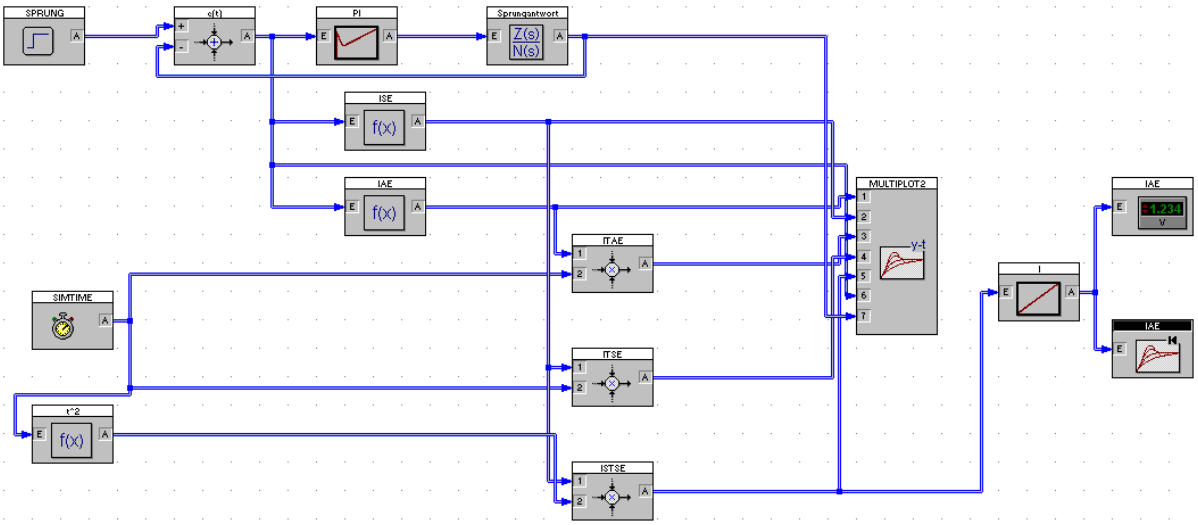
$$F_\mathrm{S}(s) = \frac{1}{20s^3 + 32s^2 + 13s + 1} \tag{18}$$

$$F_\mathrm{PID}(s) = k_\mathrm{R}\left(1 + \frac{1}{sT_\mathrm{N}} + sT_\mathrm{V}\right) \tag{19}$$

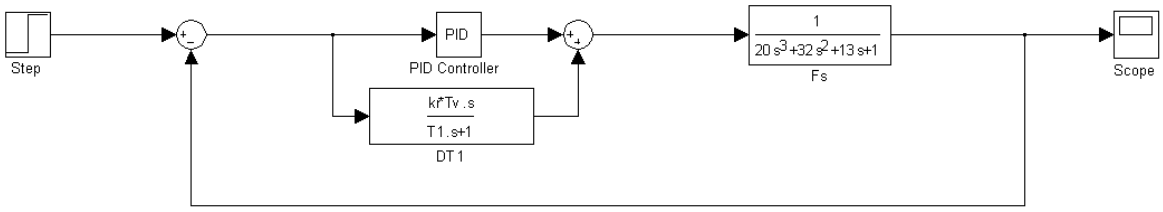
Da der ideale PID-Regler praktisch nicht realisierbar ist (das ideale D-Verhalten kann nicht erreicht werden) wird in dieser Simulation der so genannte reale PID-Regler, der durch ein $\mathrm{PIDT_1}$-System beschrieben wird, verwendet [19].

$$F_{\mathrm{PIDT_1}}(s) = k_\mathrm{R}\left(1 + \frac{1}{sT_\mathrm{N}} + \frac{sT_\mathrm{V}}{1+sT_1}\right) \tag{20}$$

In den Abbildungen 24(b) (Simulink) und 24(a) sind die jeweiligen Simulationsmodelle dargestellt.



(a) WinFACT



(b) Simulink

Abbildung 24: Modelle zur Bestimmung der Integral-Gütemaße

Das in WinFACT/BORIS erstellte Modell weist eine komplexere Struktur auf, weil hier das Gütemaß bereits während der Simulation berechnet wird. In MATLAB/Simulink hingegen wird diese Berechnung erst nach der Simulation, in MATLAB-Code, durchgeführt.

Der von MATLAB zur Verfügung gestellte PID Regler wird duch folgende Übertragungsfunktion charakterisiert:

$$F_{\mathrm{PID\,M}}(s) = P + \frac{I}{s} + D\,s \tag{21}$$

Um mit der oben beschriebenen Übertragungsfunktion des $\mathrm{PIDT_1}$-Reglers arbeiten zu können, ist ein Koeffizientenvergleich durchzuführen. Die Umrechnungsformeln für die Standardform:

$$P = k_\mathrm{R} \tag{22}$$

$$I = \frac{k_\mathrm{R}}{T_\mathrm{N}} \tag{23}$$

$$D = 0 \tag{24}$$

Anstelle des D-Gliedes des PID-Reglers wird ein $\mathrm{DT_1}$-Element mit nachfolgender Übertragungsfunktion parallel zum Regler geschaltet:

$$F_{\mathrm{DT_1}}(s) = \frac{k_\mathrm{R}\, T_\mathrm{V}}{1 + sT_1} \tag{25}$$

Bis zu diesem Zeitpunkt werden im Modell keine konkreten Zahlenwerte für k_R, T_N, T_V und T_1 gesetzt. Im nachfolgenden Abschnitt geklärt, wie diese Parameter Werte erhalten.

4.1.4 Änderung der Modellparameter

Das Ziel dieser Simulation ist nun, durch Variation der Parameter k_R, T_N und T_V das Minimum für ein Integralkriterium zu berechnen, T_1 wird konstant gehalten.

Um mit Simulink eine Simulation mit veränderbaren Parametern durchführen zu können, werden im Simulationsmodell als Werte nur die Variablen k_R, T_N, T_V und T_1 angegeben (siehe Abschnitt 4.1.3). Dieses Modell wird nun von MATLAB aufgerufen, wobei entsprechende Werte für die Variablen übergeben werden.

Da für die Berechnung des Minimums die Simulation mit je n verschiedenen Werten für k_R, T_N und T_V durchgeführt werden muss, wurde ein m-Skript erstellt. In diesem Skript wird das Modell n-mal mit jeweils unterschiedlichen Werten für die Parameter aufgerufen.

In folgendem Codebeispiel wird die Simulation n-mal durchgeführt und dabei der Parameter T_N verändert:

```
for j = 1 : steps
    sim('models/PT3_PIDT1');     % Simulation starten

    Tn = Tn + Tn_step_size;
end
```

Für die Parameter k_R, T_N und T_V wird jeweils ein *Startwert* bzw. *Endwert* angegeben. Das Eingabefeld *Schritte* definiert die Anzahl der Schritte n zwischen Startwert und Endwert. Der Parameter T_1 wird während der Simulation nicht verändert, da er auch bei realen Reglern fest vorgegeben ist [5].

Da die Zahl der veränderbaren Werte 3 ist (k_R, T_N, T_V), muss die Simulation insgesamt n^3 Zyklen durchlaufen. Dies kann bereits bei einer relativ kleinen Schrittzahl zu einer sehr langen Simulationsdauer führen. Beispielsweise sind für $n = 10$ Schritte $n^3 = 1000$ Zyklen durchzuführen.

Für ein Testsystem ergeben sich folgende Vergleichswerte:

- Gesamtdauer in MATLAB: *ca. 5 Minuten*
- Gesamtdauer in WinFACT: *ca. 75 Minuten*

Die Simulationsparameter sind für beide Programm identisch:

- Simulationsdauer: *100s*
- Simulationsschrittweite: *0.01s*

Auch der Wertebereich der Reglerparameter ist in den Testfällen gleich:

- $k_\mathrm{R} = 1 \ldots 5$
- $T_\mathrm{N} = 8 \ldots 13$
- $T_\mathrm{V} = 3 \ldots 8$
- $T_1 = 5$ (für MATLAB; in WinFACT wird der Standard-PID-Regler verwendet)

Diese Werte bilden auch die Grundlage für die in den nachfolgenden Abschnitten beschriebenen Ergebnisse.

Ist die Simulation beendet, so stehen die resultierenden Daten für die weitere Verarbeitung zur Verfügung. Durch Auswahl eines Gütekriteriums auf der GUI-Oberfläche (siehe nächster Abschnitt) wird das entsprechende Kriterium auf die Ergebnisdaten angewendet und grafisch dargestellt.

4.1.5 Graphical User Interface (GUI)

Damit der Benutzer der Simulation die Simulationsparameter nicht umständlich im m-File ändern muss, wurde eine grafische Benutzerschnittstelle (*Graphical user Interface*, GUI) gestaltet.

Durch Eintippen des Befehls `guide` im MATLAB-Command-Window öffnet sich ein Softwaretool zum Erstellen interaktiver Benutzeroberflächen.

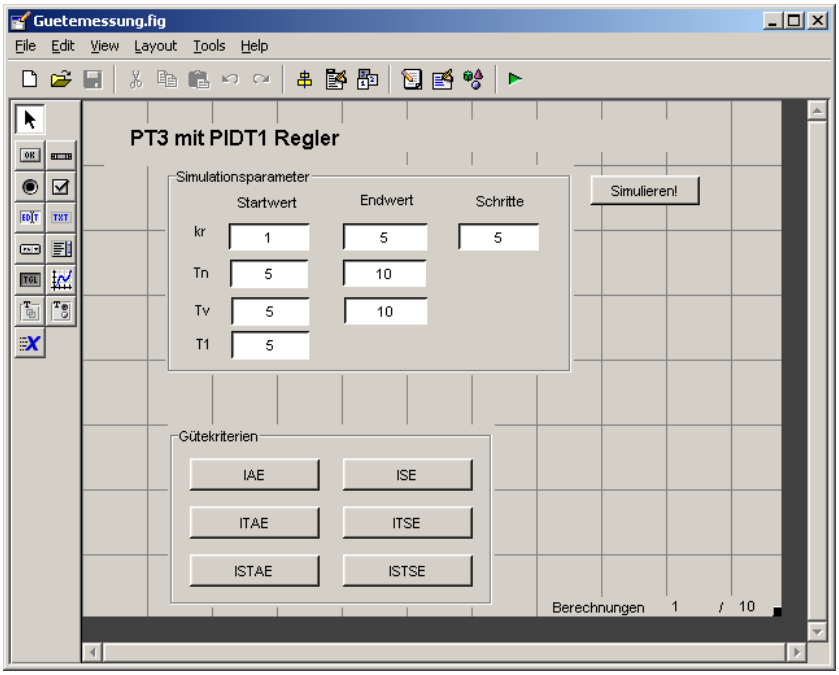


Abbildung 25: Software-Tool *GUIDE* in MATLAB

Die diversen Buttons erhalten ihre Funktionen, indem man ihnen ein m-Skript zuweist. Von diesen Skripts kann auch auf Eingabefelder der Benutzeroberfläche zugegriffen werden. Die fertiggestellte GUI-Oberfläche zeigt Abb. 26.

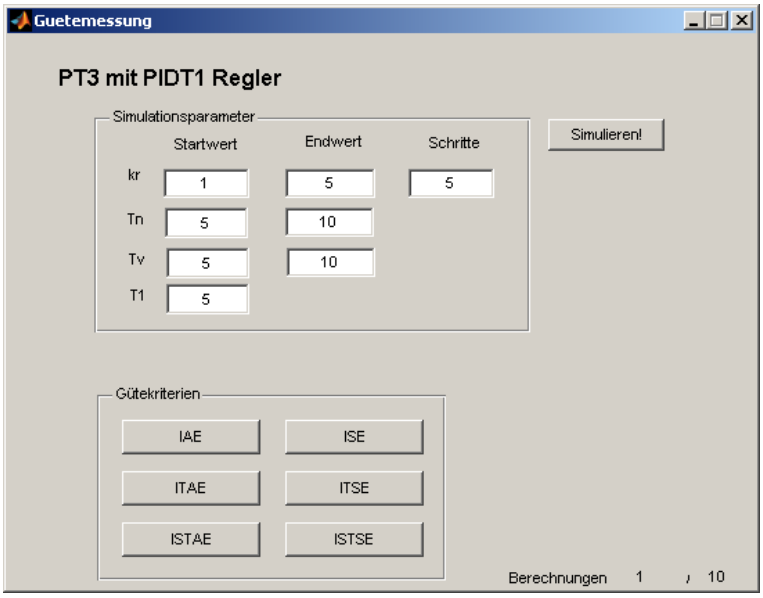


Abbildung 26: Grafische Schnittstelle für die Simulation

4.1.6 Auswertung der Daten

Nach jeder Simulation werden folgende Zeitverläufe in einer Matrix mit vier Spalten ausgegeben:

- Einheitssprung
- Sprungantwort
- Regelabweichung
- Simulationszeit

Aus dieser Matrix wird der Zeitverlauf der Regelabweichung entnommen und in einer Matrix *error* gespeichert. Diese Matrix enthält nach Abschluss aller Simulationen n^3 Zeitverläufe der Regelabweichung. Parallel dazu werden auch die aktuellen Werte der Parameter k_R, T_N und T_V in den Vektoren *kr_vec*, *Tn_vec* und *Tv_vec* abgespeichert.

Diese Werte können anschließend dem entsprechenden Verlauf der Regelabweichung zugewiesen werden:

$$\mathrm{kr\_vec}[k], \mathrm{Tn\_vec}[k], \mathrm{Tv\_vec}[k] \rightarrow \mathrm{error}[k]$$

Die Parameter an der k-ten Stelle des Vektors rufen also die Regelabweichung an der k-ten Stelle der Matrix *error* hervor.

Nun wird auf jeden einzelnen Regelabweichungsverlauf der Matrix error das Integralkriterium angewendet und anschließend der Minimalwert und dessen Position in der Matrix ermittelt. Durch die Position kann auf die Reglerparameter k_R, T_N und T_V rückgeschlossen werden.

4.1.7 Berechnung des Minimums

Hier wird nun beispielhaft die Berechnung eines Gütekriteriums durchgeführt, im konkreten Fall behandeln wir das IAE-Kriterium (Integral of Absolute value of Error). Es liegen n Zeitverläufe der Regelabweichung in einer $(m \times n)$-Matrix vor, wobei jede Spalte einem Zeitverlauf entspricht. In einer while-Schleife wird je Durchlauf eine Spalte dieser Matrix selektiert und mit den ausgewählten Daten die Berechnung durchgeführt.

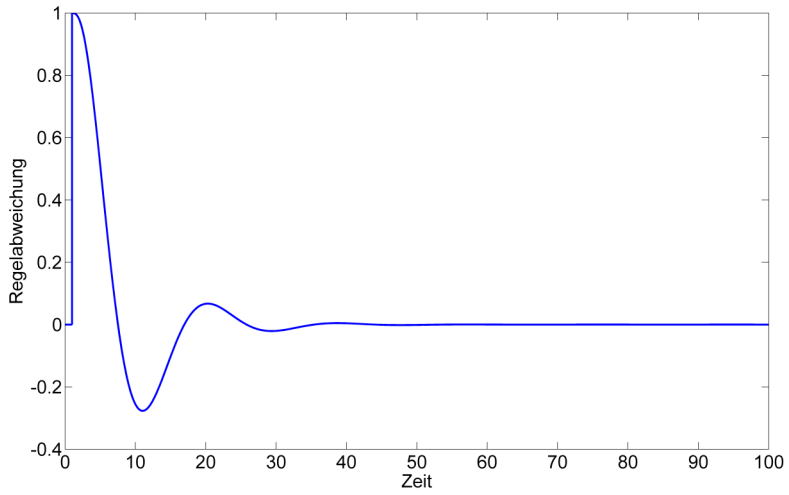


Abbildung 27: Zeitverlauf der Regelabweichung

Zuerst werden die Beträge (*absolute value*) aller Punkte der Regelabweichung gebildet. Dies geschieht in MATLAB mit der Funktion `abs()`:

```
IAE =  abs(error(:,k));
```

Der Doppelpunkt bedeutet, dass alle Daten der Spalte k ausgewählt werden. Nach der Bildung der Beträge erhält man einen Zeitverlauf nach Abb. 28.

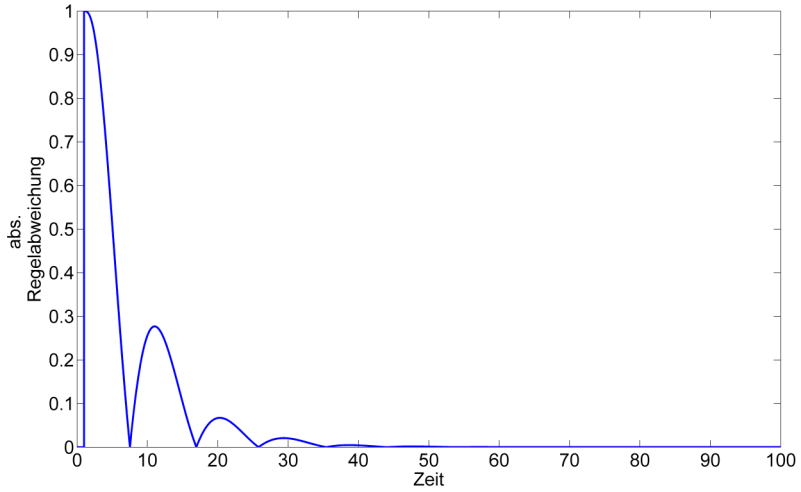


Abbildung 28: Zeitverlauf des Betrags der Regelabweichung

Anschließend wird die Fläche (Integral) des resultierenden Zeitverlaufs mittels dem Integral nach Riemann gebildet. Die Summe wird mit der Matlab Funktion `sum()` errechnet – sie addiert alle Werte eines Vektors.

```
IAE = sum(IAE*dt);
```

Die Variable *dt* stellt dabei die Simulationsschrittweite dar.

In *IAE* steht nun der skalare Wert des Gütekriteriums und wird an eine Matrix *IAE_temp* angehängt:

```
IAE_temp = [IAE_temp;IAE];
```

Wie bereits erwähnt, werden diese Schritte bei jedem Durchlauf der while-Schleife wiederholt, es stehen also anschließend insgesamt n Werte in *IAE_temp*. Durch die von MATLAB zur Verfügung gestellte Funktion `min()` können nun der kleinste Wert aller Gütekriterien sowie der zugehörige Index herausgefunden werden:

```
[min_guete index] = min(IAE_temp);
```

In *min_guete* steht nun der beste (= kleinste) Wert des Gütemaßes, durch die Variable index können nun die Werte für k_R, T_N und T_V aus den Vektoren *kr_vec*[*index*], *Tn_vec*[*index*], *Tv_vec*[*index*] bestimmt werden.

In Simulink liefert die beste Berechnung ein $IAE = 4.64$, wobei dies für folgende Parameter gilt:

- $k_\mathrm{R} = 3.4$
- $T_\mathrm{N} = 11.5$
- $T_\mathrm{V} = 3$
- $T_1 = 5$ (vorab fixiert)

Im Vergleich dazu erreicht WinFACT/BORIS einen Wert von $IAE = 4.21$ mit dem Paramtersatz:

- $k_\mathrm{R} = 3.2$
- $T_\mathrm{N} = 11.2$
- $T_\mathrm{V} = 3$

4.1.8 Grafische Darstellung

Um den Einfluss der Regelparameter auf die Regelabweichung optisch darzustellen, wird eine dreidimensionale Grafik erstellt. Der Wert des Gütekriteriums wird in Abhängigkeit von 2 änderbaren Werten k_R und T_N in einem 3D-Modell dargestellt. Die Visualisierung wird mit der MATLAB-Funktion `surface(X,Y,Z)` erstellt, wobei die k_R- bzw. T_N-Werte die X- und Y-Matrizen und Z eine Matrix der berechneten IAE-Werte darstellen.

Zuerst müssen also die Vektoren *kr_vec* und *Tn_vec* so aufbereitet werden, dass Sie an die Funktion `surface()` übergeben werden können. Das ist mittels der Anweisung `meshgrid()` zu bewerkstelligen [13]:

```
[X,Y] = meshgrid(kr_vec, Tn_vec);
```

Aus den Vektoren

$$kr = \begin{pmatrix} 1.0 & 1.8 & 2.6 & 3.4 & 4.2 \end{pmatrix}$$

$$Tn = \begin{pmatrix} 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

enstehen die Matrizen

$$X = \begin{pmatrix} 1.0 & 1.8 & 2.6 & 3.4 & 4.2 \\ 1.0 & 1.8 & 2.6 & 3.4 & 4.2 \\ 1.0 & 1.8 & 2.6 & 3.4 & 4.2 \\ 1.0 & 1.8 & 2.6 & 3.4 & 4.2 \\ 1.0 & 1.8 & 2.6 & 3.4 & 4.2 \end{pmatrix}$$

$$Y = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 \\ 7 & 7 & 7 & 7 & 7 \\ 8 & 8 & 8 & 8 & 8 \\ 9 & 9 & 9 & 9 & 9 \end{pmatrix}$$

Nun muss noch eine Matrix *IAE_plot* mit IAE-Werten erstellt werden, die den jeweiligen X und Y (bzw. kr und Tn) Werte zuweisen. Diese Matrix wurde bereits bei der Berechnung der IAE-Kriterien im Schleifendurchlauf erstellt und sieht folgendermaßen aus:

$$IAE\_plot = \begin{pmatrix} 10.75 & 9.2 & 8.91 & 9.2 & 9.92 \\ 10.11 & 8.18 & 7.68 & 7.76 & 8.18 \\ 9.76 & 7.54 & 6.96 & 6.95 & 7.25 \\ 9.6 & 7.1 & 6.48 & 6.44 & 6.67 \\ 9.62 & 6.78 & 6.14 & 6.08 & 6.27 \end{pmatrix}$$

Schlussendlich wird die Grafik mit nachfolgendem Befehl erstellt, das Ergebnis zeigt Abb. 29.

```
surface(X,Y,IAE_plot)
```

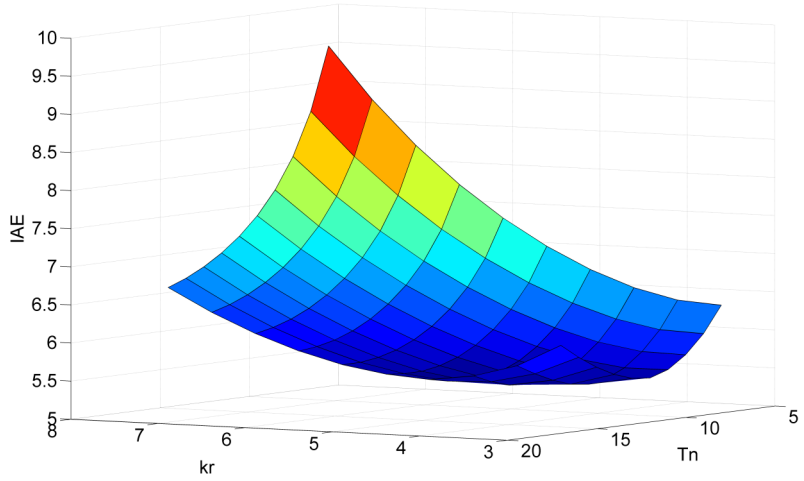


Abbildung 29: IAE in Abhängigkeit von k_R und T_N

4.1.9 Vergleich WinFACT/BORIS mit MATLAB/Simulink

Der Aufbau des Simulationsmodells ist in beiden Programmen, WinFACT/BORIS und MATLAB/Simulink sehr ähnlich, da jeweils auf das Datenflussprinzip aufgebaut wird. Das Datenflussmodell beruht darauf, dass ein Funktionsblock nur dann ausgeführt wird, wenn an jedem Eingang ein gültiges Datum anliegt. Idealerweise wird ein Modell so entworfen, dass die Daten von links nach rechts fließen. Eine Simulation durchzuführen, bei der sich die Parameter mit jedem Durchlauf verändern, ist jedoch in Simulink wesentlich aufwändiger als in WinFACT.

In der Software von Mathworks muss nämlich ein eigenes Skript geschrieben werden, das das Modell wiederholt aufruft und dabei die Parameter verändert. Um für die Änderung der Start- und Stoppwerte der Parameter nicht in das Skript eingreifen zu müssen, ist es erforderlich, eine Grafische Benutzerschnittstelle zu erstellen, was einen weiteren Aufwand darstellt. Ein Vorteil

besteht vorrangig für all jene, die das MATLAB-Skripting beherrschen und somit eine sehr flexible Simulationsumgebung erzeugen können.

Im Programm des Ingenieurbüros Dr. Kahlert hingegen gibt es dazu eine eingebaute Funktion, in der lediglich die Start- und Endwerte der Parameter sowie die Anzahl der Schritte vom eingegeben werden müssen. Danach werden die Simulationen automatisch ausgeführt und die gewünschten Signale in Abhängigkeit der Parameter auf einem *Mehrfachplot* ausgegeben. Dies ist zwar sehr praktisch, wenn man schnell zu Simulationsergebnissen kommen will, jedoch ist man eingeschränkt, wenn man zum Beispiel Ergebnisse benutzerdefiniert ausgeben möchte.

Ein weiterer, ausschlaggebender Punkt ist die Dauer der Simulation. Bei exakt demselben Beispiel mit 1000 Simulationen benötigt das in dieser Arbeit entwickelte Simulationstool für die MATLAB-Umgebung etwa 5, das in WinFACT vorhandene Pendant jedoch über 70 Minuten. Ein wesentlicher Teil dieses Leistungsvorsprungs beruht dabei auf dem mathematischen Kernel von MATLAB, welcher primär auf numerische Operationen und damit die Matrizen-Verarbeitung ausgelegt ist.

4.2 Sinusgeführte PWM

4.2.1 Ziel

Das Ziel ist es, eine Simulation einer sinusgeführten Pulsbreitenmodulation (engl. Pulse Width Modulation, PWM) durchzuführen. Das Hauptaugenemerk wird darauf gelegt, welche Signale dieses Steuersignal auf einer Last erzeugt. Diese Lastsignale werden sowohl im Zeit-, als auch im Frequenzbereich untersucht.

4.2.2 Prinzip

Das Prinzip der sinusgeführten PWM beruht auf dem Vergleich einer Referenzspannung (grüne Kurve in Abbildung 30), von der die Lastspannung abgeleitet wird, mit einer Dreieckspannung (blaue Kurve). Das Ergebnis dieses Vergleichs ist eine pulsbreitenmodulierte Rechteckspannung (roter Verlauf), mit der die Last gesteuert wird.

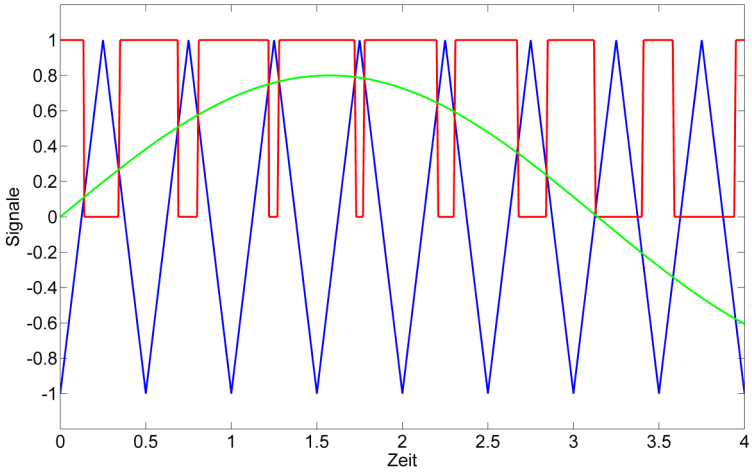


Abbildung 30: Zeitverläufe der Eingangs- und Ausgangssignale

4.2.3 Signalerzeugung

Zu Beginn werden die beiden Eingangssignale, die Referenzsinuskurve und das Dreieckssignal, erzeugt.

Das Sinussignal wird durch den Funktionsblock *Sine Wave* erzeugt, in dem Signalamplitude und -frequenz konfiguriert werden können.

Das Dreiecksignal wird durch einen Simulink-Block *Repeating Sequence* generiert. Hierbei werden 2 Vektoren übergegeben, die die Zeit und den entsprechenden Signalwert angeben. Diese Signalsequenz wird wiederholt von diesem Block ausgegeben. Für das Dreiecksignal wurden folgende Werte übergeben:

```
[0 0.25 0.5]   % Zeit
[-1 1 -1]      % Signal
```

Diese Werte führen auf das blaue Signal in Abbildung 30. Es ist deutlich zu erkennen, dass beim Zeitwert 0.5 der Signalwert -1 vorliegt.

4.2.4 Pulsgenerierung

Diese beiden Signale werden nun durch einen Komparator miteinander verglichen. Ist der Wert des Refernzsinus größer als jener der Dreieckspannung, wird das Ausgangssignal auf High gesetzt, andernfalls auf Low.

Da in Simulink Signale nicht direkt miteinander an einem Komparator verglichen werden können, wird der Simulationsblock *Compare to Zero* verwendet. Der Wert des Sinussignals wird daher vom Wert des Dreiecksignals subtrahiert, und anschließend mit 0 verglichen. Das Ergebnis ist das selbe pulsbreitenmodulierte Rechtecksignal, welches durch einen direkten Vergleich der Signale erzeugt werden würde. Die rote Kurve in Abbildung 30 stellt dieses Ausgangssignal dar.

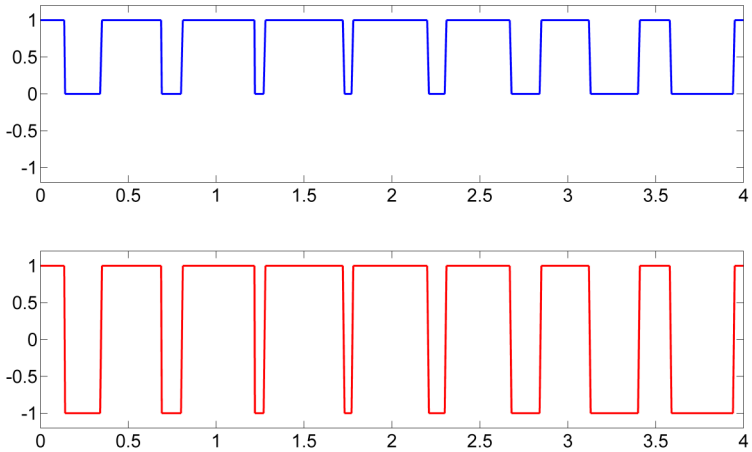


Abbildung 31: PWM-Ausgangssignal vor und nach der Entfernung des Gleichanteils

Zu diesem Zeitpunkt kann das Ausgangssignal nur positive Werte annehmen. Um den Referenzsinus, der auch negative Werte darstellt, am Ausgang korrekt darzustellen, wird aus dem Rechtecksignal der Gleichanteil entfernt. Dies geschieht, indem das Signal mit dem Wert 2 multipliziert wird und anschließend um 1 verringert wird (das Ergebnis zeigt Abb. 31).

Um zu sehen, welchen Laststrom diese Steuerspannung an einer Last bewirkt, wird es anschließend an ein PT_1-Element geführt. Dieses PT_1-System könnte einen Motor in einfachster Näherung approximieren (Gl. 26).

$$G_{\text{Last}}(s) = \frac{1}{1+s} \tag{26}$$

Durch das Tiefpassverhalten dieses Elements werden die höherfrequenten Anteile des Steuersignals herausgefiltert. Aus dem entstandenen Signal leitet sich der Laststrom ab. Am Ausgang finden wir nun die Signale aus Abb. 32 vor.

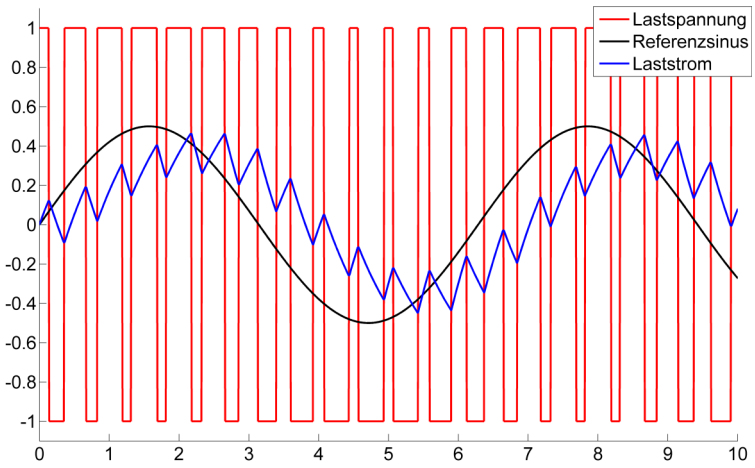
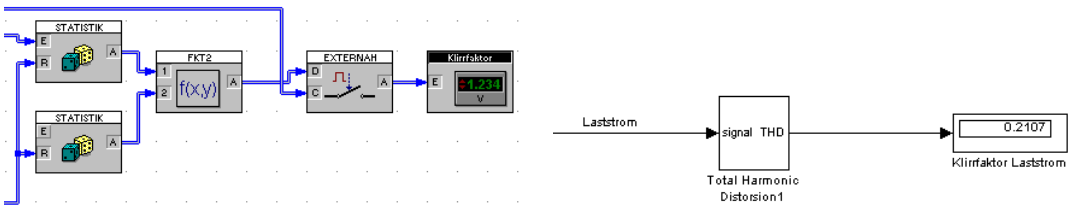


Abbildung 32: Lastsignale der PWM

In Abbildung 32 ist sehr gut zu erkennen, dass der Laststrom im Wesentlichen dem Referenzsignal entspricht. Auch die durch die Last hervorgerufene Phasenverschiebung des Stromes ist einzusehen. In der grünen Kurve sind noch immer „Unebenheiten“ vorhanden, die höherfrequente Signalanteile erzeugen. Um diese Anteile besser beurteilen zu können, werden nun der *Klirrfaktor* sowie die *Fast Fourier Transform* (FFT) des Laststromes berechnet.

4.2.5 Klirrfaktor

Der Klirrfaktor wird in Simulink durch den Funktionsblock *Total Harmonic Distortion* berechnet. Am Eingang des Blockes liegt das zu untersuchende Signal an, in den Konfigurationsparametern des Blockes wird die Frequenz der Grundschwingung eingegeben, welche in unserem Fall die Frequenz des Referenzsinus ist. Nach Ende der Simulation liegt am Ausgang ein einziger Wert vor, der Klirrfaktor.



(a) WinFACT (b) Simulink

Abbildung 33: Klirrfaktorberechnung – Modelle

In WinFACT ist die Berechnung des Klirrfaktors etwas aufwändiger, da kein fertiger Simulationsblock dafür vorliegt. Es müssen hier die Effektivwerte der Grund- bzw. Oberschwingungen manuell berechnet werden. Die Frequenz des Referenzsinus beträgt $f = 1\frac{\text{rad}}{\text{s}}$, die Amplitude 50% vom Maximalwert des Rechtecksignals. Die Frequenz des Dreiecksignales beträgt $f = 2\text{Hz}$.

Die Ergebnisse:

- WinFACT/BORIS: $k = 28.8\%$
- MATLAB/Simulink: $k = 21.07\%$

4.2.6 Fast Fourier Transform

Zur Berechnung der FFT stellt MATLAB/Simulink unter anderem die Funktion *Magnitude FFT* zur Verfügung, die den Betrag der Amplitude im Frequenzbereich ausgibt. Es gibt für diese Berechnung noch weitere Methoden in MATLAB, die z. B. den Imaginär- und Realteil getrennt ermitteln. In dieser Simulation wird jedoch nur auf die Magnitude FFT zurückgegriffen.

Diese Funktion berechnet das kontinuierliche Spektrum von n Signalpunkten. Da nicht exakt angegeben werden kann, wie viele Simulationspunkte insgesamt in der Simulation berechnet werden, wird ein Signalabschnitt von 2048 Werten in einem *Buffer* zwischengespeichert, auf den die FFT angewendet wird. Die Größe des Buffers wird idealerweise als Potenz von 2 angegeben, da in diesem Fall die Berechnung der FFT beschleunigt wird [13, 21].

Durch das *Zero-Order-Hold*-Glied wird das Signal, in diesem Fall der Laststrom, abgetastet und der aktuelle Wert in den Buffer geschrieben. Sobald dieser Buffer vollständig gefüllt ist, werden die 2048 Werte an den FFT-Block übergeben und das Frequenzspektrum berechnet.

Da die Simulationsschrittweite 0.01s beträgt, entspricht ein Teilsignal von 2048 Punkten ca. 20s. Die Frequenz des Referenzsignales ist 1 $\frac{\text{rad}}{\text{s}}$, was eine Periodendauer von ca. 6s bedeutet. Durch das Zeitfenster von 20s werden für die Berechnung also 3 komplette Sinusschwingungen in Betracht gezogen.

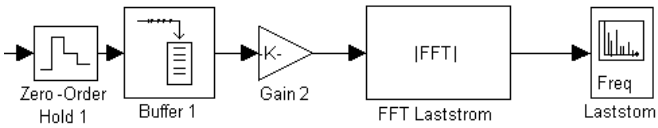


Abbildung 34: FFT-Berechnung in Simulink

Das *Gain*-Glied mit der Verstärkung $K = \frac{1}{\text{Bufferlänge}} = \frac{1}{2048}$ bewirkt eine Normierung der Ergebniswerte der FFT. Die Gleichung, die dem verwendeten Simulink-FFT-Block zugrunde liegt, lautet für eine diskrete Transformation mit N Abtastpunkten [13]:

$$X(k) = \sum_{j=1}^{N} x(j)\, \omega_n^{(j-1)(k-1)} \tag{27}$$

mit

$$\omega_n = e^{\frac{-2\pi \mathrm{i}}{N}} \tag{28}$$

Durch diese Normierung werden die Werte im Frequenzspektrum im Bereich der Amplitude des Eingangssignales angezeigt. Anschließend wird das Frequenzspektrum durch ein *Vector Scope* mit dem Titel *Laststrom* ausgegeben.

In WinFACT/BORIS erfolgt die Berechnung der FFT etwas einfacher, da nur ein einziger Funktionsblock benotigt wird (s. Abb. 36).

Die Auflösung im Frequenzbereich kann in beiden Programmen festgelegt werden. In WinFACT berechnen sich die kleinste darstellbare Frequenz und die Bandbreite aus

$$\omega_{\min} = \frac{2\,\pi}{T} \tag{29}$$

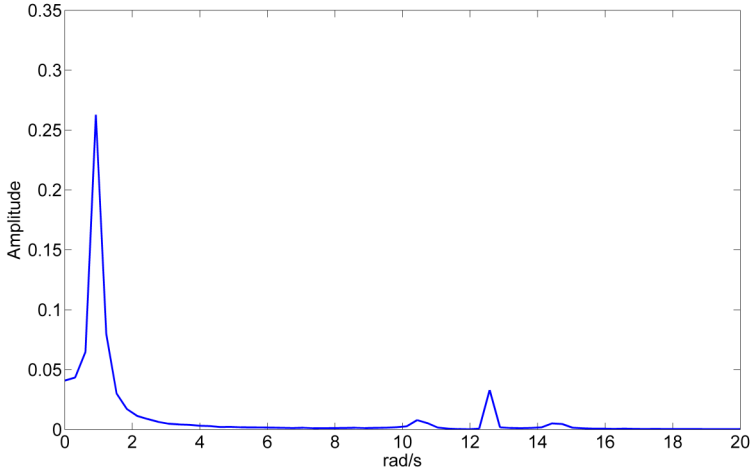


Abbildung 35: Frequenzspektrum, Grundschwingung bei 1 $\frac{\text{rad}}{\text{s}}$

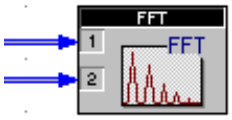


Abbildung 36: Funktionsblock zur Berechnung der FFT in WinFACT/BORIS

und

$$\omega_{\max} = \left(\frac{n}{2} - 1\right) \omega_{\min} \tag{30}$$

wobei n die Stützpunkte und T das Zeitfenster ist [9]. Diese beiden Parameter sind frei wählbar.

In MATLAB/Simulink errechnet sich die Frequenzauflösung zu

$$\Delta f = \frac{1}{N\,T_{\text{a}}} \tag{31}$$

Die Bandbreite B ergibt sich ähnlich wie bei WinFACT:

$$B = \frac{N}{2}\Delta f \tag{32}$$

Hier stellt N wiederum die Anzahl der Stützpunkte, T_{a} die Abtastzeit dar.

4.2.7 Vergleich WinFACT/BORIS mit MATLAB/Simulink

Bei diesem Beispiel sind die Unterschiede zwischen MATLAB/Simulink und WinFACT/BORIS gering. Betrachtet man die Erzeugung der Signale, so gibt es in beiden Programmen entsprechende Signalquellen, mit denen beliebige Signale erzeugt werden können.

Das erste wesentliche Unterscheidungsmerkmal macht sich beim Vergleich der beiden Signale durch einen Komparator bemerkbar. Da in Simulink die beiden Signale nicht direkt miteinander vergleichbar sind, muss ein Umweg mit einer Compare to Zero Funktion gewählt werden (siehe Abschnitt 4.2.4).

Ein weiterer Punkt, der einen gewissen Aufwand für die Portierung schafft, ist die Signalanalyse im Frequenzbereich mittels FFT. In WinFACT/BORIS ist dafür nur ein einziger Block erforderlich, in Simulink werden insgesamt 4 Blöcke benötigt.

In BORIS werden im Funktionsblock FFT die Berechnung und die Visualisierung durchgeführt. Parameter wie zum Beispiel das Zeitfenster und die Anzahl der Stützpunkte für das Frequenzspektrum werden in diesem Block konfiguriert. In Simulink hingegen erfolgt die Festlegung des Zeitfensters durch die beiden Funktionen Zero Order Hold und Buffer. Die Anzahl der Stützpunkte wird im Funktionsblock FFT „Laststrom“ angegeben und die Darstellung der Daten erfolgt wiederum separat mit dem Vector Scope.

Die Simulation in MATLAB/Simulink erfordert zwar mehr Einarbeitungszeit, schlussendlich ist man jedoch auch in diesem Beispiel wesentlich flexibler, was die benutzerdefinierten Berechnungen anbelangt.

5 Praktischer Teil – Erweiterte Strukturen, Visualisierung, Digitale Regelungstechnik

In diesem Kapitel werden mit dem Phasenregelkreis (*Phase Locked Loop*, PLL) und dem Modell einer Laufkatze auf einem Verladekran erweiterte Regelkreisstrukturen behandelt. Zusammen mit der Füllstandsregelung zeigt die Umsetzung des Laufkatzenmodells zudem die Möglichkeiten der Visualisierungen auf, welche MATLAB/Simulink und WinFACT bieten. Abschließend wird das Verhalten beider Programme bei Simulation von Abtastsystemen (digitaler Regelkreise) verglichen.

5.1 PLL – Phasenstarre Schleife

Ein phasenstarrer Regelkreis findet besonders in der Nachrichtentechnik weitreichende Anwendungen, die von der Frequenzsynthese über die Verwendung in Modems bis hin zur Dekodierung von FSK-/PSK-modulierten Signalen reichen. Detaillierte Informationen zur schaltungstechnischen Realisierung sind u. a. in [16, 17] zu finden.

In den vorliegenden Beispielen kommt der Phasenregelkreis im Wesentlichen als PSK-Decoder zum Einsatz. Dabei werden Phasensprünge im Empfangssignal erkannt und PLL-intern ausgeregelt.

5.1.1 Erzeugung synchroner Phasensprünge

Für die Simulation wird ein Quellenmechanismus benötigt, der Phasensprünge synchron zu einem Referenzsignal (Referenzphase $\varphi = 0°$) erzeugen kann. In der WinFACT-Lösung wird hierfür die Blockschaltung aus Abb. 37 eingesetzt.

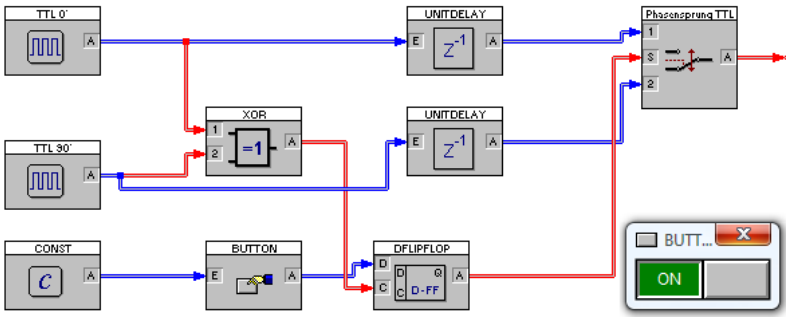


Abbildung 37: Signalquelle mit Erzeugung synchroner Phasensprünge (WinFACT)

Die beiden Signalquellen versorgen das System mit identischen Rechtecksignalverläufen, wobei die Signale eine Phasendifferenz von 90° aufweisen. Das XOR-Gatter erzeugt einen zu den Signalen flankensynchronen Takt, der ein D-Flipflop ansteuert. Mit dem Umschalt-Button, der am Eingang des Flipflops angeschlossen ist, kann der Analog-Switch am Ausgang taktsynchron zwischen den Signalen wechseln. Mit diesem Konstrukt ist es möglich, im Ausgangssignal abwechselnd eine Phasenänderung von ±90° zu erwirken.

Beim Aufbau in Simulink wurde das vorliegende Konzept dahingehend erweitert, dass nun beliebige Phasensprünge von ±90° möglich sein sollen. Das Ergebnis zeigt Abb. 38. Hier wird eine Auswahllogik implementiert (im Wesentlichen ein Up/Down-Counter), die synchron zum Referenzsignal zwischen vier Signalen mit je 90° Phasendifferenz selektiert, wobei die Logik in einem (in diesem Modell integrierten) Subblock zusammengefasst wird. Mit den beiden farbig

hinterlegen Switches kann nun während der Simulation durch einen Doppelklick auf den jeweiligen Block (bei *jedem* Umschalten) ein Phasensprung am Ausgang initiiert werden. Die aktuelle Phasenlage der Quelle wird im farbig hervorgehobenen Display dargstellt.

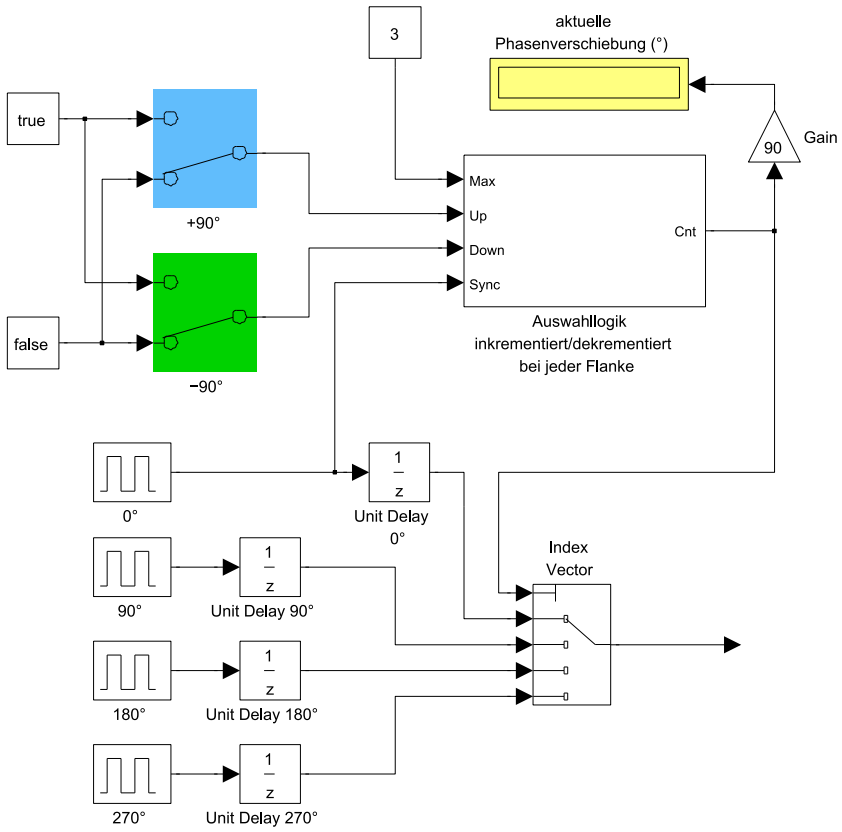


Abbildung 38: Signalquelle mit Erzeugung synchroner Phasensprünge (Simulink)

5.1.2 Implementierung des Phasenregelkreises

Allgemein kann ein Phasenregelkreis gem. Bild 39 aufgebaut werden [5]. Ein Phasenkomparator vergleicht das Eingangssignal mit der aktuellen, im Regelkreis präsenten Schwingung. Die Phasendifferenz wird am Ausgang über eine Mittelwertbildung (Tiefpass) geführt, woraus ein der Phasendifferenz proportionales analoges Signal gewonnen wird, mit dem ein spannungsgesteuerter Oszillator (*Voltage Controlled Oscillator*, VCO) beaufschlagt wird. Der VCO wird so geregelt, dass sich sein Ausgang dem eingehenden Signal anpasst. Die Information eines Phasensprungs kann hier aus einer *Änderung des VCO-Steuersignals* gewonnen werden.

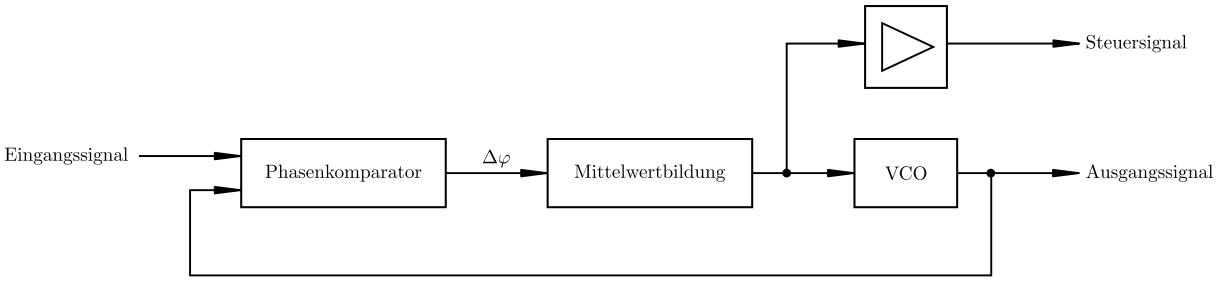
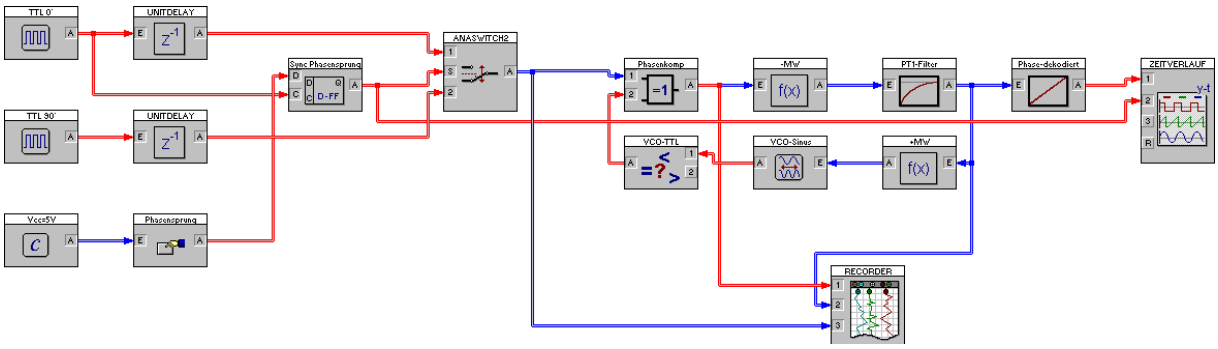


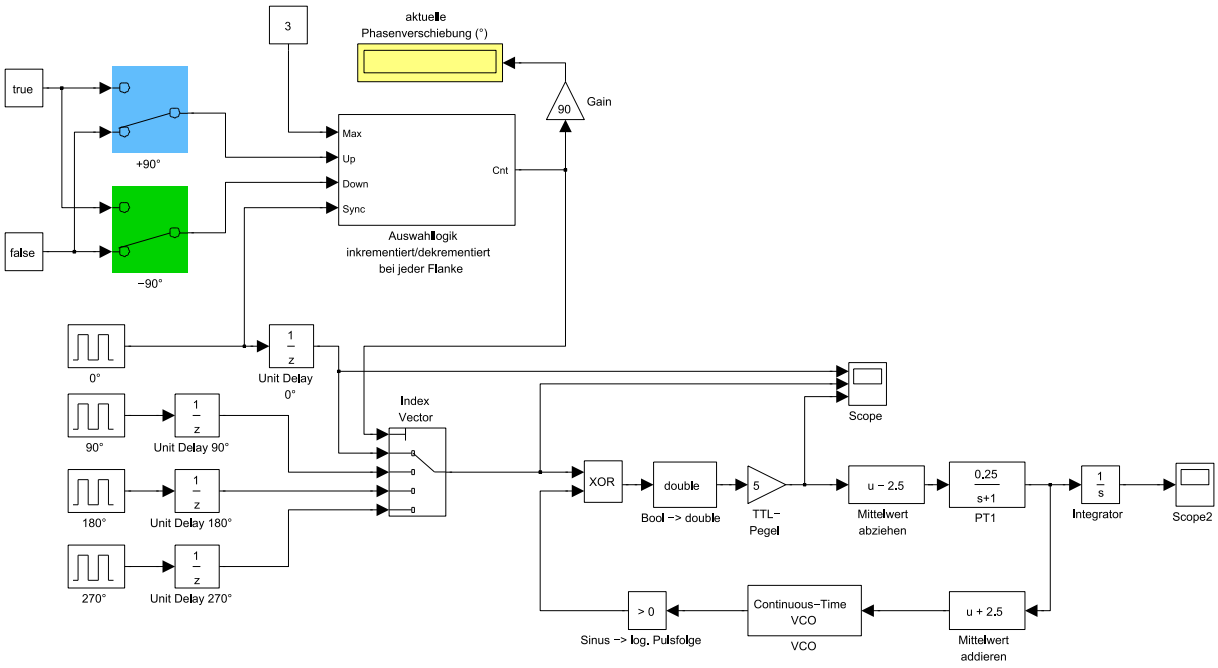
Abbildung 39: Prinzipaufbau PLL (nach [5])

Für TTL-Logik-Signale kann als Phasenkomparator ein XOR-Gatter eingesetzt werden, zur Mittelwertbildung wird hier ein Tiefpass 1. Ordnng (PT_1) verwendet. Die WinFACT-Lösung ist

in Bild 40(a), die Simulink-Variante ist in 40(b) dargestellt. Wie zuvor festgestellt, kann aus dem Steuersignal für den VCO die Phaseninformation gewonnen werden.



(a) WinFACT



(b) Simulink

Abbildung 40: PLL-Implementationen

Bemerkenswert ist hier zum Einen, dass Simulink aufgrund der vorhandenen Datentypunterscheidung Blöcke zur Typkonvertierung benötigt, zum Andern differieren die Einstellungen der jeweiligen VCO-Blöcke. WinFACT parametriert den *Sinus*generator durch Angabe von Kreisfrequenz (ω_0; entspricht der Empfindlichkeit), Amplitude und Offset (Gleichanteil). In der Betriebsart *linear* ermittelt sich die Ausgangskreisfrequenz ω gemäß Gleichung 33 [9].

$$\omega(x) = x\,\omega_0 \tag{33}$$

Die Ausgangsfrequenz hängt damit unmittelbar linear vom Eingang ab. Simulink hingegen bedient sich der Form ([13]):

$$y(t) = A_c \cos\left(2\pi f_c t + 2\pi k_c \int_0^t u(\tau)d\tau + \varphi\right) \tag{34}$$

wobei A_c die Amplitude des Ausgangssignals, f_c die Frequenz im Ruhezustand (*Quiescent frequency*), k_c die Empfindlichkeit (*Input Sensitivity*, $\left[\frac{\text{Hz}}{\text{V}}\right]$) und φ die Phasenlage im Ausgangszustand (*Initial phase*) bezeichnet. Um dasselbe Ergebnis wie unter WinFACT zu erreichen, müssen

folgende Voraussetzungen gegeben sein:

- Die Frequenz im Ruhezustand muss $f_c = 0$ sein.
- Die Phasenlage muss auf $-\frac{\pi}{2}$ eingestellt werden (BORIS betreibt einen Sinusgenerator, Simulink verwendet den Cosinus)
- Die Amplitude muss mit dem Wert von BORIS übereinstimmen.
- Die Sensitivität muss der Empfindlichkeit von BORIS angeglichen werden: $k_c = \frac{\omega_0}{2\pi}$ (Simulink verwendet Frequenzwerte, BORIS Kreisfrequenzwerte).

Im eingeschwungenen/ausgeregelten Zustand herrscht zwischen dem Eingangssignal und der Schwingung am VCO-Ausgang eine Phasenverschiebung von 90° vor. Das XOR-Gatter liefert dann ein Rechtecksignal mit einem Tastverhältnis von 50%, was nach Mittelwertbildung und Entfernung des Gleichanteils zu einem Ausgangssignal von Null führt. Das System befindet sich in Ruhelage.

Das VCO-Steuersignal wird über einen Integrator geführt. Bei jedem Phasenwechsel durch Schalter-Umlegen in der Signalquelle ist am Ausgang ein Sprung erkennbar (Abb. 41).

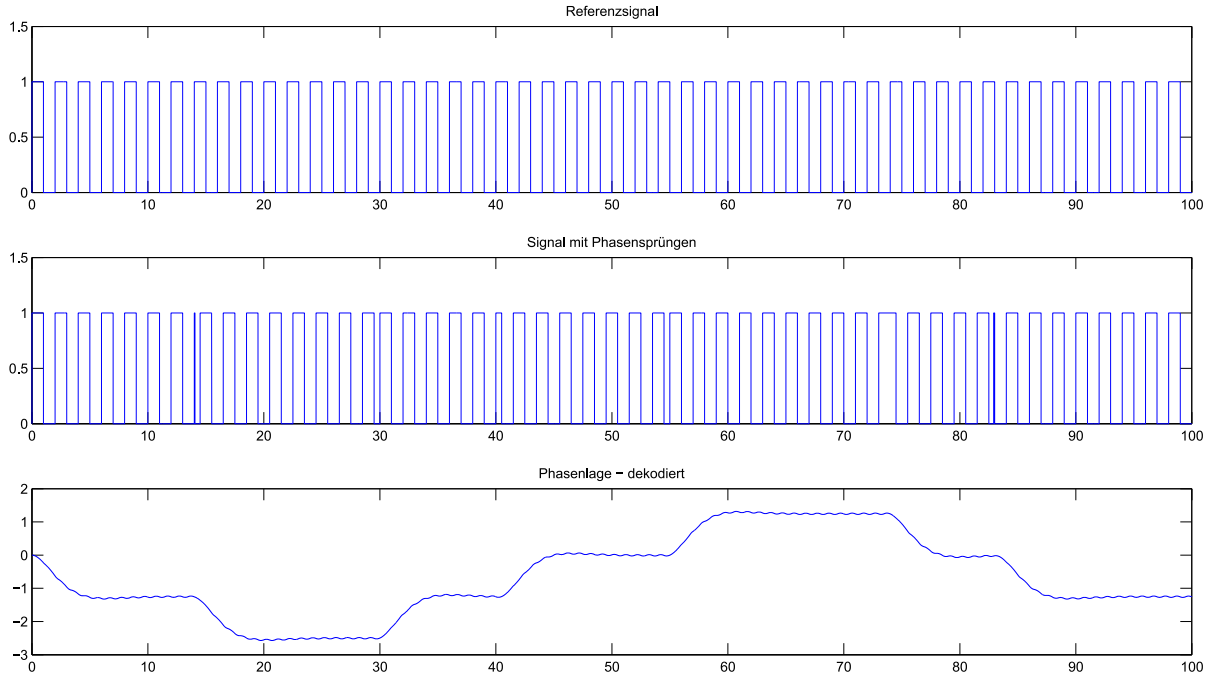


Abbildung 41: PLL – Ausgangssignal bei Phasensprüngen

5.1.3 Alternative Implementierung

Alternativ zur im vorigen Abschnitt vorgestellten Lösung könnte anstatt des PT_1 eine tatsächliche arithmetische Mittelwertbildung in Verbindung mit einem Sample&Hold-Glied eingesetzt werden (Bild 42).

5.1.4 Vergleich WinFACT/BORIS mit MATLAB/Simulink

Die Aufgabenstellung lässt sich problemlos in beiden Software-Systemen umsetzen. Bemerkenswerte Unterschiede ergaben sich in der im theoretischen Kapitel angemerkten Ausrichtung von WinFACT hinsichtlich Benutzerfreundlichkeit, da bspw. der Umschalter in der Signalquellenschaltung grafisch ausgeführt werden kann und viel mehr Möglichkeiten zur Signalverlaufsdarstellung

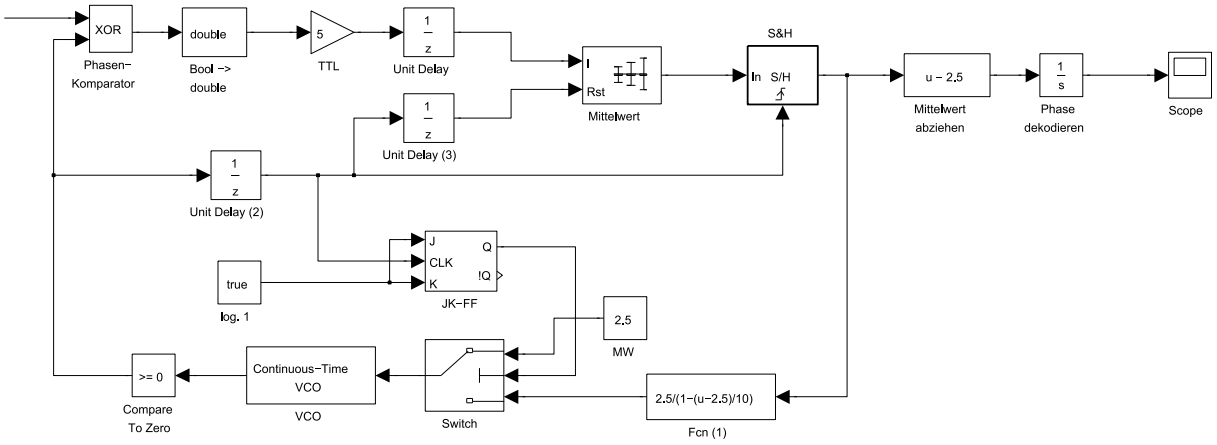


Abbildung 42: PLL – Alternative Implementation

gegeben sind. Unterschiede ergaben sich zudem durch die Typisierung von MATLAB/Simulink und die unterschiedliche Konfiguration der VCO-Blöcke. Dies sind aber allesamt unwesentliche Differenzen.

Zudem ist zu bemerken, dass die Logikblöcke in WinFACT/BORIS mit TTL-Pegel (0V Low- bzw. 5V High-Pegel) arbeiten, während die Boolean-Werte der Simulink-Blöcke in die Double-Werte 0.0 und 1.0 gewandelt werden. Daher ist in der Simulink-Schaltung ein zusätzliche Wandlung auf TTL-Signale notwendig.

5.2 Füllstandsregelung

In diesem Abschnitt soll die Modellumsetzung eines Tanksystems bzw. einer Füllstandsregelung vorgestellt werden. Im Zuge dessen wird auch ein erster Einblick in die Visualisierungsmöglichkeiten von WinFACT/BORIS bzw. MATLAB/Simulink gegeben.

5.2.1 Dynamisches Modell

Ein Flüssigkeitsbehälter mit Zu- und Abflussystem kann näherungsweise als Integrator betrachtet werden, wobei sich die Ausgangsgröße der Regelstrecke, der Füllstand h [m], bei gegebenem Zu- und Abfluss – q_{zu} bzw. q_{ab}, jeweils in $\left[\frac{\mathrm{m}^3}{s}\right]$) – und bekannter Querschnittsfläche des Tanks A $\left[\mathrm{m}^2\right]$ näherungsweise nach folgender Gleichung 35 berechnen lässt:

$$h(t) = \frac{1}{A} \int_0^t q_{\mathrm{zu}}(\tau) - q_{\mathrm{ab}}(\tau)\, d\tau \tag{35}$$

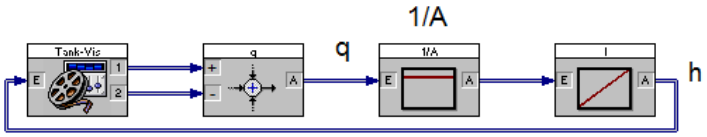
Im s-Bereich folgt daraus:

$$H(s) = \frac{1}{A}\frac{1}{s}\left(Q_{\mathrm{zu}}(s) - Q_{\mathrm{ab}}(s)\right) \tag{36}$$

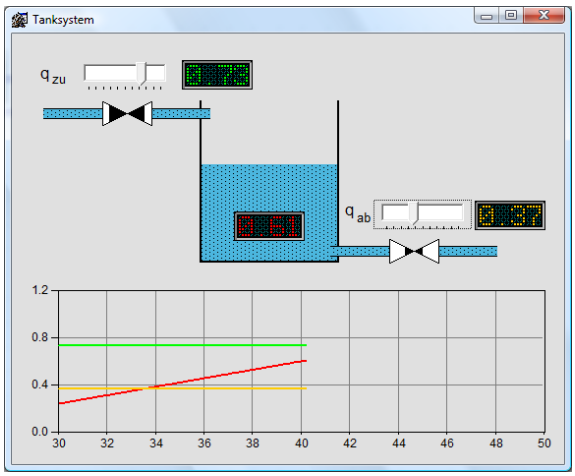
Der Tank kann somit als Integrator modelliert werden. Für die Modellierung im SW-System ist zu beachten, dass dieser Integrator mit Begrenzung betrieben wird – damit können Nullpegel und Überlauf nachgebildet werden. Im nächsten Abschnitt wird eine Beispielimplementation des Grundsystems vorgestellt. Beide Versionen (WinFACT bzw. MATLAB) beinhalten bereits Visualisierungen, die eine grafische Veranschaulichung des Problems bieten sollen.

5.2.2 Visualisierungsmöglichkeiten

Visualisierung in WinFACT/BORIS WinFACT bietet mit dem sog. *Flexible Animation Builder* (FAB) ein leistungsfähiges Werkzeug, um für Modelle und Regelkreise ein grafisches Äquivalent zu erstellen. Als Beispiel soll ein Tank mit Zu- und Abfluss dargestellt werden. Ein mögliches Ergebnis ist in Abb. 43 einzusehen.



(a) BORIS-Blockschaltung



(b) Tank-Vis-Block – FAB-Control

Abbildung 43: Tank - Visualisierung mittels FAB

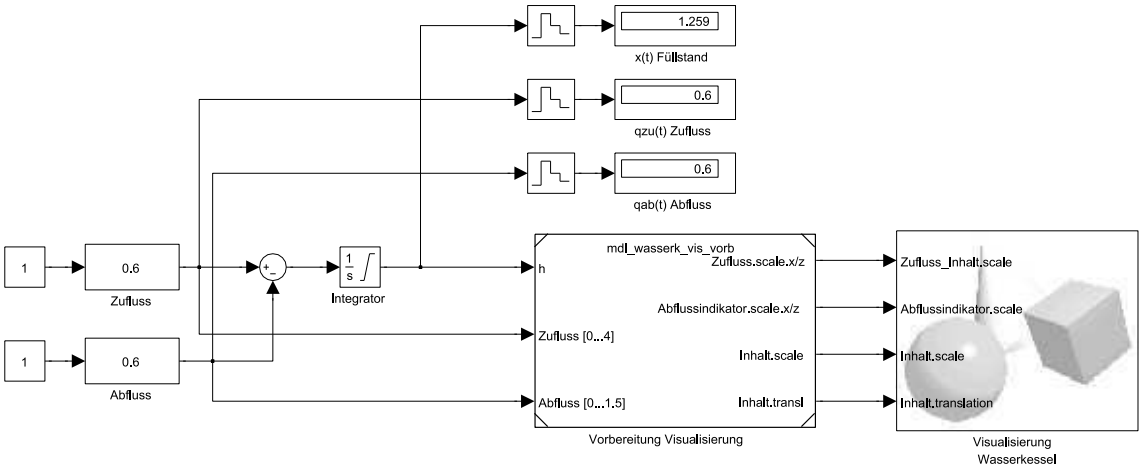
Das von FAB generierte Visualisierungsfenster wird per User-DLL (FAB.dll) in das System eingebunden. Neben der Darstellung der div. Zustandsgrößen können ebenso Controls eingebunden werden, die ein Ausgangssignal erzeugen. Im gezeigten Beispiel sind dies bspw. zwei Slider-Elemente, die die Zu- und Abflussventile steuern. Die Füllhöhe am Ausgang des (auf einen Wertebereich von 0...1 begrenzten) Integrators wird als Eingang geführt und entsprechend in die grafische Anzeige eingebunden.

Neben FAB können auch selbsterstellte User-DLLs zur Animation genutzt werden, wie es beim Laufkatzenmodell in Abschnitt 5.3 geschieht.

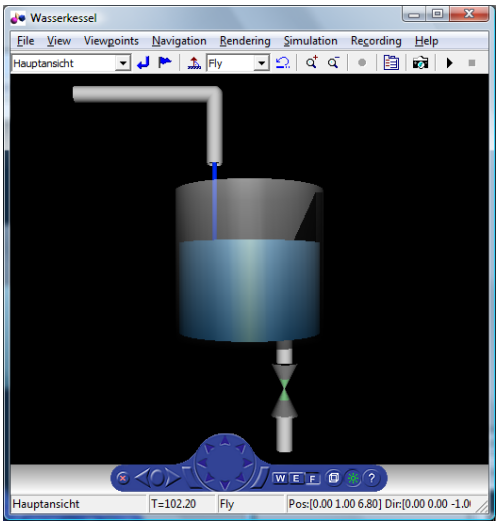
Visualisierung in MATLAB/Simulink In MATLAB/Simulink ist kein mit FAB gleichzustellendes Tool vorhanden. Wenngleich es möglich ist, mit MATLAB GUIs zu erstellen, so setzt dies doch ein gewisses Maß an Erfahrung in der Erstellung und Programmierung grafischer Oberflächen voraus. Zudem gestaltet sich die Einbindung in Simulink eher schwierig. Einfache Animationen werden meist als s-Functions eingebunden, bei denen die komplette Animation in Programmcode erstellt wird.

Ein grafisch leistungsfähigeres Werkzeug wird mit der Bibliothek „Simulink 3D Animation“ bereitgestellt. Diese Library ermöglicht die Darstellung und Veränderung von 3D-Modellen während der Simulation. Das 3D-Rendering basiert dabei auf Basis der Virtual Reality Modeling Language (VRML), einem Textformat zur Beschreibung von 3D-Szenen [6, 7].

Die Simulink-Schnittstelle bietet einen Block zur Darstellung des 3D-Modells. In diesem Block können variable Parameter des 3D-Modells festgelegt werden, die über Eingänge verändert werden sollen. Im Beispiel in Abb. 44(a) wurde die komplette Ansteuerungslogik für die Visualisierung in ein externes Modell abstrahiert, welches hier mittels Model-Referencing eingebunden wird und die Signale des Systems entsprechend für den 3D-Block aufbereitet.



(a) Simulink-Blockschaltung



(b) Visualisiertes Tankmodell

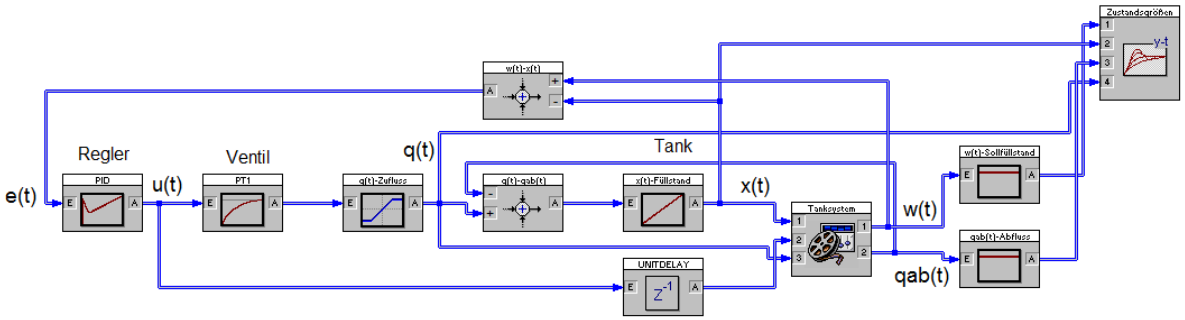
Abbildung 44: Tank - Visualisierung mittels Simulink 3D Animation

In der fertigen Visualisierung (Abb. 44(b)) verändert sich zum Einen der Pegel gemäß dem Modell, zum Andern sind Zu-/Abflussystem eingearbeitet (der „Querschnitt“ des Zuflusses bzw. die Ventilstellung des Abflusses werden dabei verändert). Die Steuerung der Flüsse erfolgt über Simulink-Slider-Gain-Elemente.

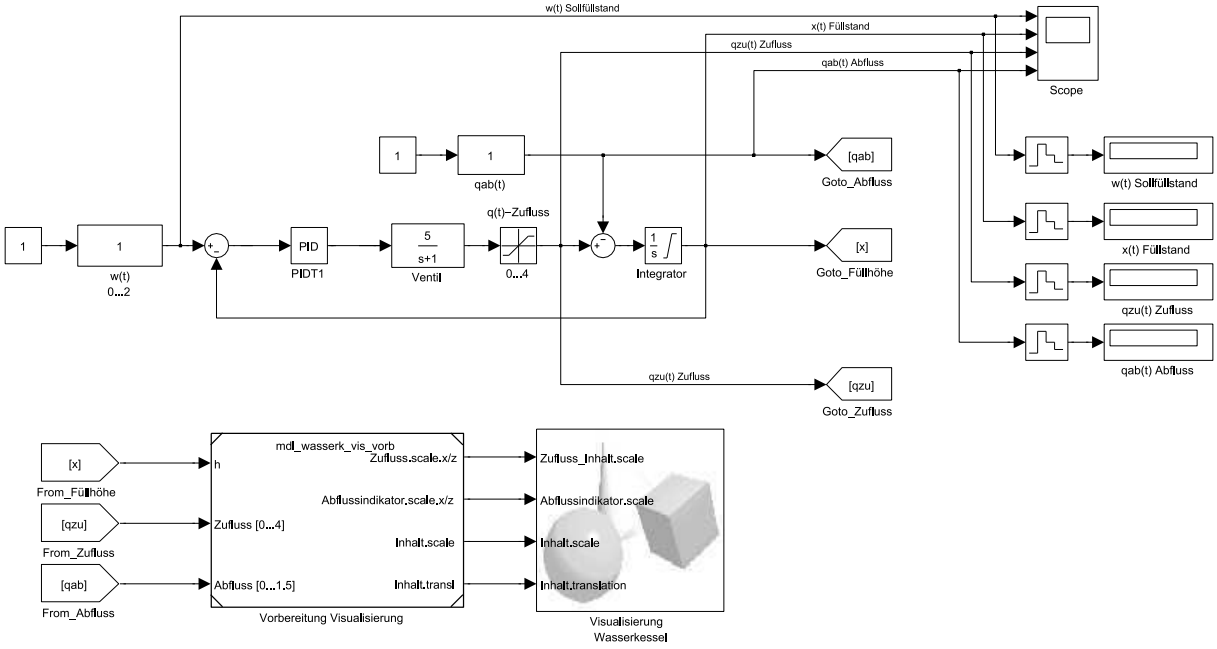
Das Erstellen des 3D-Modells kann vollständig in Code geschehen. Vorzuziehen ist aber das mit dem 3D-AddOn mitgelieferte Programm *VRBuild*, dass eine grafische Oberfläche zur Gestaltung von 3D-Szenen bietet. Neben der Platzierung von Grundkörpern (Quader, Zylinder, Kegel etc.) kann es auch sog. *Face-Sets* und *Extrusions* erzeugen, die komplexere Gestaltungsoptionen bieten.

5.2.3 Füllstandsregelkreis

Der Tank soll durch einen PID-Regler kontrolliert werden. Der Regler steuert ein Ventil, das durch ein PT_1-Element genähert wird und einen Zufluss $q_{zu} = 0...4$ erlaubt. Zur Visualisierung werden die bereits vorhandenen Möglichkeiten wiederverwendet. Die entsprechende WinFACT-Lösung zeigt Bild 45(a), Abb. 45(b) stellt eine mögliche Simulink-Umsetzung dar.



(a) WinFACT



(b) Simulink

Abbildung 45: Implementationen der Füllstandsregelung

Der PID-Regler, der in BORIS verfügbar ist, weist eine Besonderheit auf, die es in dieser Form in Simulink nicht gibt: eine Anti-Windup-Begrenzung. *Windup* bezeichnet die Tatsache, dass ein digitaler Integrator (der in der Regel einen wesentlich höheren Wertebereich aufweist, als das physikalische System abgeben kann) beständig aufintegriert, obwohl sich die tatsächliche, physikalische Größe bereits in der Begrenzung befindet. Der reale Regler-Ausgangswert verbleibt daher in der Sättigung, während der (digitale) Integrator beständig aufsummiert. Ist der gewollte Wert schließlich erreicht, benötigt der Integrator noch zusätzlich Zeit, um wieder den gewünschten Sollwert zu erreichen. Dadurch kann die Schwingneigung des Systems entscheidend erhöht werden, dem Windup-Effekt muss daher in der Praxis entgegengewirkt werden.

Dies kann durch Aktivieren der Begrenzung im PID-Block geschehen. Bei Erreichen der Anti-Windup-Grenze kann wahlweise ein *Anti-Windup-Halt* oder ein *-Reset* durchgeführt werden. Beim Reset wird der Integrator zurückgesetzt, während im Haltemodus der Begrenzungswert am Ausgang ausgegeben und der Integrationsvorgang nicht weitergeführt wird. In Simulink gibt es diese Option nicht.

Bei der Implementation wurden mehreren Versionen erstellt, die sich nur in der Reglereinstellung unterscheiden:

- Einstellung auf 65° Phasenrand
- Einstellung auf 5% Überschwingen
- Empirische Einstellung

Für beiden erstgenannten existieren zudem je zwei Versionen, bei denen sich die Einstellung der Reglerparameter hinsichtlich Ausregelgeschwindigkeit unterscheiden.

5.2.4 Vergleich WinFACT/BORIS mit MATLAB/Simulink

Neben der im vorigen Abschnitt erwähnten Anti-Windup-Funktionalität streicht dieses Beispiel deutlich der Vorteil von WinFACT/BORIS heraus: das User-Interface. Die bereitgestellten Controls erlauben es, mit GUI-Elementen die Simulation aktiv zu beeinflussen. Die einfache Erstellung mit dem Flexible Animation Builder ist ein wesentlicher Pluspunkt gegenüber Simulink. Die Möglichkeit der 3D-Darstellung kann dies nur im Hinblick auf die visuelle Präsentation ausgleichen.

Zweifelsohne lässt sich dieses vergleichsweise einfache Modell in beiden Software-Systemen simulieren, für die Verwendung in der Lehre sind aber die Möglichkeiten von WinFACT besser geeignet, um gewisse für die Lehre interessante Aspekte des physikalischen System herauszuarbeiten.

5.3 Modell Verladekran/Laufkatze

5.3.1 Dynamisches Modell

Als „Laufkatze“ wird der Wagen bezeichnet, der sich auf einem Kran bewegt und an dem die Last befestigt ist, wie es in Bild 46 schematisch dargestellt ist. Für diese Simulation wird angenommen, dass sowohl die Seillänge l als auch die Katzmasse m_{K} und die Greifermasse m_{G} konstant sind (ansonsten wäre das System zeitvariant!). Das Modell basiert auf Überlegungen, die im Detail in [3] wiedergegeben sind.

Unter Annahme einiger Vereinfachung (u. a. der Vernachlässigung der Rückwirkung der Greiferbewegung auf das Katzfahrzeug) ergibt sich für die horizontale Vorwärtsbewegung in s-Richtung des Greifers folgende Beziehung:

$$m_{\mathrm{G}}\,\ddot{y}(t) = S(t)\,\sin(\varphi(t)) - r\dot{y}(t) \tag{37}$$

wobei S die Seilkraft und $r\dot{y}(t)$ den Luftwiderstand bezeichnet. Für die Vertikalrichtung z kann ebenfalls eine vereinfachte Gleichung angegeben werden:

$$S(t)\,\cos(\varphi(t)) = mg \tag{38}$$

woraus folgt:

$$S(t) = \frac{mg}{\cos(\varphi(t))} \tag{39}$$

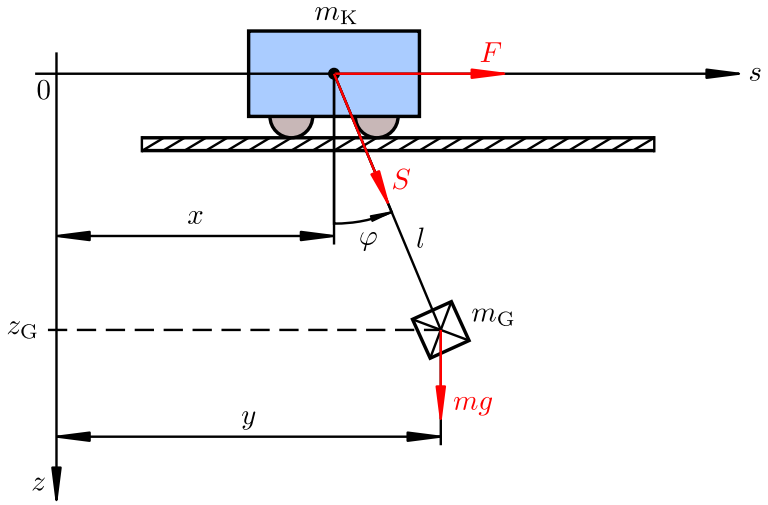


Abbildung 46: Laufkatze mit Greifer (nach [3])

Gleichung 39 in Formel 37 eingesetzt ergibt schließlich eine Differentialgleichung 2. Ordnung.

$$m_G\,\ddot{y}(t) = mg\,\tan(\varphi(t)) - r\dot{y}(t) \tag{40}$$

Im Weiteren wird $\tan(\varphi(t))$ durch $\tan(\varphi(t)) \approx \frac{x(t)-y(t)}{l}$ angenähert und die Differentialgleichung nach der höchsten Ableitung aufgelöst.

$$\ddot{y}(t) = \frac{x(t)-y(t)}{l}\,g - \frac{r}{m}\dot{y}(t) \tag{41}$$

Abschließend erfolgt eine Umformung zur Trennung nach x- und y-Position und die Transformation in den Laplacebereich.

$$\begin{aligned} \ddot{y}(t) + \frac{r}{m}\,\dot{y}(t) - \frac{g}{l}\,y(t) &= \frac{g}{l}\,x(t) \\ s^2\,Y(s) + \frac{r}{m}\,s\,Y(s) - \frac{g}{l}\,Y(s) &= \frac{g}{l}\,X(s) \end{aligned}$$

$$G_{LK}(s) = \frac{\frac{g}{l}}{s^2 + \frac{r}{m}\,s - \frac{g}{l}} = \frac{1}{\frac{l}{g}\,s^2 + \frac{lr}{mg}\,s - 1} \tag{42}$$

Dieses System beschreibt die Abhängigkeit der Greiferposition $y(t)$ in Abhängigkeit von der Katzposition $x(t)$. Die Katzposition wird nun durch ein Antriebssystem $G_A(s)$ vorgegeben. Da der Antrieb nicht sprungförmig erfolgen kann, wird hier als einfacher Fall ein PT_2-Element angenommen (was einen bereits *geregelten* Motor annähert – das Gesamtsystem verhält sich also wie eine Kaskadenregelung).

$$G_{A\,PT_2}(s) = \frac{k_S}{(1+sT_1)(1+sT_2)} \tag{43}$$

Alternativ dazu kann der Antrieb auch, ungeregelt, als IT_1 modelliert werden (Gl. 44), was bedeutet, dass der zu entwerfende Regler auch den Antrieb mitregelt.

$$G_{A\,IT_1}(s) = \frac{k_S}{s(1+sT_1)} \tag{44}$$

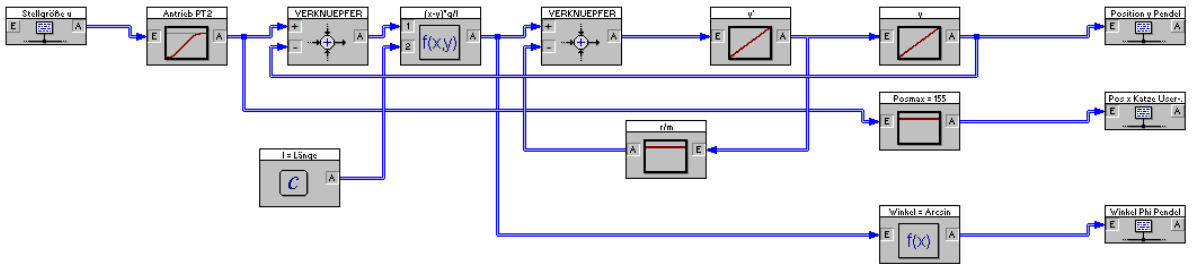
Die Übertragungsfunktion der Gesamtstrecke ergibt sich schließlich durch die Zusammenführung des Antriebselements mit der Laufkatzenstrecke. Gl. 45 beschreibt den Zusammenhang für einen PT_2-Antrieb, in Gl. 46 wird der Aktor mittels IT_1-Element approximiert.

$$G_S(s) = G_{A\,PT_2}(s)\,G_{LK}(s) = \frac{k_S}{(1+sT_1)(1+sT_2)}\,\frac{1}{\frac{l}{g}\,s^2+\frac{lr}{mg}\,s-1} \tag{45}$$

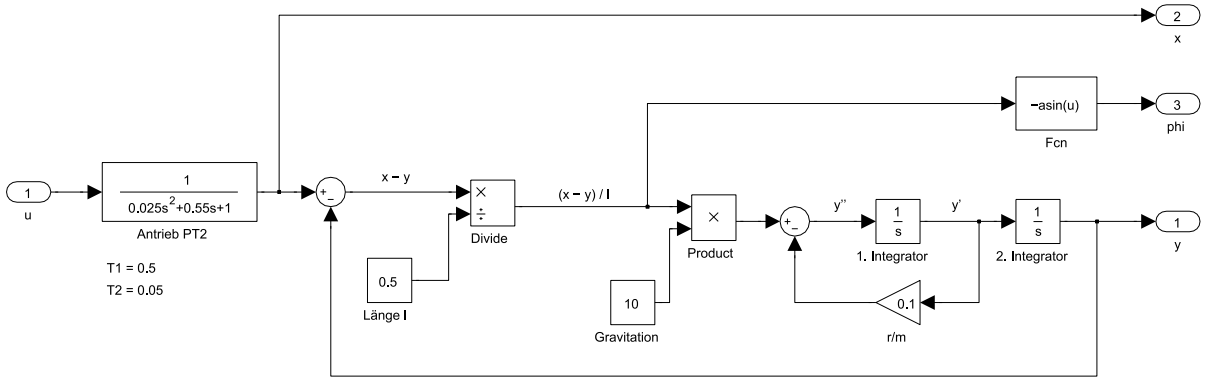
$$G_S(s) = G_{A\,IT_1}(s)\,G_{LK}(s) = \frac{k_S}{s(1+sT_1)}\,\frac{1}{\frac{l}{g}\,s^2+\frac{lr}{mg}\,s-1} \tag{46}$$

5.3.2 Simulationsmodelle

Für die Simulation wurde das im vorigen Abschnitt entwickelte Modell jeweils in Simulink und BORIS implementiert und in eigene Dateien ausgelagert. Diese können dann als einzelne Blöcke in Regelkreismodellen eingesetzt werden. WinFACT/BORIS bezeichnet solche abgekapselten Systeme als *Superblöcke* (sprich Blöcke, die tiefergreifende Funktionen kapseln), in Simulink wird das Einbinden von Sekundärmodellen als *Model Referencing* bezeichnet.



(a) WinFACT



(b) Simulink

Abbildung 47: Blockschaltungen für linearisiertes Laufkatzenmodell mit PT_2-Antrieb

Die Ausarbeitung des (linearisierten) Modelles mit PT_2-Antrieb führt auf die in Bild 47 gezeigten Varianten. Der Standardentwurf beginnt dabei mit dem Signal $\ddot{y}(t)$, von dem durch je einen Integrator $\dot{y}(t)$ und $y(t)$ gewonnen werden. Mithilfe dieser Größen wird nun Gl. 41 entsprechend umgesetzt. Die Verwendung der Gleichung im Zeitbereich ist notwendig, damit auch der Greiferwinkel $\varphi(t)$ einfach aus dem Modell ermittelt werden kann. Es wird dabei angenommen, dass der Winkel nur in geringem Maße variiert, wodurch statt der Arkustangensfunktion eine Anwendung des Arkussinus ausreicht. Der Greiferwinkel wird also durch $\varphi(t) \approx \arcsin\left(\frac{x(t)-y(t)}{l}\right)$ errechnet.

Das BORIS-Modell wird als eigener Dateityp mit Endung **.sbl* abgespeichert und kann in anderen Modellen mit einem *Superblock* referenziert werden, die Eingangs- und Ausgangsgrößen werden durch Labels markiert. Der Block *Posmax* ist nur für die spätere Visualisierung

vorgesehen, erfüllt aber hinsichtlich des mathematischen Modells keine Funktion.

In Simulink wird die Datei als einfaches Model-File (**.mdl*) abgelegt. Durch *Model Referencing* (Einbindung mittels Model-Block) wird es in entsprechenden Schaltungen eingefügt. Zu beachten ist hier, dass ein externes Modelfile nur dann mehrfach in einer Simulation verwendet werden kann, wenn die Blöcke, die das File referenzieren, im *Accelerator mode* (kann in den Blockeigenschaften/*ModelReference Parameters* aktiviert werden) arbeiten.

Weiters ist es wichtig, dass die Simulationseinstellungen des eingebundenen Modells mit denen des einbindenden Systems übereinstimmen müssen, wobei hier im Wesentlichen die Solver-Optionen stimmen müssen.

5.3.3 Weitere Implementationen

Neben der gezeigten Variante existieren auch Modelle mit IT_1-Antrieb, die sich nur darin unterscheiden, dass das PT_2-Glied entsprechend ersetzt wurde.

Zusätzlich ist noch ein physikalisch genaueres („exaktes“) Modell verfügbar, das zwar großteils dem linearisierten Modell ähnelt, aber nicht mehr linear und damit mittels Laplace-Transformation nicht behandelbar ist. Die Simulink-Datei ist in Abb. 48 wiedergegeben.

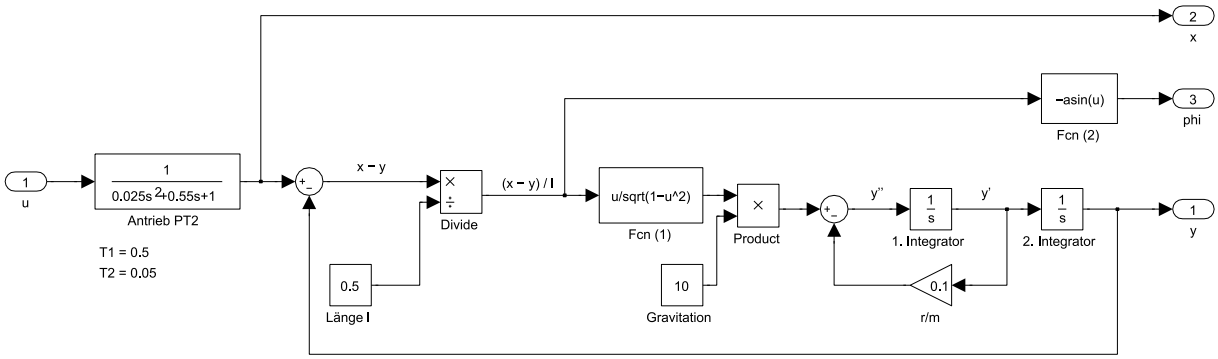


Abbildung 48: „Exaktes“ Laufkatzenmodell mit PT_2-Antrieb

5.3.4 Visualisierungsmöglichkeiten

Visualisierung in WinFACT/BORIS In diesem Beispiel wird zur Visualisierung ein sog. *User-DLL* eingesetzt. Dies ist eine proprietär, meist in C erstellte DLL, die BORIS um zusätzliche Funktionen (wie hier die grafische 2D-Darstellung) erweitert. User-DLLs müssen nur der Schnittstellenspezifikation für WinFACT folgen und stellen dadurch eine Möglichkeit für dne Benutzer dar, WinFACT selbstständig um (auch komplexere) Funktionen zu erweitern.

Anders als das FAB-Tank-Modell aus Abschnitt 5.2.2 benötigt dieses Modell keinerlei Controls und generiert keine Ausgangsgrößen, fungiert deshalb als Signalsenke – aus Katzposition $x(t)$ und Greiferwinkel $\varphi(t)$ wird in einem eigenen Fenster ein grafisches Äquivalent des Verladekrans gezeichnet, wie Abb. 49 wiedergibt.

Visualisierung in MATLAB/Simulink In Simulink wird, wie schon bei der Füllstandsregelung, ein VRML-Modell zur 3D-Darstellung verwendet. Eingangsgrößen sind Katzposition und Greiferwinkel, die Steuerungslogik für die Visualisierung wird in einem eigenen Model gekapselt. Die fertige 3D-Szene ist in Bild 50 einzusehen.

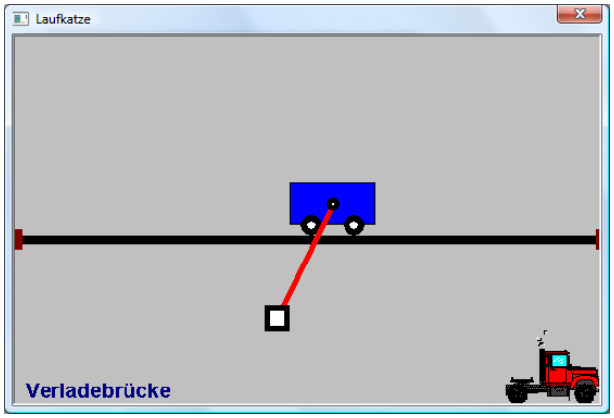


Abbildung 49: WinFACT – 2D-Visualisierung durch UserDLL

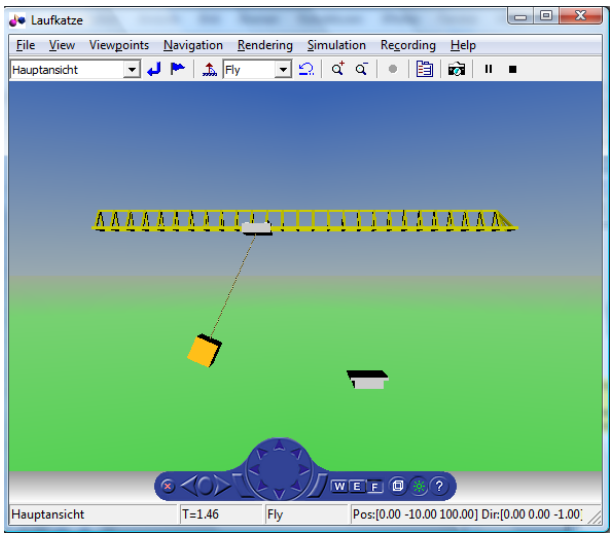


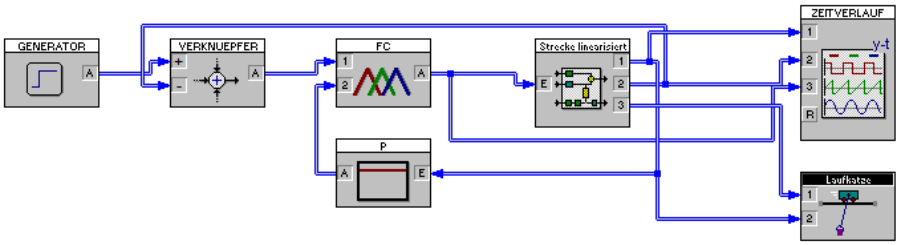
Abbildung 50: Simulink – 3D-Visualisierung

5.3.5 Regelkreise

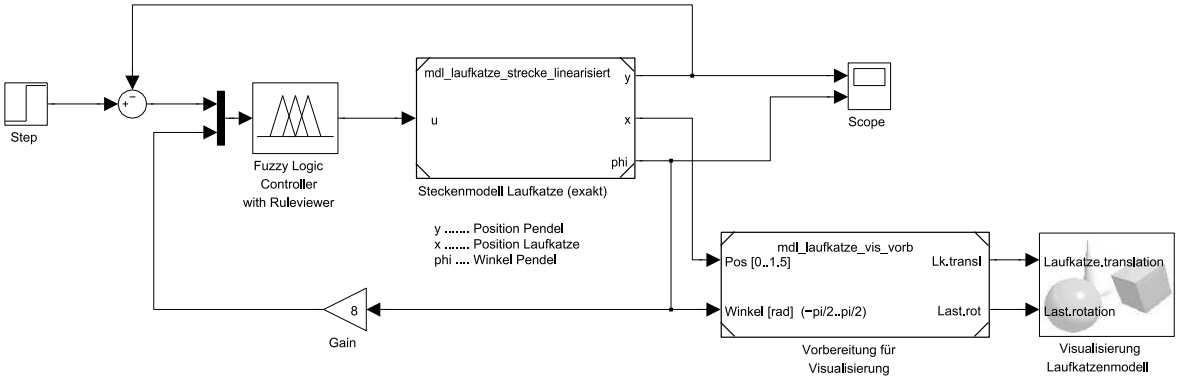
Das Anfahren der Soll-Greiferposition soll nun geregelt erfolgen. Neben Regelkreismodellen mit PID-Regler in betragsoptimaler Einstellungen wurde auch ein Fuzzy-geregeltes System vorgesehen. Der Fuzzy-Regler verarbeitet dabei die Regelabweichung $e(t) = w(t) - y(t)$ sowie den Greiferwinkel $\varphi(t)$ und versucht, den Greifer möglichst schwingungsfrei an die Zielposition zu bewegen. Der Aufbau ist in Abb. 51 ersichtlich.

Zur Einstellung des Fuzzy-Reglers in Simulink ist anzumerken, dass grundsätzlich die selben Verfahren zur Fuzzyfizierung und Inferenzbildung vorhanden sind wie in WinFACT. Dem ist nicht so bei der Gewinnung eines scharfen Ausgangswerts bei der Defuzzyfizierung – in der WinFACT-Version wird die randwerterweiterte Schwerpunktmethode verwendet, die in Simulink standardmäßig nicht vorhanden ist. Die Einstellung *Centroid* entspricht der Schwerpunktmethode (CoA: Center of Area oder auch CoG: Center of Gravity) und kommt damit der WinFACT-Einstellung am nächsten. Es ist möglich, das randwerterweiterte Verfahren proprietär zu implementieren, was jedoch händisch und in Code erfolgen muss.

Wie bereits erwähnt wurden neben der gezeigten Variante mit Fuzzy-Regelungen noch das linearisierte und das exakte Modell jeweils mit PID-Regler und Einstellung gemäß dem Betragsoptimum umgesetzt. Zusätzlich sind noch für alle Modelle (linearisiertes Modell und exaktes



(a) WinFACT



(b) Simulink

Abbildung 51: Laufkatze mit Fuzzy-Regler

Modell mit PT_2-Antrieb, linearisiertes Modell mit IT_1-Antrieb) Simulationen für den ungeregelten Fall erstellt worden.

Für alle Systeme ist anzumerken, dass die Verwendung der 3D-Visualisierung und des $PIDT_1$-Reglers in Simulink unter Umständen zu einem äußerst langsamen Simulationsvorgang führen kann. Hier bietet es sich an, verschiedene Solver zu testen. Für PID-geregelte Systeme mit 3D-Visualisierung haben sich die Solver für steife Systeme (z. B. *ode15s*) als performant erwiesen, für Fuzzy-Systeme ist evtl. ein Umstieg auf einen Fixed-Step-Solver vorteilhaft.

5.3.6 Vergleich WinFACT/BORIS mit MATLAB/Simulink

Verglichen mit dem Tanksystem weist das Laufkatzenmodell eine höhere Komplexität auf, was jedoch für die Modellierung in beiden Programmen keine wesentlichen Probleme aufwirft. Da bei diesem Modell der Visualisierungsaspekt wesentlich ausgeprägter ist als noch beim Tanksystem, ist hier die Möglichkeit der 3D-Animation als vorteilhaftere Option (im Vergleich zur 2D-UserDLL) anzusehen. Einziger Nachteil ist, dass es auf rechenschwachen System möglicherweise Geschwindigkeitsprobleme durch die 3D-Berechnung geben kann.

Die große Funktionsvielfalt von MATLAB/Simulink wirkt sich bei diesen Modellen teilweise als Problem aus. Der voreingestellte Variable-Step-Solver erweist sich teilweise als äußerst unperformant, eine empirische Suche nach dem optimalen Solver ist notwendig. Nichtsdestotrotz kann eine solche Einstellung gefunden werden und die Komplexität und Konfigurierbarkeit der Umgebung bleibt als Positivum anzusehen.

5.4 Digitale Regelungstechnik

Dieser Abschnitt behandelt einige Modelle der digitalen Regelungstechnik, im Einzelnen werden folgende Themen behandelt:

- Diskretisierung kontinuierlicher Systeme
- Diskrete periodische Folgen
- Diskrete Systeme und PWM
- Diskreter Regler nach Takahashi
- FIR-Regelkreise und Dead Beat

5.4.1 Diskretisierung kontinuierlicher Systeme

Um kontinuierliche Systeme mit diskreten Reglern behandeln zu können, müssen auch die Streckenmodelle diskretisiert werden. Dazu wird die z-Transformation eingesetzt (siehe u. a. [2, 20]). Die Anwendung dieser Transformation erzeugt ein Abtastsystem, das (in Abhängigkeit der Abtastzeit) dem kontinuierlichen System exakt entspricht. Die Abtastung kann mit Hilfe sog. *Halteglieder nullter Ordnung (Zero Order Hold, ZOH)* beschrieben werden, wodurch dieses Diskretisierungsverfahren im Englischen oft schlicht als ZOH-Diskretisierung bezeichnet wird.

Neben der exakten Transformation existieren diverse Näherungsverfahren, die dann zur Anwendung kommen, wenn keine exakte Lösung ermittelbar oder der Rechenaufwand zu hoch ist. Die hier behandelten Approximationen sind:

- Rechtecknäherung Typ I (linksseitig)
- Rechtecknäherung Typ II (rechtsseitig)
- Trapeznäherung (auch als bilineare Transformation oder in der Literatur häufig als Tustin-Formel bezeichnet)

Diese Verfahren beschreiben einfache Substitutionen, mit den vom s- in den z-Bereich umgerechnet werden kann (siehe hierzu [1]). Bezeichnet T die Abtastzeit, so gilt für die linksseitige Rechtecknäherung die Substitution

$$s \to \frac{z-1}{T} \tag{47}$$

Für die rechtsseitige Rechtecknäherung:

$$s \to \frac{z-1}{T\,z} \tag{48}$$

Die Tustin-Formel ist etwas genauer [20], ihre Substitutionsregel lautet:

$$s \to \frac{2}{T}\,\frac{z-1}{z+1} \tag{49}$$

In den behandelten Beispielsimulationen wurde jeweils ein kontinuierliches System mit der exakten Approximation und sowohl einer Rechtecknäherung als auch der Tustin-Formel verglichen. Ein Beispiel zeigt Abb. 52.

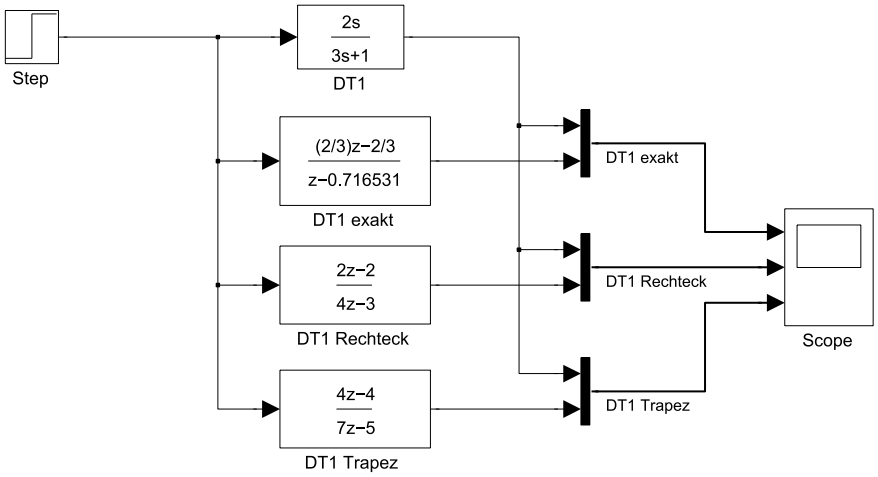


Abbildung 52: Diskretisierung eines DT_1-Systems

Die Signalverläufe zeigt Bild 53. Deutlich zu erkennen ist die Ungenauigkeit der Rechtecknäherung. Die Tustin-Formel weist ebenfalls eine Abweichung vom Optimum – der exakten ZOH-Diskretisierung – auf.

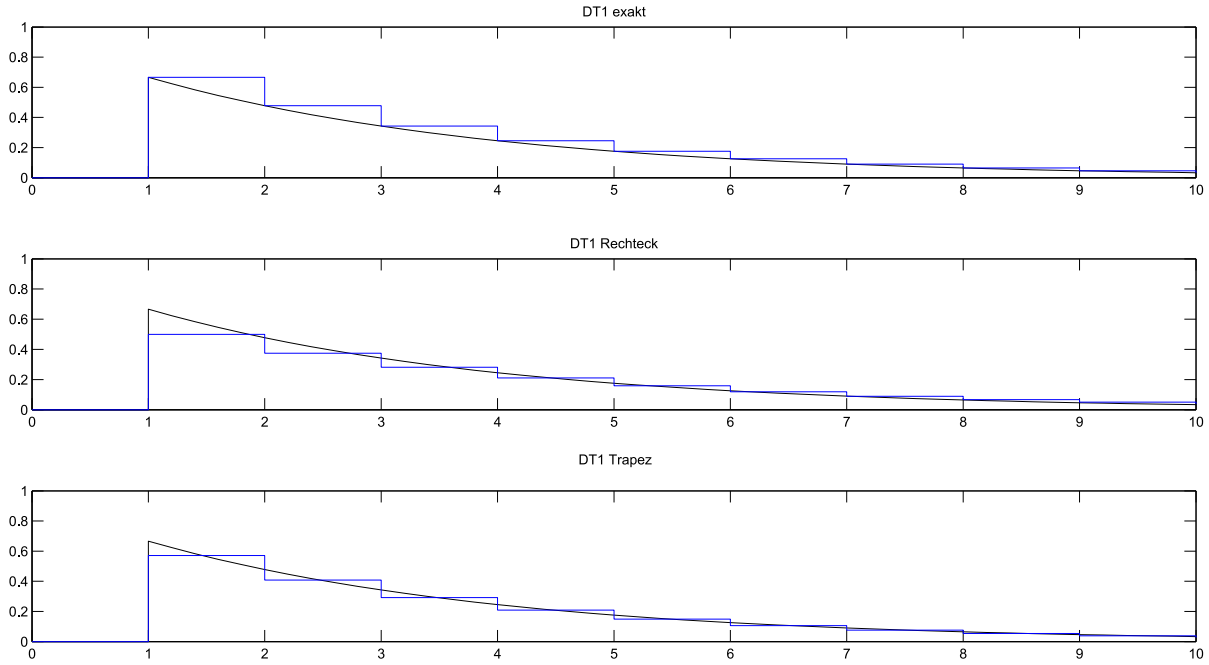


Abbildung 53: Diskretisierung eines DT_1-Systems – Signalverläufe

Dieses Beispiel zeigt auch gut die Möglichkeiten der Signalverlaufsdarstellung. In WinFACT/-BORIS ist das Standard-Control der *Zeitverlauf*, der allerdings nur maximal 3 Signale darstellen kann. Für bis zu 50 Eingänge kann ein sog. *Mehrfach-Zeitverlauf* eingesetzt werden. Beide Blöcke unterstützen wahlweise die Darstellung aller Signale in einem einzigen Diagramm oder ein Diagramm pro Zeitverlauf.

Flexibler gestaltet sich hier die Arbeit mit Simulink: Durch Multiplexer-Bausteine können mehrere Signale in einem Diagramm zusammengefasst werden (in Abb. 53 werden bspw. je ein digitales und das Analogsignal zusammengefasst). Zudem kann die Anzahl der separaten Zeitverlaufsdiagramme im *Scope* auf der Darstellungsoberfläche unter *Parameters/Number of axis* eingestellt werden – in Abb. 53 sind drei Achsen konfiguriert.

5.4.2 Diskrete periodische Folgen

Mit Hilfe der z-Transformation ist es möglich, durch Angabe aller Punkte einer Periode eine periodische Folge zu erzeugen. Das Ergebnis ist eine z-„Übertragungsfunktion“, die allerdings als *Stoßantwort* interpretiert werden muss. Als Beispiel soll die Folge $y = (0, 1, 1, 0, 0, -1, -1, 0)$ periodisch ausgegeben werden.

Allgemein gilt für eine Folge $x = (x_0, x_1, \ldots, x_{N-1})$ mit N Werten, die periodisch wiederholt werden soll, die Beziehung:

$$X(z) = \frac{z^N}{z^N - 1} \sum_{i=0}^{N-1} x_i z^{-i} \tag{50}$$

Auf die Beispielfolge angewendet bedeutet dies:

$$\begin{aligned} X(z) &= \frac{z^8}{z^8 - 1} \left(0z^0 + 1z^{-1} + 1z^{-2} + 0z^{-3} + 0z^{-4} - 1z^{-5} - 1z^{-6} + 0z^{-7}\right) \\ &= \frac{z^7 + z^6 - z^3 - z^2}{z^8 - 1} \end{aligned}$$

Wie Abb. 54 veranschaulicht, wird diese z-Funktion als Übertragungssystem betrachtet und mit einem Einheitsimpuls beaufschlagt.

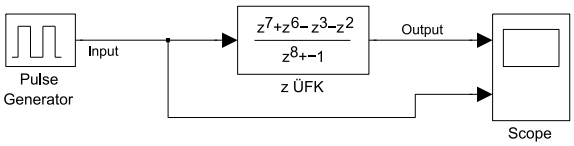
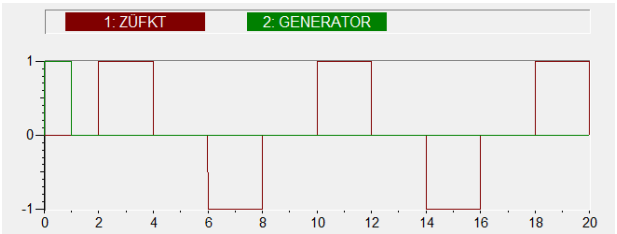
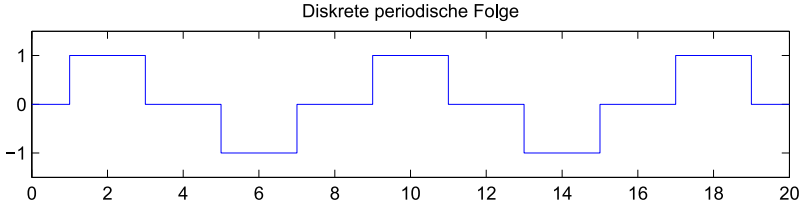


Abbildung 54: Schaltung zur Simulation einer diskreten periodischen Folge

Die Ergebnissignalverläufe zeigt Bild 55. Auffällig ist, dass WinFACT im Vergleich zu Simulink einen Abtastschritt später reagiert. Der Grund hierfür konnte nicht ermittelt werden – nahe liegt, dass WinFACT erst nach einmaligem Verstreichen der Abtastzeit zum ersten Mal abtastet, während Simulink bereits beim allerersten Abtastwert reagiert. Belege hierfür konnten jedoch nicht gefunden werden.



(a) WinFACT



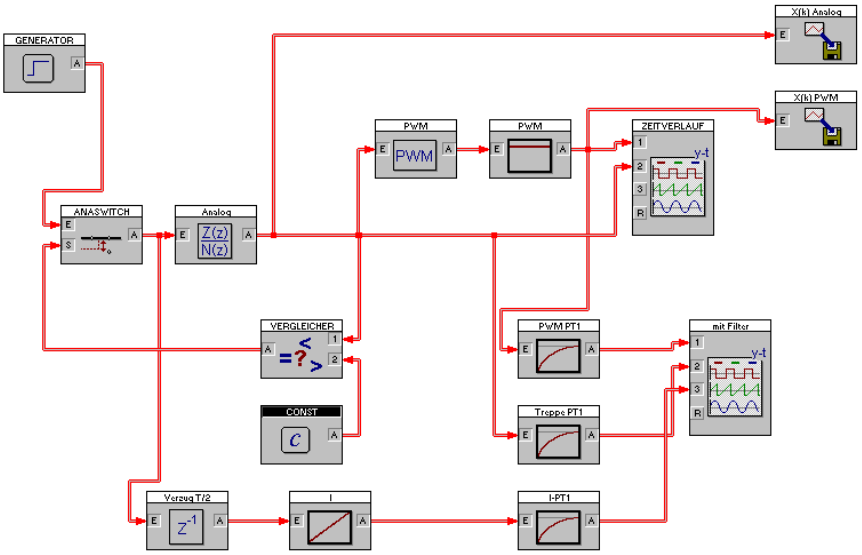
(b) Simulink

Abbildung 55: Diskrete periodische Folge – Signalverläufe

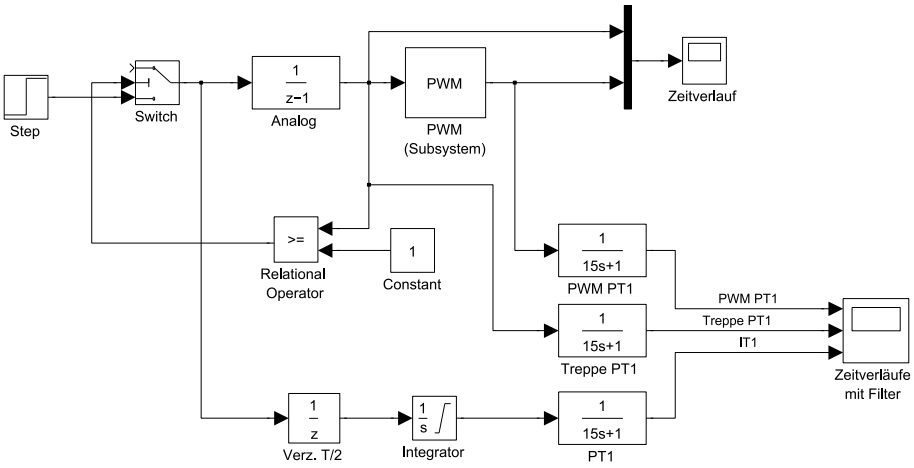
5.4.3 Diskrete Systeme und PWM

Um kontinuierliche Systeme ansteuern zu können, muss der digital berechnete Zahlenwert in ein analoges, auf kontinuierliche Systeme anwendbares Signal gewandelt werden. In der Industrie wird hierfür oftmals die Pulsweitenmodulation (PWM) als kostengünstige Variante eingesetzt. Ein analoger Ausgangswert, $x(k)$, wird in Form eines Pulses mit der Pulsweite $p(k)$ ausgegeben. Die Steuerperiode ist gleich der Abtastzeit T. Wird über das PWM-Signal der Mittelwert gebildet (z. B. durch eine träge Regelstrecke), so entspricht dieser Mittelwert dem Analogwert von $x(k)$.

Zur Simulation dieses Verhaltens wurden die Schaltungen in Abb. 56 entworfen. In ihr steuert ein Summierer (Block ist als „Analog“ bezeichnet), der ein kontinuierlich steigendes Treppensignal erzeugt, einen PWM-Block an. Dieser erzeugt ein pulsweitenmoduliertes Signal, dessen Mittelwert ein analoges Äquivalent zum Ausgangswert des Summierers darstellt.



(a) WinFACT



(b) Simulink

Abbildung 56: PWM-Simulation

Für Simulink wurde ein simples PWM-Subsystem erstellt, das den entsprechenden PWM-Block aus WinFACT/BORIS nachbildet. Ein solcher Block war nicht vorhanden, konnte aber rasch entwickelt und in einem im Modell eingebetteten Subsystem gekapselt werden. Zusätzlich zum PWM-Signal werden noch einige interessante Signale aus dem System über je ein Tiefpassfilter

1. Ordnung geleitet, um die Filterwirkung träger Systeme zu evaluieren. Das hier interessante PWM-Signal in Verbindung mit dem Ausgangswert des Summierers zeigt Abb. 57.

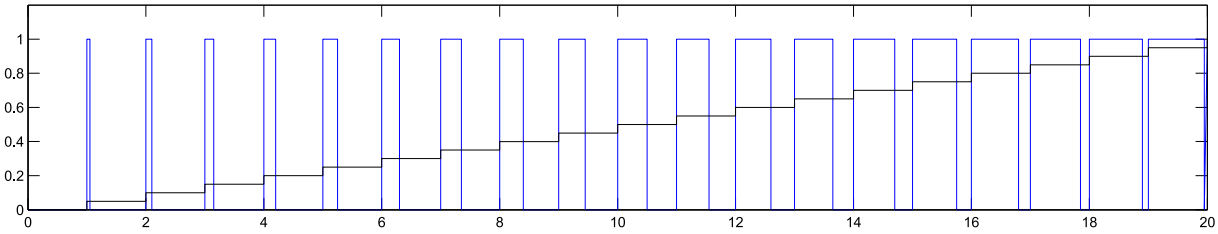


Abbildung 57: Pulsweitenmoduliertes Ausgangssignal

5.4.4 Diskreter Regler nach Takahashi

Zur Einstellung eines digitalen Regler kann der Reglerentwurf nach Takahashi herangezogen werden. Basierend auf der empirischen Einstellung nach Ziegler-Nichols (jedoch ohne das System in den schwingenden Zustand bringen zu müssen) erfolgt der Reglerentwurf durch Berücksichtigung der Abtastzeit und der Speicherung der Stellgröße im System [20].

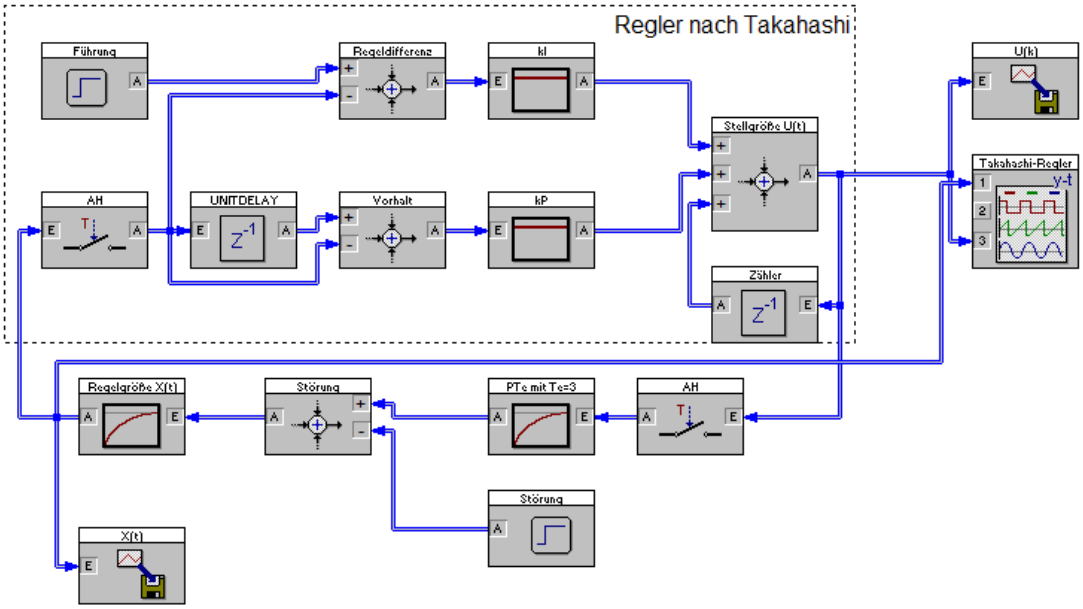


Abbildung 58: Takahashi-Regler – WinFACT/BORIS

Ein nach Takahashi entworfener Regler ist in Bild 58 bzw. Bild 59 dargestellt. Die Umsetzung in Simulink erfolgt ohne jede Probleme und entspricht genau der BORIS-Vorgabe.

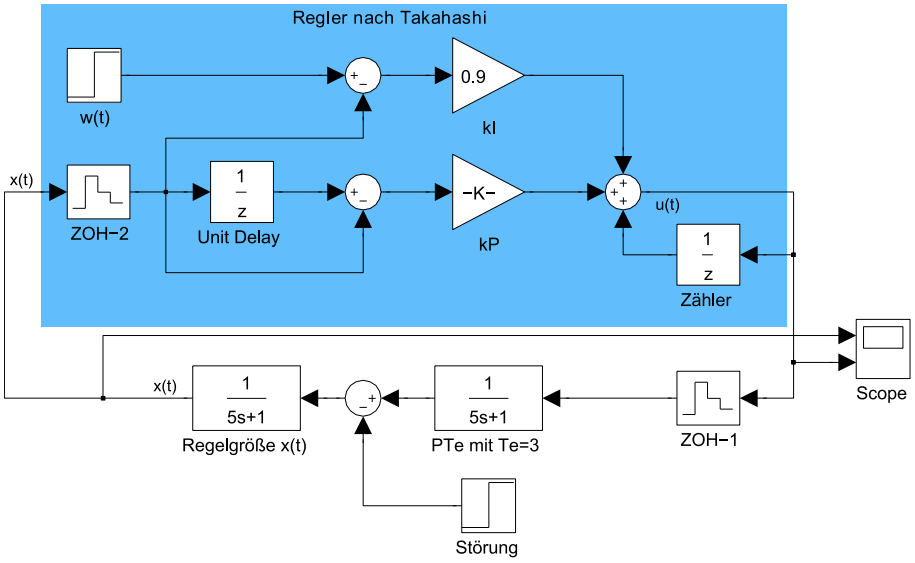


Abbildung 59: Takahashi-Regler – Simulink

Die Sprungantwort ist in Abb. 60 gezeigt – bei einer Zeit $t = 50$ wird das System mit einer Störung belastet, um das Störverhalten des Reglers zu evaluieren. Der Ausschlag in der Ausgangsgröße ist deutlich erkennbar.

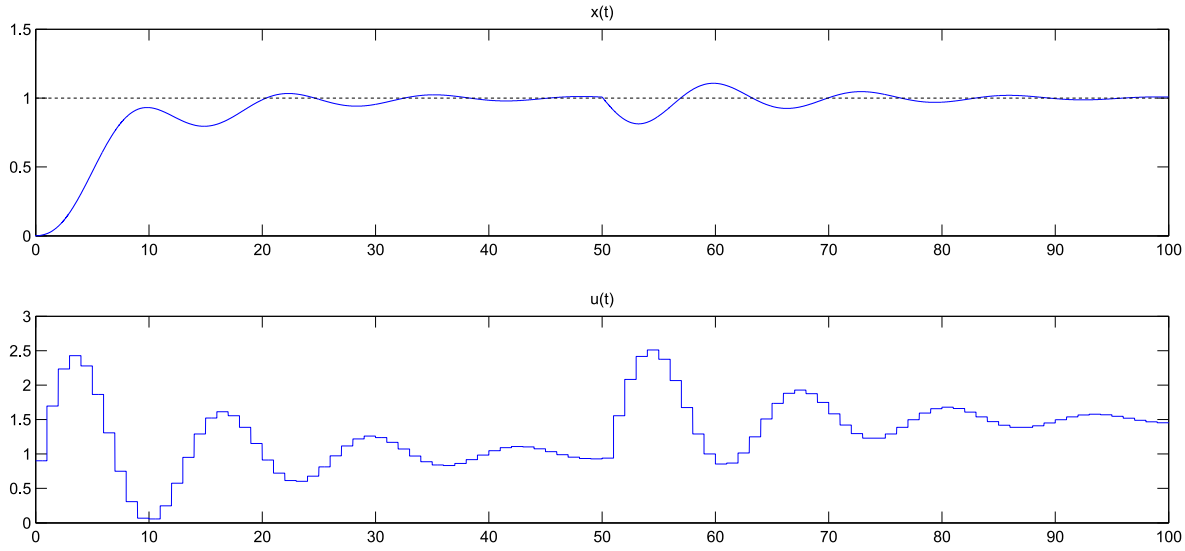


Abbildung 60: Takahashi – Signalverlauf mit Störung bei $t = 50$

5.4.5 FIR-Regelkreise und Dead-Beat-Regler

Diskrete Systeme, deren Impulsantwort endlich ist, werden als *Finite-Impulse-Response*-Systeme (FIR-Systeme) bezeichnet. Ein wesentlicher Vorteil (speziell für die Filtertechnik, vgl. [4]) ist, dass solche Systeme niemals instabil werden.

Für die Regelungstechnik von Bedeutung ist der auf FIR basierende diskrete Reglerentwurf nach dem *Dead-Beat*-Prinzip, der einen Regler mit der Zielsetzung dimensioniert, den Regelvorgang *zeitoptimal* auszuführen. Liegt die exakte z-Transformation einer Strecke n-ter Ordnung vor, so kann ein Dead-Beat-Regler entworfen werden, der nach n Schritten ausgeregelt hat (sprich die Regelaktivität nach n Schritten beendet hat).

Eines der Simulationsbeispiele ist in Bild 61 gezeigt. Diese Schaltung realisiert eine Gegenüberstellung des Dead-Beat-geregelten mit dem ungeregelten Fall.

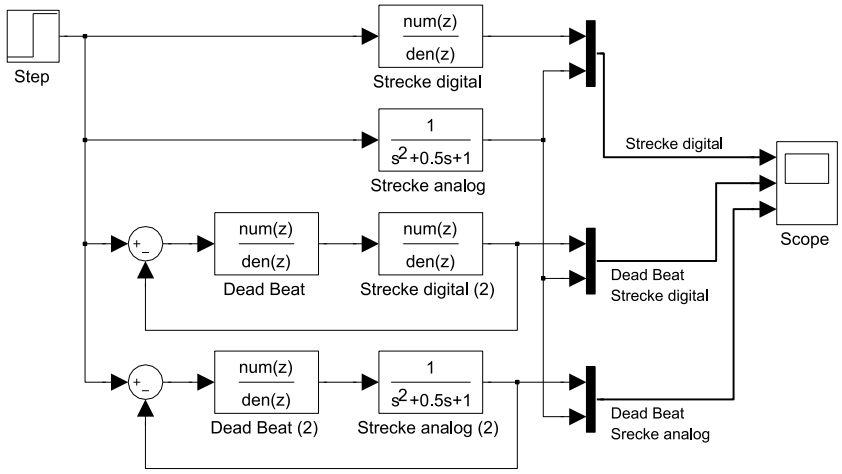


Abbildung 61: Dead-Beat-Regelung

Die Signalverläufe in Abb. 62 zeigen dabei:

- Die Sprungantwort der kontinuierlichen, ungeregelten Strecke zusammen mit der diskreten, ungeregelten Strecke.
- Die Sprungantwort des Dead-Beat-geregelten diskreten Systems
- Die Sprungantwort des Dead-Beat-geregelten kontinuierlichen Systems

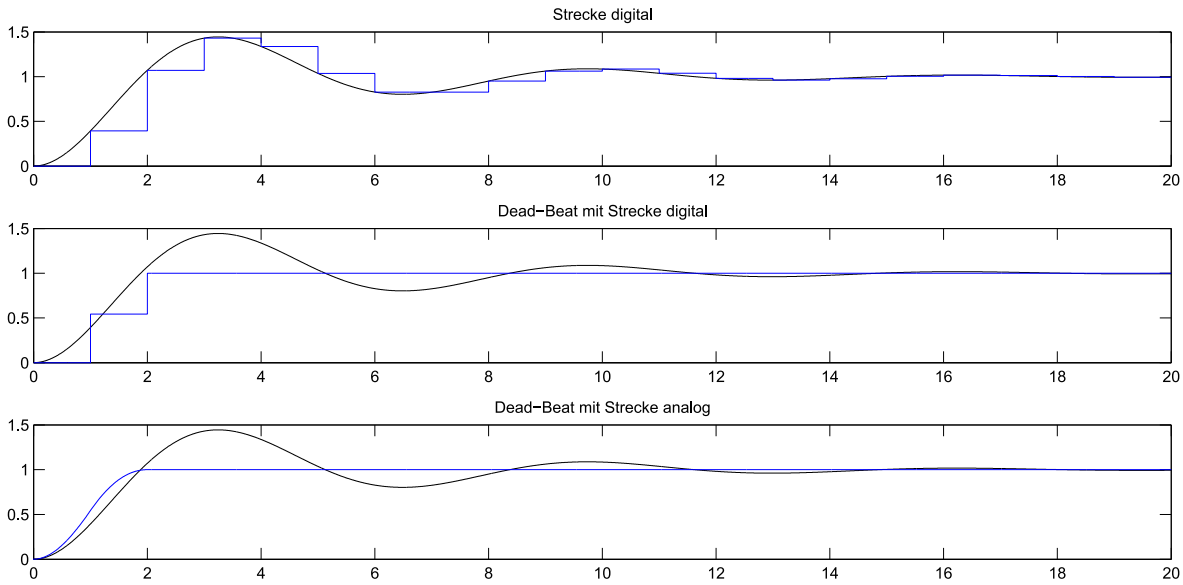


Abbildung 62: Signalverläufe einer Dead-Beat-Regelung an einem PT_2-Glied

Als Regelstrecke dient ein schwingungsfähiges PT_2-System. Wie die Signalverläufe belegen, regelt der Dead-Beat-Controller das System 2. Ordnung tatsächlich nach 2 Abtastschritten aus.

5.4.6 Vergleich WinFACT/BORIS mit MATLAB/Simulink

Zur Behandlung digitaler/diskreter Systeme und Regelkreise bieten beide Systeme ausreichende und weitgehend idente Blöcke an. Die einzige wesentliche Diskrepanz, die festgestellt werden

konnte, trat bei der Behandlung diskreter periodischer Folgen auf, bei denen sich die „Startzeitpunkte“ der jeweiligen Folge um einen Abtastschritt unterschieden. Die Ursache hierfür konnte jedoch nicht festgestellt werden. Umso interessanter wird dieser Punkt, wenn man abseits der Folgenbetrachtung die weiteren Simulationsbeispiele untersucht: Hierbei treten keine Verzögerungen im Anfangszeitpunkt auf. Eine weitere Untersuchung könnte daher in Richtung des Verhaltens bei Impulsantworten erfolgen.

Hinsichtlich Bedienung haben sich in diesen Beispielen Unterschiede bei der Darstellung der Signalverläufe aufgetan – WinFACT/BORIS beschränkt sich auf Zeitverläufe und Mehrfach-Zeitverläufe, die 3 bzw. mehrere Plots gleichzeitig darstellen können, allerdings im Vergleich zu Simulink mit eingeschränkter Flexibilität. Bei Simulink können Signalgruppierungen und Plot-Anzahl frei gestaltet und kombiniert werden, bei WinFACT/BORIS ist das in diesem Umfang nicht möglich.

6 Zusammenfassung und Ausblick

Anhand der vorgestellten Beispiele sind einige Charakteristika der beiden Probanden klar erkennbar. Prinzipiell ist aber festzustellen, dass alle hier behandelten Probleme der Regelungstechnik zweifelsohne in jedem der Softwaresysteme lösbar ist, was aufgrund der vergleichsweise geringen Komplexität der Modelle jedoch zu erwarten war. Dass Unterschiede in den Ausführungen auftreten, liegt in der Natur der Sache, kann allerdings auch durch die abweichenden Grundausrichtungen der Anwendungen begründet werden.

Trotz der Tatsache, dass sowohl WinFACT/BORIS als auch MATLAB/Simulink sowohl in der Ausbildung, als auch im tatsächlichen industriellen Umfeld in vielfältiger Weise und ausgiebig Anwendung finden, verfolgen die Softwaresysteme unterschiedliche Ziele: WinFACT/BORIS weist einen dedizierten Hang zur Benutzerinteraktion auf, während sich MATLAB/Simulink auf die Lösung des eigentlichen, mathematisch-physikalischen Problems beschränkt und durch die schiere Funktionsvielfalt besticht.

Besonders in der Lehre ist der Aspekt der Interaktion mit dem Modell von Bedeutung – in diesem Bereich zeigt WinFACT klare Vorteile. MATLAB/Simulink weist nur sehr eingeschränkte Möglichkeiten auf, das Modell während der Simulation zu beeinflussen. Mit dem Flexible Animation Builder lassen sich für BORIS Oberflächen gestalten, die Regelkreise in 2D visualisieren und steuerbar machen. Einzig bei der grafischen Darstellung ist mit Simulink 3D ein leistungsfähiges Analogon vorhanden, das jedoch ein gewisses Mindestmaß an Einarbeitung und Entwicklungszeit voraussetzt, während im FAB eine GUI durch vergleichsweise geringen Aufwand entsteht.

Im Gegenzug bietet Simulink jedoch eine technische/mathematische Flexibilität, an die WinFACT/BORIS nur schwer anschließen kann. Die Leistungsfähigkeit des Mathematik-Kerns und eine reichliche Anzahl an fertigen Erweiterungsmodulen reduzieren viele aufwändige Probleme auf das Einfügen einiger weniger Funktionsblöcke. Die starke numerische Orientierung führt zudem auf eine (zum Teil beträchtliche) Verringerung der Rechenzeit, was besonders am Beispiel der Güteoptimierung nach den Integralkriterien deutlich wurde.

Als Conclusio kann festgehalten werden, dass sowohl WinFACT/BORIS als auch MATLAB/-Simulink für den Unterricht (unbestreitbar) geeignet sind. Zu beachten ist allerdings, dass diese Schlussfolgerung rein aufgrund der hier bearbeiteten Beispiele gezogen wird – viele weitere in der Regelungstechnik maßgebliche Themen konnten aus dem Grund mangelnder zeitlicher Ressourcen nicht behandelt werden. Für künftige Untersuchungen hinsichtlich der Leistungsfähigkeit der beiden Systeme bieten sich u. a. Themen aus den Gebieten der Zustandsraumdarstellung, der nichtlinearen Regelungen oder der Systemidentifikation an. Auch eine Evaluierung der Anbindungsmöglichkeiten an externe Hardwarekomponenten (z. B. Mess-/Steuermodule) wäre von Interesse. Letzteres könnte besonders für praktische Demonstrationen im Unterricht eine Bereicherung darstellen.

Literaturverzeichnis

[1] Angermann, A., M. Beuschel, M. Rau und U. Wohlfarth: *MATLAB© – Simulink© – Stateflow©*. Oldenbourg Wissenschaftlicher Verlag, München, 6. Auflage, 2009.

[2] Föllinger, O.: *Laplace-, Fourier- und z-Transformation*. Hüthig Verlag, Heidelberg, 9. Auflage, 200.

[3] Föllinger, O.: *Regelungstechnik. Einführung in die Methoden und ihre Anwendung*. Hüthig Verlag, Heidelberg, 10. Auflage, 2008.

[4] Grünigen, Daniel Ch. v.: *Digitale Signalverarbeitung*. Fachbuchverlag Leipzig im Carl Hanser Verlag, München, Wien, 3. Auflage, 2004.

[5] Haager, W.: *Regelungstechnik*. öbv & hpt VerlagsgmbH & Co. KG, Wien, 2. Auflage, 2004.

[6] ISO/IEC: *ISO/IEC 14772-1:1997: Information technology – Computer graphics and image processing – The Virtual Reality Modeling Language (VRML) – Part 1: Functional specification and UTF-8 encoding*. International Organization for Standardization/International Electrotechnical Commission, 1997.

[7] ISO/IEC: *ISO/IEC 14772-2:2004: Information technology – Computer graphics and image processing – The Virtual Reality Modeling Language (VRML) – Part 2: External authoring interface (EAI)*. International Organization for Standardization/International Electrotechnical Commission, 2004.

[8] Kahlert: *Ingenieurbüro Dr. Kahlert – Software-Engineering & Automatisierungstechnik*. `http://www.kahlert.com` (04.02.2010).

[9] Kahlert: *WinFACT 7 – BORIS 7.1.1.307 Hilfedatei*. Ingenieurbüro Dr. Kahlert, 2005.

[10] Kahlert, J.: *Einführung in WinFACT*. Carl Hanser Verlag, München, 2009.

[11] Lutz, H. und W. Wendt: *Taschenbuch der Regelungstechnik mit MATLAB und Simulink*. Wissenschaftlicher Verlag Harri Deutsch, Frankfurt am Main, 7. Auflage, 2007.

[12] MathWorks: *The MathWorks - MATLAB and Simulink for Technical Computing*. `http://www.mathworks.com` (04.02.2010).

[13] MathWorks: *MATLAB 7.8.0 (R2009a) Student Version Help File*. The MathWorks, Inc., 2009.

[14] Papula, L.: *Mathematik für Ingenieure und Naturwissenschaftler – Band 2. Ein Lehr- und Arbeitsbuch für das Grundstudium*. Vieweg+Teubner Verlag, Wiesbaden, 11. Auflage, 2008.

[15] Reuter, M. und S. Zacher: *Regelungstechnik für Ingenieure*. Friedr. Vieweg + Sohn Verlag, Wiesbaden, 11. Auflage, 2004.

[16] Seifart, M.: *Analoge Schaltungen*. Verlag Technik, Berlin, 6. Auflage, 2003.

[17] Tietze, U. und Ch. Schenk: *Halbleiter-Schaltungstechnik*. Springer Verlag, Berlin, 12. Auflage, 2002.

[18] Tröster, F.: *Steuerungs- und Regelungstechnik für Ingenieure*. Oldenbourg Wissenschaftsverlag, München, 2. Auflage, 2005.

[19] Unbehauen, H.: *Regelungstechnik I*. Vieweg Verlag, Wiesbaden, 12. Auflage, 2002.

[20] Unbehauen, H.: *Regelungstechnik II. Zustandsregelungen, digitale und nichtlineare Regelsysteme*. Vieweg+Teubner Verlag, Wiesbaden, 9. Auflage, 2009.

[21] Weisstein, E. W.: *Fast Fourier Transform*. From MathWorld – A Wolfram Web Resource. `http://mathworld.wolfram.com/FastFourierTransform.html` (08.01.2010).