Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Dennis Stark

Entwicklung und Implementierung eines modellbasierten Reglers in eine C-Code-basierte Software im Automotive Kontext

Fakultät Technik und Informatik
Department Maschinenbau und Produktion

Faculty of Engineering and Computer Science
Department of Mechanical Engineering and Production Management

Dennis Stark

Entwicklung und Implementierung eines modellbasierten Reglers in eine C-Code-basierte Software im Automotive Kontext

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Konstruktionstechnik und Produktentwicklung im Maschinenbau
am Department Maschinenbau und Produktion
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

in Zusammenarbeit mit:
Vibracoustic SE & Co. KG
System Engineering
Lotsekai 10
21079 Hamburg

Erstprüfer/in: Prof. Dr. Thomas Frischgesell
Zweitprüfer/in: Vincent Poncet

Abgabedatum: 11.12.2024

Zusammenfassung

Dennis Stark

Thema der Masterthesis

Entwicklung und Implementierung eines modellbasierten Reglers in eine C-Code-basierte Software im Automotive Kontext

Stichworte

Magnetventil, Stromregelung, Reglerauslegung, Model-Based Design, Software-Entwicklung, Luftfedersystem, Simulation

Kurzzusammenfassung

Die Thesis befasst sich mit dem Aufbau und der Dimensionierung eines Stromreglers zur Schaltung eines Magnetventils in einem Model-Based Design. Es wird ein grundlegendes Modell eines PID-Reglers in *Simulink* aufgebaut. Des Weiteren wird der Regelkreis zur Dimensionierung des Reglers mit einem selbst modellierten PWM-Generator und dem Magnetventil als *Simscape*-Modell aufgebaut. Zur Dimensionierung der Regelparameter wird die Methode der Summenzeitkonstante verwendet. Der iterativ ermittelte PI-Regler wird anschließend über den Seriencode Generator *TargetLink*, in einen C-Code konvertiert und in die Software integriert und am Prüfstand validiert. Abschließend wird Simulation und Realität verglichen und ein allgemeiner Vergleich zwischen dem Model-Based Design und der manuellen Code-Entwicklung gezogen.

Dennis Stark

Title of the paper

Development and implementation of a model-based controller in a C-code-based software in an automotive context

Keywords

Solenoid valve, current control, controller design, model-based design, software development, air spring system, simulation

Abstract

The thesis deals with the design and dimensioning of a current controller for switching a solenoid valve in a model-based design. A basic model of a PID controller is built in *Simulink*. Furthermore, the control loop, which is used for dimensioning the controller, is set up with a self-modelled PWM generator and the solenoid valve as a *Simscape* model. The sum time constant method is used to dimension the control parameters. The iteratively determined PI controller is then converted into C code using the *TargetLink* production code generator, integrated into the software and validated on the test bench. Finally, simulation and reality are compared and a general comparison is drawn between model-based design and manual code development.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ABS	Antiblockiersystem
ASIL	*Automotive Safety Integrity Level*
ASW	Anwendungssoftware
AUTOSAR	*Automotive Open System Architecture*
BSW	Basissoftware
CDD	*Complex Device Driver*
D-Glied	Differentialglied
DT1-Glied	Differentialglied mit Verzögerung erster Ordnung
EHB	elektrohydraulische Bremse
I-Glied	Integralglied
ICtrlrMeas	*Current Control Measurement*
ICtrlrSp	*Current Control Setpoint*
ICtrlrSSPSts	*Current Control Setpoint Status*
MBD	*Model-Based Design*
MCAL	*Microcontroller Abstraction Layer*
MISRA	*Motor Industry Software Reliability Association*
NC	*Normally Closed*
NO	*Normally Open*
P-Glied	Proportionalglied
PHndlgActrReq	*Pressure Handling Actuator Request*
PT1-Glied	Verzögerungsglied erster Ordnung
PWM	Pulsweitenmodulation
PwmICtrlr	*PWM Current Controller*
PwrSply	*Power Supply*
QM	Qualitätsmanagement
RTE	*Runtime Environment*
SIO	*Safety Input Output*

Symbolverzeichnis

Symbol	Bezeichnung	Dimension
$A(\omega)$	Amplitudengang	/
A_dB(ω)	Amplitudengang in Dezibel	dB
$A_0(\omega)$	Amplitudengang des offenen Regelkreises	/
$A_M(\omega)$	Amplitudengang des Messgliedes	/
$A_R(\omega)$	Amplitudengang des Reglers	/
$A_S(\omega)$	Amplitudengang der Regelstrecke	/
A1	Erste Fläche zur Methode der Summenzeitkonstante	/
A2	Zweite Fläche zur Methode der Summenzeitkonstante	/
af	Abtastfrequenz zur Stromrückmessung	Hz
Amp	Amplitude des Dreieckssignales zur PWM-Generierung	/
C	Kapazität des Kondensators	F
D	Durchmesser der Spule	m
dV	Spannungsdifferenz bei simuliertem Spannungsabfall	V
e	Regeldifferenz	A
f_c	Grenzfrequenz des Tiefpassfilters	Hz
fail	Fehlerzähler im Zustandsdiagramm	/
fPWM	Grundfrequenz des PWM-Signales	Hz
G(s)	Übertragungsfunktion im Bildbereich	/
$G_0(s)$	Übertragungsfunktion des offenen Regelkreises	/
$G_M(s)$	Übertragungsfunktion des Messgliedes	/
$G_R(s)$	Übertragungsfunktion der Regelstrecke	/
$G_S(s)$	Übertragungsfunktion des Reglers	/
H	Magnetische Feldstärke	A/m
Hold	*Hold*-Strom	A
I	Stromstärke	A
Im	Imaginärteil der Übertragungsfunktion	/

Ireal	Direkt gemessene Stromstärke in der Simulation	A
Isoll	Sollstrom aus dem Zustandsdiagramm	A
K	Verstärkung	/
K_D	Differentierbeiwert	/
K_I	Integrierbeiwert	/
K_P	Proportionalbeiwert	/
K_S	Verstärkung der Regelstrecke	/
l	Länge der Spule	m
L	Induktivität	H
Limit	Grenzwert in der *Push-Phase*	A
N	Windungsanzahl der Spule	/
N_{PT1}	Filterkoeffizient des PT1-Gliedes	1/s
Push	*Push-Strom*	A
PushMax	Maximal mögliche Stromstärke zur Ventilschaltung	A
r	Rückführgröße	A
R	Gesamtwiderstand	Ω
Re	Realteil der Übertragungsfunktion	/
RC	Vorwiderstand des Kondensators	Ω
RL	Vorwiderstand der Spule	Ω
RS	Widerstand des elektrischen Schalters	Ω
RSenseDV	Widerstand des Shunt	Ω
s	Komplexe Operationsvariable	/
STime	Schaltzeit zwischen *Push* und *Hold*	s
T	Verzögerungszeit	s
T_g	Ausgleichszeit	s
T_n	Nachhaltzeit	s
T_t	Totzeit	s
T_U	Verzugszeit	s

T_V	Vorhaltzeit	s
T_Σ	Summenzeitkonstante	s
Toleranz	Toleranzbereich für die *Push-Phase*	/
u	Steuergröße	%
U	Spannung	V
u(t)	Eingangssignal im Zeitbereich	/
U(s)	Eingangssignal im Bildbereich	/
uD	Steuergröße des D-Gliedes	A
uDT	Steuergröße des DT1-Gliedes	A
uI	Steuergröße des I-Gliedes	A
uP	Steuergröße des P-Gliedes	A
US	Grenzwert für den Unterschwinger in der Regelung	A
v(t)	Ausgangssignal im Zeitbereich	/
V(s)	Bildfunktion	/
VK	Spannungsverstärkung für Voltage-Controlled Voltage Source	V
VHold	Haltezeit des simulierten Spannungsabfalles	s
VTime	Zeitpunkt des simulierten Spannungsabfalles	s
w	Führungsgröße	/
x	Regelgröße	A
y	Stellwert	/
δ	Dämpfungsgrad	/
$\varphi(\omega)$	Phasengang	°
$\varphi_0(\omega)$	Phasengang des offenen Regelkreises	°
$\varphi_M(\omega)$	Phasengang des Messgliedes	°
$\varphi_R(\omega)$	Phasengang des Reglers	°
$\varphi_S(\omega)$	Phasengang der Regelstrecke	°
υ_m	Überschwingweite	/
ω	Kreisfrequenz	1/s

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

1 Einleitung

In der Software-Entwicklung und allgemeinen Programmierung haben sich über die Jahre stetig der Funktionsumfang der Sprachen und die Code-Generierung optimiert, um der gleichzeitig ansteigenden Komplexität von Systemen entgegenzuwirken. Während zu Beginn die textbasierte Programmierung vereinfacht wurde, gibt es mittlerweile Möglichkeiten über neue graphische Systeme Code zu entwickeln. *Model-Based Design*, auch MBD abgekürzt, ist einer dieser modernen Entwicklungsansätze, bei dem ein System oder Prozess nicht als reiner Code, sondern als ein mathematisch visuelles Modell entworfen und simuliert werden kann. Durch die Möglichkeit der Simulation ist eine Validierung und Verifikation an einem digitalen Zwilling möglich. Die frühzeitige Erkennung von Schwachstellen und Fehlern ohne die Verwendung von physischen Prototypen reduziert demnach die Anzahl an Entwicklungszyklen und somit die Entwicklungszeit und Kosten. (*Warum sollten Sie Model-Based Design einführen?*, 2024f) Während zum Aufbau der Simulation noch iterative Schritte im Zusammenhang mit dem realen System nötig sind, kann durch die finale Simulation die allgemeine Prozessiteration rein virtuell vorgenommen werden. Um allerdings MBD in der Software-Entwicklung zu nutzen, muss die Konvertierung des Modelles in einen Code erfolgen. Hierfür gibt es bereits Möglichkeiten der automatisierten Code-Generierung, wovon eine in dieser Arbeit verwendet und validiert wird.

1.1 Motivation

Die Masterarbeit wird im Bereich *System Engineering* des Unternehmens *Vibracoustic SE & Co. KG* erarbeitet. Die Abteilung befasst sich mit Luftfederung- und *Leveling*-Systemen für Fahrzeuge. Die Software-Architektur im Automobilbereich weist infolge von langjähriger Weiterentwicklung eine hohe Komplexität auf, weshalb nach Alternativen zum textbasierten Coding in der Programmierung gesucht wird. Während eine standardisierte Architektur bereits seit langem Stand der Technik ist, wird die modell-basierte Programmierung im Unternehmen hauptsächlich zur Simulation und Verifikation eingesetzt und nur wenig in der direkten Entwicklung des Software-Codes. Das Ziel der Arbeit ist zum einen der Aufbau einer Prozesskette zum Einsatz eines *Simulink*-Modells in der Softwareentwicklung und zum anderen der Vergleich zwischen textbasiertem Coding und MBD. Zur grundlegenden Architektur und Integration von einfachen Operationen wird MBD dabei bereits verwendet, allerdings noch nicht zur Integration von Basisfunktionen, wie zum Beispiel die Ansteuerung der Aktuatoren. Das Ziel ist es zu zeigen, dass durch die Verwendung von MBD eine einfachere Parametrisierung und Anpassung von Funktionen möglich ist. Hierzu wird eine geeignete Funktion benötigt, welche sinnvoll in einem MBD aufgebaut werden kann und die Möglichkeit der Integration in eine bereits bestehende Software besitzt. Somit ergibt sich die im nachfolgenden Abschnitt vorgestellte Aufgabenstellung.

1.2 Aufgabenstellung

Die folgende Masterarbeit befasst sich mit der Entwicklung und Auslegung eines Reglers im Automobilbereich bei Verwendung eines *Model-Based Designs*. Der Regler soll in der Softwareentwicklung eingesetzt werden und dient zur Ansteuerung eines Magnetventils durch die Anpassung der Stromstärke als Regelgröße. Dabei dient ein PWM-Signal als Stellgröße.

Zu Beginn wird der elektrische Aufbau des Magnetventils als virtuelles Modell in *Matlab Simulink* aufgebaut. Anschließend wird der PWM-Regler aufgebaut und gemeinsam mit dem Modell des Ventiles in einen Regelkreis integriert. Am finalen Modell sollen daraufhin verschiedene Reglerarten dimensioniert, analysiert und miteinander verglichen werden, um so in einem iterativen Prozess den optimalen Regler für die Strecke zu finden.

Die Software wird in C-Code implementiert und nach dem Prinzip von AUTOSAR aufgebaut, welches eine allgemeine Software-Architektur im Automobilbereich darstellt. Folglich muss der final ausgelegte Regler in *Simulink* als C-Code generiert und in die Software integriert werden. Daraufhin kann die Software auf die Steuereinheit geladen werden. Zur Verifizierung der Simulation dient ein *Hardware-in-the-Loop* Aufbau. Dabei wird die Steuereinheit an einem Prüfstand an die elektrischen Komponenten des Fahrzeugs angeschlossenen, wodurch das Fahrzeug selbst bei den Tests nicht benötigt wird.

In dieser Arbeit wird die Regelung nur für ein Magnetventil ausgelegt. Allerdings soll das finale Modell künftig zur schnellen Auslegung von weiteren Magnetventilen verwendet werden und somit die reine Code-basierte Programmierung ersetzen. Folglich wird ein Vergleich zwischen *Model-Based Design* und reiner Code-basierter Programmierung aufgebaut und konkret die Vor- und Nachteile in der Anpassung von Reglern aufgezeigt.

1.3 Herangehensweise

In den folgenden Kapiteln wird zunächst die Theorie behandelt. Dabei wird auf die Grundlagen zu den Thematiken Magnetventile und Regelung sowie Software-Architektur eingegangen. Abschließend wird der aktuelle Stand im Unternehmen zur Integration von MBD und der momentanen Art der Stromregelung näher erläutert. Im darauffolgenden Abschnitt wird der Regelkreis zur Dimensionierung und Simulation des Stromreglers entwickelt. Im ersten Schritt werden die einzelnen Systeme des Regelkreises aufgebaut. Daraufhin wird der Regler über die Sprungantwort und verschiedene mathematische Faustformel-Verfahren ausgelegt und in der Simulation analysiert. Zusätzlich zur mathematischen Ermittlung der Regelparameter wird in einem iterativen Prozess innerhalb der Simulation eine weitere Anpassung am Regler vorgenommen. Anschließend findet die Anpassung des Modells zur Generierung des C-Codes und zur Integration in die Software statt. Die finale Software wird daraufhin auf eine Steuerein-

heit geladen und am Prüfstand validiert. Dabei wird sowohl die reine Funktionalität als auch die Parametrisierung des Reglers betrachtet. Des Weiteren kann in einem erneuten Iterationsprozess sowohl der Regler als auch die Simulation anhand des Verhaltens des realen Ventils angepasst werden. Am resultierenden Regler wird abschließend das Regelverhalten analysiert und eine allgemeine Bewertung der Reglerentwicklung über MBD gezogen. Final wird ein Vergleich zwischen MBD und reinem Coding dargelegt, bei dem die verschiedenen Aspekte von der Entwicklung bis hin zur Anpassung und Verwendung in anderen Systemen betrachtet werden.

2 Theorie

In diesem Kapitel werden die Grundlagen, welche zum Verständnis der darauffolgenden Ausarbeitung benötigt werden, erläutert. Zu Beginn wird der Aufbau und die Funktion eines Magnetventils erläutert. Daraufhin wird die Ansteuerung über die Pulsweitenmodulation und allgemeine Grundlagen im Bereich der Regelungstechnik beschrieben. Zum Abschluss wird der Aufbau der Software-Architektur und der aktuelle Stand im Unternehmen dargelegt.

2.1 Aufbau und Funktionsweise eines Magnetventils

„Elektromagnete oder Magnetventile werden sowohl als schaltende Stellglieder, z.B. Einspritzventile, Hydraulikventile für automatische Getriebe oder ABS, sowie als proportional wirkende Stellglieder, z.B. stellbare Stoßdämpfer, elektrohydraulische Bremse (EHB) eingesetzt“ (Isermann, 2006, S. 400). In dieser Arbeit werden im Fahrzeugsystem schaltende Magnetventile zur Ansteuerung von pneumatischen und hydraulischen Aktuatoren betrachtet. Der grundlegende Aufbau eines Magnetventils kann mit wenigen Komponenten beschrieben werden.

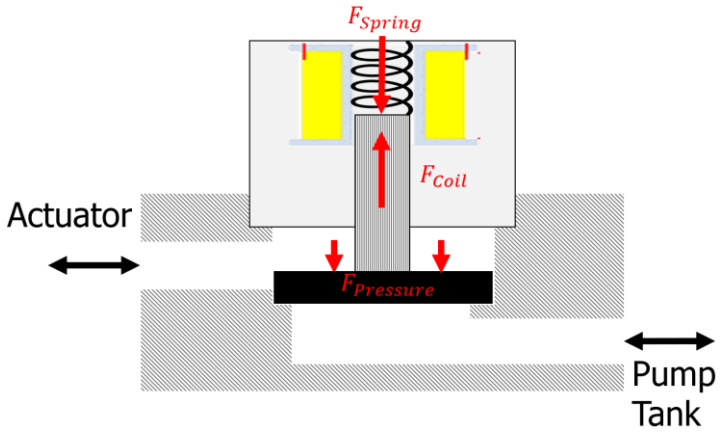


Abbildung 1: Aufbau eines NC Magnetventils am Beispiel eines Luftfeder-Ventils mit Darstellung der Kräfte (*Vibracoustic* intern)

Die Hauptkomponente stellt die elektromagnetische Spule dar. Diese wird über eine angelegte Spannung aufgeladen, wodurch ein magnetisches Feld im Kern der Spule erzeugt wird. Die magnetische Feldstärke H einer Zylinderspule errechnet sich wie folgt:

$$H = \frac{I * N}{\sqrt{D^2 + l^2}} \tag{1}$$

Formel 1: Berechnung der magnetischen Feldstärke

Während die Windungszahl N sowie der Durchmesser D und die Länge l der Spule konstant bleiben, kann die Stromstärke I und folglich die Feldstärke über die Auf- und Entladung der Spule angepasst

werden. Die zeitliche Abhängigkeit zwischen der über der Spule anliegenden Spannung *U* und der Stromstärke ergibt sich über die folgende Formel:

$$U = i * R + L * \frac{di}{dt} \tag{2}$$

Formel 2: Abhängigkeit zwischen der Spannung und der Stromstärke an der Spule

Die Stromstärke nähert sich, wie in Abbildung 2 zu erkennen, logarithmisch einem Maximum an, welches über die anliegende Spannung und den Gesamtwiderstand *R* definiert wird. Der Anstieg der Stromstärke wird über die Induktivität *L* der Spule definiert.

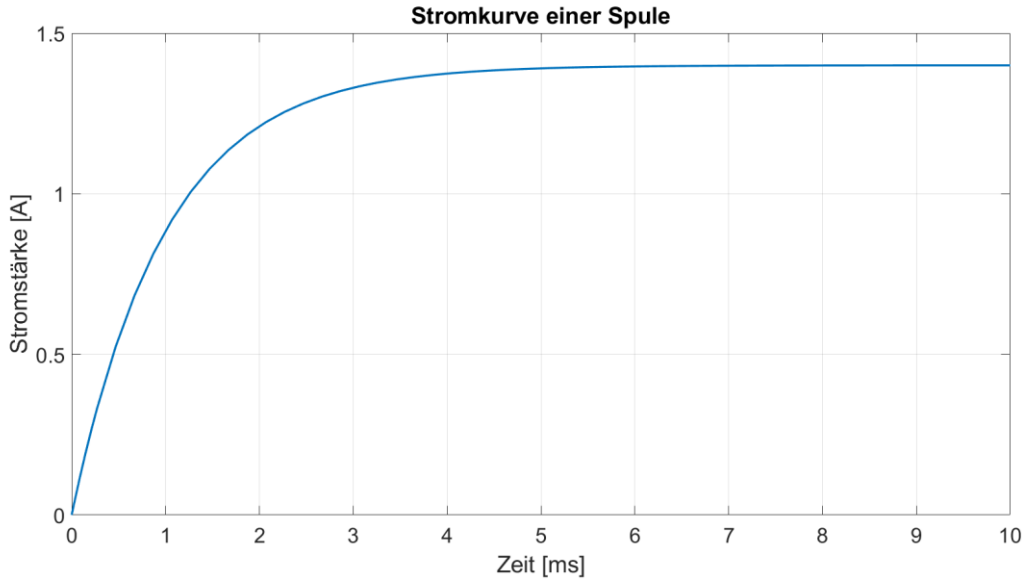


Abbildung 2: Stromverlauf bei einer Spannung von 14 V einem Widerstand von 10 Ω und einer Induktivität von 10 mH

Das magnetische Feld wirkt eine dazu proportionale Kraft auf einen Anker aus. Als Anker wird ein magnetischer Kern im Inneren der Spule bezeichnet, welcher zum Öffnen oder Schließen des Ventiles dient. Der Anker wird im Ruhezustand über den Druck einer anliegenden Feder, wie in Abbildung 1 dargestellt, je nach Betriebsart in einem der beiden möglichen Ventilzustände gehalten. Hierbei wird zwischen stromlos geschlossenen (engl. Normally Closed - NC), welches in Abbildung 3 dargestellt wird, und stromlos offenen (engl. Normally Open - NO) Magnetventilen unterschieden. Durch den sich bewegenden Eisenkern im Magnetfeld der Spule kommt es zu einer elektromagnetischen Induktion. Dabei verändert sich der magnetische Widerstand im System, weshalb diese Selbstinduktion in der zuvor beschriebenen Beziehung (Formel 2) zwischen Stromstärke und Spannung berücksichtigt werden muss. In der folgenden Formel 3 wird die Position des Ankers mit *x* dargestellt. (Balakrishnan & Kumar N, 2015)

$$U = i * R + L(i,x) * \frac{di}{dt} + i * \frac{dL(i,x)}{dx} * \frac{dx}{dt} \tag{3}$$

Formel 3: Integration der Selbstinduktion in die Abhängigkeit zwischen Spannung und Stromstärke an der Spule

Bei Betrachtung der Stromstärke über der Zeit zeigt sich während der Aufladung ein Anstieg bis zu einem ersten Spitzenwert. Ab diesem Punkt reicht die magnetische Feldstärke aus, um den Anker anzuheben, wobei jedoch durch die Bewegung der magnetische Widerstand erhöht wird und somit die Stromstärke kurzzeitig abfällt. Sobald der Anker in seiner Endposition ist, pendelt sich die Stromstärke auf den definierten Endwert ein. Dieser Verlauf kann in der späteren Auswertung, zum Beispiel in Abbildung 53, betrachtet werden.

Am Kopf des Ankers befindet sich eine Ventilmembran, welche den Zufluss des Mediums im geschlossenen Zustand abdichtet. Hierbei gibt es zwei Bauarten. Zum einen kann die Ventilmembran fest mit dem Anker verbunden sein. Somit wird direkt über die Bewegung des Ankers das Ventil geöffnet und geschlossen. Zum anderen kann die Ventilmembran frei auf dem Zufluss liegen und beim Öffnen des Ventiles durch den Druckunterschied in den Leitungen angehoben werden. Während die erste Variante meist in Systemen mit niedrigen Durchflussraten eingesetzt wird, wird die zweite Variante häufig bei Systemen mit einem hohen Betriebsdruck verwendet.

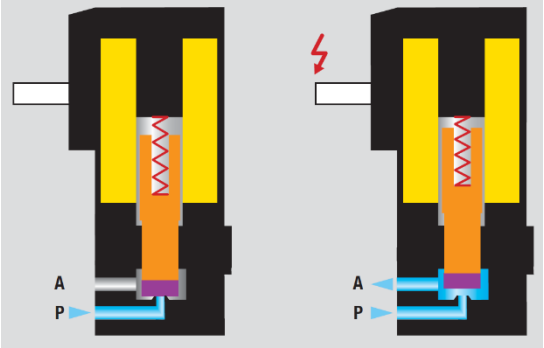


Abbildung 3: Schaltung eines NC Magnetventils (Quelle: https://www.norgren.com/de/service/blog/everything-you-need-to-know-about-solenoid-valves-part-1)

Zur Verringerung des Stromverbrauches kann die Stromstärke über die Auf- und Entladungszeiten der Spule so eingestellt werden, dass nur ein definierter Wert zwischen 0 und dem Maximum erreicht wird. Zur Steuerung der Auf- und Entladung wird die Pulsweitenmodulation eingesetzt.

2.2 Pulsweitenmodulation

Die Pulsweitenmodulation dient als Schnittstelle zwischen analogen und digitalen Signalen. Dabei wird ein analoges Signal auf zwei diskreten Zuständen, *high* und *low*, abgebildet. Auf der zeitlichen Ebene wird das Signal in einer Pulsform übertragen, wodurch ein Rechtecksignal erzeugt wird. (Hüning, 2016, S. 157) Während die Frequenz des PWM-Signales konstant bleibt, kann die Breite des Pulses, also die high-Phase, variiert werden. Somit ist das Signal zwar wertediskret, allerdings auf zeitlicher Ebene kontinuierlich. Die zwei Zustände können im System über verschiedene physikalische Signale, wie zum Beispiel die Eingangsspannung, beschrieben werden, werden jedoch auf logischer Ebene, wie in Abbil-

dung 4 zu sehen ist, mit „1“ für *high* und „0“ für *low* definiert. Die über eine PWM-Schnittstelle übertragenden Informationen werden über die Pulsdauer dekodiert. Das Verhältnis der *High*-Phase zur Periodendauer wird als Tastgrad oder auch *Duty-Cycle* bezeichnet. (*PWM - Pulsweitenmodulation (PDM PLM PBM)*, 2024b) Bei Verwendung der logischen Zustandswerte beschreibt der Tastgrad somit einen relativen Wert zwischen 1 und 0. Bei der Pulsweitenmodulation wird folglich als Eingangssignal ein Wertebereich und ein relativer Wert vorgegeben, wodurch als Ausgangssignal ein Rechtecksignal mit einem errechneten Tastgrad vorliegt.

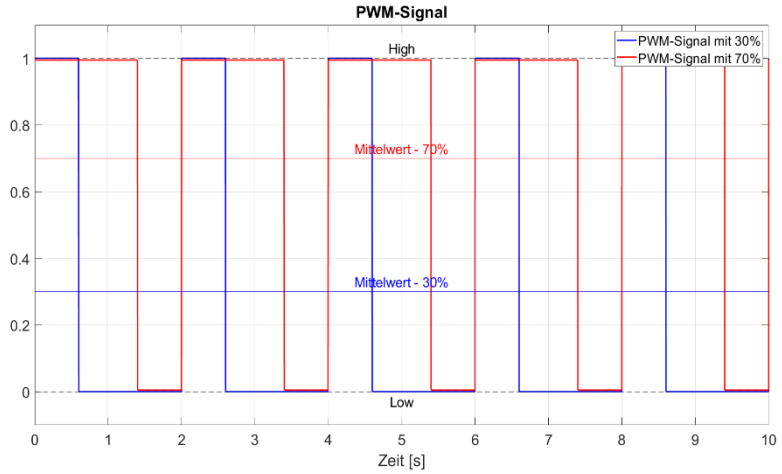


Abbildung 4: Darstellung der Pulsweitenmodulation anhand von einem Signal mit einem Tastgrad von 30% und einem Signal mit einem Tastgrad von 70%

Zur Erzeugung des PWM-Signales wird zunächst ein Sägezahnsignal mit einer konstanten Frequenz und Amplitude benötigt. Der prozentuale Eingangswert wird an die Amplitude des Sägezahnsignales angepasst. Anschließend werden beide Signale addiert und über einen sogenannten Komparator mit 0 verglichen. Wenn das resultierende Signal über 0 liegt, wird *high* ausgegeben und wenn es unter 0 liegt, wird *low* ausgegeben.

Das Verfahren der Pulsweitenmodulation wird neben der Übertragung von Informationen ebenfalls zur Steuerung in vielen technischen Systemen eingesetzt. Dabei liegt ein Vorteil bei der einfachen Realisierung des PWM-Signales. Der PWM-Generator für das Sensorsignal kann, wie vorhin beschrieben, leicht mit Hilfe eines Komparators aufgebaut werden. Auf der Empfängerseite gibt es bei den meisten Microcontrollern bereits ein Modul zur Dekodierung von PWM-Signalen. (Hüning, 2016, S. 158) In dieser Arbeit wird jedoch die Dekodierung auf Empfängerseite nicht benötigt. Über das erzeugte PWM-Signal wird die Auf- und Entladezeit der Spule direkt gesteuert, worüber wiederum die resultierende Stromstärke definiert werden kann. Mit einem Tastgrad von 50 % wird folglich nur die Hälfte der maximalen Stromstärke an der Spule erreicht.

Ein Nachteil der PWM-Modulation stellt die notwendige Bandbreite des Übertragungsmediums dar. Je nach Frequenz des PWM-Signales und Bandbreite des Übertragungsmediums können die Flanken des PWM-Signales abgeflacht werden, wodurch eine Verfälschung der kodierten Informationen vorliegt. (Hüning, 2016, S. 158-159)

2.3 Magnetventile im System

Die Magnetventile werden zur Ansteuerung von hydraulischen und pneumatischen *Leveling*- und Luftfedersystemen am Fahrzeug eingesetzt. Es werden sowohl ein- als auch zweiachsige Systeme von *Vibracoustic* angeboten. *Levelingsysteme* dienen zur Einstellung der Fahrwerkshöhe. Hierbei werden durch den Druck im System je nach Art zwei oder vier Aktuatoren angesteuert. Bei dem Luftfedersystem kommt zu dieser Funktion außerdem die Einstellung des Federverhaltens über den Druck hinzu. Beide Systeme dienen zur Unterstützung des Komforts in verschiedenen Szenarien. Dabei soll vor allem die Veränderung der Fahrzeuglage durch unterschiedliche Beladungszustände kompensiert und folglich eine bessere Ausnutzung der Federwege erzielt werden. (Isermann, 2006, S. 265) Des Weiteren dient die variable Fahrzeughöhe zur Anpassung der Aerodynamik in verschiedenen Fahrsituationen. Beispielsweise kann somit die Fahrzeughöhe bei hohen Geschwindigkeiten reduziert und somit der Luftwiderstand und der resultierende Verbrauch verringert werden. Außerdem wird durch die Anpassung der Fahrzeughöhe ebenfalls die Höhe des Fahrzeugschwerpunktes verändert. Eine Verringerung des Schwerpunktes sorgt bei hohen Geschwindigkeiten für mehr Fahrsicherheit und eine bessere Fahrdynamik. (Isermann, 2006, S. 267) Es werden verschiedene Szenarien, wie beispielsweise der Bremsvorgang oder die Kurvenlage, über die Sensorik am Fahrzeug erfasst und berücksichtigt. Da in dieser Arbeit jedoch nur die Steuerung des Magnetventils betrachtet wird und solche Szenarien bereits in der vorherigen Befehlskette berücksichtigt werden, kann im Folgenden auf diese Erkennung verzichtet werden.

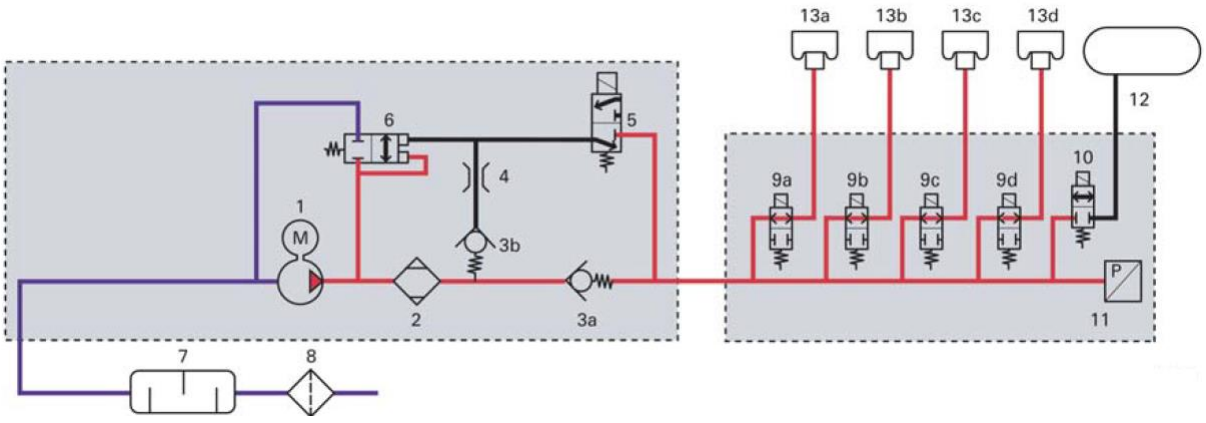


Abbildung 5: Systemübersicht eines *Four-Corner-Luftfedersystems* mit der Luftzufuhr (links) und dem Ventilblock (rechts) bei Druckaufbau (*Vibracoustic* intern)

Bei dem in dieser Arbeit verwendeten System handelt es sich um ein pneumatisches *Four-Corner-Luftfedersystem*. Eine grundlegende Systemübersicht bietet die Abbildung 5. Es müssen insgesamt vier Luftfedern (Abbildung 5, 13a-d) mit je einem Magnetventil (Abbildung 5, 9a-d) angesteuert werden. Im Ventilblock ist außerdem ein weiteres Magnetventil (Abbildung 5, 10) vorgesehen, welches den Zufluss zum Reservoir (Abbildung 5, 12) steuert. Im Reservoir kann über den Kompressor ein definierter Druck

gespeichert und anschließend anstelle des Kompressors zum Anheben der Aktuatoren verwendet werden. Der Kompressor (Abbildung 5, 1) befindet sich im linken System zur Luftzufuhr. Wie in Abbildung 5 dargestellt, kann bei eingeschaltetem Kompressor das Rückschlagventil (Abbildung 5, 3a) über den Druck geöffnet und somit der Druck im Ventilblock aufgebaut werden. Bei geöffneten Magnetventilen für die Luftfedern kann somit das Fahrzeug angehoben werden. Abschließend wird für den Druckabbau ein Auslassventil benötigt. Hierbei findet der Druckabbau über zwei Stufen statt. Zunächst muss der Kompressor ausgeschaltet und das elektrische Auslassventil (Abbildung 5, 5) geöffnet werden. Wenn die Magnetventile für die Luftfedern geöffnet werden, wird der Druck in den Luftfedern am pneumatischen Auslassventil (Abbildung 5, 6) aufgebaut. Dieses Ventil wird durch den Druck geöffnet, wodurch der Druck über das Rückschlagventil (Abbildung 5, 3b) und das pneumatische Auslassventil abgebaut und folglich die Fahrzeughöhe verringert werden kann. Zur Ermittlung der Fahrzeughöhe ist an jeder Luftfeder ein Höhensensor implementiert. Des Weiteren wird der Druck im Ventilblock über den Drucksensor (Abbildung 5, 11) überprüft. Die Systemübersicht zum Druckabbau ist im Anhang I vorzufinden.

Für diese Arbeit wird die Regelung des elektrischen Auslassventiles betrachtet. Es handelt sich dabei um ein NC Magnetventil mit einer festen Ventilmembran. Zur Öffnung des Ventiles werden zwei Phasen betrachtet. Im unbestromten Zustand ist das Ventil geschlossen. Dabei drückt die Luft aus den Luftfedern auf den Ventilteller, weshalb die aus der magnetischen Feldstärke resultierenden Kraft auf den Anker sowohl gegen die Federkraft als auch die Druckdifferenz arbeiten muss. Der hierzu benötigte Strom wird auch als *Push-Strom* bezeichnet. Sobald das Ventil vollständig geöffnet ist, liegt der Ventilteller am Gehäuse an, weshalb keine Kraft infolge des Systemdrucks wirkt. Folglich kann die Feldstärke und somit der benötigte Strom verringert werden. Dieser Wert wird auch als *Hold-Strom* bezeichnet. Mit Berücksichtigung der Einregelungszeit auf den *Push-Strom* und der Schaltzeit des Ventiles wird eine Zeitspanne definiert, nach welcher der *Push-Strom* auf den *Hold-Strom* verringert werden kann. Die Schaltphasen können in Abbildung 20 bei der Definition der Anforderungen näher betrachtet werden. Die Regelung der Stromstärke ist abhängig von der gewählten Spule sowie den Aktualisierungsraten der Eingangssignale aus der Software.

2.4 Regelung

„Regeln ist ein Vorgang, bei dem fortlaufend eine Größe, die zu regelnde Größe (Regelgröße = Istwert), erfasst, mit einer anderen Größe, die vorgegebene Führungsgröße (Sollwert), verglichen und abhängig vom Ergebnis dieses Vergleichs im Sinne einer Angleichung an die Führungsgröße beeinflusst wird“ (DIN 19 226).

Die Regelung beschreibt dabei einen geschlossenen Wirkungsablauf, den sogenannten Regelkreis. Dabei wird zu Beginn des Kreislaufes ein definierter Sollwert gefordert. Dieser wird als sogenannte Führungsgröße *w* zugeführt und mit dem Istwert verglichen. Der Istwert wird dabei auch als Rückführgröße

r bezeichnet. Die Differenz zwischen Sollwert und Istwert wird als Regelabweichung *e* an den Regler weitergeleitet. Der Regler selbst ist ein sogenanntes Übertragungssystem, von dem es verschiedene vordefinierte Arten gibt. Der Regler berechnet je nach Regelkreis aus der Regelabweichung die neue Steuergröße *u* oder den direkten Stellwert *y*. Dies ist abhängig von dem geforderten Eingangssignal der Regelstrecke. Die Regelstrecke stellt das mathematische Modell des zu regelnden physikalischen Systems dar. In einigen Regelkreisen entspricht das Eingangssignal der Regelstrecke nicht der zu regelnden physikalischen Größe. In diesem Fall muss die aus dem Regler berechnete Steuergröße zunächst im Stellglied in die passende Stellgröße *y* umgerechnet werden. Die Reaktion der Regelstrecke, welche in dieser Arbeit das Magnetventil darstellt, wird über eine Messung der Regelgröße *x* ermittelt. Sowohl beim Messvorgang als auch bei der Rückführung können Veränderungen des Signales auftreten. Hierzu gehört zum Beispiel eine Faktorisierung durch einen Einheitenwechsel oder eine zeitliche Verzögerung, bedingt durch das reale System. Diese werden im Messglied zusammengefasst. Als Ausgangssignal des Messgliedes wird die Rückführgröße *r* erneut mit der Führungsgröße *w* verglichen. Während dieses Wirkungsablaufes kann es durch Einflüsse aus der Umgebung zu einem Störverhalten und daraus resultierenden Abweichungen kommen. Die von außen eingreifenden Größen werden auch als Störgröße *z* bezeichnet. (Beier & Wurl, 2013, S. 14-15) Im Folgenden wird die Beschreibung des Übertragungsverhalten, so wie das Verhalten von bereits bekannten Reglerarten erläutert.

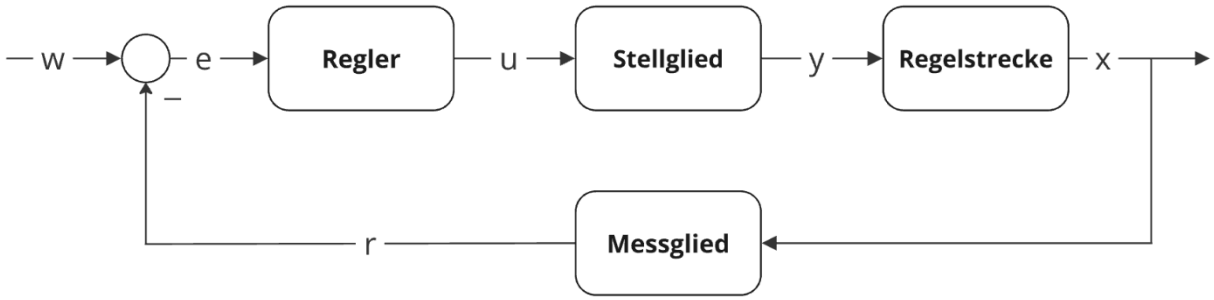


Abbildung 6: Grundaufbau des Regelkreises

2.4.1 Übertragungsverhalten

Zur Beschreibung von technischen Systemen und deren Verhalten wird das System als ein Übertragungssystem betrachtet. Jedes System wird dabei mit einer definierten Menge an Eingangsgrößen und dazugehörigen Ausgangsgrößen sowie einer Verarbeitung der Signale beschrieben. Übertragungssysteme werden zunächst in statische und dynamische Modelle unterteilt. Bei einer zeitlichen Änderung der Ein- und Ausgangsgrößen wird von einem dynamischen System gesprochen. Des Weiteren wird eine Unterteilung in lineare und nichtlineare Systeme vorgenommen, welche am statischen Verhalten ermittelt wird. Das statische Verhalten bezieht sich auf den Zusammenhang zwischen der Eingangs- und Ausgangsgröße im stationären Zustand, bei dem eine konstantes Eingangssignal vorliegt. Es ergibt sich eine Kennlinie, welche über *v=f(u)* dargestellt werden kann. Die Eingangsgröße wird hier mit *u* und die

Ausgangsgröße mit *v* beschrieben. Wenn eine proportionale Abhängigkeit zwischen den Ein- und Ausgangsgrößen vorliegt, wird das System als linear bezeichnet. (Grote & Feldhusen, 2014, X2.1)

Die mathematische Beschreibung des dynamischen Verhaltens eines Übertragungsgliedes wird als Übertragungsfunktion *G* definiert. Die Funktion beschreibt das Verhältnis von Ausgangssignal *v* zu Eingangssignal *u*. Zur Bestimmung der Übertragungsfunktion im Zeitbereich wird zunächst der Zusammenhang zwischen Ein- und Ausgangssignal über die physikalischen Gleichgewichtsbedingungen ermittelt. Anschließend wird dieses Gleichungssystem nach *v* umgestellt, um somit wie bei der statischen Kennlinie eine Funktion für die Ausgangsgröße in Abhängigkeit der Eingangsgröße aufzustellen. Diese Funktion zeichnet sich meist durch eine gewöhnliche Differentialgleichung höherer Ordnung aus. Um die Untersuchung einer solchen Funktion zu erleichtern, wird eine Laplace-Transformation durchgeführt. Bei der Laplace-Transformation wird eine zeitliche Funktion in eine komplexe Funktion umgewandelt. Die zeitliche Funktion *v(t)* wird mit dem Faktor e^{-st} multipliziert und anschließend über die Grenzen *t=0* und *t=∞* integriert. Die komplexe Operationsvariable *s* wird mit $s = \delta + i * \omega$ beschrieben. Dabei steht δ für die Dämpfung und ω für die Kreisfrequenz des Systems. Durch die Integration entfällt das t und der Term hängt nur noch von *s* ab. Die transformierte Funktion wird auch als Bildfunktion *V(s)* bezeichnet. (Grote & Feldhusen, 2014, X2.2.3)

$$V(s) = \int_0^\infty e^{-st} * f(t)dt \tag{4}$$

Formel 4: Allgemeine Laplace-Transformation

Die Übertragungsfunktion *G(s)* im Bildbereich errechnet sich anschließend genauso wie im Zeitbereich als Quotient des Eingangssignal *U(s)* und des Ausgangssignal *V(s)*.

$$V(s) = G(s) * U(s) \tag{5}$$

Formel 5: Darstellung der Bildfunktion

$$G(s) = \frac{V(s)}{U(s)} \tag{6}$$

Formel 6: Umstellung der Bildfunktion nach der Übertragungsfunktion

Zur Untersuchung des dynamischen Verhaltens wird sowohl das Verhalten im Zeitbereich als auch das im Frequenzbereich betrachtet. Das Verhalten im Zeitbereich lässt sich über die Sprungantwort des Systems bestimmen. Hierbei wird die Reaktion des Ausgangssignales auf eine sprungförmige Veränderung des Eingangssignales analysiert. Als sprungförmiges Testsignal dient der Einheitssprung. Die Sprungantwort beschreibt dabei das zeitliche Verhalten des Ausgangssignales. Das Verhalten des Übertragungsgliedes selbst wird auch als Übergangsfunktion bezeichnet. Die Übergangsfunktion stellt somit eine spezielle Form der Übertragungsfunktion dar. Zur Beurteilung der Sprungantwort werden mehrere Kennwerte aus dem Kurvenverlauf ermittelt. Der erste Wert ist die Verzugszeit T_U. Diese beschreibt die

Zeitspanne zwischen dem Beginn des Einheitssprunges und dem Schnittpunkt der ersten Wendetangente der Sprungantwort mit der Zeitachse. Die Verzugszeit stellt somit die zeitliche Verzögerung des Übertragungsgliedes dar. Der zweite Kennwert ist die Ausgleichszeit T_g. Hierbei wird die Zeitspanne zwischen dem Schnittpunkt der ersten Wendetangente mit der Zeitachse und dem Schnittpunkt der Wendetangente mit der Sollwert-Grenze ermittelt. Über das Verhältnis von Verzugszeit zu Ausgleichszeit kann die Regelbarkeit einer Regelstrecke bestimmt werden. Der letzte Kennwert ist die Überschwingweite v_m. Die Überschwingweite ist, wie in Abbildung 7 zu erkennen, die größte Abweichung der Sprungantwort vom Sollwert nach dem erstmaligen Überschreiten des Sollwertes. (Gieck & Gieck, 2013, T2) Die beschriebenen Kennwerte werden in den künftigen Kapiteln zur Dimensionierung der Regler eingesetzt.

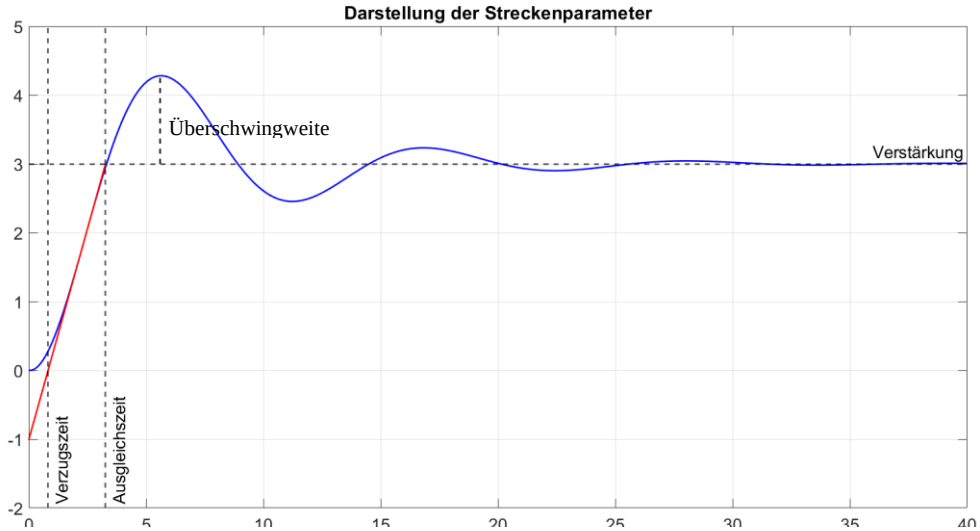


Abbildung 7: Darstellung der Streckenparameter

Zur Untersuchung des Übertragungsverhaltens im Frequenzbereich werden der Frequenzgang und die Ortskurve betrachtet. Das Frequenzverhalten beschreibt die Reaktion des Systems auf ein periodisches Testsignal. Hierfür wird vorzugsweise ein Eingangssignal in Form eines Kosinus verwendet. Das Eingangssignal stellt somit im stationären Zustand eine periodische Schwingung mit einer definierten Amplitude $\hat{u}$ und einer Kreisfrequenz ω dar. Am Ausgangssignal kann ebenfalls eine harmonische Schwingung ermittelt werden, dessen Amplitude und Phasenlage jedoch abweichen können. Der Frequenzgang $G(j\omega)$ wird im Allgemeinen als komplexe Übertragungsfunktion mit einer Trennung nach Real- und Imaginärteil definiert, es lässt sich aber die Veränderung zwischen Ein- und Ausgangssignal ebenfalls über die Trennung nach Betrag und Phase beschreiben: (Grote & Feldhusen, 2014, X2.2.2)

$$G(j\omega) = Re(\omega) + j * Im(\omega) \tag{7}$$

Formel 7: Berechnung des Frequenzganges aus dem Real- und Imaginärteil

$$G(j\omega) = A(\omega) * e^{j\varphi(\omega)} \tag{8}$$

Formel 8: Berechnung des Frequenzganges aus dem Betrag und der Phase

In der Formel 8 beschreibt *φ(ω)* die Phasendifferenz zwischen Ein- und Ausgangssignal, während *A(ω)* das frequenzabhängige Verhältnis der beiden Amplituden darstellt. *A(ω)* wird daher auch als Amplitudengang und *φ(ω)* als Phasengang bezeichnet. Mathematisch werden diese Größen wie folgt definiert:

$$A(\omega) = |G(j\omega)| = \sqrt{Re^2(\omega) + Im^2(\omega)} \tag{9}$$

Formel 9: Berechnung des Amplitudenganges

$$\varphi(\omega) = argG(j\omega) = arctan\frac{Im(\omega)}{Re(\omega)} \tag{10}$$

Formel 10: Berechnung des Phasenganges

Der Amplitudengang und der Phasengang werden gemeinsam im sogenannten Bode-Diagramm dargestellt. In diesem Diagramm wird allerdings eine logarithmische Achsenskalierung verwendet, weshalb die Amplitude in Dezibel umgerechnet werden muss.

$$A_{dB}(\omega) = 20 * log_{10}\big(A(\omega)\big) \tag{11}$$

Formel 11: Umrechnung des Amplitudenganges in Dezibel

Des Weiteren wird die auch Abszisse, auf der die Kreisfrequenz beschrieben wird, logarithmisch dargestellt.

Eine weitere Darstellung des Frequenzganges zeigt die Ortskurve, bei der beide Komponenten des Frequenzganges in der komplexen Ebene aufgetragen werden. Dabei beschreibt der Phasengang den Winkel zur Real-Achse und der Amplitudengang die Entfernung zum Koordinatenursprung, was wiederum dem Betrag des Vektors entspricht. Die Ortskurve wird besonders zur Analyse der Stabilität von Systemen eingesetzt. Hierzu wird das *Nyquist-Kriterium* verwendet. Laut diesem Kriterium gilt ein Regelkreis als stabil, „wenn die Ortskurve der Übertragungsfunktion des offenen Regelkreises den Punkt -1 bei zunehmender Frequenz nicht umfährt“ (Beier & Wurl, 2013, S. 152). Der offene Regelkreis bezeichnet dabei eine Trennung der Rückführung, wodurch Regler, Stellglied, Regelstrecke und Messglied als eine Reihenschaltung betrachtet werden.

Das Verhalten von technischen Systemen kann im Allgemeinen durch die Kombination weniger Grundformen beschrieben werden. Hierzu gibt es die Unterteilung in proportionales, integrales, differentielles, Verzögerungs- und Laufzeitverhalten. Durch das bereits bekannte Verhalten dieser grundlegenden Übertragungsglieder, kann die Übertragungsfunktion einer Regelstrecke experimentell und damit schneller ermittelt werden. Im Folgenden werden die Übertragungsglieder und deren Verhalten genauer beschrieben.

2.4.1.1 Proportionalglied

Das Proportionalglied, auch P-Glied abgekürzt, dient zur reinen Verstärkung des Eingangssignales. Ein einfaches technisches System ist zum Beispiel ein starrer Hebel. (Grote & Feldhusen, 2014, X3.2.1) Das P-Glied verknüpft die Regeldifferenz mit der Steuergröße über den Proportionalbeiwert K_P. Damit ist dieses Übertragungsglied frequenz- und zeitunabhängig. (Grote & Feldhusen, 2014, X2.3.1) Die Übertragungsfunktion G genauso wie der Amplitudengang werden für t>0 durch den konstanten Wert K_P beschrieben, während der Phasengang konstant bei 0° liegt.

$$v(t) = K_P * u(t) \tag{12}$$

Formel 12: Zeitverhalten des P-Gliedes

$$G(s) = K_P \tag{13}$$

Formel 13: Übertragungsfunktion des P-Gliedes

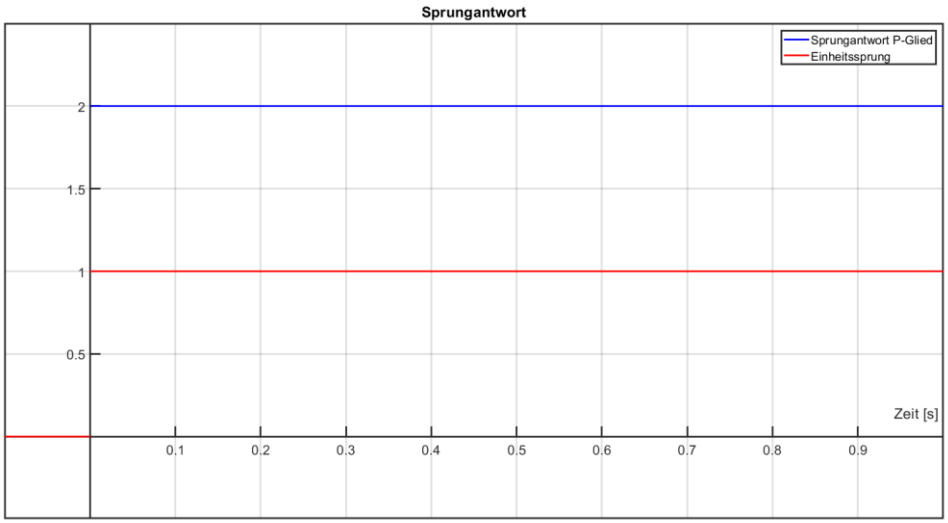


Abbildung 8: Sprungantwort des P-Gliedes bei einem Proportionalbeiwert K_P=2

Da diese Übertragungsfunktion keinen imaginären Anteil besitzt, lässt sich der Frequenzgang leicht über den Realteil definieren.

$$A(\omega) = \sqrt{K_P^2} = K_P \tag{14}$$

Formel 14: Amplitudengang des P-Gliedes

$$\varphi(\omega) = \arctan\left(\frac{0}{K_P}\right) = 0° \tag{15}$$

Formel 15: Phasengang des P-Gliedes

Da der Frequenzgang ebenfalls unabhängig von der Kreisfrequenz ist, beschreibt die Ortskurve einen einfachen Punkt auf der reellen Achse, wie in Abbildung 9 dargestellt. Somit ist das *Nyquist-Kriterium* für den reinen P-Regler erfüllt, so lange der Proportionalbeiwert größer als -1 ist.

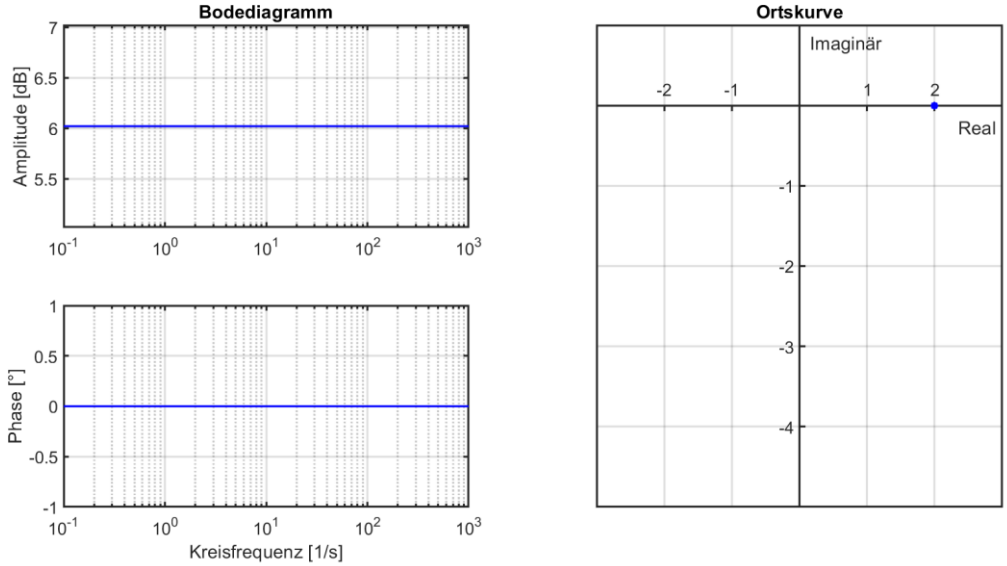


Abbildung 9: Bodediagramm (links) und Ortskurve (rechts) des P-Gliedes bei einem Proportionalbeiwert K_P=2

2.4.1.2 Integralglied

Beim Integralglied, auch I-Glied abgekürzt, wird die Eingangsgröße über der Zeit integriert und mit dem Integrierbeiwert K_I multipliziert. Ein technisches Beispiel hierfür ist das Befüllen eines Behälters. (Grote & Feldhusen, 2014, X3.3.1) Die Übertragungsfunktion beim Einheitssprung beschreibt somit eine Gerade mit einer positiven Steigung.

$$v(t) = K_I * \int u(t)dt \tag{16}$$

Formel 16: Zeitverhalten des I-Gliedes

$$G(s) = \frac{K_I}{s} \tag{17}$$

Formel 17: Übertragungsfunktion des I-Gliedes

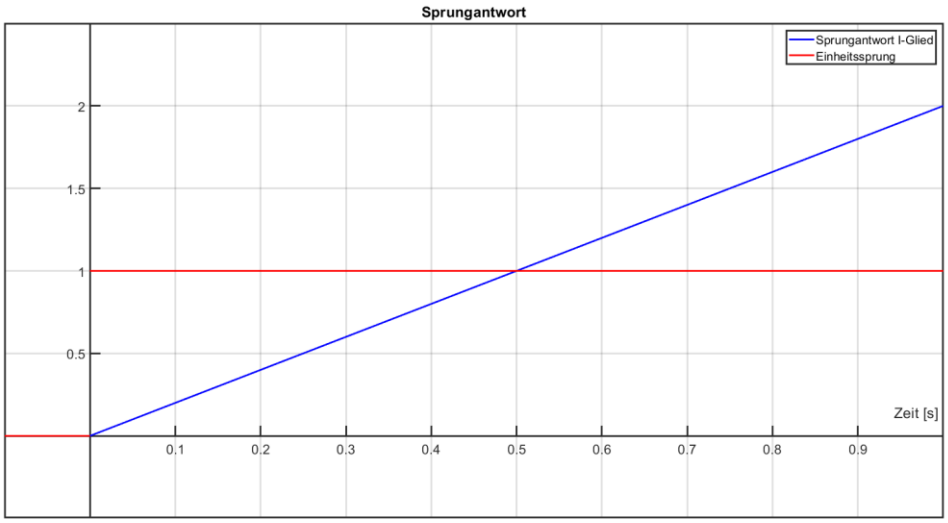


Abbildung 10: Sprungantwort des I-Gliedes bei einem Integrierbeiwert K_I=2

Die Änderungsgeschwindigkeit und somit die Steigung der Geraden wird über den Integrierbeiwert definiert. Für den Frequenzgang wird die Übertragungsfunktion erweitert, wodurch der Real- und Imaginärteil deutlich wird.

$$G(j\omega) = \frac{K_I}{j\omega} * \frac{j}{j} = -\frac{K_I}{\omega} * j \tag{18}$$

Formel 18: Frequenzgang des I-Gliedes

$$Re(j\omega) = 0 \tag{19}$$

Formel 19: Realteil des Frequenzganges beim I-Glied

$$Im(j\omega) = -\frac{K_I}{\omega} \tag{20}$$

Formel 20: Imaginärteil des Frequenzganges beim I-Glied

Während sich der Amplitudengang leicht bestimmen lässt, kann der Phasengang mit einem Realteil von 0 nicht direkt ausgerechnet werden. Bedingt durch das Vorzeichen des Imaginärteils liegt der Phasengang im negativen Unendlichen. Der Grenzwert des Arkustangens liegt bei -π/2.

$$A(\omega) = \sqrt{\left(-\frac{K_I}{\omega}\right)^2} = \frac{K_I}{\omega} \tag{21}$$

Formel 21: Amplitudengang des I-Gliedes

$$\varphi(\omega) = -\frac{\pi}{2} \tag{22}$$

Formel 22: Phasengang des I-Gliedes

Der Amplitudengang fällt folglich mit 20 dB pro Dekade, während der Phasengang konstant bei -90° liegt. Die Ortskurve fängt folglich im Nullpunkt an und fällt auf der imaginären Achse bis ins negative Unendliche.

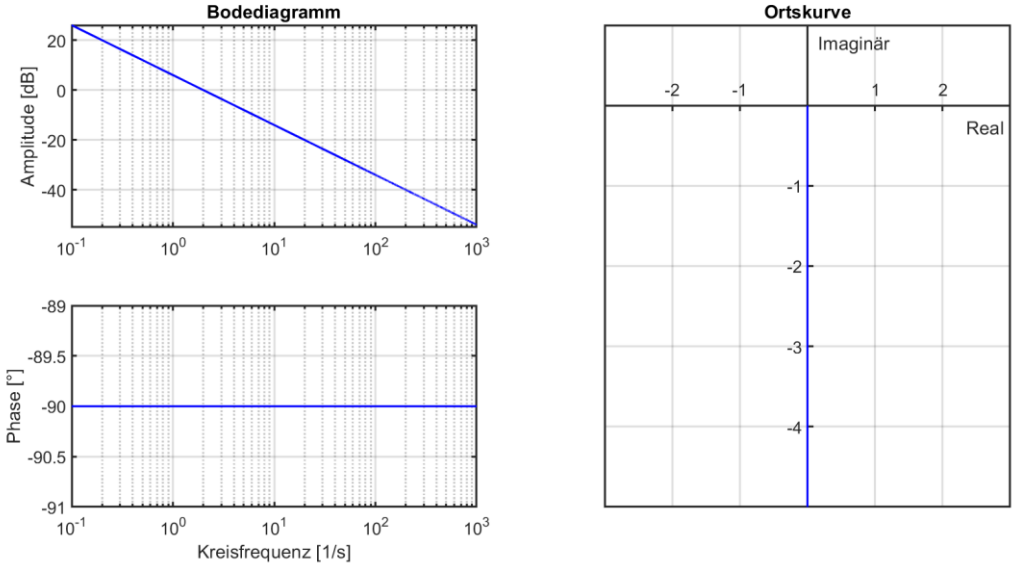


Abbildung 11: Bodediagramm (links) und Ortskurve (rechts) des I-Gliedes bei einem Integrierbeiwert K_I=2

2.4.1.3 Differentialglied

Das Differentialglied, welches als D-Glied abgekürzt wird, beschreibt die zeitliche Ableitung der Eingangsgröße, welche ebenfalls mit einem Differenzierbeiwert K_D multipliziert wird. Die Übertragungsfunktion der Sprungantwort beschreibt somit einen Dirac-Impuls. Folglich kann dieses Glied nur in der Theorie betrachtet werden und würde in der Praxis nur in Verbindung mit einer Verzögerung realisierbar sein. (Beier & Wurl, 2013, S. 137) Die Übertragungsfunktion besteht aus dem Produkt des Differenzierbeiwertes und der Bildvariablen s.

$$v(t) = K_D * \frac{du(t)}{dt} \tag{23}$$

Formel 23: Zeitverhalten des D-Gliedes

$$G(s) = K_D * s \tag{24}$$

Formel 24: Übertragungsfunktion des D-Gliedes

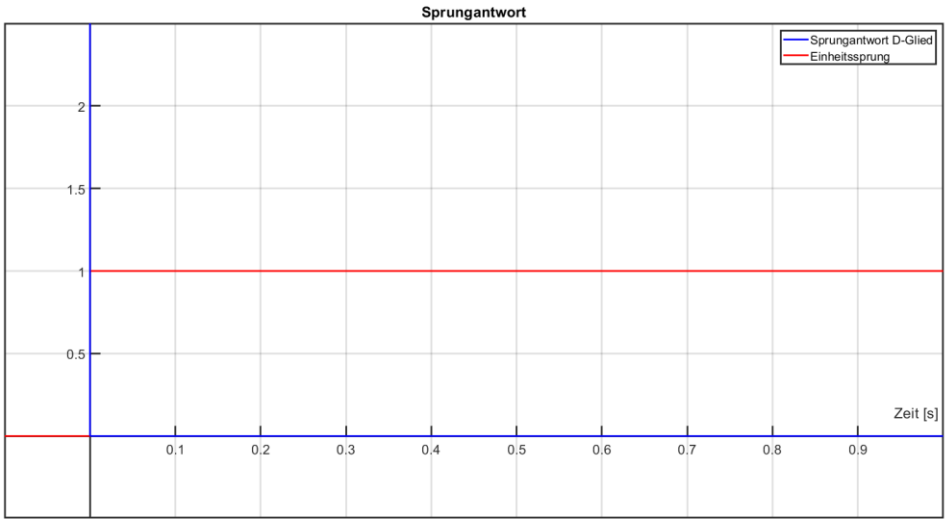


Abbildung 12: Sprungantwort des D-Gliedes bei einem Differenzierbeiwert K_D=2

Bei der Aufteilung der Funktion in Realteil und Imaginärteil ergibt sich ein ähnliches Bild zum Integralglied.

$$Re(\omega) = 0 \tag{25}$$

Formel 25: Realteil des Frequenzganges beim D-Glied

$$Im(\omega) = K_D * \omega \tag{26}$$

Formel 26: Imaginärteil des Frequenzganges beim D-Glied

$$A(\omega) = \sqrt{(K_D * \omega)^2} = K_D * \omega \tag{27}$$

Formel 27: Amplitudengang des D-Gliedes

$$\varphi(\omega) = \arctan\left(\frac{K_D * \omega}{0}\right) = \frac{\pi}{2} \tag{28}$$

Formel 28: Phasengang des D-Gliedes

Der Grenzwert des Arkustangens im positiven Unendlichen liegt bei π/2. Die Ortskurve beginnt folglich erneut im Nullpunkt, steigt jedoch auf der imaginären Achse ins positive Unendliche.

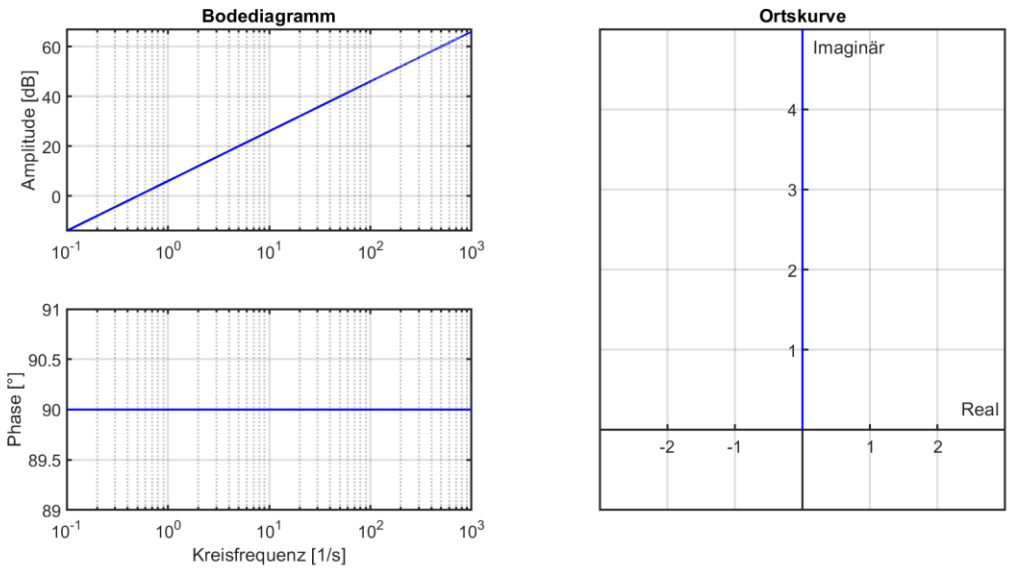


Abbildung 13: Bodediagramm (links) und Ortskurve (rechts) des D-Gliedes bei einem Differenzierbeiwert K_D=2

2.4.1.4 Verzögerungsglied erster Ordnung

Übertragungsglieder, welche in der Lage sind, Energie zu speichern, sorgen für eine zeitliche Verzögerung bei der Freisetzung der selbigen Energie. (Beier & Wurl, 2013, S. 83) Das Verzögerungsglied erster Ordnung, welches als PT1-Glied abgekürzt wird, zeichnet sich durch die Verzögerungszeit *T* aus. Die Übertragungsfunktion der Sprungantwort beschreibt eine Kurve mit abfallender Steigung, welche sich im Unendlichen einem stationären Endwert annähert.

$$v(t) = K * \left(1 - e^{-\frac{t}{T}}\right) \tag{29}$$

Formel 29: Sprungantwort des PT1-Gliedes

$$G(s) = \frac{K}{T * s + 1} \tag{30}$$

Formel 30: Übertragungsfunktion des PT1-Gliedes

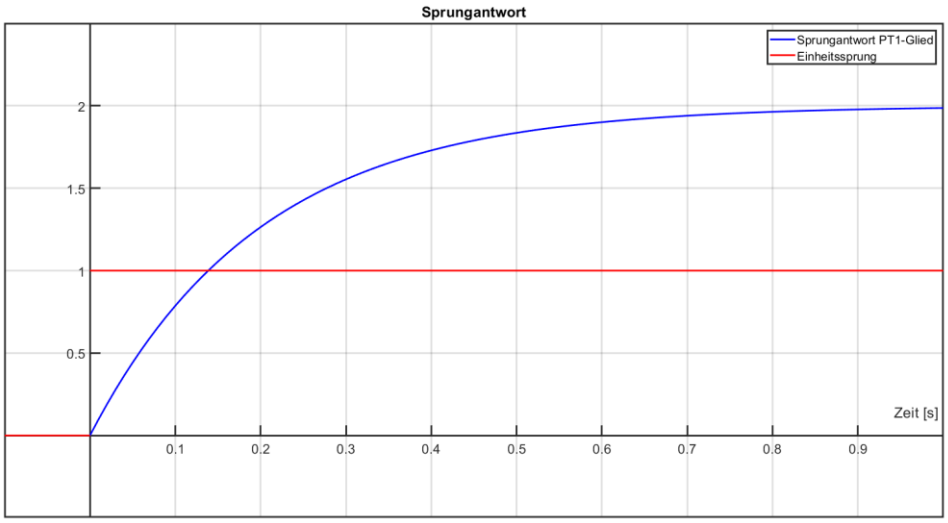


Abbildung 14: Sprungantwort des PT1-Gliedes bei einer Verstärkung K=2 und einer Verzögerungszeit T=0,2 s

Sowohl die Verstärkung K als auch die Verzögerungszeit T können anhand der Sprungantwort eines Systems ermittelt werden. Da der Sollwert der Sprungantwort durch den Einheitssprung mit 1 festgelegt ist, kann der stationäre Endwert direkt als Verstärkung der Übertragungsfunktion abgelesen werden. Die Verzögerungszeit ist als Zeitspanne definiert, welche das System benötigt, um auf 63,2 % des Endwertes anzusteigen. (Beier & Wurl, 2013, S. 84) Des Weiteren ergibt sich über das Verhältnis von Endwert zur Verzögerungszeit die Steigung bei t=0. Zur Betrachtung des Frequenzganges muss wie beim I-Glied eine Erweiterung der Übertragungsfunktion vorgenommen werden. Daraufhin kann diese in den Real- und Imaginärteil aufgeteilt werden.

$$G(j\omega) = \frac{K}{T * j\omega + 1} * \frac{1 - T * j\omega}{1 - T * j\omega} = \frac{K * (1 - T * j\omega)}{1 + T^2 * \omega^2} \tag{31}$$

Formel 31: Frequenzgang des PT1-Gliedes

$$Re(\omega) = \frac{K}{1 + T^2 * \omega^2} \tag{32}$$

Formel 32: Realteil des Frequenzganges beim PT1-Glied

$$Im(\omega) = -\frac{K * T * \omega}{1 + T^2 * \omega^2} \tag{33}$$

Formel 33: Imaginärteil des Frequenzganges beim PT1-Glied

$$A(\omega) = \sqrt{\left(\frac{K}{1 + T^2 * \omega^2}\right)^2 + \left(-\frac{K * T * \omega}{1 + T^2 * \omega^2}\right)^2} = \frac{K}{1 + T^2 * \omega^2} * \sqrt{1 + T^2 * \omega^2} \tag{34}$$

Formel 34: Amplitudengang des PT1-Gliedes

$$\varphi(\omega) = \arctan\left(\frac{-K * T * \omega}{K}\right) = \arctan(-T * \omega) \tag{35}$$

Formel 35: Phasengang des PT1-Gliedes

Der Amplitudengang beginnt bei 0 und ab der Verzögerungszeit *T* fällt die Amplitude mit 20 dB pro Dekade. Der Phasengang beginnt bei 0° und fällt bis -90° ab, wobei bei *T* die Phase -45° beträgt. Der Verlauf der Ortskurve kann bei diesem Übertragungsglied nicht direkt aus dem Frequenzgang abgelesen werden. Erkennbar ist, dass bei ω=0 die Phase 0 und die Amplitude *K* beträgt. Bei ω=∞ liegt die Phase bei -π/2 und der Betrag bei 0. Somit endet die Ortskurve im Nullpunkt. Dazwischen beschreibt die Ortskurve einen Halbkreis, welcher nach oben hin geöffnet ist.

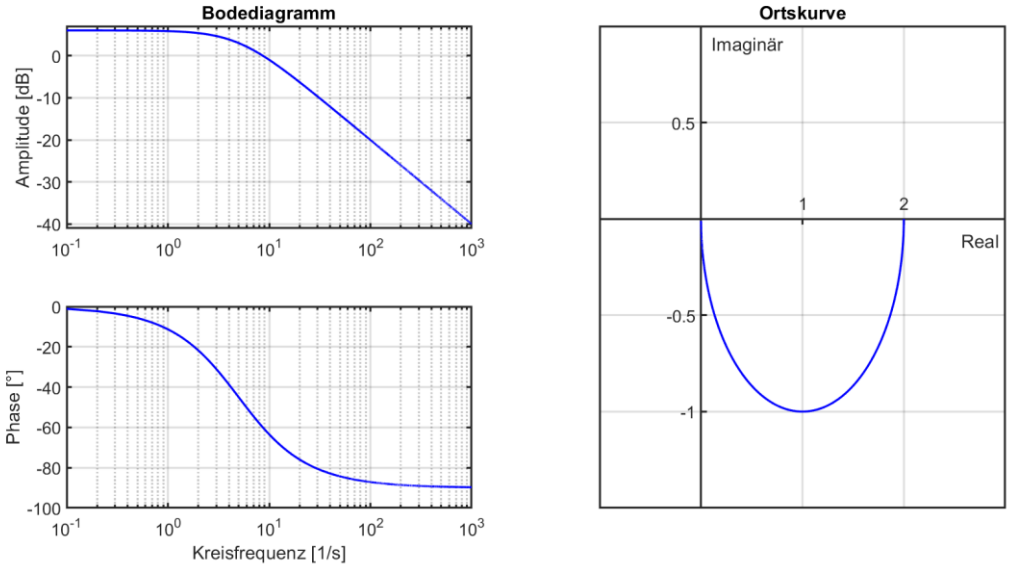


Abbildung 15: Bodediagramm (links) und Ortskurve (rechts) des PT1-Gliedes bei einer Verstärkung K=2 und einer Verzögerungszeit T=0,2 s

2.4.1.5 Totzeitglied

Das Totzeitglied beschreibt das Laufzeitverhalten mit der definierten Totzeit T_t. Dieser Wert gibt vor, nach welcher Zeitspanne sich die Veränderung des Eingangssignales am Ausgangssignal bemerkbar macht. Das Laufzeitverhalten tritt besonders in physikalischen Systemen mit Materialtransport. Beispielsweise wird bei einem Bandförderer die Zeit zwischen Beladung und Abwurf als Totzeit betrachtet. (Grote & Feldhusen, 2014, X3.2.4) Die Sprungantwort entspricht folglich dem Einheitssprung mit einem zeitlichen Versatz. Die Übertragungsfunktion im Bildbereich entspricht dabei einer e-Funktion.

$$v(t) = u(t - T_t) \tag{36}$$

Formel 36: Zeitverhalten des Totzeitgliedes

$$G(s) = e^{-s*T_t} \tag{37}$$

Formel 37: Übertragungsfunktion des Totzeitgliedes

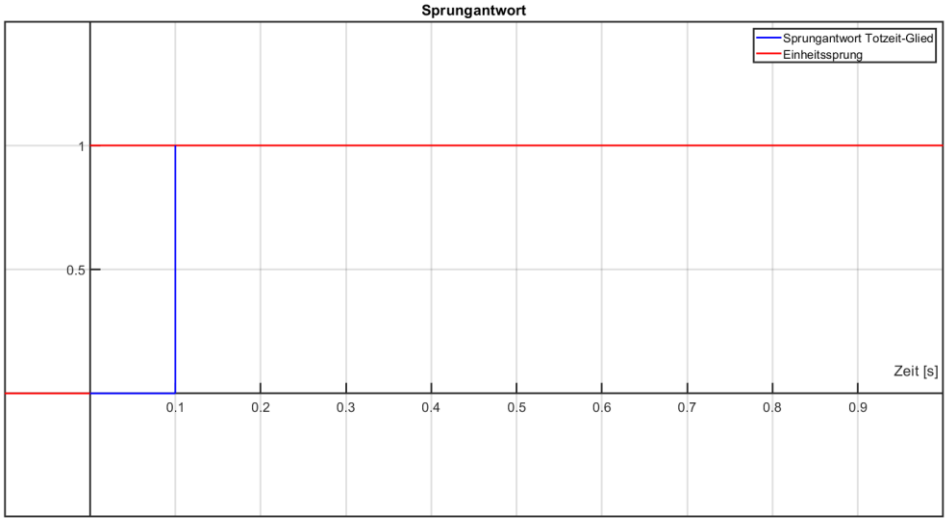


Abbildung 16: Sprungantwort des Totzeitgliedes

Beim Totzeitglied ist die Aufteilung der Übertragungsfunktion in Realteil und Imaginärteil aufgrund der e-Funktion deutlich schwieriger. Daher wird in diesem Abschnitt nicht auf die konkrete Bestimmung des Phasen- und Amplitudenganges eingegangen. Der Betrag der Übertragungsfunktion beträgt zu jeder Frequenz 1. Der Amplitudengang in Dezibel liegt folglich auf der Nulllinie. Der Phasengang beginnt im Nullpunkt und nimmt anschließend mit dem Wert der Totzeit als Steigung konstant ab. In der logarithmischen Darstellung sinkt daher die Phase mit einer zunehmend negativen Steigung bis ins negative Unendliche. Durch den konstanten Betrag von 1 und der konstant sinkenden Phase weist die Ortskurve einen Kreis um den Nullpunkt mit dem Radius 1 auf. (Beier & Wurl, 2013, S. 105)

$$A(\omega) = 1 \tag{38}$$

Formel 38: Amplitudengang des Totzeitgliedes

$$\varphi(\omega) = -T_t * \omega \tag{39}$$

Formel 39: Phasengang des Totzeitgliedes

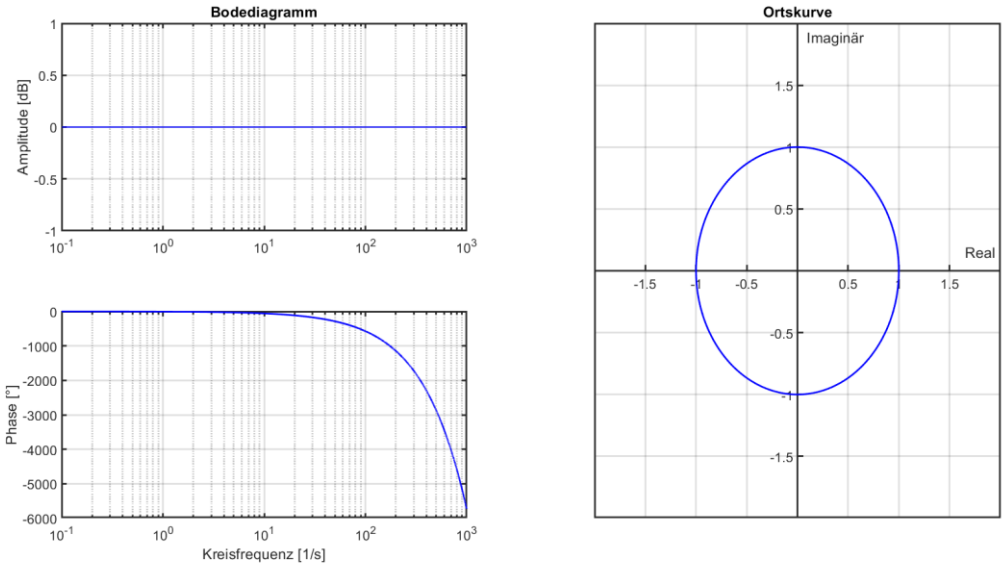


Abbildung 17: Bodediagramm (links) und Ortskurve (rechts) des Totzeitgliedes bei einer Totzeit von 0,1 s

2.5 AUTOSAR

AUTOSAR steht für „**AUT**omotive **O**pen **S**ystem **AR**chitecture“ und ist eine standardisierte Software-Architektur für elektronische Steuergeräte in der Fahrzeugindustrie. Das Ziel ist eine Referenzarchitektur aufzubauen, welche allgemein verwendet werden kann, um so die wachsende Komplexität der Software in modernen Fahrzeugen einzudämmen. Des Weiteren werden durch eine einheitliche Architektur die Kosten reduziert und die Qualität verbessert. (*AUTOSAR | Vector*, 2024a)

Der AUTOSAR Standard befasst sich mit drei Bereichen der Software-Entwicklung: Methodik, Architektur und Schnittstellen. Über die Methodik wird der Herstellungsprozess der Software definiert. Hierbei wird die Architektur, die Kommunikation und die Code-Generierung in einzelne Prozesse aufgeteilt und über die Integration zusammengeführt. (Rost, 2024) Die Methodik stellt nur einen geringen Teil der Ausarbeitung dar und wird demnach nicht weiter vertieft.

Wesentlich relevanter für die folgende Ausarbeitung ist die Architektur. Die Referenzarchitektur beschreibt ein Schichtmodell, welches eine Abstraktion der Software von der Hardware schafft. (Rost, 2024) Auf der untersten Ebene liegt die Microcontroller-Schicht. Diese Ebene dient als reine Schnittstelle zur Hardware, wozu die Steuereinheit sowie Ventile und Sensoren zählen.

Über dieser Ebene liegt die Basissoftware-Schicht. Sie beinhaltet das gesamte Betriebssystem der Software und entkoppelt die Hardware von der Software. Da diese Ebene deutlich komplexer ist, kann sie, wie in Abbildung 18 dargestellt, in eine Tabelle aus vier vertikalen Säulen und drei horizontalen Ebenen aufgeteilt werden. Durch die Säulen wird die Basissoftware, auch als BSW abgekürzt, in verschiedene

Dienste unterteilt. Benötigt werden dabei Systemdienste, Speicherverwaltungsdienste, Kommunikationsdienste und Hardware-Ein-/Ausgabedienste. Über die drei horizontalen Ebenen wird ebenfalls eine Abstraktion der Software zur Hardware geschaffen. (Neue, 2024, S. 8) Auf der untersten Ebene befindet sich die Microcontrollerabstraktionsschicht (engl. Microcontroller Abstraction Layer), auch MCAL abgekürzt. In dieser Ebene befindet sich die Konfiguration und Initialisierung des Microcontrollers und der integrierten Geräte. Sie enthält somit die internen Treiber und direkten Zugriff auf den Microcontroller. Der MCAL ist controllerspezifisch und muss folglich beim Wechsel der Hardware ebenfalls ersetzt werden. (Kindel & Friedrich, 2009, S. 122-123) Darüber liegt die Steuergeräteabstraktionsschicht, welche die Eigenschaften des Steuergerätes abstrahiert. Dabei stellt sie ein Interface zum Zugriff auf die elektrischen Signale des Steuergerätes bereit. Durch dieses Interface wird die darüberliegende Software von der Hardware entkoppelt. (Rost, 2024) Als oberste Ebene der BSW existiert die Serviceschicht. Sie stellt den Anwendungen und Basissoftwaremodulen die grundlegenden Dienste zur Verfügung. Dazu gehören beispielsweise die Speicherverwaltungs-, Betriebsystem-, Diagnose- und Kommunikationsfunktionen. (Kindel & Friedrich, 2009, S. 121)

Die oberste Ebene in der Gesamtarchitektur ist die Anwendungssoftware, auch als ASW abgekürzt. Diese Schicht beinhaltet alle Anwendungssoftwarekomponenten und Sensor-/Aktor-Softwarekomponenten. Bei einer Softwarekomponente handelt es sich um ein Strukturelement, welches über deren Ein- und Ausgänge zur Kommunikation mit der restlichen Software definiert wird. Des Weiteren wird eine Softwarekomponente durch seine Aufgaben innerhalb der BSW spezifiziert. Zur Verarbeitung von Signalen und Ausführung von Funktionen besitzt eine Softwarekomponente sogenannte *Runnables*. „Runnables sind Codesequenzen in den Softwarekomponenten, die durch Events, wie beispielsweise Timer oder den Empfang von Daten, aktiviert werden können“ (Kindel & Friedrich, 2009, S. 90). *Runnables* haben Zugriff auf die Ein- und Ausgänge der Softwarekomponente und können somit Daten empfangen, Operationen ausführen und neue Daten versenden. Die Sensor/Aktor-Softwarekomponenten stellen Softwarekomponenten dar, welche in der Anwendungsschicht nicht von der Steuereinheit sondern von den Eigenschaften der konkreten Sensoren und Aktoren abstrahiert werden. (Kindel & Friedrich, 2009, S. 89) Während die Anwendungsschicht lediglich die Softwarekomponenten beinhaltet, wird für die Kommunikation zwischen diesen und für die Kommunikation zur BSW eine weitere Ebene benötigt.

Zwischen der BSW und der ASW liegt die *Runtime Environment* Schicht, auch RTE abgekürzt. In dieser Schicht befindet sich die genaue Beschreibung der benutzten Schnittstellen, wodurch die geforderten Kommunikationsverbindungen geschaffen werden können. (Kindel & Friedrich, 2009, S. 119)

Damit ist die gesamte Kommunikation der Software über die verschiedenen Abstraktionsebenen klar definiert. Allerdings existiert eine Ausnahme, welche auch als fünfte Säule der BSW betrachtet wird. Hierbei geht es um die komplexen Treiber (engl. Complex Device Driver), auch CDD abgekürzt. Hierüber werden nicht-standardisierte Software-Module vorerst unkompliziert in die Software-Architektur

implementiert und können je nach Notwendigkeit im Nachhinein in das Schichtmodell eingebunden werden. Da komplexe Treiber keine Abstraktionsebene besitzen, wird ein direkter Zugriff der RTE auf den Microcontroller erzeugt. (Neue, 2024, S. 8)

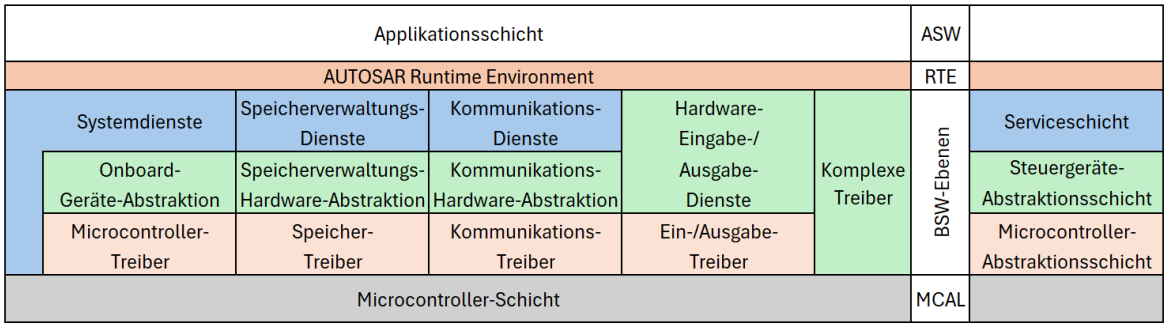


Abbildung 18: Software-Architektur im Schichtmodell nach dem AUTOSAR Standard, farbliche Aufteilung der BSW in die verschiedenen Ebenen zwischen der RTE und der Microcontroller-Schicht

Nach der fertigen Definition der AUTOSAR-Architektur müssen abschließend die Schnittstellen der Gesamtsoftware beschrieben werden. Diese Schnittstellen besitzen keine eigene Schicht in der Architektur. Stattdessen liefern sie eine vollständige Beschreibung aller technischer Schnittstellen, welche zur Herstellung der Software auf unterschiedlicher Hardware benötigt wird. Sie werden folglich durch die Hardware vorgegeben und müssen in der Software-Architektur berücksichtigt werden. Dabei muss im hier betrachteten Luftfedersystem berücksichtigt werden, dass die Steuereinheit nicht nur mit den selbstintegrierten Sensoren, sondern auch mit dem Fahrzeug kommuniziert. Somit kann auf bereits vorhandene Sensorik und deren Messdaten zurückgegriffen werden.

2.6 Aktueller Stand

Zur vereinfachten Beschreibung der aktuellen Vorgehensweise der Software-Entwicklung wird zunächst der Signalverlauf zur Schaltung der Ventile im *DaVinci Developer* betrachtet. Dieses Programm dient zum Architektur-Entwurf der Software. Hierbei kann eine graphische Definition der Softwarekomponenten inklusive derer Ein- und Ausgänge, Datentypen und weiterer Spezifikationen erfolgen. Des Weiteren ist das Verbinden von Softwarekomponenten sowie der Zusammenschluss zu Kompositionen möglich. In der Abbildung 19 wird der Signalfluss zur Ansteuerung des Auslassventiles beschrieben. Im Folgenden werden der Prozess der Ventilansteuerung sowie die implementierten Softwarekomponenten und Signale nur grob für das Verständnis erläutert. Die Ansteuerung beginnt bei der *QM*.

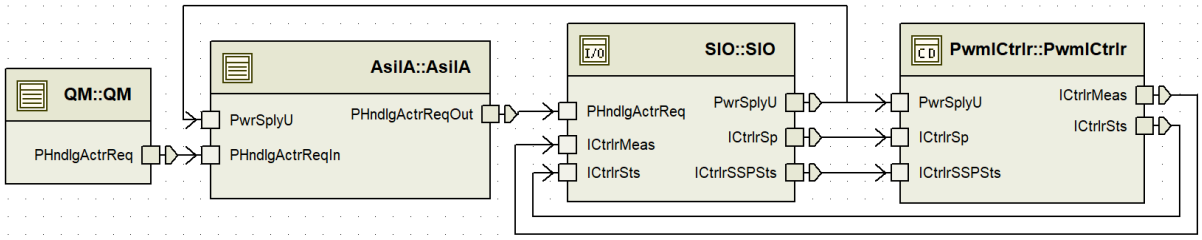


Abbildung 19: Reduzierter Aufbau der Software-Architektur im DaVinci Developer

Die Softwarekomponente *QM*, dessen Abkürzung für Qualitätsmanagement steht, implementiert alle nicht sicherheitsrelevanten Funktionen der Anwendungssoftware. Zur Ansteuerung des Auslassventiles wird eine Anfrage über das Signal *PHndlgActrReq (Pressure Handling Actuator Request)* ausgelöst. Dabei handelt es sich um ein Bus-Signal, welches die Anfragen für das Boost-, Kompressor-, Auslass- und Reservoir-Ventil beinhaltet. Das gesamte Bus-Signal wird an die *AsilA* weitergeleitet.

Die Softwarekomponente *AsilA* befasst sich mit Teilen des funktionalen Verhaltens der Software, das in ASIL behandelt wird. ASIL steht für *Automotive Safety Integrity Level*, zu Deutsch *Funktionale Sicherheit für Automotive-Systeme*. Es handelt sich dabei um ein festgelegtes Maß für das Risiko von Funktionen/Systemen im Serienfahrzeug, welche über die ISO 26262 definiert wird. Dazu gehören im Leveling-System die Vorverarbeitung der Höhensensordaten, die Plausibilitätsprüfung der Daten und die Kalibrierung der Höhensensoren sowie die grundlegende Prüfung der Höhenbegrenzung. Des Weiteren trägt das in dieser Arbeit angesteuerte Auslassventil zur funktionalen Sicherheit des Systems bei. Die *AsilA* berücksichtigt folglich definierte Sicherheitsabfragen bei der Schaltung der Ventile und schickt das überarbeitete Bus-Signal *PHndlgActrReqOUT* an die nächste Softwarekomponente, den *SIO*.

Die Softwarekomponente *SIO* befasst sich mit allen sicherheitsrelevanten Ein- und Ausgängen. Neben dem Bus-Signal der *AsilA* empfängt der *SIO* ebenfalls die Ventilanfragen für die Luftfederventile und bestimmt für jedes Ventil den Sollwert für die Stromregelung. Die acht Sollwerte der verschiedenen Ventile werden zu einem *Array* zusammengefasst und mit dem Signal *ICtrlrSp (Current Control Setpoint)* an den *PwmICtrlr (PWM Current Controller)* übertragen. Zusätzlich findet eine Statusabfrage statt, welche über das Signal *ICtrlrSSPSts (Current Control Setpoint Status)* ebenfalls an den *PwmICtrlr* geschickt wird.

Die Softwarekomponente *PwmICtrlr* ist aktuell für die Stromregelung zuständig. Es handelt sich dabei um einen komplexen Treiber, welcher aus den Sollwerten des *ICtrlrSp* das notwendige PWM-Signal ermittelt. Dieses Signal wird anschließend über die BSW an den Microcontroller weitergeleitet, was in der Darstellung im *DaVinci Developer* nicht ersichtlich ist. Die gemessene Stromstärke wird zurückgeleitet an den *PwmICtrlr*, welcher wiederum über das Signal *ICtrlrMeas (Current Control Measurement)* die gemessene Stromstärke zurück an den *SIO* schickt.

Die aktuelle Stromregelung im *PwmICtrlr* wird als reiner C-Code entwickelt. In der folgenden Entwicklung eines neuen Stromreglers, soll dieser Prozess in *Simulink* stattfinden. Dabei handelt es sich um eine blockbasierte Programmierung. Über das zusätzliche Programm *TargetLink* kann das Block-Modell des Reglers in einen C-Code konvertiert werden und in die Software-Architektur integriert werden. *TargetLink* ist ein Seriencode Generator, welcher effizienten C-Code direkt aus *Mathworks Simulink/Stateflow* generiert. (*TargetLink*, 2024e) Dieser Prozess wird bereits in der Software-Entwicklung im Unternehmen verwendet, jedoch im Rahmen einer anderen Software-Komponente und zur reinen Architektur und einfachen Abfragen. Eine komplette Funktion, wie in diesem Fall die Stromregelung, wurde bisher noch nicht über *TargetLink* aufgebaut. Aus dem Architektur-Entwurf des *DaVinci Developers* kann jede Software-Komponente in dem AUTOSAR-Format *ArXML* exportiert werden. Dabei handelt es sich um eine *XML*-Datei, welche zur Darstellung hierarchisch strukturierter Daten im Format einer Textdatei dient. Allerdings wird im *ArXML*-Format ein von AUTOSAR definiertes Schema berücksichtigt. Über *TargetLink* kann aus der Datei sowohl eine Datenbank erzeugt als auch ein Blockbild in *Simulink* generiert werden. Dieser Schritt wird für die *AsilA* bereits eingesetzt. Die anschließenden Funktionen der *Runnables* müssen daraufhin in *Simulink* erstellt werden. Während die *AsilA* jedoch reine Sicherheitsabfragen beinhaltet, soll nun getestet werden, ob dieser Prozess ebenfalls für einen komplexen Treiber, wie einen Stromregler, geeignet ist.

3 Entwicklung des Regelkreises

Im folgenden Kapitel wird die Simulation des Magnetventils in einem Regelkreis aufgebaut. Hierzu werden zunächst die grundlegenden Vorgaben durch die Hardware und Anforderungen an den Regler dargelegt. Daraufhin wird der Aufbau der einzelnen Module sowie der Zusammenschluss in einen Regelkreis erläutert. Final wird die Dimensionierung und die aus der Simulation ermittelte Auslegung der Regelparameter erläutert.

3.1 Anforderungen

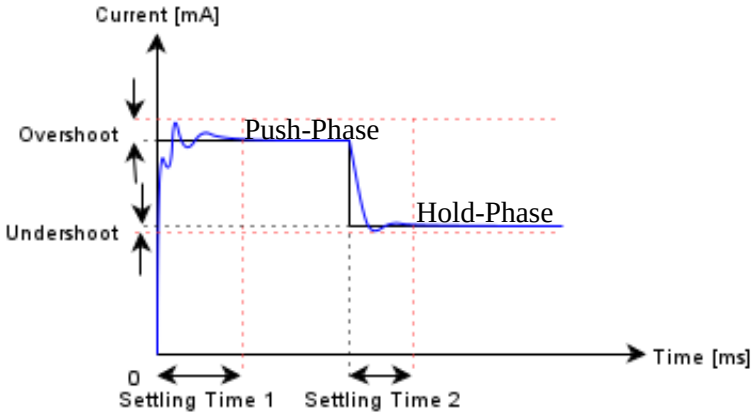


Abbildung 20: Allgemeine Anforderungen an den Stromverlauf bei der Ventilansteuerung (*Vibracoustic* intern)

Zu Beginn werden die grundlegenden Anforderungen an die Regelung definiert. Hierzu können alle notwendigen Werte in der Dokumentation von *Vibracoustic* zu den einzelnen Ventilen, bzw. in diesem Fall konkret zum Auslassventil betrachtet werden. Das Regelverhalten zum Schalten des Ventiles wird, wie bereits erwähnt, in zwei Phasen unterteilt, die *Push*- und die *Hold-Phase*. Beim Auslassventil wird zu Beginn ein *Push-Strom* von 750 mA benötigt, um den Anker in Bewegung zu versetzen. Beim Einregeln auf diesen Wert sind zwei Bedingungen vorgegeben. Einerseits darf die Einregelzeit, welche in Abbildung 20 als *Settling Time 1* beschrieben wird, nicht mehr als 50 ms betragen und andererseits darf die Überschwingweite nicht einen Grenzwert von 50 mA überschreiten. Diese Anforderungen beziehen sich jedoch nicht auf die direkt gemessene Stromstärke, sondern ein durch ein Tiefpassfilter gefiltertes Signal. Die Filterung wird später näher erläutert. Sobald das Ventil vollständig geöffnet ist, kann die Stromstärke auf den vorgegebenen *Hold-Strom* reduziert werden. Da sich eine Detektion des Ventilzustandes als äußers schwierig herausstellt, wird eine Zeitvorgabe für die *Push-Phase* definiert. Nach einem Intervall von 250 ms kann bei ausbleibenden Komplikationen eine vollständige Öffnung des Ventiles angenommen werden. Daraufhin wird ein *Hold-Strom* von 580 mA vorgegeben. Beim Absinken auf diese Stromstärke kann es je nach Reglerauslegung zu einem Unterschwinger kommen. Dabei darf

die Unterschwingweite ebenfalls nicht mehr als 50 mA betragen, um ein erneutes Schließen des Ventiles auszuschließen. Als Einregelzeit ist erneut ein Grenzwert von 50 ms vorgegeben. Abschließend muss die Schließung des Ventiles geregelt werden. Dabei sind allerdings keine großen Anforderungen an den Regler gegeben. Sobald der Befehl zum Schließen auftaucht, muss das System sofort abschalten und die Einregelzeit wird somit nur noch von der Stromkurve der Spule vorgegeben.

Neben der Stromstärke muss außerdem die Spannung bei der Regelung berücksichtigt werden. Hierbei wird ein Spannungsbereich definiert, in welchem die Ansteuerung der Ventile rein funktional möglich ist. Es wird eine Mindestspannung von 8,8 V benötigt. Durch die Mindestanforderung muss sichergestellt werden, dass der *Push-Strom* trotz schwankender Spannung bei dem vorhandenen Gesamtwiderstand im System erreicht werden kann. Des Weiteren wird eine maximale Spannung von 16,2 V vorgegeben. Bei einer zu hohen Eingangsspannung kann es zu einer Schädigung der Hardware kommen. Daher wird bei einer Überschreitung der Obergrenze das System abgeschaltet.

Für die reine Regelung sind damit alle notwendigen Anforderungen definiert. Für die vollständige Simulation müssen noch weitere Werte ermittelt werden. Für die Regelstrecke werden die durch die Hardware definierten Spezifikationen des Magnetventils benötigt. Da allerdings die Arbeit künftig für weitere Ventile eingesetzt werden soll, werden die Werte im Folgenden nur als Variablen definiert, welche somit flexibel für jedes Ventil angepasst werden können. Abschließend werden die verschiedenen Aktualisierungsraten zwischen den Signalen durch die Software vorgegeben. Diese werden erst später näher definiert, da hierzu zunächst die Software-Architektur und die benötigten Signale ermittelt werden müssen.

3.2 Simulation des Magnetventils

Zur Simulation des Magnetventils in *Simulink* wird die *Foundation Library* der Toolbox *Simscape* verwendet. Damit können physikalische Systeme auf Grundlage von physischen Verbindungen direkt im Blockdiagramm aufgebaut werden. (*Simscape,* 2024c) Der Aufbau des Magnetventils findet in einem iterativen Prozess statt, um auf diese Weise die einzelnen Funktionen zu validieren. Der erste Entwicklungsschritt befasst sich mit der Aufladung der Spule. Der Grundaufbau ist ein einfacher elektrischer Kreislauf mit einer Spule und einem Vorwiderstand. Zu Beginn wird eine Spannungsquelle benötigt, die das Magnetventil mit Gleichstrom versorgt. Um in der künftigen Simulation die Spannung während der Simulation flexibel zu gestalten, wird ein *Controlled Voltage Source*-Block verwendet. Dieser Block besitzt neben dem standardmäßigen Ein- und Ausgang zur Integration in den Regelkreis einen weiteren Eingang, zur externen Definierung der Eingangsspannung, welche im Modell mit *U* gekennzeichnet ist. Da in der Software die Spannung in mV und in der Simulation die Spannung in V benötigt wird, muss zuvor eine Umrechnung stattfinden. Als nächstes wird die Spule mit dem *Inductor*-Block integriert.

Hierbei werden drei Parameter abgefragt: die Induktivität, der Widerstand und die parallele Leitfähigkeit. Die Leitfähigkeit kann zunächst vernachlässigt werden und wird daher mit 0 1/Ω festgelegt. Der Widerstand wird ebenfalls mit 0 Ω definiert. Ersatzweise wird dafür ein externer Widerstand vor der Spule integriert. Dieser Schritt dient sowohl zur Veranschaulichung als auch der flexibleren Anpassung des Widerstandes. Für den Widerstand wird die Variable *RL* angelegt. Allgemein werden alle entscheidenden Parameter zunächst als Variablen angelegt, um diese später in einem *Matlab*-Skript flexibel anpassen zu können und so die Simulation ebenfalls für weitere Ventile verwenden zu können. Für die Induktivität wird die Variable *L* angelegt. Zur funktionalen Simulation wird bei physikalischen Systemen immer eine Referenz. Des Weiteren benötigt jedes eindeutig verbundene *Simscape*-Blockdiagramm genau einen sogenannten *Solver Configuration*-Block. (*Solver Configuration*, 2024d) *Simscape* arbeitet mit eigenen Einstellungen für den *Solver*, welche über diesen Block definiert werden. Es besteht jedoch ebenso die Möglichkeit, die Einstellungen des lokalen *Solvers* von *Simulink* zu übernehmen. Diese *Solver*-Definition wird hier eingesetzt.

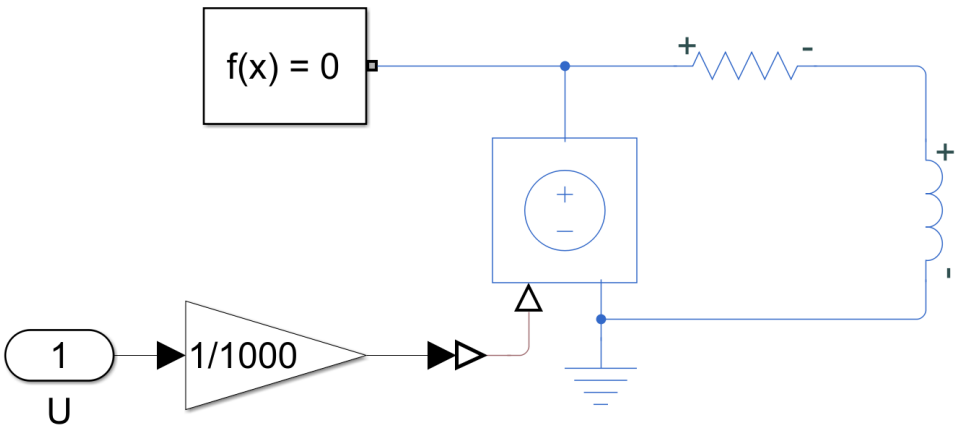


Abbildung 21: Grundlegender Kreislauf zur Aufladung der Spule

Somit ist der grundlegende Aufbau zur Aufladung einer Spule abgeschlossen. Um jedoch die maximale Stromstärke über ein PWM-Signal zu regeln, wird eine Schaltung benötigt, welche die Spule je nach Regelung auf- und entlädt. Hierzu werden zwei Schalter in den elektrischen Kreislauf integriert. Der erste Schalter wird direkt in Reihe mit der Spule geschaltet, um so die Aufladung zu steuern. Der zweite Schalter wird hinter den ersten Schalter und parallel zur Spule eingebunden und dient zur Steuerung der Entladung. Der verwendete *Switch* aus der elektrischen *Simscape Foundation Library* besitzt einen Anschluss für ein externes physikalisches Signal. In den Block-Parametern kann der sogenannte *Threshold*, zu Deutsch Schwellenwert, definiert werden. Dieser Parameter gibt einen Grenzwert vor, welcher vom externen Signal überschritten werden muss, damit der Schalter schließt. Des Weiteren kann der Widerstand im geschlossenen und die Leitfähigkeit im offenen Zustand definiert werden. Da das PWM-Signal als *Simulink*-Signal beschrieben wird, muss über einen *Simulink-PS-Converter* das Signal in ein physikalisches Signal konvertiert werden. Dieses kann daraufhin mit dem Anschluss des ersten Schalters verbunden werden. Als Schwellenwert wird ein Wert von 0,5 gewählt. Da das PWM-Signal, welches im Modell mit *y* als Stellgröße gekennzeichnet ist, nur 1 oder 0 ausgibt, können allgemein alle Werte

dazwischen als Schwellenwert gewählt werden. Somit wird der erste Schalter geschlossen, wenn das Signal eine 1 ausgibt und geöffnet, wenn das Signal eine 0 ausgibt. Damit der zweite Schalter ein gegensätzliches Verhalten aufweist, muss das PWM-Signal zunächst invertiert werden. Dazu wird ein *Simulink Switch* verwendet. Dieser ist mit zwei weiteren Signalen verbunden, den Konstanten 1 und 0. Wenn das Eingangssignal größer 0 ist, wird eine 0 ausgegeben und wenn das Signal kleiner gleich 0 ist, wird eine 1 ausgegeben. Wenn somit der erste Schalter öffnet, wird der zweite Schalter geschlossen und die Spule kann sich über den kurzgeschlossenen Kreislauf entladen. Der Widerstand wird bei beiden Schaltern über die eigene Variable *RS* festgelegt. Die Leitfähigkeit wird in *Simulink* grundlegend auf $1*10^{-8}$ 1/Ω gesetzt und nicht weiter geändert.

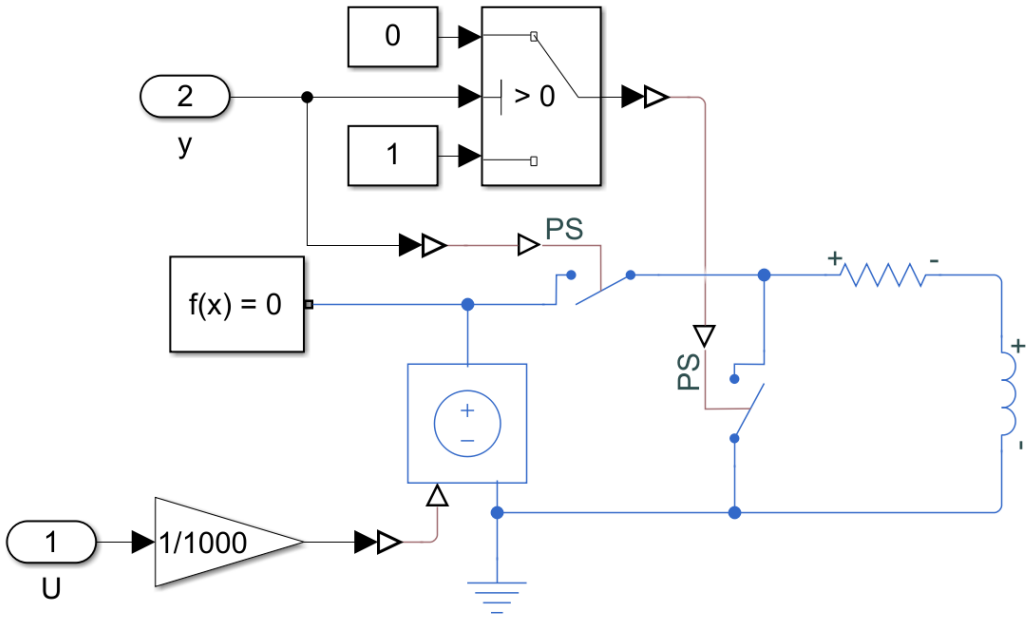


Abbildung 22: Integration der PWM-Steuerung in den Kreislauf der Spule

Im nächsten Schritt wird die Strommessung für den künftigen Regelkreis implementiert. Zunächst wird ein einfacher *Current Sensor* hinter der Spule in Reihe geschaltet. Hierüber wird die direkt gemessene Stromstärke abgenommen und als Ausgangssignal *Ireal* weitergeführt. Das ausgehende physikalische Signal muss über einen *PS-Simulink-Converter* in ein *Simulink*-Signal konvertiert werden, um die Messung in den Regelkreis zurückzuführen. Bei Betrachtung des Strommesswertes zeigt sich ein starkes Rauschen, wie in Abbildung 23 zu erkennen. Dieses Rauschen entsteht infolge der Steuerung der Stromstärke. Um über das PWM-Signal eine geringere Stromstärke zu regeln, muss die Spule immer wieder auf- und entladen werden. Folglich schwankt die Stromstärke zwischen dem maximalen Wert und 0 A.

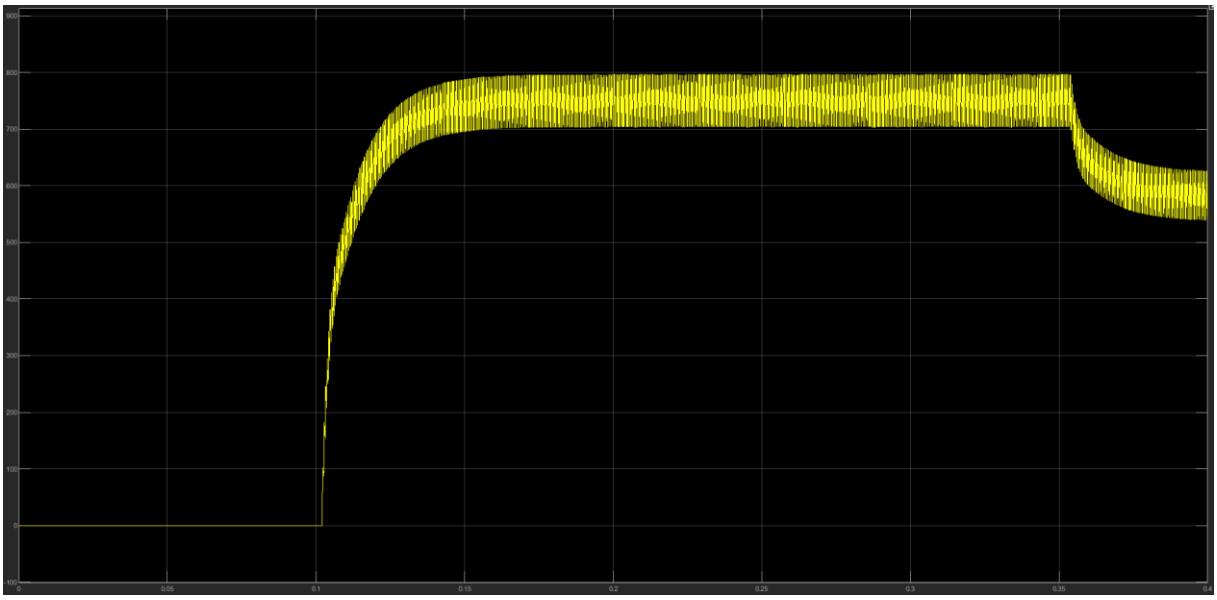


Abbildung 23: Darstellung der rauschenden Stromkurve im *Simulink Scope*

Das Rauschen kann über die Grundfrequenz des PWM-Signales und die Aufladungsgeschwindigkeit der Spule angepasst werden. Die Geschwindigkeit wird über die Eingangsspannung, die Induktivität und den Widerstand definiert. Diese Werte sind jedoch durch die verwendete Hardware fest definiert. Eine Alternative zur Verringerung des Rauschens ist die Filterung des Signales. Hierzu wird ein passiver Tiefpassfilter integriert, welcher ebenfalls im realen System bereits verwendet wird. Die Schaltung eines Tiefpassfilters besteht aus einem Widerstand und einem Kondensator. Das gefilterte Signal wird dabei als Spannung über dem Kondensator abgegriffen. Durch diese Schaltung können unerwünschte hohe Frequenzen eines elektrischen Signales modifiziert, bzw. zurückgewiesen werden. (Storr, 2018) Zunächst wird ein weiterer Widerstand in Reihe zur Spule eingebaut, über den die abfallende Spannung gemessen wird. Dieser Widerstand wird auch *Shunt* genannt und über die Variable *RSenseDV* beschrieben. Zur Spannungsabnahme wird ein *Voltage-Controlled Voltage Source*-Block verwendet. Dieser Block misst die Spannung über dem Widerstand und gibt diese als Quelle mit einer definierten Verstärkung in einem weiteren Kreislauf aus. Die Spannungsverstärkung wird über die Variable *VK* definiert. Die neue Spannungsquelle wird mit einem Widerstand *RC* und einem Kondensator in Reihe geschaltet. Über die Definition des Widerstandes und der Kapazität des Kondensators kann die Grenzfrequenz des Tiefpassfilters angepasst werden. Der Kondensator besitzt in *Simulink* neben der Kapazität ebenfalls die Möglichkeit, einen Vorwiderstand und die Leitfähigkeit einzustellen. Die Kapazität wird mit der Variable *C* gekennzeichnet. Zur Definierung des Tiefpassfilters muss eine Grenzfrequenz f_C angegeben werden. Der Filter lässt somit nur Frequenzen zwischen 0 Hz und der Grenzfrequenz durch. (Storr, 2018) Im Beispielsystem wird ein Tiefpassfilter, dessen Grenzwert als Kreisfrequenz mit 1000 Hz angegeben ist. Zur Ermittlung der eigentlichen Grenzfrequenz muss die Kreisfrequenz durch 2π dividiert werden. Somit ergibt sich eine Grenzfrequenz f_C von 159,2 Hz. Mit diesem Wert kann der Widerstand *RC* vor dem Kondensator über den folgenden Zusammenhang ermittelt:

$$RC = \frac{1}{2\pi * f_c * C} \tag{40}$$

Formel 40: Zusammenhang zwischen dem Vorwiderstand und der Grenzfrequenz beim RC-Tiefpassfilter

Die Einstellung der Leitfähigkeit des Kondensators wird nicht geändert und beträgt grundsätzlich 0 1/Ω. Des Weiteren wird für den elektrischen Kreislauf erneut eine elektrische Referenz benötigt. Zur Messung des gefilterten Signales wird ein *Voltage Sensor* parallel zum Kondensator geschaltet. Der gemessene Wert wird ebenfalls als physikalische Größe ausgegeben und muss erneut in ein *Simulink*-Signal konvertiert werden. Abschließend wird der Spannungswert durch die Eingangsverstärkung und den *Shunt*-Widerstand geteilt, um somit die eigentliche Stromstärke zu erhalten. Da dieses Ausgangssignal zurück an den Regler geführt wird, wird das Ausgangssignal mit *x* für die Regelgröße definiert. Da in der Regelung mit der Stromstärke in mA gerechnet wird und sowohl das gefilterte Signal, als auch die direkt gemessene Stromstärke in A ausgeben werden, müssen beide Signal vor dem Ausgang wieder umgerechnet werden.

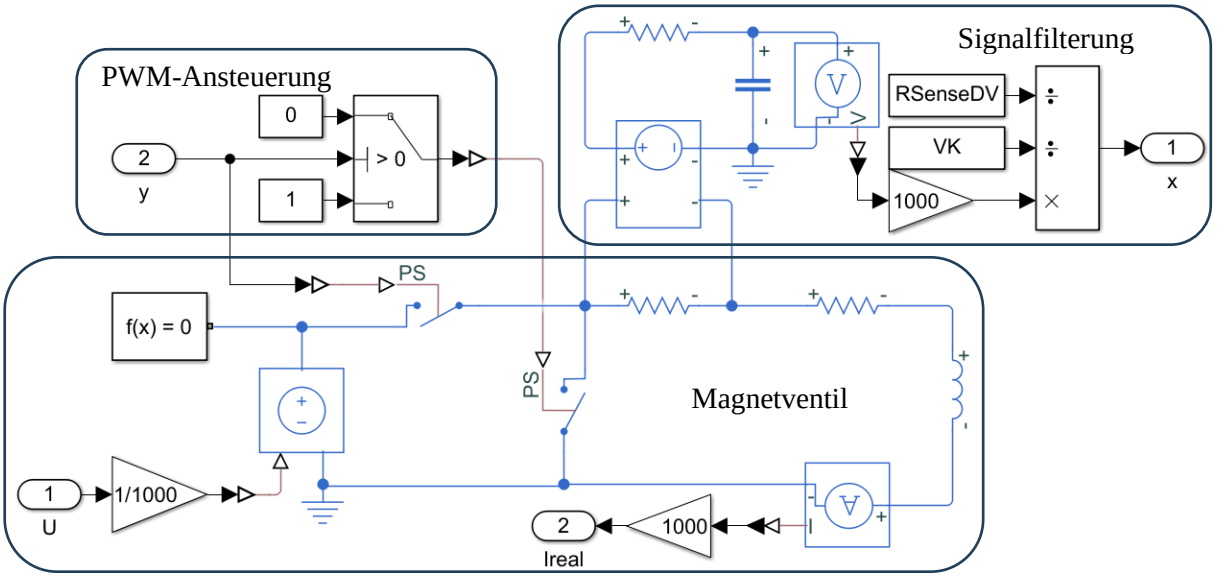


Abbildung 24: Finaler Aufbau des Magnetventils mit Integration der gefilterten Strommessung

3.3 Aufbau des Reglers

Zur grundlegenden Architektur des Reglers wird der Aufbau eines vollständigen PID-Reglers betrachtet und in seine einzelnen Grundfunktionen unterteilt. Zur strukturierten Darstellung des Regelkreises wird für den Regler ein Subsystem in *Simulink* erzeugt. Während in der ersten Version die Regelabweichung als Eingangssignal verwendet wird, muss in der finalen Version die Einbindung in die Software mit den vorhandenen Ein- und Ausgangssignalen berücksichtigt werden.

Zu Beginn wird das P-Glied des Reglers betrachtet. Hierbei wird die Regelabweichung durch den Proportionalbeiwert verstärkt. In *Simulink* wird dazu ein *Gain*-Block mit dem Wert K_P verwendet. Um künftig eine effiziente Dimensionierung des Reglers künftig durchzuführen, werden alle notwendigen Parameter wie bei der Regelstrecke über ein *Matlab*-Skript definiert. Zur graphischen Übersicht werden für die verschiedenen Regler-Anteile Subsysteme in *Simulink* erstellt. Somit ist das P-Glied final gestaltet.

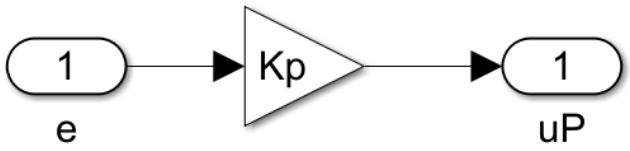


Abbildung 25: Aufbau des P-Gliedes mit der Regeldifferenz *e*, der Verstärkung K_P und der resultierenden Steuergröße *uP*

Als nächstes folgt der Aufbau des I-Gliedes. Bei dem Integral-Glied wird die Regelabweichung über der Zeit aufintegriert. Der rückgemessene Strom, der als Eingangssignal dient, wird jedoch nicht kontinuierlich, sondern über eine definierte Abtastfrequenz *af* gemessen. Der rückgemessene Strom wird im künftigen System mit 500 Hz aktualisiert. Folglich liegt am Eingang ein zeitdiskretes Signal vor, welches ebenso zeitdiskret integriert werden muss. Zur Berechnung der diskreten Flächen wird die Regelabweichung mit dem Zeitintervall pro Abtastung multipliziert. Daraufhin muss dieser Wert mit der vorherigen Gesamtfläche addiert werden. Die finale Regelgröße wird somit zurückführt und um die Zeit der Abtastrate verzögert. Hierzu wird ein *Unit Delay* in *Simulink* verwendet. Der zeitverzögerte Wert entspricht der vorherigen Gesamtfläche und kann anschließend mit dem Wert der neuen diskreten Fläche summiert werden. Für die erste Abtastung wird der Initialwert des *Unit Delays* auf 0 gesetzt. Für das finale I-Glied wird wie beim P-Glied ein Integrierbeiwert K_I über einen *Gain*-Block hinzugefügt. Außerdem muss eine Problematik, die während der ersten Testphasen ermittelt wurde, angepasst werden. Bei einer schnellen Absenkung des Stromes kann es geschehen, dass der I-Anteil negativ wird. Sobald anschließend der Ist-Strom den Nullwert erreicht und somit die Regeldifferenz ebenfalls 0 ergibt, gibt es keine weitere Anpassung des I-Anteils, weshalb bei der darauffolgenden Einschaltung des Ventiles ein abweichendes Verhalten auftreten kann. Daher wird ein *Saturation*-Block integriert, welcher das Minimum auf 0 begrenzt.

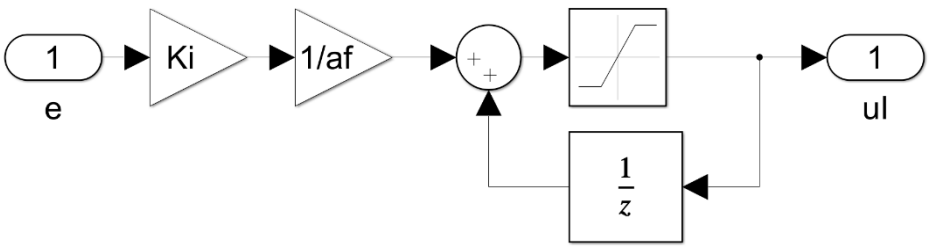


Abbildung 26: Aufbau des I-Gliedes mit der Regeldifferenz *e*, der Verstärkung K_I, der Abtastfrequenz *af* und der resultierenden Steuergröße *uI*

Abschließend wird das Differential-Glied betrachtet. Hierbei wird die zeitliche Veränderung der Regelabweichung betrachtet. Bei der Ableitung muss ebenso wie bei der Integration das diskrete Eingangssignal berücksichtigt werden. Für die zeitdiskrete Ableitung muss somit der Wert der Regelabweichung von der vorherigen Abtastung von der aktuellen Regelabweichung abgezogen werden. Das Eingangssignal wird direkt zu Beginn in zwei Signale aufgeteilt, wobei eines durch einen *Unit Delay* um den Wert der Abtastrate zeitlich verzögert wird. Anschließend wird der zeitverzögerte Wert vom neuen Wert der Regelabweichung abgezogen. Wie bei dem I-Glied wird auch hier der Initialwert des *Unit Delays* für die erste Abtastung auf 0 gesetzt. Die Differenz der Regelabweichung muss anschließend zur Berechnung der Steigung durch die Abtastzeit dividiert werden. Der finale Wert wird ebenfalls über einen *Gain*-Block mit dem Differenzierbeiwert K_D multipliziert.

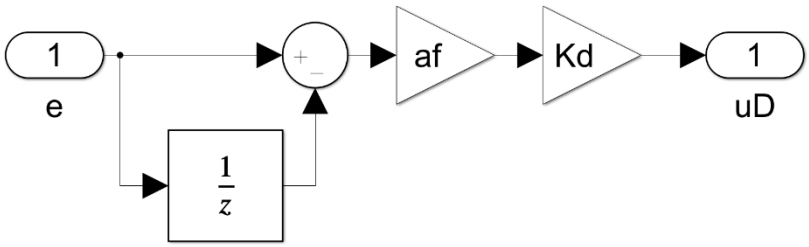


Abbildung 27: Aufbau des D-Gliedes mit der Regeldifferenz *e*, der Abtastfrequenz *af*, der Verstärkung KD und der resultierenden Steuergröße *uD*

Eine Besonderheit beim Differentialglied ist die Tatsache, dass das Übertragungsverhalten durch den Dirac-Impuls nur in der Theorie und nicht in der Realität möglich ist. Zur Stabilisierung des Differentialanteiles wird daher statt eines reinen D-Gliedes ein DT1-Glied verwendet. Der reine D-Anteil wird in einem eigenen Subsystem zusammengefasst, während die Glättung durch die Verzögerungszeit T_U in einem weiteren Subsystem erfolgt. Das Eingangssignal entspricht somit dem Regelwert des reinen D-Gliedes. Anstelle der Verzögerungszeit T_U wird der Filterkoeffizient N gewählt, welcher $1/T_U$ entspricht. Die Übertragungsfunktion des DT1-Gliedes im Bildbereich entspricht:

$$G(s) = \frac{K_D * s}{\frac{1}{N}s + 1} \tag{41}$$

Formel 41: Übertragungsfunktion des DT1-Gliedes im Bildbereich

Die Übertragungsfunktion setzt sich somit aus der Funktion für das D-Glied und der Übertragungsfunktion für das PT1-Glied zusammen. Für das zweite Subsystem wird somit das Zeitverhalten des PT1-Gliedes benötigt.

$$\frac{1}{N} * \dot{v}(t) + v(t) = K * u(t) \tag{42}$$

Formel 42: Übertragungsfunktion des PT1-Gliedes im Zeitbereich

Da der Differenzierbeiwert K_D bereits im D-Anteil reingerechnet wurde, kann die Verstärkung *K* des DT1-Gliedes in der Funktion mit 1 definiert werden. Um die Formel in *Simulink* aufzubauen, wird die Übertragungsfunktion nach $\dot{v}(t)$ umgestellt.

$$\dot{v}(t) = \big(u(t) - v(t)\big) * N \tag{43}$$

Formel 43: Umformung der Übertragungsfunktion des PT1-Gliedes

Zu Beginn muss somit eine Summationsstelle aufgebaut werden, an der das Ausgangssignal *v(t)* vom Eingangssignal *u(t)* subtrahiert wird. Um eine mathematische Schleife zu vermeiden, muss für die Rückführung ein Unit Delay mit einer Verzögerung von einer Abtastung eingebaut werden. Anschließend wird über einen *Gain*-Block die Differenz mit dem Filterkoeffizienten N_{PT1} multipliziert. Um daraufhin das eigentliche Ausgangssignal *v(t)* zu erhalten, muss das Signal zeitdiskret integriert werden. Hierzu wird wie beim I-Glied vorgegangen. Das Signal wird mit dem Zeitintervall pro Abtastung multipliziert, um die Fläche einer konkreten Abtastung zu erhalten. Daraufhin wird über ein *Unit Delay* das Ergebnis mit der vorherigen Fläche summiert.

Das DT1-Glied, bestehend aus den beiden Subsystemen für den D- und den PT1-Anteil wird final in einem Subsystem zusammengefasst. Die drei Subsysteme für die verschiedenen Anteile des PID-Reglers werden an einem *Add*-Block zusammengeführt. Als Eingangssignal des Gesamtreglers dient die Regelabweichung.

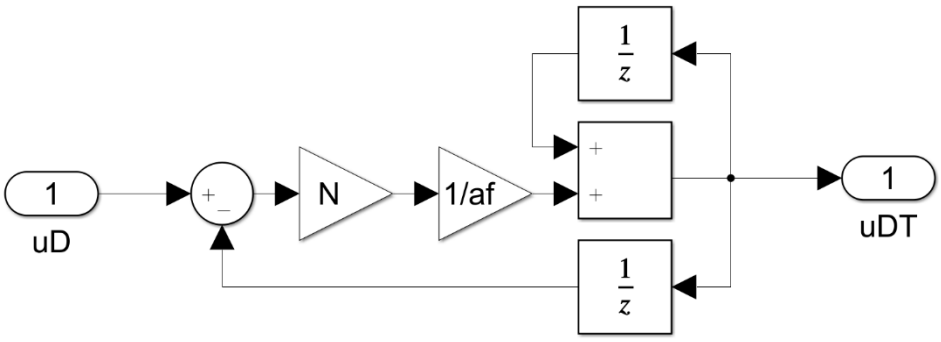


Abbildung 28: Aufbau des PT1-Gliedes mit der Steuergröße des D-Gliedes als Eingang, des Filterkoeffizienten N_{PT1}, der Abtastfrequenz *af* und der insgesamt resultierenden Steuergröße *uDT*

Die resultierende Steuergröße kann noch nicht unmittelbar zur Erzeugung des PWM-Signales verwendet werden. Hierzu wird der prozentuale Anteil der vorgegebenen Stromstärke von der maximalen Stromstärke benötigt. Als weiteres Eingangssignal des Subsystems wird daher die Eingangsspannung benötigt. Die Regelgröße wird über den Gesamtwiderstand des elektrischen Systems in eine Spannung umgerechnet und anschließend durch die Eingangsspannung geteilt. Zur Eingrenzung wird das ausgehende Signal mit einem *Saturation*-Block als Wert zwischen 0 und 1 limitiert.

Somit ist die reine Stromregelung abgeschlossen. Da allerdings als Eingangssignal nur eine einfache Ventilanfrage dient, welche den Wert 1 für *offen* und 0 für *geschlossen* beträgt, muss vor der Stromre-

gelung eine Ermittlung des Sollstromes stattfinden. Wie zuvor erwähnt, wird bei der Öffnung des Ventiles zunächst der *Push-Strom* eingestellt, für ein definiertes Zeitintervall gehalten und anschließend auf den *Hold-Strom* abgesenkt. Dabei müssen einige Bedingungen zeitgleich überprüft werden. Für die Einstellung des Sollstromes eignet sich daher ein Zustandsdiagramm, welches über die *Toolbox Stateflow* von *Simulink* erstellt wird. Mit dieser *Toolbox* kann zunächst ein Diagramm erstellt werden, in welchem Zustände, Bedingungen und Aktionen definiert werden können.

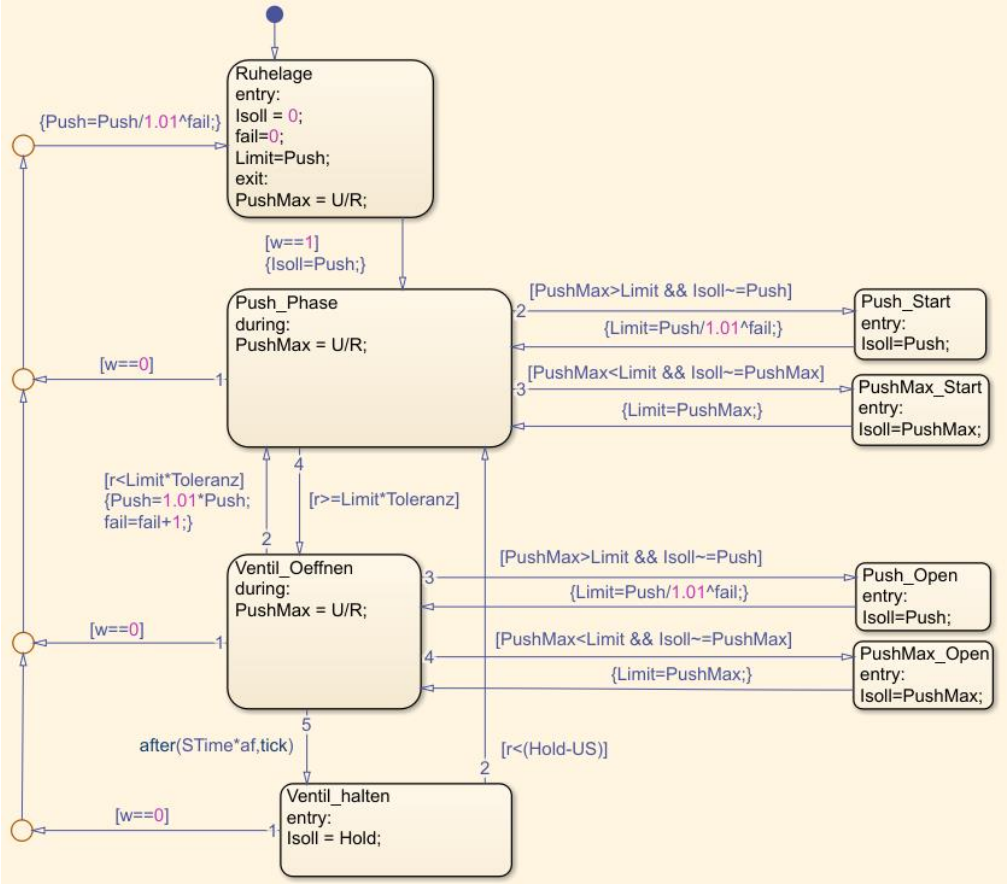


Abbildung 29: Zustandsdiagramm zur Bestimmung des Sollstromes

Als Eingangssignale werden die Ventilanfrage *w*, die Spannung *U* und die Rückführgröße *r* benötigt. Das Ausgangssignal stellt der ermittelte Sollstrom *Isoll* dar. Der Anfangszustand wird als Ruhelage bezeichnet und setzt zunächst alle variablen Werte auf 0. Des Weiteren wird der Grenzwert *Limit* mit dem *Push-Strom Push* gleichgesetzt, was später im Näheren erläutert wird. Sobald die Ventilanfrage gleich 1 ist, wird die Ruhelage verlassen und der Sollstrom auf *Push* gesetzt. Beim Verlassen des Zustandes wird außerdem die Variable *PushMax* berechnet. Hierbei handelt es sich um die maximal erreichbare Stromstärke, welche im nächsten Zustand benötigt wird. Dieser Zustand wird *Push_Phase* genannt. Von hier aus sind mehrere Szenarien möglich. Dabei müssen die Bedingungen nach der Reihenfolge der Abfrage priorisiert werden. Die wichtigste Bedingung stellt die Führungsgröße dar. Wenn während des Zustandes die Ventilanfrage wieder auf 0 gesetzt wurde, muss das System direkt in die Ruhelage zurück und der Sollstrom ebenfalls mit 0 A definiert werden. Die zwei folgenden Bedingungen ergeben sich aus einer Problematik, welche bei vorherigen Systemen festgestellt wurde. Bei der

Ansteuerung der Ventile kann es durch eine spontane Absenkung der Spannung vorkommen, dass die maximal mögliche Stromstärke im Kreislauf geringer ist als der geforderte *Push-Strom*. In diesem Fall wird die Annahme getroffen, dass der maximale Strom ausreicht, um das Ventil zu schalten. Hierzu wird die zuvor erwähnte Variable *PushMax* benötigt, welche während des Zustandes immer wieder aktualisiert wird. Wenn der Wert von *PushMax* unterhalb des geforderten *Push-Stroms* liegt, wird der anfangs definierte Grenzwert *Limit* an den Maximalwert angepasst. Gleichzeitig muss für eine nachträgliche Änderung die umgekehrte Abfrage gestellt und der Grenzwert wieder zurück auf den *Push-Strom* gesetzt werden. Damit sind alle relevanten Randbedingungen abgefragt und die eigentliche Prozesskette kann weitergehen. Sobald der rückgemessene Strom den Grenzwert *Limit* erreicht, geht das Diagramm über in den nächsten Zustand. Es wird in dieser Bedingung allerdings eine Toleranz für den Grenzwert integriert. Je nach Anforderungen kann darüber definiert werden, dass der Zustand bereits ab einer definierten Prozentzahl des Grenzwertes gewechselt werden kann. Im folgenden Zustand wird angenommen, dass durch den erreichten Grenzwert das Ventil schalten kann. Dabei müssen erneut einige Randbedingungen abgefragt werden.

Ebenso wie im Zustand zuvor besitzt die Frage nach der Führungsgröße die größte Relevanz. Außerdem wird weiterhin der Grenzwert über die zwei Abfragen angepasst und der maximale Strom während der Zeit im Zustand mit jeder Abtastung neu berechnet. Anschließend muss abgesichert werden, dass während der Schaltphase kein Einbruch der Stromstärke aufkommt, weshalb in regelmäßigen Zeitabschnitten ein erneuter Abgleich zwischen dem Istwert *r* und dem Sollwert *Limit* stattfinden muss. Dabei wird erneut die zuvor einbezogene Toleranz berücksichtigt. Wenn der Grenzwert unterschritten wird, muss eine Anhebung des *Push-Stromes* und somit des ausgehenden Sollstroms stattfinden, ohne den Grenzwert zu ändern. Daher wird beim Eintreffen der Bedingungen der *Push-Strom* um den Faktor 1,01 angehoben. Aus dieser Notwendigkeit wird deutlich, warum der zusätzliche Grenzwert *Limit* direkt zu Beginn integriert werden musste. Der ursprünglich definierte *Push-Strom* soll weiterhin als Vorgabe für den rückgemessenen Strom gelten, allerdings muss eine Anhebung des Sollstromes stattfinden, wenn der reale Strom durch Störeinflüsse diesen Grenzwert nicht erreichen kann. Des Weiteren kann mit dieser Bedingung die nachträgliche Anpassung des Grenzwertes beschrieben werden. Um nachzuvollziehen wie oft, der *Push-Strom* angehoben wurde, wird der Zähler *fail* integriert. Mit jeder Erhöhung wird der Zähler um 1 erhöht. Um nachträglich wieder den ursprünglichen Wert für den *Push-Strom* zu ermitteln, muss der neue Wert durch den Faktor $1{,}01^{fail}$ geteilt werden. Dieser Faktor ergibt sich aus dem Zusammenhang, dass mit jeder Anhebung des *Push-Stromes* der aktuelle Wert um den Faktor 1,01 erhöht wird. Somit muss der finale Wert bei der Rückrechnung genauso oft durch diesen Wert geteilt werden. Die Anzahl der Divisionen ergibt sich aus dem Zähler *fail*. Die Berechnung wird bei der Anpassung des Sollstromes auf den *Push-Strom* eingesetzt, um den Grenzwert Limit wieder auf den ursprünglichen Wert zurückzusetzen. Wenn sich, während der Schaltphase des Ventiles keine weitere Änderung ergibt, kann die finale Bedingung eintreten. Nach einer wie zu Beginn erwähnten Schaltzeit von

200 ms kann der Sollstrom auf den Betrag des *Hold-Stroms* reduziert werden. Für die zeitliche Bedingung wird in Stateflow der Befehl *after(Time,sec)* verwendet. Das Diagramm geht somit über in den Zustand *Ventil_halten*.

Beim Eintreten des Zustandes wird der Sollstrom *Isoll* mit dem *Hold-Strom Hold* gleichgesetzt. Zum Austreten aus dem Zustand gibt es zwei Bedingungen. Zunächst wird erneut die Ventilanfrage überprüft. Des Weiteren muss die in den Anforderungen erwähnte Bedingung der maximalen Unterschwingweite berücksichtigt werden. Wenn beim Regeln auf den *Hold-Strom* die Unterschwingweite über dem vorgegebenen Grenzwert *US* von 50 mA gerät, muss die *Push-Phase* erneut durchgeführt werden, da nicht garantiert werden kann, dass das Ventil weiterhin offen ist. Um das Diagramm final abzuschließen, wird die Rückrechnung für den ursprünglichen *Push-Strom* vor dem Eintritt in den Anfangszustand integriert.

Das fertige Zustandsdiagramm wird vor der Regelung integriert und der Sollstrom aus dem Diagramm mit der Rückführgröße verglichen, um somit die Regeldifferenz für die anschließende Regelung zu erhalten. Das Gesamtsystem wird als Library für den Regler im Projekt abgespeichert. Die Gesamtdarstellung der Regelung ohne Subsysteme ist im Anhang II vorzufinden.

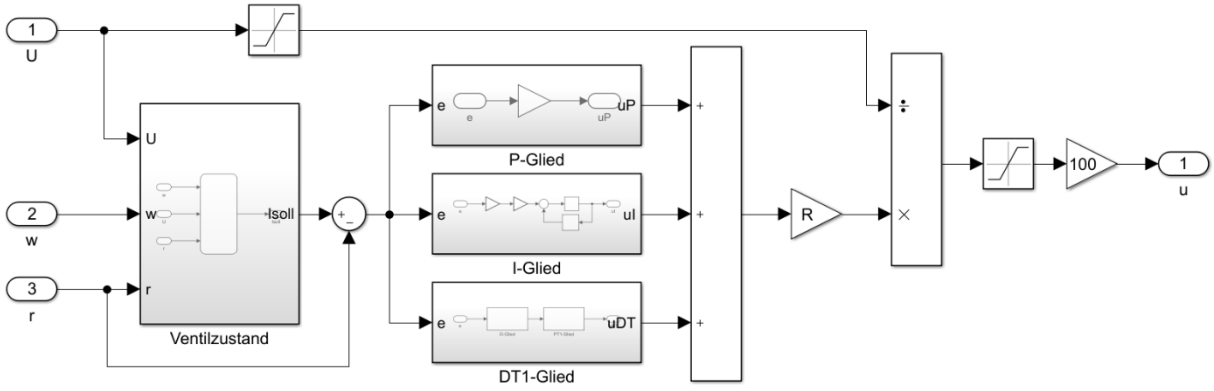


Abbildung 30: Gesamtaufbau des Reglers mit der Eingangsspannung *U*, der Führungsgröße *w*, der Rückführgröße *r* sowie der resultierenden Steuergröße *u* für den PWM-Generator

3.4 Aufbau des PWM-Generators

Zur Generierung eines PWM-Signales in *Simulink* wird ein Grundsignal mit einer konstanten Frequenz benötigt. Hierzu dient ein Dreieckssignal mit einer definierten Grundfrequenz *fPWM* und Amplitude *Amp*. Die Grundfrequenz ist vorgegeben mit 2 kHz. Die Amplitude kann frei gewählt werden. Sie sollte jedoch nicht zu klein sein, um auch geringe Wert gut erfassen zu können. Daher wird ein Wert von 50 gewählt. Das Dreieckssignal wird mit dem zuvor beschriebenen Eingangssignal summiert. Durch das Eingangssignal wird somit die Ruhelage der Schwingung angepasst. Das resultierende Signal wird an ein Relais geschlossen. Ein Relais besitzt nur zwei Zustände *on* und *off*, welche in *Simulink* mit einem Ausgangswert definiert werden. Bei einem PWM-Signal werden diese Zustände mit *high* und *low* oder

auch den logischen Zahlen 1 und 0 beschrieben. Des Weiteren müssen die Grenzwerte in *Simulink* beschrieben werden, an denen das Relais zwischen diesen Zuständen wechselt. Hierbei wird für beide Wechsel der Grenzwert 0 eingetragen. Wenn das Eingangssignal somit größer als 0 ist, wird das Relais auf *on* geschaltet, und wenn das Signal kleiner als 0 ist, wird das Relais auf *off* geschaltet.

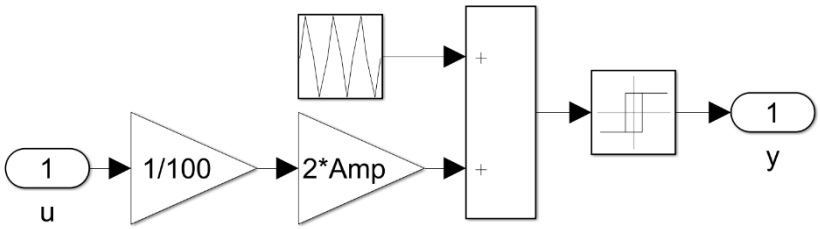


Abbildung 31: Aufbau des PWM-Generators mit der Steuergröße *u*, der Amplitude *Amp* des Dreieckssignales und der Regelgröße *y*

Im Weiteren müssen Anpassungen vorgenommen werden, damit das PWM-Signal den Prozentwert richtig moduliert. Wenn eine Stromstärke von 0 A vorgegeben wird, liegt am Relais das reine Dreiecksignal an. Wenn sich die Ruhelage des Signales bei 0 befindet, würde das Relais eine halbe Periode lang eine 1 und anschließend eine halbe Periode lang eine 0 ausgeben. Nach der Definition der Pulsweitenmodulation würde somit 50 % der maximalen Stromstärke vorgegeben werden. Der maximale Punkt des Dreieckssignales muss somit bei 0 liegen, damit das Relais einen konstanten Nullwert für das PWM-Signal ausgibt. Daher wird die Ruhelage des Signales um den Wert der Amplitude gesenkt. Final muss das Eingangssignal vor der Summierung angepasst werden. Wenn ein Eingangswert von 50 % vorliegt, muss die Ruhelage, wie vorhin ermittelt, bei 0 liegen. Die Ruhelage muss somit um den Wert der Amplitude angehoben werden. Bei einem Eingangswert von 1, muss die Ruhelage um den zweifachen Wert der Amplitude angehoben werden. Somit muss das Eingangssignal mit der doppelten Amplitude multipliziert werden. Dadurch wird die Führungsgröße über das PWM-Signal korrekt vermittelt und die Stellgröße kann an die Regelstrecke geleitet werden.

3.5 Zusammenführung des Regelkreises

Zur Zusammenfassung des Reglers, des PWM-Generators und der Regelstrecke wird ein weiteres Subsystem als Eingangssignal benötigt. Allgemein wird als Eingang für den Regler die vorgegebene Führungsgröße *w*, der gemessene Ist-Strom *r* und die Batteriespannung *U* benötigt. Um eine realistische Batteriespannung zu simulieren, wird vor der Zuführung eine Filterung über einen RC-Filter wie bei der Regelstrecke vorgenommen. So werden impulsartige Änderungen der Spannung vermieden. Des Weiteren werden für den Eingangswert zwei *Step*-Funktionen eingesetzt, welche als Summe an den Tiefpassfilter weitergegeben werden. Beim ersten Block wird der Initialwert auf die halbe Spannung gesetzt und nach einer definierten Schrittzeit *VTime* um einen vorgegebenen Spannungsverlust *dV* reduziert. Der zweite Block besitzt ebenfalls als Initialwert die halbe Spannung, jedoch wird die Schrittzeit um

den Wert einer Haltezeit *VHold* erhöht und anschließend wird die Spannung um den Betrag der Verlustspannung erhöht. Somit kann über beide *Step*-Funktionen ein diskreter Spannungsverlust über einen definierten Zeitbereich simuliert werden.

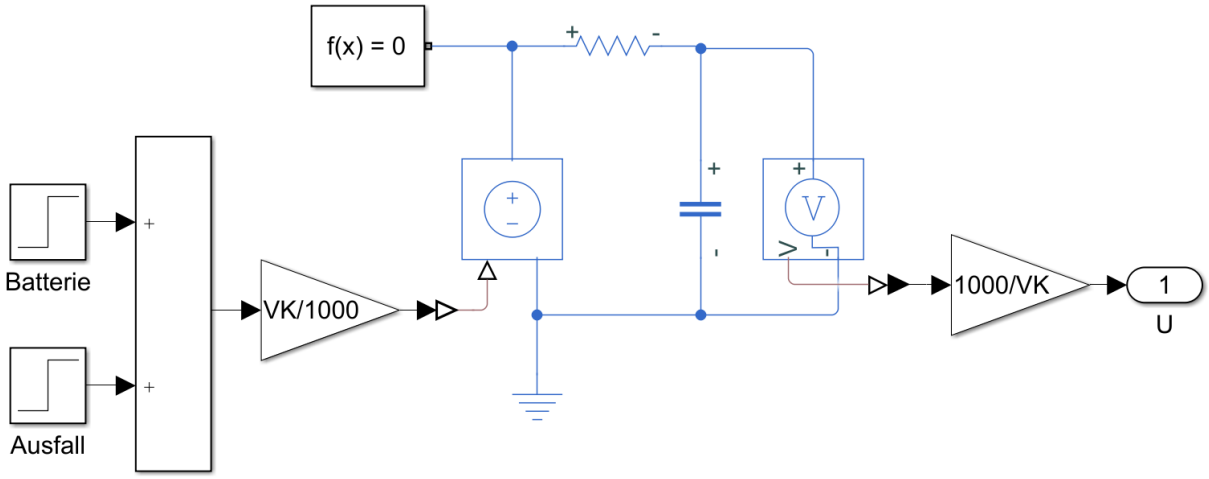


Abbildung 32: Modell zur Simulation der Batteriespannung

Ein weiterer Aspekt für die Simulation ist die Begrenzung der Betriebsspannung. Hierbei ist durch die zuvor genannten Vorgaben eine maximale Spannung von 16,2 V und eine minimale Spannung von 8,8 V definiert. Diese Begrenzung ist relevant für die Schaltung und Regelung des Ventiles und übt sich daher direkt auf die Führungsgröße aus. Daher werden die Führungsgröße und die Batteriespannung in einer *Matlab*-Funktion zusammengeführt und durch eine einfache If-Bedingung wird überprüft, ob die Spannung im Betriebsbereich und das Ventil folglich geschaltet werden kann. Die somit überarbeitete Führungsgröße kann daraufhin an den Regler weitergeleitet werden.

Neben der Führungsgröße *w* wird ebenfalls die Batteriespannung *U* und der rückgemessene Strom *r* an den Regler weitergegeben. Die aus dem Regler resultierende Steuergröße *u* wird über den PWM-Generator in ein PWM-Signal umgewandelt, welches als Stellgröße *y* für die Regelstrecke fungiert. Zusammen mit der Batteriespannung wird das Signal als Eingang an die Regelstrecke weitergeleitet. Aus der Regelstrecke ergeben sich sowohl der direkt gemessene Realstrom *Ireal* sowie der über den Tiefpassfilter geglättete Messstrom, welcher die Regelgröße *x* darstellt. Der Realstrom und der Messstrom werden beide zur Analyse über ein Scope graphisch veranschaulicht. Des Weiteren wird der Messstrom als Rückführgröße *r* zurück an den Regler geleitet. Um eine algebraische Schleife, bei der der Eingang direkt mit dem Ausgang ohne Verzögerung verglichen wird, zu vermeiden, wird ein *Unit Delay* mit dem Wert der *Sample Time* der Gesamtsimulation eingesetzt.

Der grundlegende Aufbau des Regelkreises ist abgeschlossen, allerdings werden in dieser Simulation noch nicht die Aktualisierungsraten der Signale in der Software berücksichtigt. Hierzu werden *Rate Transition*-Blöcke im Folgenden eingesetzt. Die Spannung wird in der Software über das Signal *PwrSply (Power Supply)* mit einer Rate von 100 Hz aktualisiert. Daher wird am Ausgang der Batteriespannung ein *Rate Transition* mit einer *Sample Time* von 10 ms eingefügt. Die Ventilanfrage und der

rückgemessene Strom werden mit einer Rate von 500 Hz und somit einer *Sample Time* von 2 ms aktualisiert. Die genauen Signale und die Verknüpfung mit dem Regler werden in Kapitel 4.3 behandelt. Damit sind die Raten durch die drei *Rate Transition*-Blöcke an die Software angepasst. In der Abbildung 33 werden die verschiedenen Aktualisierungsraten des Regelkreises farblich dargestellt. Dabei stellt blau ein diskretes Signal mit einer Rate von 100 Hz, grün ein diskretes Signal mi einer Rate von 500 Hz und rot ein diskretes Signal mit der Rate des im Modell verwendeten *Solvers* dar. Die *Rate Transition*-Blöcke werden in Gelb dargestellt, da diese den Übergang zwischen zwei verschiedenen Raten bilden. Neben der Darstellung der Aktualisierungsraten ist in Abbildung 33 die Ausgabe der einzelnen Regler-Anteile über die Signale *P_OUT*, *I_OUT* und *D_OUT* zu erkennen.

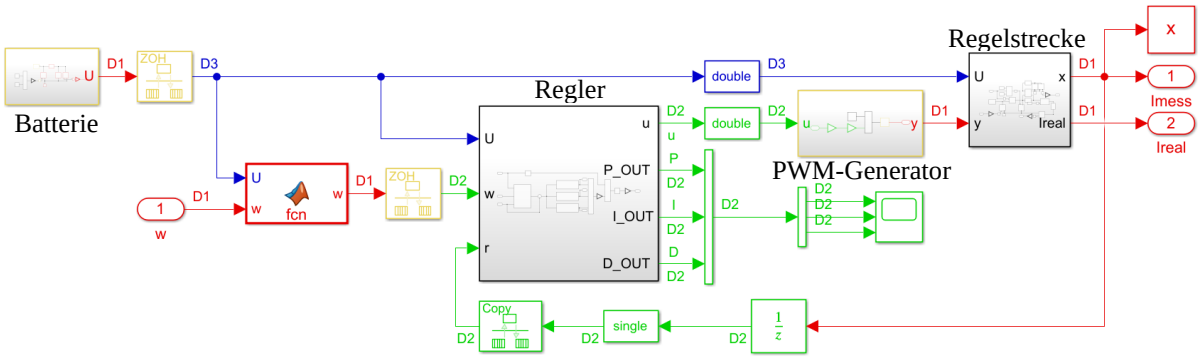


Abbildung 33: Darstellung der verschiedenen Aktualisierungsraten im Regelkreis

Abschließend muss die *Sample Time* der Gesamtsimulation definiert werden. Um das PWM-Signal fehlerfrei zu erfassen, sollte die *Sample Time* mindestens das Hundertfache der PWM-Grundfrequenz betragen. Um außerdem Eigenschwingungen durch die verschiedenen Frequenzen zu vermeiden, wird iterativ ein geeigneter Wert ermittelt. Final ergibt sich dadurch eine *Sample Time* von 4,5 µs. Für den *Solver* wird ein einfacher Euler *ode1* verwendet, da dieser Rechnung der später verwendeten Steuereinheit entspricht.

3.6 Dimensionierung des Reglers

Zur Dimensionierung wird die Sprungantwort der Regelstrecke in *Simulink* ermittelt und über ein *Matlab*-Skript automatisch ausgewertet. Hierzu werden verschiedene mathematische Faustformel-Verfahren getestet. Des Weiteren wird die Sprungantwort zur experimentellen Ermittlung der Übertragungsfunktion der Regelstrecke verwendet. Über die einzelnen Übertragungsfunktionen kann die Gesamtfunktion des offenen Regelkreises zur Betrachtung der Stabilitätsgrenzen eingesetzt werden. Aus den Ergebnissen der Faustformel-Verfahren werden unter Berücksichtigung der Gesamtstabilität die Reglerparameter für verschiedene Reglertypen hergeleitet und über die Darstellung von unterschiedlichen Szenarien in der Simulation miteinander verglichen.

3.6.1 Sprungantwort der Regelstrecke

Für die Ermittlung der Sprungantwort der Regelstrecke wird ein neues *Simulink*-Modell aufgebaut. Hierbei wird anstelle einer Regelung eine einfache Steuerung mit einem Einheitssprung erstellt. Der Einheitssprung wird dabei als Eingangssignal an den PWM-Generator weitergeleitet. Es wird somit nicht der Sollwert der Stromstärke auf 1, sondern der prozentuale Anteil der Stromstärke von der Gesamtstromstärke auf 1 % gesetzt. Der daraus resultierende Realstrom vom *Current Sensor* und der gefilterte Messstrom können über ein Scope visualisiert werden und zeigen die Sprungantwort der Regelstrecke. Um vergleichbare Ausgangsbedingungen sicherzustellen, wird für den *Solver* ebenfalls ein einfacher Euler mit der derselben *Sample Time* eingestellt. Zur anschließenden Verarbeitung der Sprungantwort in einem *Matlab*-Skript werden die Werte des Messstromes als *Timeseries* im *Matlab Workspace* abgespeichert. Mit diesen Daten wird im folgenden Kapitel gearbeitet.

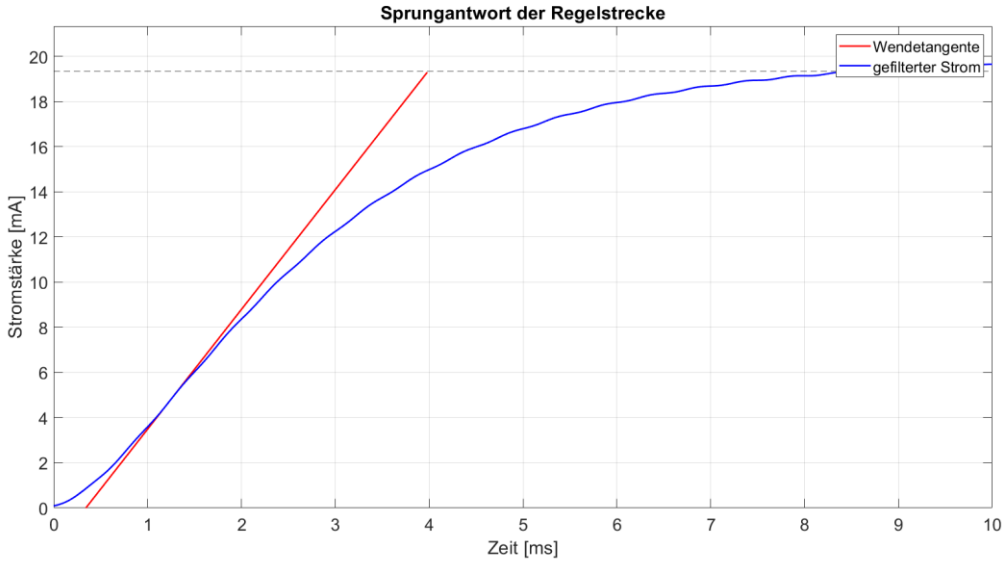


Abbildung 34: Ermittlung der Wendetangente an der Sprungantwort der Regelstrecke

3.6.2 Übertragungsfunktion der Regelstrecke

Zur künftigen Dimensionierung und Stabilitätsanalyse wird in einem weiteren Skript aus der Sprungantwort die Übertragungsfunktion G_S der Regelstrecke ermittelt. Dafür wird zunächst nach einer bereits bekannten Übertragungsfunktion mit einem ähnlichen Sprungverhalten gesucht. Es zeigt sich, dass die Sprungantwort des Verzögerungsgliedes erster Ordnung deutliche Parallelen zum Verhalten der Regelstrecke aufweist. Die Übertragungsfunktion des PT1-Gliedes ist bereits im Kapitel 2.4.1.4 beschrieben worden.

Die Verstärkung *K* und die Verzögerungszeit *T* der Regelstrecke lassen sich, wie ebenfalls im Kapitel 2.4.1.4 beschrieben, aus der Sprungantwort ablesen. Für die vorliegende Regelstrecke einschließlich des PWM-Generators kann somit eine Verstärkung von 18,46 und eine Verzögerungszeit von 3,1 ms ermittelt werden. Zur Validierung der Ergebnisse wird die mathematische Sprungantwort über eine Zeitspanne von 5 ms in einem Plot dargestellt und mit dem Resultat der Simulation verglichen. Das Skript, welches zur Ermittlung der Parameter verwendet wurde, ist im Anhang III vorzufinden.

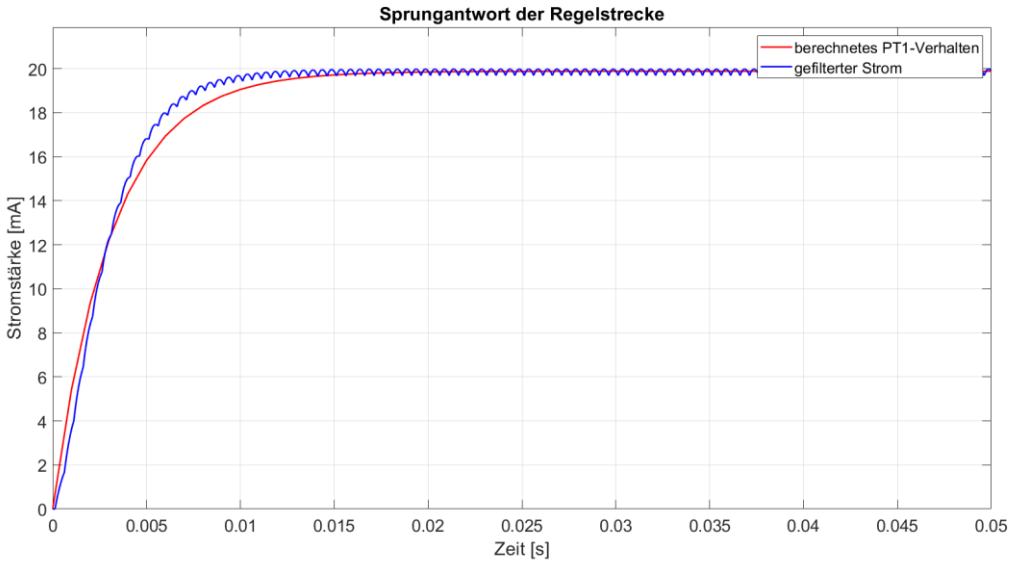


Abbildung 35: Vergleich zwischen der Sprungantwort aus der Simulation und dem berechneten PT1-Verhalten

3.6.3 Mathematische Verfahren zur Dimensionierung

Es gibt viele verschiedene Faustformel-Verfahren zur Ermittlung geeigneter Regelparameter, von denen in diesem Kapitel einige erläutert werden. Mehrere Verfahren arbeiten dabei mit der Wendetangente der Sprungantwort. Hierzu wird zunächst die *Timeseries* in zwei *Arrays*, eines für die Zeit und das andere für die Messwerte, aufgeteilt. Daraufhin muss das *Array* für die Messwerte als Funktion über der Zeit betrachtet und abgeleitet werden. Die abgeleitete Funktion wird in einem neuen *Array* abgespeichert. Die Wendetangente der Sprungantwort befindet sich am Punkt der maximalen Steigung. Daher wird über die Funktion *max* sowohl der Extremwert der Ableitung als auch dessen Position im *Array* ermittelt. Über die Position kann anschließend die gemessene Stromstärke der Sprungantwort ermittelt werden und über den Steigungswert die Wendetangente an diesem Punkt eingezeichnet werden. Hierbei zeigt sich jedoch eine Problematik. Da auch das gefilterte Signal meist immer noch ein leichtes Rauschen aufweist, weist die Ableitung ebenfalls eine starke Schwingung auf. Somit erweist es sich als kritisch, die realen Werte für die Wendetangente zu ermitteln. Eine leichte Veränderung der Ausgangswerte sorgt somit für eine starke Änderung der Steigung und folglich der Streckenparameter. Des Weiteren kann nicht die mathematische Darstellung der Regelstrecke über das PT1-Glied verwendet werden,

da bei den folgenden Verfahren eine Verzugszeit in der Sprungantwort erwartet wird. Daher wird eine Glättung der Stromkurve über eine interne Filterfunktion von *Matlab* durchgeführt. Hierzu wird die Funktion *smoothdata* mit der Definition eines Gauß-Filters verwendet. Die geglättete Stromkurve wird in Abbildung 36 dargestellt.

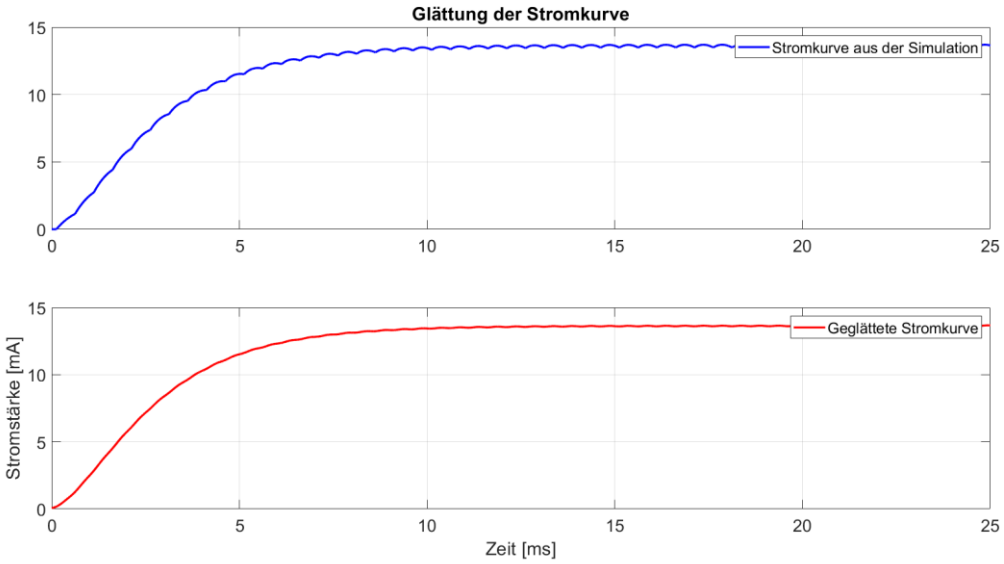


Abbildung 36: Glättung der aus der Simulation ermittelten Stromkurve

Zunächst wird das Verfahren nach Ziegler/Nichols getestet. Ziegler und Nichols haben insgesamt zwei Verfahren entwickelt, wobei hier nur die Reglerbestimmung anhand der Sprungantwort betrachtet wird. Dazu müssen einige Streckenparameter mithilfe der vorherig definierten Wendetangente ermittelt werden. Als erstes wird die Verzugszeit T_U ermittelt. Hierbei handelt es sich um die Zeit, in der die Sprungantwort keine nennenswerte Änderung aufweist. Sie wird über die Zeit definiert, welche zwischen dem Sprung des Eingangssignales und dem Schnittpunkt der Wendetangente mit der x-Achse liegt. Als nächstes wird die Ausgleichszeit T_g ermittelt. Dieser Streckenparameter beschreibt die Zeit, die für den Übergang zwischen den stabilen Zuständen benötigt wird. Der Wert wird über die Zeitspanne definiert, welche zwischen dem Schnittpunkt der Wendetangente mit der x-Achse und dem Schnittpunkt der Wendetangente mit dem Beharrungswert der Sprungantwort liegt. Final wird die Verstärkung der Strecke K_S ermittelt. Hierbei handelt es sich um die Verstärkung des Einheitssprunges durch die Regelstrecke. Mit der vollständigen Ermittlung der notwendigen Streckenparameter kann über die folgende Tabelle 1 eine bereits annähernde Bestimmung der Regelparameter für die verschiedenen Reglertypen durchgeführt werden: (Beier & Wurl, 2013, S. 198-199)

Reglertyp	K_P	T_n	T_V
P	$\frac{T_g}{K_s * T_U}$	-	-
PI	$0{,}8 * \frac{T_g}{K_s * T_U}$	$3 * T_U$	-
PID	$1{,}2 * \frac{T_g}{K_s * T_U}$	$2 * T_U$	$0.42 * T_U$
PD	$1{,}2 * \frac{T_g}{K_s * T_U}$	-	$0{,}25 * T_U$

Tabelle 1: Bestimmung der Regelparameter nach Ziegler/Nichols

Während in dem vorher aufgebauten Modell für den Regler der Integrierbeiwert K_I und der Differenzierbeiwert K_D verwendet wird, wird über diese Tabelle die Nachstellzeit T_n des I-Anteiles und die Vorhaltzeit T_V des D-Anteiles verwendet. Bei der Nachstellzeit handelt es sich um die Zeit, um die ein PI-Regler schneller als ein reiner I-Regler ist. Die Vorhaltzeit gibt wiederum die Zeit an, um die ein PD-Regler schneller als ein reiner P-Regler ist. Beide Zeiten können über die folgenden Formeln in die entsprechenden Beiwerte umgerechnet werden:

$$K_I = \frac{K_P}{T_n} \tag{44}$$

Formel 44: Zusammenhang zwischen dem Integrierbeiwert und der Nachstellzeit

$$K_D = K_P * T_V \tag{45}$$

Formel 45: Zusammenhang zwischen Defferenzierbeiwert und Vorhaltzeit

Über die Einstellparameter nach Ziegler/Nichols kann allerdings nur ein definiertes Regelverhalten eingestellt werden. Für eine Anpassung der Parameter nach dem gewünschten Regelverhalten gibt es das Verfahren nach Chien, Hrones und Reswick, welches auf dem vorherigen Verfahren aufbaut. Hierzu werden die gleichen Streckenparameter wie zuvor benötigt. Für die Bestimmung der Einstellparameter kann jedoch zwischen verschiedenen Optimierungen gewählt werden. Zum einen kann zwischen einem optimalen Führungsverhalten und Störverhalten unterschieden werden und zum anderen kann ein aperiodischer Verlauf oder ein 20 % Überschwinger für die Einregelung gewählt werden. Die Werte für die verschiedenen Reglertypen lassen sich über die folgenden Tabellen ermitteln: (Beier & Wurl, 2013, S.201, S. 202)

Reglertyp	Parameter	Aperiodischer Verlauf	20% Überschwinger
P	K_P	$0{,}3 * \frac{T_g}{K_s * T_U}$	$0{,}7 * \frac{T_g}{K_s * T_U}$
PI	K_P	$0{,}35 * \frac{T_g}{K_s * T_U}$	$0{,}6 * \frac{T_g}{K_s * T_U}$
	T_n	$1{,}2 * T_g$	T_g
PID	K_P	$0{,}6 * \frac{T_g}{K_s * T_U}$	$0{,}95 * \frac{T_g}{K_s * T_U}$
	T_n	T_g	$1{,}35 * T_g$
	T_V	$0{,}5 * T_U$	$0{,}47 * T_U$

Tabelle 2: Bestimmung der Regelparameter nach Chiens, Hrones, Reswick für ein optimales Führungsverhalten

Reglertyp	Parameter	Aperiodischer Verlauf	20% Überschwinger
P	K_P	$0{,}3 * \frac{T_g}{K_s * T_U}$	$0{,}7 * \frac{T_g}{K_s * T_U}$
PI	K_P	$0{,}6 * \frac{T_g}{K_s * T_U}$	$0{,}7 * \frac{T_g}{K_s * T_U}$
	T_n	$4 * T_U$	$2{,}3 * T_U$
PID	K_P	$0{,}95 * \frac{T_g}{K_s * T_U}$	$1{,}2 * \frac{T_g}{K_s * T_U}$
	T_n	$2{,}4 * T_U$	$2 * T_U$
	T_V	$0{,}42 * T_U$	$0{,}42 * T_U$

Tabelle 3: Bestimmung der Regelparameter nach Chiens, Hrones, Reswick für ein optimales Störverhalten

In der Abbildung 34 wird die aus der gefilterten Sprungantwort ermittelten Wendetangente dargestellt. Bei der Dimensionierung des Reglers über diese Verfahren zeigen sich jedoch einige Probleme. Bei der Filterung muss darauf geachtet werden, dass zwar das Rauschen reduziert, allerdings dabei der Verlauf dennoch nicht zu stark verfälscht wird. Des Weiteren muss bei Betrachtung des Zeitraumes, in dem sich die Wendetangente befindet, beachtet werden, dass eine leichte Veränderung der Steigung bereits eine starke Auswirkung auf die Ergebnisse haben kann. Bei der Dimensionierung des Reglers über diese Methoden zeigen sich daher sehr unterschiedliche Parameter, welche allerdings in nur wenigen Situati-

onen für ein gutes Regelverhalten sorgen. Daher muss eine neue Methode gewählt werden, welche robuster gegen periodische Störanteile reagiert. Hierfür eignet sich die Methode der Summenzeitkonstante, auch *T-Summen Regel* genannt.

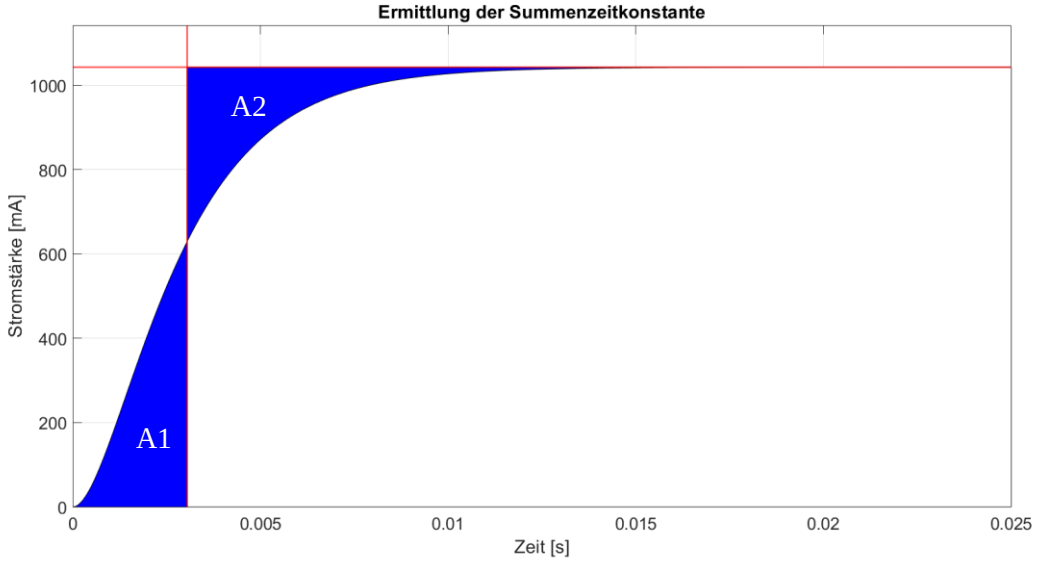


Abbildung 37: Darstellung der Flächen A1 und A2 für die Methode der Summenzeitkonstante

Die Methode der Summenzeitkonstante arbeitet nicht mit der Wendetangente der Sprungantwort, weshalb keine weitere Glättung der Messdaten notwendig ist. Für diese Methode werden zwei Streckenparameter benötigt. Zu Beginn wird erneut die Streckenverstärkung K_S abgelesen. Die zweite Kenngröße ist die Summenzeitkonstante T_Σ. Hierbei handelt es sich um den Zeitpunkt, an dem die Fläche *A1* genauso groß ist wie die Fläche *A2*, welche in der Abbildung 37 dargestellt werden.

Zur automatisierten Ermittlung der Regelverstärkung wird der maximale Wert aus dem *Array* der Messwerte ermittelt. Des Weiteren muss über die Funktion *numel* die Anzahl der Messwerte ermittelt werden. Daraufhin wird eine Vorschleife mit *i=1:1:n* gebildet, wobei *n* die zuvor ermittelte Anzahl der Messwerte darstellt. In jedem Durchlauf werden die beiden Flächen berechnet und miteinander verglichen. Die Fläche *A1* ergibt sich aus der Summe aller Messwerte von 1 bis zum aktuellen Durchlauf *i*. Für die zweite Fläche *A2* wird zunächst die Fläche unterhalb des Beharrungswertes vom aktuellen Durchlauf bis zum Endwert über $K_S*(n-i)$ ermittelt und anschließend davon die Gesamtfläche unterhalb der Sprungantwort abgezogen. Die beiden Flächen berechnen sich in *Matlab* somit wie folgt:

$$A1 = sum(y(1{:}\,i);$$

$$A2 = K_S * (n - i) - sum\big(y(i{:}\,n)\big);$$

Während die Fläche *A1* bei 0 beginnt und mit jedem Durchlauf zunimmt, beginnt die Fläche *A2* bei ihrem Maximum und nimmt anschließend mit jedem Durchlauf ab. Da durch den diskreten Vergleich der Flächen der exakte Zeitpunkt übersprungen werden kann, wird folglich über eine *if*-Bedingung er-

mittelt, in welchem Durchlauf die Fläche *A1 >= A2* ist. Sobald diese Bedingung zutrifft, wird der Zeitpunkt über das *Zeitarray* mit dem aktuellen Iterationsindex *i* ermittelt. Das Skript ist im Anhang IV vorzufinden. Wenn beide Kenngrößen vorliegen, können die Regelparameter für die verschiedenen Reglertypen aus der folgenden Tabelle ermittelt werden: (Horn)

Reglertyp	$\mathbf{K_P}$	$\mathbf{K_I}$	$\mathbf{K_D}$
P-Regler	$\frac{1}{K_S}$	-	-
PI-Regler	$\frac{1}{2 * K_S}$	$\frac{K_P}{0{,}5 * T_\Sigma}$	-
PD-Regler	$\frac{1}{K_S}$	-	$K_P * 0{,}33 * T_\Sigma$
PID-Regler	$\frac{1}{K_S}$	$\frac{K_P}{0{,}66 * T_\Sigma}$	$K_P * 0{,}17 * T_\Sigma$

Tabelle 4: Bestimmung der Regelparameter für verschiedene Reglertypen über die Summenzeitkonstante

Die verschiedenen Regelparameter werden abschließend in einer allgemeinen Matrix für die Methode der Summenzeitkonstante zusammengefasst. Dabei muss berücksichtigt werden, dass sich die Werte auf die Regelung der Steuergröße für den PWM-Generator beziehen. Der eigentliche Regler jedoch berechnet eine Stromstärke, welche über eine nachträgliche Faktorisierung erst in die eigentliche Steuergröße umgerechnet wird. Daher müssen die einzelnen Regelparameter ebenfalls umgerechnet werden. Dabei wird die Matrix mit der anliegenden Spannung multipliziert und anschließend durch den Widerstand und den Faktor 100 dividiert. Somit ergeben sich die finalen Regelparameter.

3.6.4 Analyse und Vergleichskriterien verschiedener Reglertypen

Zur Ermittlung eines geeigneten Reglers wird das Reaktionsverhalten in verschiedenen Szenarien sowie die Stabilität des Regelkreises betrachtet. Hierzu muss eine Simulation aufgebaut werden, die bei jedem Durchlauf die identischen Ausfälle darstellt. Dabei werden verschiedene Störsignale betrachtet. Das untersuchte Szenario wird über ein zeitliches Intervall von einer Sekunde betrachtet. Zunächst wird die Ventilanfrage bei 0,02 s auf 1 gesetzt und das Einschwingen auf den *Push-Strom* analysiert. Daraufhin wird bei 0,1 s die Ventilanfrage für 5 ms auf 0 und anschließend direkt wieder auf 1 gesetzt. Somit wird ein kurzzeitiger Abbruch während der *Push-Phase* dargestellt. Anschließend wird die Ventilanfrage erst wieder bei 0,4 s auf 0 gesetzt. Somit bleibt ein ausreichender Zeitbereich, in dem sowohl die *Push-Phase* als auch der Übergang zum *Hold-Strom* betrachtet werden kann. Im nächsten Schritt wird eine Signalschwankung direkt zu Beginn simuliert. Hierzu wird bei 0,5 s erst die Ventilanfrage für 2 ms auf 1 und

direkt wieder auf 0 gesetzt, um daraufhin die Anfrage bei 0,51 s wieder auf 1 zu setzen. Dieser Wert wird bis zum Ende der Simulation nicht weiter verändert. Für die Integration dieser Szenarien in *Simulink* wird ein sogenannter *Signal Builder* für die Führungsgröße verwendet.

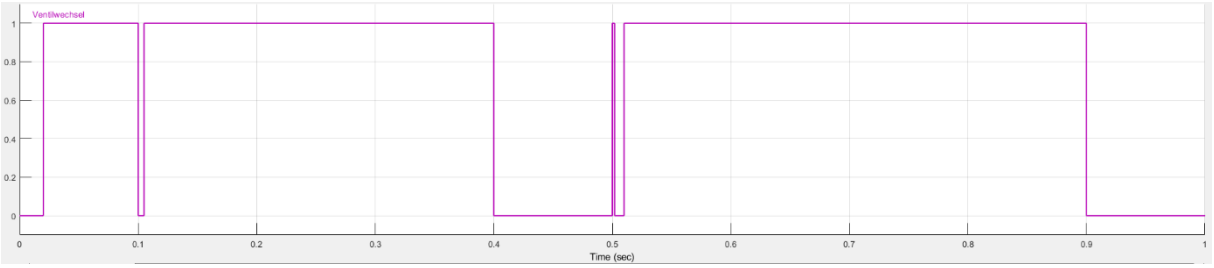


Abbildung 38: Verlauf der Führungsgröße über den *Signal Builder*

Eine weitere Störgröße ist der Spulenwiderstand. Beim Öffnen des Ventiles wird durch die Bewegung des Ankers eine Selbstinduktion erzeugt, welche für eine Verlustleistung und folglich einen Einfall der Stromstärke sorgt. Zur Simulation des Stromabfalles wird Daher bei Erreichen des *Push-Stromes* kurzzeitig der Widerstand erhöht und nach einer definierten Zeit von 5 ms wieder zurückgesetzt. Dabei wird der Widerstand um 1 Ω angehoben. Der Wert ist zunächst eine beliebige Annahme und wird im späteren Verlauf angepasst. Zur Realisierung in der Simulation werden ein weiteres Zustandsdiagramm und ein Vorwiderstand, dessen Wert extern eingestellt werden kann, verwendet. Abschließend wird das Verhalten bei einer leichten Verringerung der Spannung betrachtet. Hierzu wird die Spannung von 14 V ab 0,7 s bis zum Ende der Simulation um 4 V reduziert.

Zur Dimensionierung der Regler wird zunächst das zuvor aufgebaute Verfahren der Summenzeitkonstante eingesetzt. Nach dem Durchlauf des Skriptes ergibt sich somit die folgende Abbildung 39 für die Sprungantwort mit anschließender Auswertung:

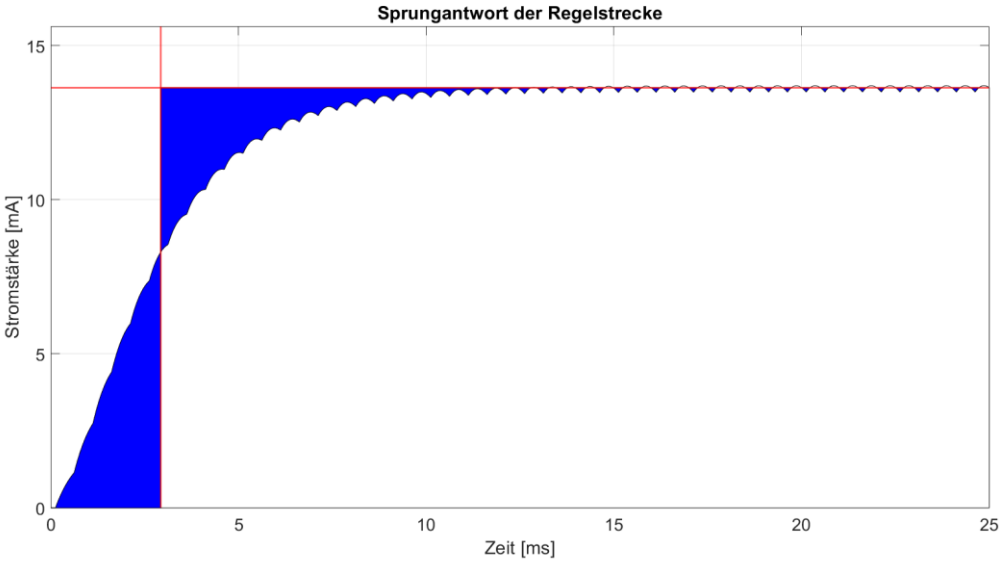


Abbildung 39: Ermittlung der Summenzeitkonstante anhand der Sprungantwort der Regelstrecke

Reglertyp	K_P	K_I	K_D
P-Regler	0,7650	0	0
PI-Regler	0,3825	261,5530	0
PD-Regler	0,7650	0	0,0007
PID-Regler	0,7650	396,2925	0,0004

Tabelle 5: Bestimmung der Regelparameter über die Regelstrecke des Magnetventils

Mit den entwickelten Szenarien und ermittelten Parametern kann im Folgenden die finale Auslegung des Stromreglers für das Auslassventil beginnen.

3.6.5 Finale Auslegung

In diesem Kapitel werden die verschiedenen Regler getestet und teils angepasst, um so den finalen Reglertyp mit den passenden Regelparametern zu ermitteln. Begonnen wird dabei mit dem P-Regler.

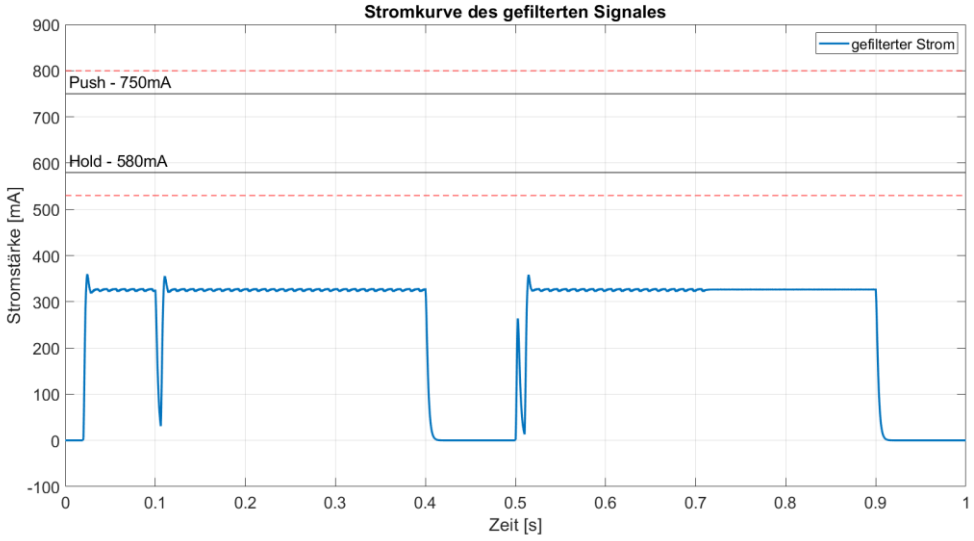


Abbildung 40: Regelung des berechneten P-Reglers in verschiedenen Szenarien

Aus dem Ergebnis wird deutlich, dass der P-Regler eine zu geringe Verstärkung besitzt, um die gewünschten Werte für die Stromstärke zu erreichen. Zum allgemeinen Verhalten kann gesagt werden, dass ein einzelner Überschwinger mit anschließender Einregelung zu erkennen ist. Es zeigt sich außerdem in den verschiedenen Szenarien ein sehr konstantes Einschwingverhalten. Durch die Verringerung der Spannung ab 0,7 s ist eine Glättung des Signales zu erkennen. Um zu verstehen, warum durch die Methode der Summenzeitkonstante eine zu geringe Verstärkung ermittelt wurde, wird das Szenario in Abbildung 41 erneut mit einem Proportionalbeiwert von 2 dargestellt.

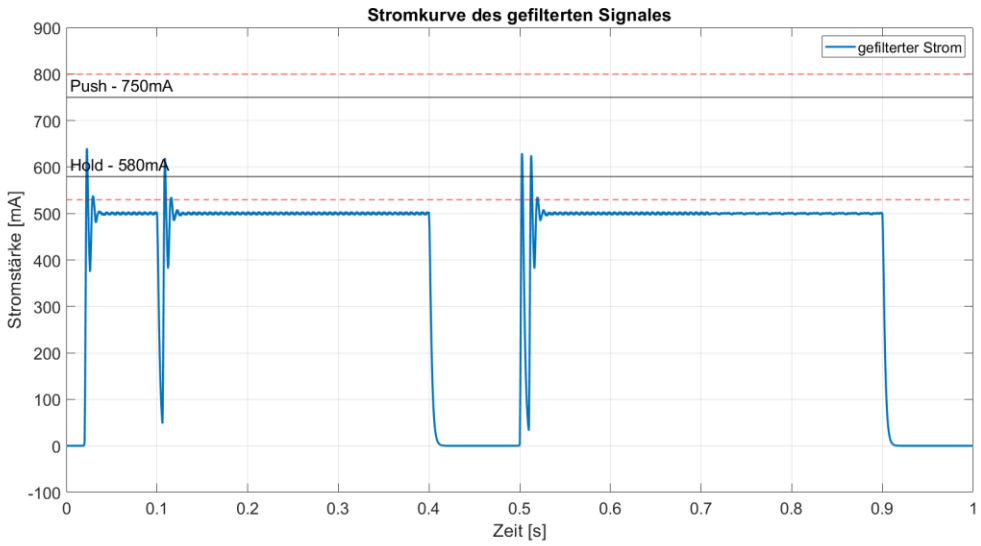


Abbildung 41: Regelung des angepassten P-Reglers in verschiedenen Szenarien

Aus diesen Ergebnissen wird deutlich, dass die Anhebung des Proportionalbeiwertes schnell für ein stark schwingendes Einregelverhalten sorgt. Durch die Methode der Summenzeitkonstante wird hingehen im Optimalfall, wie in Abbildung 40 zu sehen ist, ein Einschwingverhalten mit nur einem Überschwinger erzeugt. Somit wird deutlich, dass auch durch eine weitere Anpassung ein reiner P-Regler nicht für die Stromregelung geeignet ist.

Im nächsten Schritt wird der PI-Regler im Stromregelkreis getestet. In der Abbildung 42 ist zu erkennen, dass die erzielten Werte, für den *Push*- und den *Hold-Strom* erreicht werden können. Der Regler zeigt allgemein eine schnelle Einregelung mit einem Überschwinger, welcher jedoch nicht über den aus den Anforderungen gegebenen Grenzwert von 800 mA hinausgeht. Das Regelverhalten ist schnell und dennoch stabil. Der kleine Einbruch vor Erreichen des *Push-Stromes* ergibt sich durch die simulierte Widerstandsänderung während der Öffnung des Ventiles. Beim Absenken auf den *Hold-Strom* ist kein Unterschwingen zu erkennen, weshalb hier ebenfalls die Bedingungen erfüllt werden. Durch die Verringerung der Spannung bei 0,7 V kommt es zu einem kleinen Einbruch der Stromstärke, weshalb durch die Fehlerregelung im Zustandsdiagramm der Sollstrom im nachliegenden Verlauf leicht angehoben wurde. Die Verzögerung des Einbruches ergibt sich aus der Aktualisierungsrate von 100 Hz für die Batteriespannung. Der Regler erfüllt damit alle Anforderungen und ist für die künftige Stromregelung geeignet.

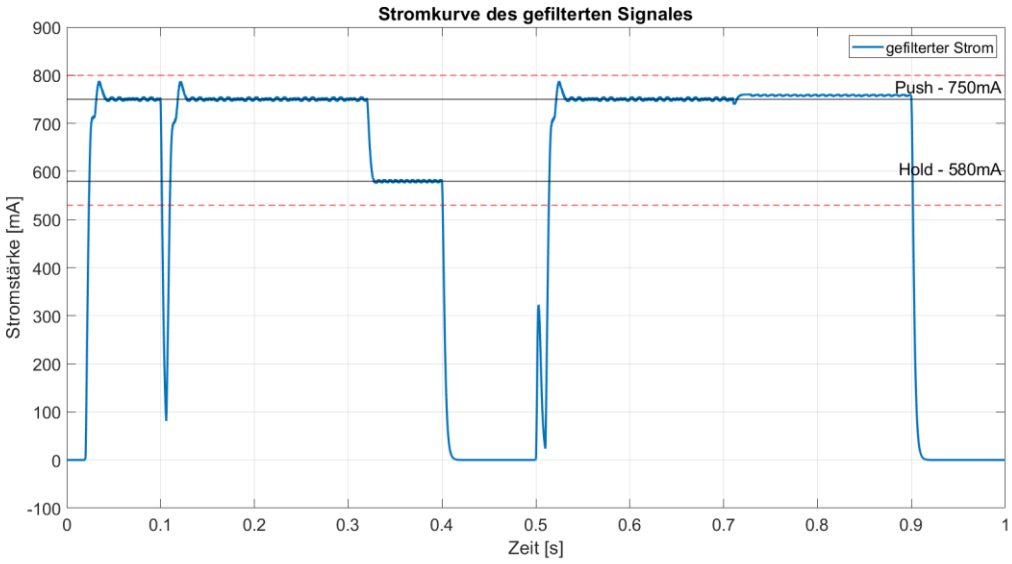


Abbildung 42: Regelung des berechneten PI-Reglers in verschiedenen Szenarien

Im folgenden Schritt wird der PD-Regler in der Anwendung analysiert. Während die Summenzeitkonstante die Werte für einen reinen PD-Regler ermittelt, wird in der Simulation weiterhin für den D-Anteil ein DT1-Glied mit dem Filterkoeffizienten N_{PT1} verwendet. Dieser Wert wird zunächst auf 1 gesetzt. Wie beim reinen P-Regler zeigt sich in Abbildung 43 ein zu niedriges Einregeln mit einem leichten Überschwinger. Durch die Anpassung des Proportional- und Differenzierbeiwerts kann zwar das Überschwingen verstärkt oder verringert werden, doch der erwünschte *Push-Strom* kann nicht erreicht werden. Daraus kann geschlussfolgert werden, dass für Stromregelung ein I-Anteil zur Gewährleistung der stationären Genauigkeit notwendig ist.

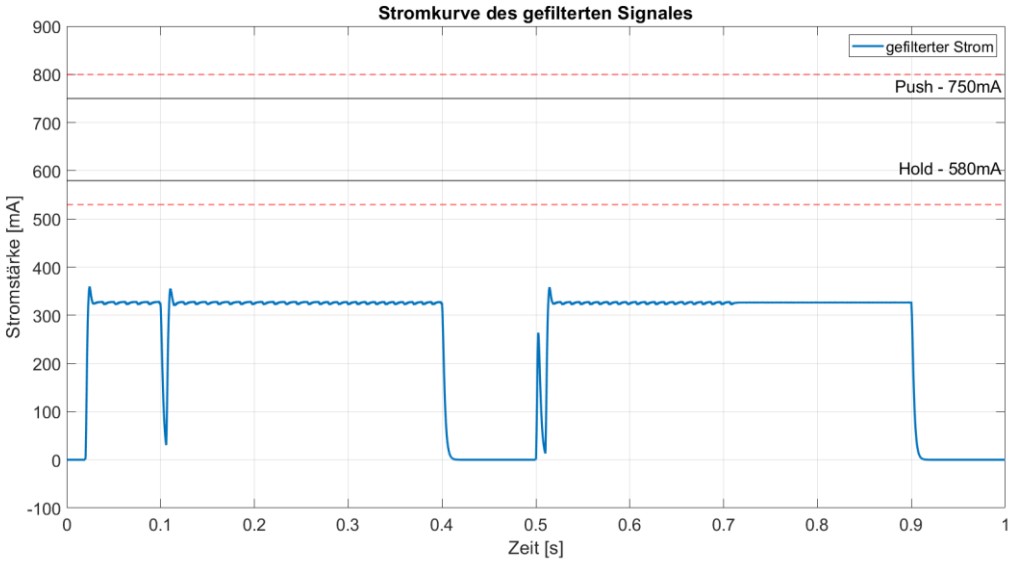


Abbildung 43: Regelung des berechneten PD-Reglers in verschiedenen Szenarien

Final wird das Verhalten des PID-Reglers in der Simulation analysiert. Auch hier wird der Filterkoeffizient des DT1-Gliedes zunächst auf 1 gesetzt. Der Regler weist ein sehr schnelles Regelverhalten auf.

Allerdings ist dadurch sowohl bei der Einregelung auf den *Push-Strom* ein Über- und Unterschwingen und bei der Einregelung auf den *Hold-Strom* ein Unterschwingen festzustellen. Durch das stärkere Einschwingverhalten greift außerdem die Fehlerregelung des Zustandsdiagrammes ein, weshalb der Beharrungswert leicht über dem *Push-Strom* liegt.

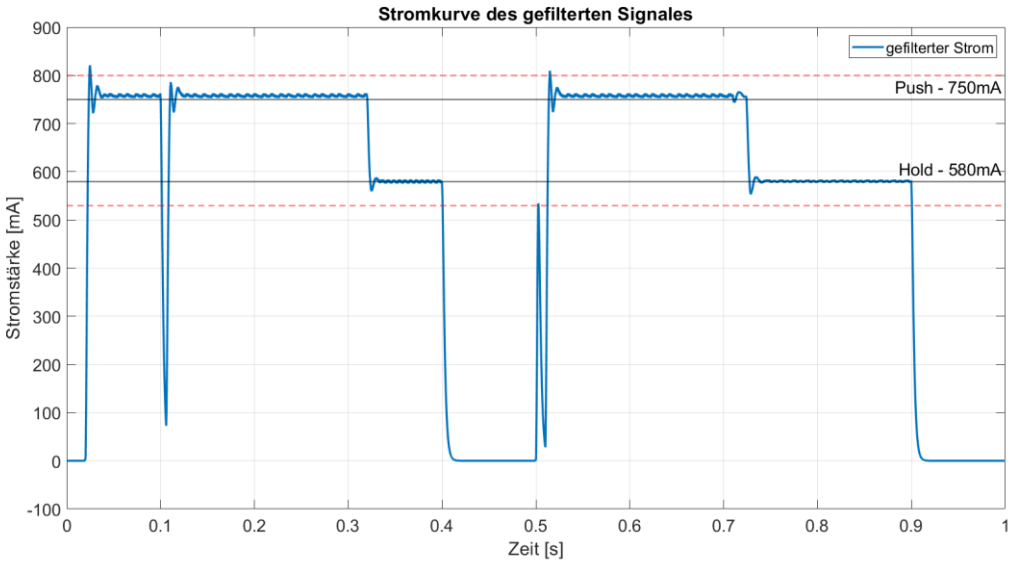


Abbildung 44: Regelung des berechneten PID-Reglers in verschiedenen Szenarien

Während die Einregelzeit im Rahmen der Anforderungen liegt, ist zu erkennen, dass der Überschwinger beim *Push-Strom* über den Grenzwert hinaus geht. Folglich kann dieser Regler ohne Anpassung nicht verwendet werden. Durch die Änderungen des Proportional- und des Differenzierbeiwerts kann zwar das Überschwingen angepasst werden, jedoch sind diese Werte sehr empfindlich und machen eine feine Anpassung daher schwierig. Durch die iterative Erhöhung des Filterkoeffizienten hingegen, kann das Überschwingen Schritt für Schritt reduziert werden. Erst ab einem Wert von 700 zeigt sich ein ähnlich gutes Regelverhalten zum PI-Regler. Allerdings ist durch die verschiedenen Szenarien zu erkennen, dass die Einregelung nicht konstant ist, und leichte Veränderungen bereits einen großen Einfluss besitzen. Daher wird im Folgenden weiterhin mit dem PI-Regler gearbeitet.

Für die finale Auslegung des PI-Reglers wird ebenfalls eine Anpassung an den Parametern vorgenommen. Das Regelverhalten liegt zwar bereits im Rahmen der Anforderungen, doch zur Absicherung soll der I-Anteil verringert werden, um so das Überschwingen weiter zu reduzieren. Dabei wird eine Reduzierung des Wertes der Überschwingweite gegen 0 angestrebt, ohne dabei die vorgegebene Einregelzeit zu überschreiten. In einem iterativen Prozess wird somit der Integrierbeiwert auf 150 reduziert. Der einzelne Schaltvorgang zeigt damit nur noch ein schwaches Überschwingen und ist somit für die simulierte Regelung des Ventiles ideal geeignet.

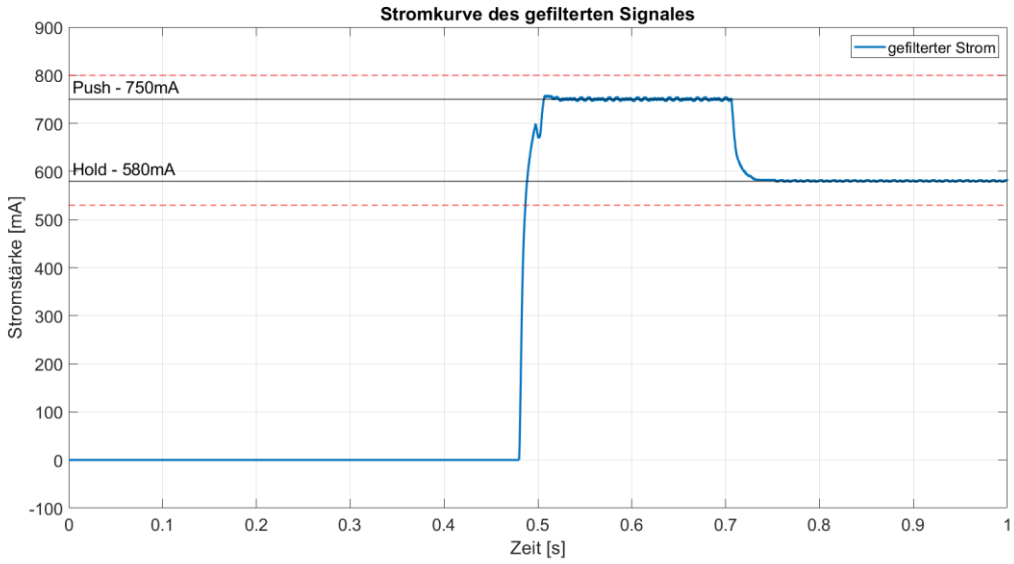


Abbildung 45: Stromkurve des angepassten PI-Reglers während eines Schaltvorganges

Abschließend wird die Stabilität des Reglers betrachtet. Hierzu wird das *Nyquist-Kriterium* verwendet. Zur Bestimmung des Kriteriums wird die Ortskurve des offenen Regelkreises benötigt. Beim offenen Regelkreis werden der Regler, die Regelstrecke und das Messglied als Reihenschaltung betrachtet. Während die Übertragungsfunktion G_S der Regelstrecke experimentell ermittelt werden musste, können die Übertragungsfunktionen des Reglers und des Messgliedes direkt definiert werden. Beim Regler werden die Übertragungsfunktionen der verschiedenen Reglerarten verwendet, welche bereits im Kapitel 2.4.1 erläutert wurden. Für den PI-Regler werden die Übertragungsfunktionen des P-Reglers und des I-Reglers addiert. Dabei muss die nachträgliche Umrechnung in die finale Steuergröße berücksichtigt werden. Der Gesamtfaktor wird als zusätzliche Verstärkung K_V definiert. Der Regler besitzt somit folgende Übertragungsfunktion G_R:

$$G_R(s) = \frac{100 * R}{U} * \left(K_P + \frac{K_I}{s}\right) = K_V * \left(K_P + \frac{K_I}{s}\right) \tag{46}$$

Formel 46: Übertragungsfunktion des PI-Reglers im Bildbereich

Da in der Rückführung des gemessenen Stromes keine Veränderung des Signales stattfindet, liegt kein Messglied vor und muss somit in den folgenden Rechnungen auch nicht berücksichtigt werden. Die Übertragungsfunktion G_0 des offenen Regelkreises errechnet sich daher wie folgt:

$$G_0(s) = G_R(s) * G_s(s)$$

$$G_0(s) = \frac{100 * R}{U} * \left(K_P + \frac{K_I}{s}\right) * \frac{K}{T * s + 1} \tag{47}$$

Formel 47: Übertragungsfunktion des offenen Regelkreises bei einem PI-Regler mit einer PT1-Regelstrecke

Zur Bestimmung der Ortskurve werden der Phasengang und der Amplitudengang der Übertragungsfunktion benötigt. Zur schnelleren Ermittlung können die Phasen- und Amplitudengänge der einzelnen

Übertragungsfunktionen ermittelt werden. Für die Regelstrecke als PT1-Glied sind diese ebenfalls im Kapitel 2.4.1.4 angegeben. Für den PI-Regler gelten folgende Formeln:

$$A_R(\omega) = K_V * \sqrt{K_P^2 + \frac{K_I^2}{\omega^2}} \tag{48}$$

Formel 48: Amplitudengang des PI-Reglers

$$\varphi_R(\omega) = \arctan\left(\frac{-K_I}{\omega * K_P}\right) \tag{49}$$

Formel 49: Phasengang des PI-Reglers

Während sich der Phasengang des offenen Regelkreises aus der Summe der einzelnen Regelkreise erschließt, wird der Amplitudengang des offenen Regelkreises über das Produkt der einzelnen Amplitudengänge berechnet. Somit ergibt sich der folgende Frequenzgang für den offenen Regelkreis.

$$A_0(\omega) = A_R(\omega) * A_S(\omega) = \frac{K * K_V}{1 + T^2 * \omega^2} * \sqrt{K_P^2 + K_P^2 * T^2 * \omega^2 + \frac{K_I^2}{\omega^2} + K_I^2 * T^2} \tag{50}$$

Formel 50: Amplitudengang des offenen Regelkreises

$$\varphi_0(\omega) = \varphi_R(\omega) + \varphi_S(\omega) = \arctan\left(\frac{-K_I}{\omega * K_P}\right) + \arctan(-T * \omega) \tag{51}$$

Formel 51: Phasengang des offenen Regelkreises

Aus dem resultierenden Phasengang und Amplitudengang des offenen Regelkreises kann zu jeder Frequenz ein Punkt auf der Ortskurve ermittelt werden. Dabei stellt die Amplitude die Länge des Ortsvektors und die Phase den Winkel zur reellen Achse dar. In der folgenden Abbildung werden das Bode-Diagramm und die Ortskurve des offenen Regelkreises dargestellt. Durch den I-Anteil im Regler beginnt die Ortskurve im negativen Unendlichen und endet im Nullpunkt. Im Diagramm ist zwar eine leichte Verschiebung auf der reellen Achse in den negativen Bereich zu sehen, diese entsteht jedoch bei Betrachtung des Phasengangs durch den leichten Einfall der Phase auf etwa 96°. Dies ist dadurch zu erklären, dass das PT1-Glied bei 0° beginnt und bei der Verzugszeit auf -90° absinkt, während der PI-Regler bei -90° beginnt und anschließend bei der Nachstellzeit T_n auf 0° steigt. Durch die zeitnahe Überlagerung ergibt sich somit nur ein kleiner Abfall der Phase. Mithilfe der Ortskurve kann über das *Nyquist-Kriterium* gesagt werden, dass der Regelkreis stabil ist.

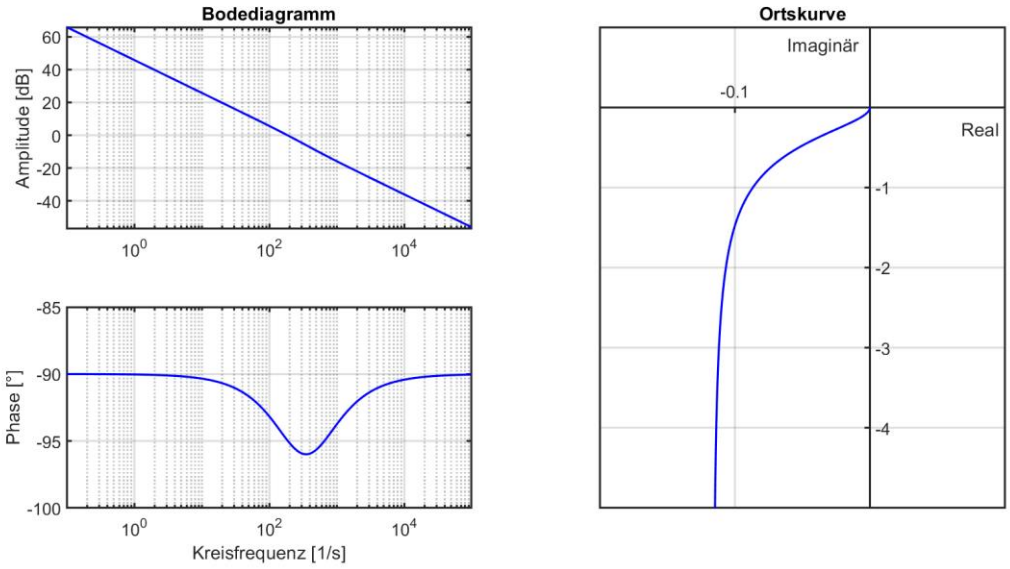


Abbildung 46: Bodediagramm (links) und Ortskurve (rechts) des offenen Regelkreises

4 Integration des Reglers in den C-Code

Im folgenden Kapitel wird das *Simulink*-Modell des Reglers erneut separat von der Simulation aufgebaut und zur Kompilierung in einen C-Code überarbeitet. Hierzu wird das Programm *TargetLink* von *DSpace* verwendet, welches eine eigene Block-Bibliothek mit einer zugeordneten Datenbank in *Simulink* integriert. Im ersten Schritt wird der reine Regler überarbeitet und kompiliert. Daraufhin wird zur Validierung der erzeugte C-Code wieder in die Simulation in *Simulink* integriert und das Verhalten mit der vorherigen Simulation verglichen. Anschließend wird die Software-Architektur überarbeitet, um eine Integration des Reglers zu ermöglichen.

4.1 Anpassung des *Model-Based Design*

Im ersten Schritt wird das *Library*-Modell des Reglers in ein separates Modell kopiert und die Verbindung zur *Library* getrennt, um so keine Veränderung an der gesamten Simulation vorzunehmen. Anschließend muss das neue Modell in ein reines Modell von *TargetLink* konvertiert werden. Hierzu besitzt *TargetLink* eine eigene Block-Bibliothek und eine dazugehörige Datenbank in *Simulink*. Es werden weitere relevante Funktionen durch *TargetLink* in *Simulink* integriert, welche im Laufe der Code-Entwicklung erläutert werden.

Die Blöcke von *TargetLink* entsprechen auf funktionaler Ebene den Blöcken von *Simulink*, besitzen allerdings eine detaillierte Eingabemaske zur Definierung von Variablen, Datentypen und weiteren Informationen, die zur Code-Generierung benötigt werden. Zur schnellen Konvertierung des Modelles besitzt *TargetLink* eine Funktion, mit der die Blöcke des *Simulink* Modelles automatisch durch entsprechende Blöcke der *TargetLink*-Bibliothek ersetzt werden. Die Variablen, welche vorher durch das Skript in *Matlab* ausgelesen wurden, müssen nun in die neu erstelle Datenbank von *TargetLink* implementiert werden. In der Datenbank reichen jedoch nicht nur der Name und der Wert der Variable, es wird ebenfalls der Datentyp und die Klasse benötigt. In *Simulink* wird, wenn keine konkrete Bedingung für den Datentypen vorliegt, standardmäßig der Datentyp *Double* definiert. *Double*, oder auch als *float64* bezeichnet, beinhaltet Gleitkommavariablen mit einer doppelten Genauigkeit, wodurch Werte in den Bereichen von -1,798E+308 bis -4,941E-324 für negative Werte und von 4,941E-324 bis 1,798E+308 für positive Werte möglich sind. Die 64 bezieht sich dabei auf die Datenmenge der Variablen. Jede Variable wird mit 64 Bits, bzw. 8 Bytes, abgespeichert. Durch die Klasse der Variable werden Definitionen, wie Speicherung und Verarbeitung der Variablen definiert. Da die meisten Variablen während der Regelung flexible Werte benötigen, wird allgemein die Klasse „Mergeable_Global“ definiert. In dieser Klasse kann global auf alle Variablen zugegriffen und eine Anpassung während der Regelung vorgenommen werden. Dabei handelt es sich um eine vordefinierte Klasse von *TargetLink*. Neben standardmäßigen Vorlagen gibt es in *TargetLink* außerdem die Möglichkeit direkte Vorlagen für den AUTOSAR-Stan-

dard zu verwenden. Diese Vorlagen werden für den späteren Aufbau der Softwarekomponente eingesetzt. Zur Verwendung der Variablen aus der Datenbank sind entweder in der Eingabemaske der Blöcke bereits konkrete Felder vorgesehen oder es wird der Befehl *ddv()* verwendet. Dabei muss der Pfad samt dem Namen der Variable in der Klammer eingegeben werden. Durch die Fertigstellung der Datenbank und die neue Definition der Blöcke kann bereits ein Großteil des Reglers konvertiert werden.

Im nächsten Schritt muss das *Stateflow*-Diagramm überarbeitet werden. Hierzu muss die Programmiersprache angepasst werden. In den Eigenschaften des *Charts* kann die Sprache unter *Action Language* von *Matlab* auf C geändert werden. In C müssen jedoch die Zeitfunktionen angepasst werden. Während in *Matlab* die Bezeichnung *sec* für Sekunde verwendet werden kann, muss der Zeitwert über C anders definiert werden. Hierzu wird der Befehl *after(Wert, Tick)* verwendet. Tick steht dabei für einen Rechenschritt der Simulation. Als Wert muss folglich keine Zeit, sondern eine Anzahl an Rechenschritten eingetragen werden. Durch die Vorgabe der Abtastfrequenz kann der vorher definierte Zeitwert mit der Frequenz multipliziert werden, um so die Anzahl der Rechenschritte in diesem Zeitraum zu erhalten. Abschließend müssen die Variablen innerhalb des *Stateflow*-Diagrammes neu definiert werden. In der Simulation wird für alle extern definierten Parameter der Typ *Parameter Data* verwendet. Dadurch wird direkt auf Parameter im *Matlab Workspace* zugegriffen. Im neuen Modell muss jedoch ein Zugriff auf die Datenbank erfolgen. Hierzu werden neue Variablen als *Local Data* angelegt und der Wert wird über die Variablen der Datenbank definiert. Für den Zugriff auf die Datenbank wird erneut der Befehl *ddv()* verwendet.

4.2 Kompilierung mit Validierung

Zur Kompilierung des Codes muss ein Rahmen definiert werden. Hierzu wird der gesamte Regler in einem *TargetLink* Subsystem zusammengefasst. Für die Testphase werden neben dem eigentlichen Ausgang des Reglers ebenfalls Ausgänge für den resultierenden Sollwert sowie den einzeln errechneten Anteilen des PI-Reglers angelegt. Über den *Main Dialog* von *TargetLink* kann final der Code generiert werden. Es wird dabei automatisch eine geeignete Ordnerstruktur angelegt und der Gesamtcode in verschiedene *Headerfiles* aufgeteilt. Die Hauptproblematik bei der Code-Generierung ist zum einen die korrekte vollständige Definierung aller Variablen sowie der Verlauf der Datentypen. Hierbei darf es keine Schnittstelle mit unterschiedlichen Datentypen geben. Es muss dazu im Voraus eine Konvertierung des Datentyps stattfinden. Zur Kontrolle können die Datentypen direkt an den Signallinien angezeigt werden. Dabei wird im späteren Verlauf außerdem ersichtlich, dass bei Verwendung von nicht geeigneten Datentypen, diese automatisch von *TargetLink* angepasst werden.

Zur Validierung des ersten fehlerfrei generierten Codes muss dieser wieder in die ursprüngliche Simulation integriert werden. Hierzu wird der bisherige Regler durch einen *C Caller*-Block ausgetauscht. Über diesen Block kann eine konkrete Funktion eines C-Codes aufgerufen werden. Die Integration des

C-Codes findet über die Konfigurationen des *Simulink* Modells statt. Hier können unter dem Abschnitt *Simulation Target* sowohl der *Source File* als auch die dazugehörigen *Header Files* definiert werden. Bei der Integration zeigt sich jedoch der erste Fehler in der Code-Generierung. Für die Software an sich reicht die aktuelle Definition der Variablen, zur konkreten Validierung einer C-Funktion in *Simulink* müssen jedoch die Ein- und Ausgänge dieser Funktion klar definiert werden. Dies geschieht über die Definition der Variablenklasse. Für die Eingänge der Reglerfunktion werden die Spannung, die Führungsgröße und die Rückführgröße als *Function Arguments* definiert. Dabei handelt es sich um nicht konstante Variablen, welche über einen externen Wert aufgerufen werden. Die resultierende Steuergröße sowie die Analyseausgänge werden als *Function Reference Arguments* definiert. Diese Variablen werden über eine Referenz aufgerufen. Bei erneuter Generierung des Codes werden nun im *C Caller* die Ein- und Ausgänge angezeigt. Im Ergebnis der Simulation zeigt sich beim ersten Durchlauf ein identisches Verhalten zur ersten Simulation. Dieses Verhalten kann allerdings bei einem neuen Durchlauf nicht reproduziert werden. Während bei der vorherigen Simulation alle Variablen über das *Matlab*-Skript wieder auf die Ausgangswerte gesetzt werden, scheint der C-Code nicht wieder auf die Anfangsdefinition zurückzuspringen. Stattdessen werden die letzten Werte verwendet. Eine Integration einer neuen Initialisierung der Variablen im Code kann die Problematik in der Simulation lösen, wird allerdings für die reale Anwendung nicht benötigt. Um alternativ ein reproduzierbares Ergebnis in der Simulation zu schaffen, muss die Führungsgröße vor der Beendigung wieder auf null gesetzt werden, damit die Variablen des Reglers ebenfalls zurückgesetzt werden. Mit der ansonsten erfolgreichen Validierung des C-Codes kann im folgenden Schritt die Integration des Reglers in die Gesamtsoftware beginnen.

4.3 Integration in den Software-Code

Zur Integration des Reglers in die Software wird zunächst die Applikationssoftware betrachtet. Dazu wird eine Kopie einer aktuellen Software im *DaVinci Developer* angelegt. In dieser Software wird, wie in Kapitel 2.6 beschrieben, die Anfrage von der *QM* über die *AsilA* an den *SIO* geschickt, welcher wiederum einen resultierenden Sollwert an den *PwmICtrlr* schickt. Eine Möglichkeit zur Integration des neuen Reglers wäre der Aufbau einer eigenen Softwarekomponente. In dieser Komponente könnte die Funktion des Reglers als *Runnable* eingebaut werden, allerdings benötigt jede Komponente weitere grundlegende *Runnables* zur funktionalen Verwendung in der Software. Da dies ein deutlich höheres Know-How in der Software-Architektur verlangt, wird eine Alternative verwendet.

Statt einer eigenen Softwarekomponente kann die Regelung als *Runnable* in einer bereits existierenden Softwarekomponente laufen. Während die meisten Komponenten CDDs sind und somit direkt über den reinen C-Code aufgebaut werden, gibt es für die *QM* und die *AsilA* bereits ein fertiges Modell in *Simulink*. Eine Möglichkeit wäre somit der direkte Einsatz des Reglers als *Runnable* in der QM. Hier kann

die Ventilanfrage direkt abgefangen und als Führungsgröße für den Regler eingesetzt werden. Allerdings würden somit die Sicherheitsüberprüfungen aus der *AsilA* und dem *SIO* überbrückt werden, was zur späteren Demolierung der Hardware beim Testen führen kann. Als finale Lösung wird daher ein *Runnable* in der *AsilA* aufgebaut. Hierzu wird im ersten Schritt die Software-Architektur im *DaVinci Developer* angepasst.

4.3.1 Anpassung der Software-Architektur

Die *AsilA* benötigt neue Ein- und Ausgänge zur Regelung des Stromes. Wie in der Simulation werden Eingangs die Spannung, die Führungsgröße und der rückgemessene Strom benötigt. Für die Spannung existiert das Bus-Signal *PwrSplyU*. Für dieses Signal wird ein Empfänger-Port an der *AsilA* integriert. Alle Signale der Software sind in einem *Port Interface* gelistet und können direkt für die Ports ausgewählt werden. In den Port-Eigenschaften muss die Signalflussrichtung, folglich in diesem Fall Empfänger, gewählt werden. Über die Kommunikationsspezifikationen können die einzelnen Signale des Busses betrachtet und bearbeitet werden. Das Bus-Signal *PwrSplyU* besteht aus den drei einzelnen Signalen *Clmp30*, *Clmp30RvsProtn* und *Clmp30Swt*. Auf die genaue Definition der Signale wird nicht weiter eingegangen, da für die Arbeit nur das Signal *Clmp30Swt* mit der für den Regler relevanten Spannung benötigt wird. Zur finalen Definition muss jedem Signal ein Initialwert zugeordnet werden. Hierzu gibt es ebenfalls in der Datenbank für alle Signale vorgefertigte Initialwerte. Damit ist der erste Port vollständig definiert. Im Weiteren wird nicht mehr auf die gesamte Definition der Ports eingegangen.

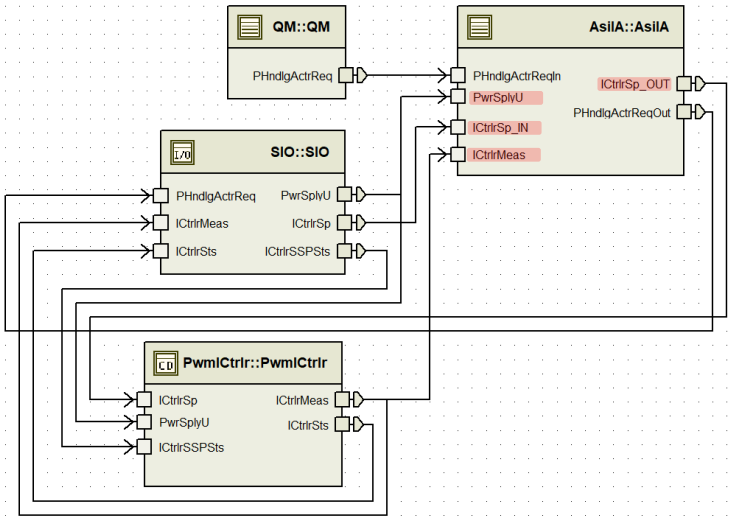


Abbildung 47: Reduzierter Aufbau der neuen Software zur Darstellung des Signalverlaufes, in Rot markiert die neuen Signale

Für die Führungsgröße muss zunächst ein geeignetes Signal gewählt werden. Die Anfrage des Auslassventiles beginnt bei der *QM*, wie in Abbildung 47 zu erkennen ist. Hierbei handelt es sich jedoch um die rohe Anfrage ohne vorherige Sicherheitsüberprüfungen, weshalb es nicht verwendet werden sollte. Das Signal wird sowohl von der *AsilA* als auch von der Softwarekomponente *SIO* überarbeitet. Für den

direkten Vergleich mit dem vorherigen Regler, wird das Signal *ICtrlrSp*, welches vom *SIO* an den *PwmICtrlr* weitergeleitet wird, verwendet. Das Signal stellt ein *Array* dar, welches die Sollströme für insgesamt acht Ventile beinhaltet. Das Signal wird über die *AsilA* umgeleitet und die überarbeiteten Sollwerte werden daraufhin an den *PwmICtrlr* weitergeleitet. Der *PwmICtrlr* besitzt zwei Modi. Zum einen kann ein Sollwert vorgegeben werden, welcher über die Softwarekomponente mit dem Istwert abgeglichen und geregelt wird. Zum anderen kann ein Prozentwert vorgegeben werden und der *PwmICtrlr* fungiert als reiner PWM-Generator, der den Prozentwert in ein entsprechendes PWM-Signal umwandelt. Da das Signal *ICtrlrSp* insgesamt die Sollwerte von acht Ventilen beinhaltet, müssen im künftigen *Simulink*-Modell sieben Sollwerte lediglich weitergeleitet werden, während der Sollwert des Auslassventiles über den eigenen Regler überarbeitet wird und als Prozentwert ausgegeben wird. Folglich muss der Modus beim *PwmICtrlr* nur für das Auslassventil angepasst werden. Im Developer selbst kann dies jedoch nicht geändert werden und wird später angepasst. Für den Architektur-Entwurf ist nur die Integration eines Ein- und Ausganges an der *AsilA* für das Signal *ICtrlrSp* relevant. Dabei beinhalten beide Signale die gleichen Spezifikationen, jedoch wird das Eingangssignal als *ICtrlrSp_IN* und das Ausgangssignal als *ICtrlrSp_OUT* bezeichnet.

Für die Rückführgröße wird das Signal *ICtrlrMeas* verwendet. Hierbei handelt es sich um den rückgemessenen Strom, welcher über die BSW an die Softwarekomponente *PwmICtrlr* und von dort auf Applikationsebene an die Software-Komponente *SIO* weitergeleitet wird. Diese Verbindung muss allerdings für die neue Regelung nicht getrennt werden, sondern es kann eine zuzügliche Verbindung zur *AsilA* geschaffen werden. Es zeigt sich hierbei die erste Problematik für die Regelung des Stromes. Während der Sollwert vom *SIO* empfangen wird, muss für den Empfang der Rückführgröße eine Verarbeitungsperiode im *PwmICtrlr* stattfinden, wodurch eine Totzeit entsteht. Da der Messstrom mit einer Abtastzeit von 2 ms aktualisiert wird, beträgt künftig auch die Totzeit im Messglied 2 ms. Dazu muss künftig eine Anpassung der Regelparameter vorgenommen werden.

Nachdem die Anpassung der Ein- und Ausgänge abgeschlossen ist, kann das neue *Runnable* der *AsilA* erzeugt werden. Hierzu kann im *DaVinci Developer* direkt ein neues *Runnable* mit eigenem Namen angelegt werden. Zur Spezifikation werden die Zugangspunkte und ein Trigger benötigt. Bei den Zugangspunkten handelt es sich um die benötigten Ein- und Ausgänge, folglich die zuvor neu definierten Ports der *AsilA*. Durch den Trigger wird definiert, wodurch das *Runnable* aufgerufen wird. In diesem Fall wird, wie in der Simulation, eine periodische Aktualisierungsrate der Steuergröße von 2 ms definiert.

Somit ist die Modifikation des grundlegenden Architektur-Entwurfes abgeschlossen. Der vollständige Entwurf mit zensierten Signalen ist im Anhang V vorzufinden. Final wird für die künftige Bearbeitung des *Simulink*-Modelles die *AsilA* als *ArXML* exportiert. Anschließend folgt die Überarbeitung der Softwarekonfiguration im *DaVinci Configurator*. Da der *Configurator* und der *Developer* auf die gleiche

Datei zugreifen, werden bereits die Änderungen der Software-Architektur im *Configurator* übernommen. Folglich müssen keine großen Überarbeitungen vorgenommen werden. Da jedoch ein neues *Runnable* mit einem neuen Trigger integriert wurde, müssen diese im *Configurator* miteinander verknüpft werden. Nach der Verknüpfung kann im Programm die neue RTE für die Kommunikation der neuen Software generiert werden. Damit sind alle Änderungen an der Software-Architektur abgeschlossen und der Stromregler kann in *Simulink* in das neue *Runnable* der *AsilA* integriert werden.

4.3.2 Aufbau des neuen *Runnables*

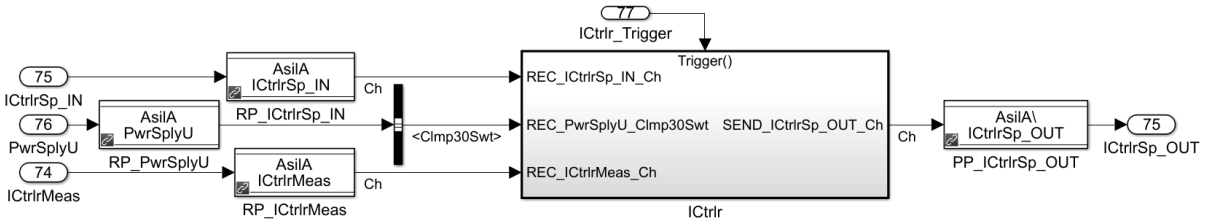


Abbildung 48: Äußerlicher Aufbau des *Runnables* in *Simulink*

Zur Überarbeitung der *AsilA* in *Simulink* wird zunächst das bereits vorhandene Modell der aktuellen Software vorwendet. Wie bereits zuvor definiert, existiert auf der obersten Ebene ein einzelnes *TargetLink* Subsystem mit allen benötigten Ein- und Ausgängen. In diesem Subsystem werden wiederum alle *Runnables* der *AsilA* als eigene Subsysteme dargestellt. Dabei erhält jedes Subsystem, die als Zugangspunkte vordefinierten Ein- und Ausgänge sowie einen Trigger. Zur Definition von Empfänger- und Sender-Ports, besitzt *TargetLink* eigene Blöcke, in denen die zugehörige Softwarekomponente und der aus der Datenbank zugehörige Port definiert werden. In einem weiteren *Simulink*-Modell wird nun die neue *ArXML*-Datei der überarbeiteten *AsilA* integriert. Diese kann in der Datenbank von *TargetLink* importiert werden. Über den *Frame-Generator* kann anschließend ein Block-Modell in *Simulink* automatisch generiert werden. Allerdings beinhaltet die Datenbank nur die Informationen zur Architektur und nicht zu den konkreten Funktionen innerhalb der *Runnables*. Daher wird das Subsystem des neuen Stromreglers im ursprünglichen *Simulink*-Modell integriert und die alte Datenbank der neu generierten Datenbank angeglichen. Somit sind bereits alle standardmäßigen *Runnables* vollständig aufgebaut und lediglich der Stromregler muss in das neue *Runnable* integriert werden.

Zu Beginn müssen die Eingänge des *Runnables* innerhalb des Subsystems angepasst werden. Für die Führungsgröße wird das Signal *ICtrlrSp_IN* verwendet. Da es sich hierbei um ein *Array* aus acht Sollströmen handelt, muss zunächst über einen *Demux* das Signal aufgespalten und nur der Sollstrom für das Auslassventil abgegriffen werden. Die restlichen Signale werden im Nachhinein wieder mit dem neu errechneten Sollwert über einen *Mux* in ein gemeinsames *Array* zusammengeführt. Über den aktuellen Software-Code der *AsilA* kann ermittelt werden, dass der Wert für das Auslassventil an der zweiten Stelle im *Array* liegt. Diese Positionierung gilt auch für den später verwendeten Messstrom. Da der neue

Stromregler eine integrierte Ermittlung des Sollstromes besitzt und nur die Ventilanfrage benötigt, wird das Eingangssignal über einen *Data Type Conversion*-Block in ein Signal mit dem Typ *Boolean* konvertiert, wodurch der Wert 1 ist, wenn der Sollstrom größer 0 ist, und 0 wenn der Sollstrom ebenfalls 0 ist.

Der Datentyp des Spannungssignals *Clmp30Swt* aus dem Bus *PwrSplyU* muss ebenfalls konvertiert werden, da dieser in der Software als *uint16* und im Stromregler als *Double* definiert ist. Die gleiche Problematik gilt für die Rückführgröße aus dem Signal *ICtrlrMeas*. Hierbei muss jedoch im Voraus wie bei der Führungsgröße das Signal über einen *Demux* aufgeteilt werden, um den rückgemessenen Strom des Auslassventiles zu erhalten. Die restlichen Rückmessungen werden nicht weiter benötigt.

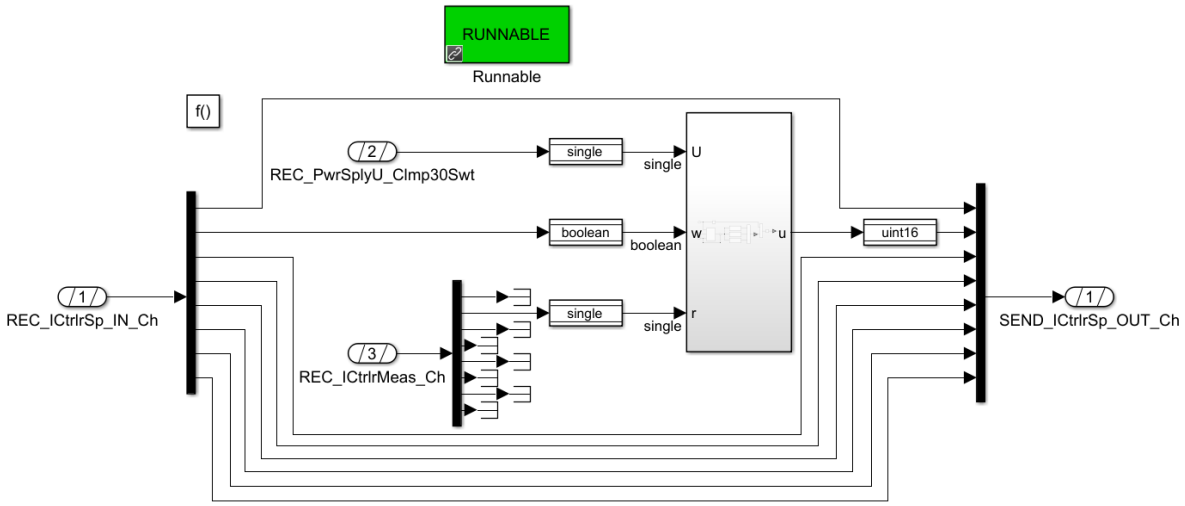


Abbildung 49: Anpassung der Ein- und Ausgänge für die Integrierung des Stromreglers in *Simulink*

Somit sind alle notwendigen Eingangssignale vorhanden. Der Stromregler wird als eigenes Subsystem in das *Runnable* integriert und mit den überarbeiteten Eingangssignalen verbunden. Die im Regler neu definierten Parameter müssen in der Datenbank der *AsilA* integriert werden. Hierzu kann die Datenbank der *AsilA* mit der Datenbank des Reglers verglichen und fehlende Variablen direkt übertragen werden. Final muss das Ausgangssignal des Reglers ebenfalls angepasst werden. Auch das Sollstromsignal *ICtrlrSp_OUT* besitzt den Datentypen *uint16*, während die Steuergröße des Reglers als *Double* definiert ist. Somit wird hier erneut der Datentyp konvertiert. Anschließend kann die Steuergröße mit den Sollwerten für die anderen Ventile über einen *Mux* wieder in einem *Array* verknüpft und an den Sender-Port weitergeleitet werden. Beim Verknüpfen ist es wichtig, auf die Reihenfolge der Signale zu achten, da in der Software definiert ist, an welcher Stelle des *Arrays* sich welcher Sollwert befindet.

Somit ist das *Simulink*-Modell der neuen *AsilA* vollständig aufgebaut und kann über den *TargetLink Main Dialog* in einen Software-Code kompiliert werden. Zur finalen Erstellung der Software muss eine Anpassung vorgenommen werden. Wie zuvor erwähnt, muss der Modus des *PwmICtrlr* konkret für das Auslassventil angepasst werden. Diese Änderung kann nur direkt im Code stattfinden. Die Einstellung kann in der sogenannten *Post-build time configuration* des CDDs geändert werden. In diesem Code sind

die Definitionen für die einzelnen Ventile und somit auch für das Auslassventil vorzufinden. Wenn alle Codes vollständig überarbeitet sind, können diese in einer vorgegebenen Ordnerstruktur aufgebaut werden, um daraufhin ein von der Abteilung erstelltes Skript zur Verknüpfung aller einzelnen Codes und Erstellung der Software durchzuführen. Bei dem ersten Versuch zeigt sich jedoch eine Problematik. Die Software kann zwar generiert werden, allerdings treten Warnungen auf, die bei der sonstigen Generierung nicht vorhanden sind. Durch eine nachträgliche Recherche ergibt sich, dass die Steuereinheit, welche für die folgende Validierung verwendet werden soll, mit maximal 32 Bit rechnen kann. Folglich ist der in der Regelung verwendete Datentyp *Double* mit 64 Bit zu groß für die Verarbeitung auf der Steuereinheit. Zunächst wird eine Anpassung der Datentypen an die Ein- und Ausgangssignale aus der Software vorgenommen. Hierbei zeigen sich jedoch die früher erwähnten Komplikationen mit abweichenden Datentypen. Zum einen werden Fehlermeldungen bei der Code-Generierung durch die Verwendung von verschiedenen Datentypen an einem Block ausgegeben und zum anderen zeigt sich, dass *TargetLink* eine automatische Anpassung der Datentypen in einigen Rechenschritten vornimmt. Während bei einigen Rechenschritten eine Rundung der Werte zur Anpassung an den vorgegebenen Datentyp erwartet wird, versucht *TargetLink* eigene Datentypen zu integrieren, um so das Ergebnis möglichst exakt darstellen zu können. Um diese Probleme zu vermeiden, wird ein allgemeiner Datentyp für alle Variablen innerhalb des Reglers verwendet. Da zur Berechnung der Regleranteile Gleitkommazahlen benötigt werden, wird das Maximum der Steuereinheit ausgenutzt und für alle Variablen der Datentyp *Single* mit 32 Bit verwendet. Durch diese Änderungen werden die Warnungen bei der Softwaregenerierung behoben. Im Folgenden müssen dennoch einige nachträgliche Änderungen an dem Regler vorgenommen werden, bevor die eigentliche Validierung am Prüfstand beginnen kann.

4.3.3 Anpassung der Regelparameter

Bei der Überarbeitung der Software-Architektur und dem Aufbau des neuen *Runnables* hat sich eine Problematik gezeigt, welche eine hohe Auswirkung auf den Regler besitzt. Durch die Ermittlung der Rückführgröße über den *PwmICtrlr* ergibt sich eine Totzeit T_t von 2 ms. Dieses Totzeitverhalten muss in der Simulation in einem Messglied berücksichtigt werden. Zur Simulation der Totzeit wird in der Rückführung ein *Unit Delay* mit einer *Sample Time* von 2 ms integriert. Da der aktuelle Regler einen hohen Integrierbeiwert aufweist, zeigt sich durch die Totzeit ein starkes überschwingendes Regelverhalten in der Simulation. Die Anpassung der Parameter findet sowohl über das entwickelte Programm zur Summenzeitkonstante als auch einer nachträglichen iterativen Optimierung statt. Die Methode der Summenzeitkonstante wird erneut mithilfe der Sprungantwort eingesetzt. Um das Totzeitglied zu berücksichtigen, kann bei dieser Methode die Totzeit als Verzögerung des Einheitssprunges einbezogen werden. Dadurch wird die Verstärkung K_s nicht verändert, doch die Summenzeitkonstante wird um den Wert der Totzeit erhöht. (Lunze, 2020, S. 284) Bei der neuen Ermittlung der Regelparameter ergibt sich für den I-Anteil somit ein Wert von 155,3351. Mit diesen Parametern zeigt sich in der überarbeiteten

Simulation, wie bereits bei der ersten Dimensionierung, ein leichtes Überschwingen mit anschließender Einregelung. Um erneut das Überschwingen zu reduzieren, wird der Integrierbeiwert in mehreren Schritten verkleinert, bis kein Überschwinger oder nur ein schwaches Überschwingen vorzuweisen ist. Der Proportionalbeiwert wird dabei konstant gehalten. Es ergibt sich somit ein reduzierter Wert von 100 für den Integrierbeiwert. Abschließend muss erneut die Stabilität des Regelkreises überprüft werden. Hierzu muss die Übertragungsfunktion des Totzeitgliedes, welche bereits in Kapitel 2.4.1.5 erläutert wurde, bei der Übertragungsfunktion des offenen Regelkreises berücksichtigt werden. Während der Amplitudengang des Totzeitgliedes als Faktor 1 keinen Einfluss auf den gesamten Amplitudengang des offenen Regelkreises besitzt, muss der Phasengang mit dem zuvor ermittelten Phasengang summiert werden. Im Bodediagramm ist dadurch ein starkes Abfallen der Phase zu erkennen. Dieses kann durch die Ortskurve deutlich besser graphisch dargestellt werden. Diese besitzt zu Beginn einen ähnlichen Verlauf, endet allerdings in einer Spirale. Dabei liegt die erste Schnittstelle mit der reellen Achse bei etwa -0,2. Eine genauere Angabe ist dabei nicht nötig, da diese bereits ausreicht, um mithilfe des *Nyquist-Kriteriums* sagen zu können, dass die Stabilität für den überarbeiteten Regelkreis gegeben ist.

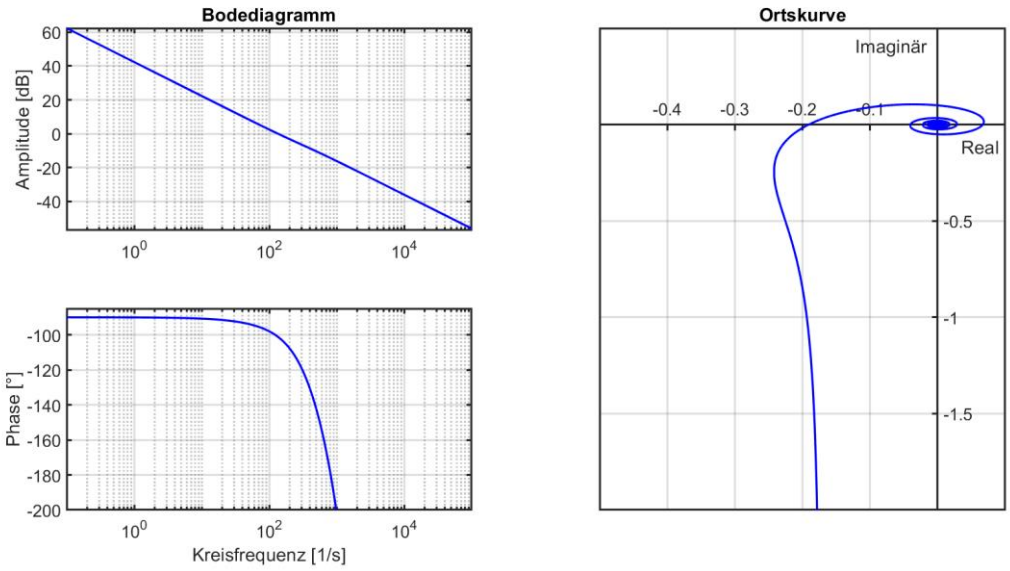


Abbildung 50: Bodediagramm (links) und Ortskurve (rechts) des offenen Regelkreises bei integrierter Totzeit

5 Validierung an der Hardware

Im folgenden Kapitel wird die Software am Prüfstand getestet und zum einen die allgemeine Funktion sowie das Verhalten des Reglers analysiert. Hierzu wird zu Beginn der Aufbau des Prüfstandes und die Übertragung der Software kurz erläutert. Daraufhin wird die Funktion des Reglers getestet und über eine Stromzange das konkrete Regelverhalten an der Stromkurve gemessen, analysiert und abgeglichen. Abschließend findet eine Bewertung der Regelung statt.

5.1 Aufbau des Prüfstandes

Zur Prüfung des Reglers muss zunächst die Software auf eine Steuereinheit übertragen werden. Hierbei gibt es verschiedene Modelle. Eine wichtige Eigenschaft für die ersten Tests ist die Status-LED. Einige Steuereinheiten besitzen eine LED, an welcher direkt erkannt werden kann, ob die Steuereinheit aufwacht und der Software-Code problemlos läuft. Dadurch kann zum einen ein direkter Fehler beim Überschreiben des Speichers mit der neuen Software und zum anderen ein Fehler beim Ausführen der Regelung festgestellt werden. Des Weiteren wird ein sogenannter *Debug-Port* an der Steuereinheit benötigt. Über diesen Anschluss kann sowohl die Software direkt auf die Steuereinheit geladen werden als auch die Änderung und Auswertung von Parametern vorgenommen werden.

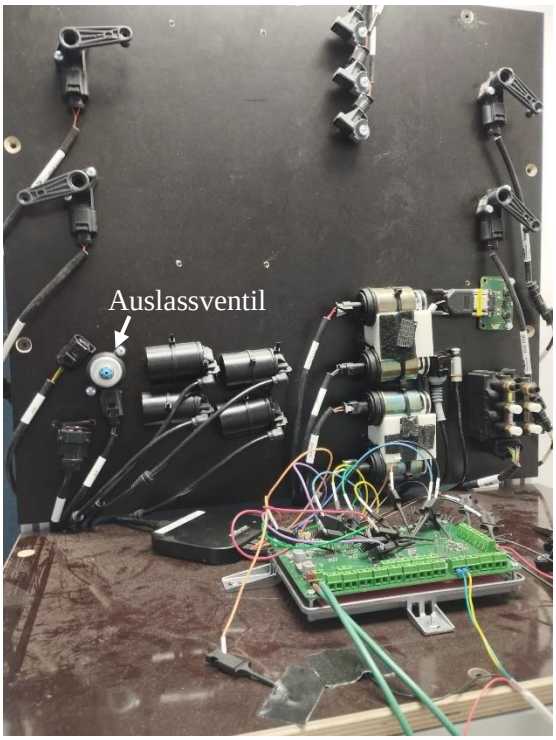


Abbildung 51: Aufbau des HIL-Prüfstandes mit den verschiedenen Sensoren

Für die Hardware wird die Steuereinheit an einem *HIL*-Prüfstand (*Hardware in the Loop)* angeschlossen. An diesem Prüfstand werden alle für das Luftfeder-System relevanten Sensoren und Aktuatoren aus dem Fahrzeug, wie in Abbildung 51 zu erkennen, an der Steuereinheit angeschlossen. Somit können die verschiedenen Funktionen getestet werden, ohne eine direkte Beschädigung am Fahrzeug zu riskieren. Während die Ventile für die Aktuatoren und für die Pumpe in einem Ventilblock verknüpft werden, wird das Auslassventil einzeln mit der Steuereinheit verbunden. Über die Spannungszufuhr des Ventiles wird eine Stromzange angeschlossen, wodurch mit einem Oszilloskop die Stromkurve während des Betriebs der Regelung nachgewiesen werden kann.

Zur Ansteuerung wird ein Computer über den *Debug-Port* mit der Steuereinheit verbunden. Über das Programm *winIDEA* kann zunächst der Software-Code auf die Steuereinheit geladen werden. Über das Programm kann außerdem Einsicht auf den gesamten Code und die verschiedenen Fehlermeldungen der Steuereinheit gewährt werden. Für die folgende Analyse des Reglers können außerdem *Breakpoints* im Code integriert und Anpassungen an den Parametern vorgenommen werden.

5.2 Analyse des Reglers

Der erste Test ergibt sich durch das Überschreiben des Speichers mit der neuen Software. Wenn kritische Fehler bei der Softwaregenerierung nicht erkannt wurden, zeigen sich diese durch das grundlegende Verhalten der Steuereinheit. Hierbei sind allerdings bei der neu entwickelten Software keine Fehler festzustellen. Die LED flackert grün. Dabei kann zum einen durch die Farbe gesagt werden, dass keine direkten Fehler beim Überschreiben des Speichers entstanden sind und zum anderen wird das Flackern durch die Rechenschritte innerhalb der Steuereinheit erzeugt. Würde sich die Software durch einen Fehler aufhängen, würde die LED konstant grün leuchten. Somit können bis zu diesem Zeitpunkt keine Fehler ermittelt werden. Die allgemeine Spannung im System wird zu Beginn wie in der Simulation auf 14 V eingestellt. Zu Beginn wird ein *Breakpoint* beim Aufruf der Reglerfunktion gesetzt, um zu überprüfen, ob die Software den korrekten Code aufruft. Hierbei können weiterhin keine Fehler festgestellt werden. Daher soll im nächsten Schritt das Ventil angesteuert werden.

Zur Validierung der Software gibt es bereits vorgesehene Parameter welche als *Bypass* fungieren und somit die direkte eigene Ansteuerung ermöglichen. Dabei werden die grundlegenden Anforderungen zur Schaltung des Ventils umgangen und folglich direkt der Wert des Auslassventiles im Signal *PHndlgActrReq* angepasst. Sowohl das *Bypass*-Signal als auch das eigentliche Steuersignal besitzen insgesamt 3 Zustände. Der erste Zustand ist mit dem Wert 0 definiert und aktiviert die automatische Schaltung des Ventiles. Der zweite Zustand ist mit dem Wert 1 definiert und steht für die Öffnung des Ventiles. Sobald dieser Zustand eintrifft, wird somit die Schaltphase aktiviert und die Sollströme angepasst. Der letzte Zustand ist mit dem Wert 2 definiert und sorgt für die direkte Schließung des Ventiles.

Beim ersten Versuch das Ventil zu öffnen, kann ein akustisches Klicken am Ventil wahrgenommen werden, was für die reine Funktion des Reglers spricht. Allerdings zeigt sich über die Auswertung der Signale, dass die Regelung während der Push-Phase abgebrochen wurde. Bei Betrachtung der Stromkurve über das Oszilloskop zeigt sich ein stark überschwingendes Verhalten bei der Regelung auf den *Push-Strom* mit einem darauffolgenden Absenken des Stromes auf 0 A. Das abweichende Verhalten ist auf zwei Problematiken zurückzuführen, welche in der Simulation nicht wahrgenommen werden konnten. Ein allgemeiner Unterschied zwischen Simulationen und Realität ist die Trägheit. Während in der Simulation Signale direkt verarbeitet und gerechnet werden können, treten bei der Hardware immer Trägheitseffekte auf. Kein physikalisches System kann in der Realität ohne Verzögerung und Störungen durch die Umgebung ablaufen. Daher ist auch zu erkennen, dass die Stromkurve im Vergleich zur Simulation langsamer ansteigt. Durch die Trägheit wird, wie bei der Totzeit der I-Anteil deutlich erhöht und fördert daher ein Überschwingen. Eine weitere Problematik stellt der bereits erwartete Einfall der Stromstärke beim Öffnen des Ventils dar. Zum einen ist daran zu erkennen, dass das Ventil bereits vor dem Erreichen des *Push-Stromes* beginnt zu schalten. Zum anderen ergibt sich ein deutlich ausgeprägterer Einsturz im Vergleich zur Simulation. Folglich muss die Änderung des Widerstandes in der Realität höher sein als in der Simulation angenommen. Die Werte des Oszilloskops können als *CSV*-Datei abgespeichert und somit in *Matlab* importiert werden. Daraufhin wird die Stromkurve erneut in einem Plot dargestellt und mit dem Ergebnis der Simulation überlagert. Über diesen Vergleich kann die Widerstandsänderung beim Öffnen des Ventils angepasst werden, bis der Stromabfall in der Simulation ein ähnliches Verhalten zur realen Messung aufweist. Dabei zeigt sich, dass der Stromabfall bei etwa 700 mA beginnt und ein Anstieg des Widerstandes um 4 Ω in der Simulation verwendet werden kann. In einer erneuten iterativen Anpassung ergibt sich mit der überarbeiteten Simulation ein Integrierbeiwert von 50. Mit diesem Wert erreicht die Stromstärke gerade so den *Push-Strom*, fällt daraufhin leicht ab und nähert sich wieder dem Sollwert an. Somit wird in der Simulation das Überschwingen besonders

stark reduziert, um unter Berücksichtigung der Trägheit auch das Überschwingen am realen System zu vermeiden.

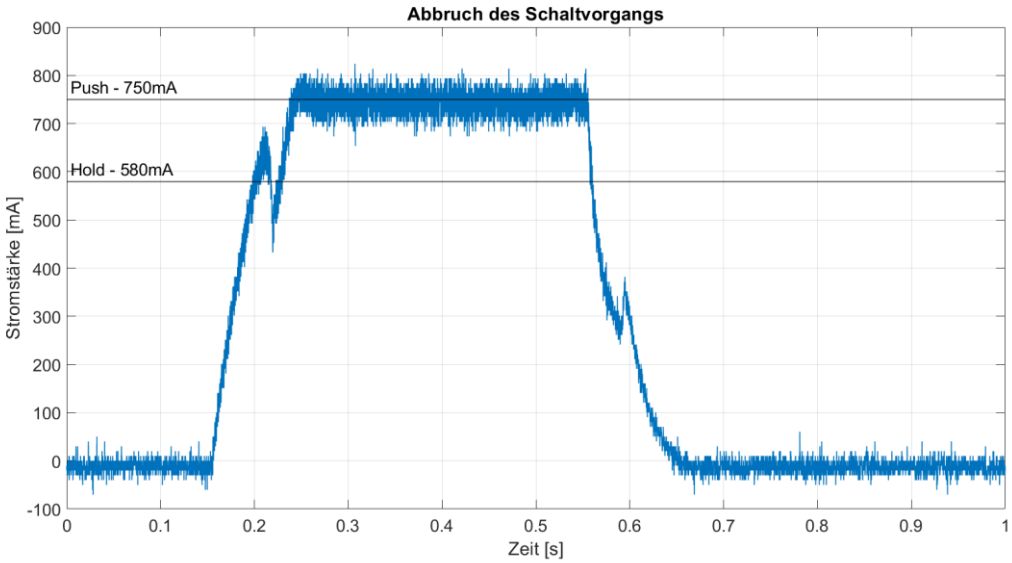


Abbildung 52: Abbruch des Schaltvorgangs während der Push-Phase

Anstelle einer neuen Softwaregenerierung kann dieser Wert vorerst über den *Debug-Port* direkt auf der Steuereinheit angepasst werden. Bei einem erneuten Versuch wird jedoch erneut der Schaltvorgang, wie in Abbildung 52 zu erkennen, während der *Push-Phase* abgebrochen. Am Oszilloskop zeigt sich dennoch das erwartete leichte Überschwingen und die darauffolgende Einregelung auf den *Push-Strom*. Folglich muss die Fehlermeldung auf Grund einer Sicherheitsabfrage innerhalb der Software aufgerufen werden. Eine Problematik, die bei vorherigen Entwicklungen aufgetreten ist, ist eine zu hohe Spannung. Daher wird in einem iterativen Versuch die Spannung schrittweise reduziert. Bei einer Spannung von 9,6 V kann zum ersten Mal das erwartete Regelverhalten ohne Fehlermeldung festgestellt werden.

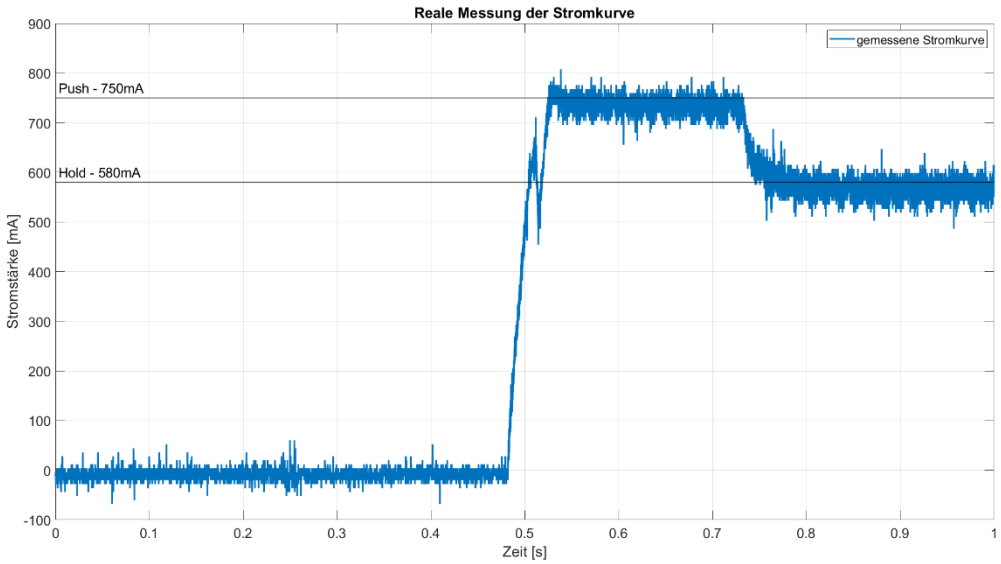


Abbildung 53: Reale Messung der Stromkurve über ein Oszilloskop bei einer Betriebsspannung von 9,6 V

Bei der Betrachtung der Abbildung 53 ist ein steiler Anstieg zu Beginn der Regelung zu erkennen. In Anbetracht der Ungenauigkeit durch das Rauschen kann die Öffnung des Ventiles und der damit einhergehende Abfall der Stromstärke ab etwa 0,51 s abgelesen werden. Nachdem der Strom auf etwa 450 mA abgesunken ist, steigt dieser anschließend wieder steil an und pendelt sich mit einem leichten Überschwinger auf dem geforderten *Push-Strom* ein. Nach dem im Zustandsdiagramm definierten Zeitintervall von 200 ms ist das Absinken auf den *Hold-Strom* zu erkennen. Die Messung wurde jedoch erneut bei einer Eingangsspannung von 9,6 V aufgenommen. Bei einer Erhöhung der Spannung zeigt sich trotz eines nur noch geringen Überschwingens ein Abbruch der Ventilschaltung durch eine Software-interne Fehlermeldung. Die Fehlermeldung kann durch bereits vordefinierte Sicherheitsabfragen ausgelöst werden, jedoch kann der genaue Grund nicht ermittelt werden. Daher wird für die weiteren Tests nur mit der erwähnten Spannung von 9,6 V gearbeitet. Um die Reproduktion des Ergebnisses nachzuweisen, werden vier weitere Aufnahmen über das Oszilloskop gemacht, in *Matlab* ausgewertet und in einem gemeinsamen Diagramm aufgetragen.

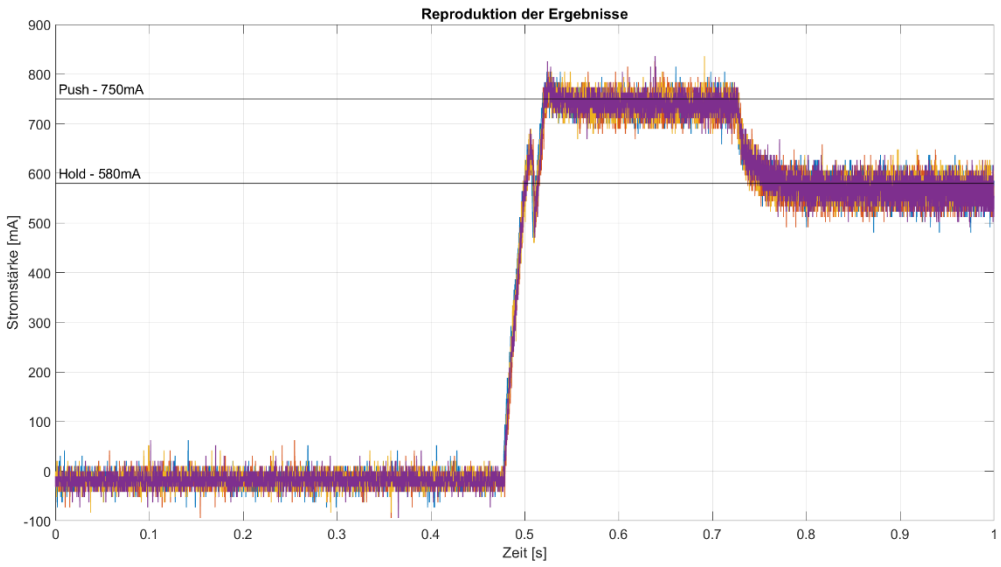


Abbildung 54: Nachweis der Reproduzierbarkeit mit vier überlagerten Stromkurven

An der Abbildung 54 ist zu erkennen, dass zwar durch das Rauschen geringe Abweichungen in den Peaks auftreten können, das grundlegende Verhalten jedoch bei allen Proben gleich ist. Der direkte Vergleich mit der Simulation und die damit einhergehende Bewertung des Reglers findet im folgenden Kapitel statt.

5.3 Bewertung der *Model-Based* Regelung

Für die Bewertung der Stromregelung aus dem MBD wird zunächst erneut auf die realen Ergebnisse in Abbildung 54 eingegangen und anhand der Messwerte aus dem Oszilloskop die Anforderungen aus

Kapitel 3.1 überprüft. Die ersten Anforderungen können aus dem Einregelverhalten auf den *Push-Strom* ermittelt werden. Die erste Bedingung befasst sich mit dem Überschwingverhalten des Reglers. Hierbei darf die Überschwingweite nicht mehr als 50 mA betragen. Beim rauschenden Signal sind vereinzelt Spitzen zu erkennen, welche diesen Wert überschreiten, allerdings muss berücksichtigt werden, dass diese Anforderung nur für das gefilterte Signal gilt. Folglich ist diese Anforderung erfüllt.

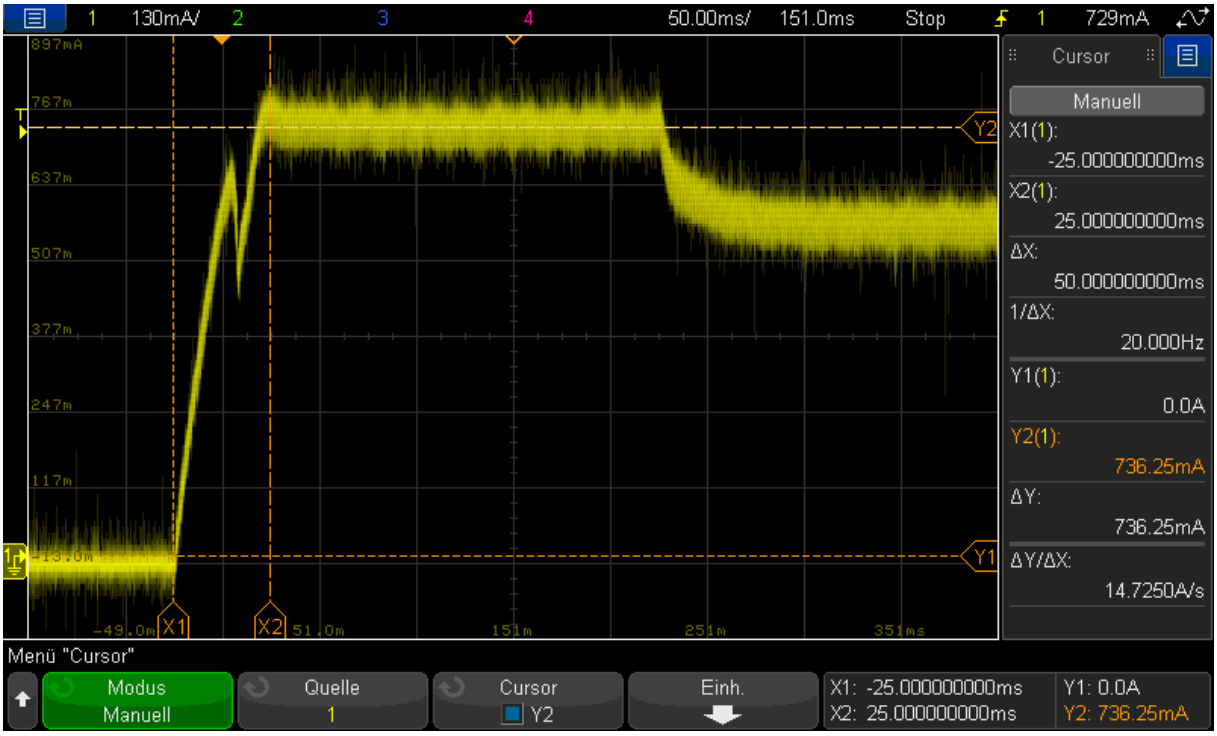


Abbildung 55: Auswertung der Stromkurve bei der Regelung auf den *Push-Strom* am Oszilloskop

Die zweite Bedingung ist die Einhaltung der maximalen Einregelzeit von 50 ms. Durch den kleinen Proportionalbeiwert in Verbindung mit der Trägheit des Systems kann diese Bedingung nicht erfüllt werden. Der *Push-Strom* ist zwar in diesem Zeitbereich erreicht und das Ventil damit geöffnet, was anhand der Abbildung 55 zu erkennen ist, allerdings kann durch das leichte Überschwingen nicht festgestellt werden, dass sich in dieser Zeitspanne die Stromstärke bereits eingeregelt hat. Da jedoch kein darauffolgender Unterschwinger vorhanden ist, kann ermittelt werden, dass die Einregelzeit zuzüglich der Haltezeit in der *Push-Phase* nicht das summierte Zeitintervall von 250 ms überschreitet. Durch das Zustandsdiagramm werden in der Haltezeit Störeinflüsse berücksichtigt und nur bei Einhaltung des *Push-Stromes* nach einer Dauer von 200 ms der Strom reduziert. Dieser Zeitbereich kann im Diagramm sehr gut abgelesen werden und zeigt, dass ab der Einregelung nach dem ersten Überschwinger, keine weitere Störung aufgetreten ist. Abschließend muss das Einregelverhalten auf den *Hold-Strom* betrachtet werden. Hierbei wird erneut die Anforderung für den Unterschwinger und die Einregelzeit überprüft. Aus dem Diagramm ist eine milde Absenkung der Stromstärke ohne Unterschwinger zu erkennen. Somit ist die erste Anforderung erfüllt. Allerdings ist das nicht vorhandene Unterschwingen auf die insgesamt langsame Einregelung zurückzuführen. Folglich wird die geforderte Einregelzeit von 50 ms nicht erfüllt.

Neben den Anforderungen ist der direkte Vergleich zwischen der Stromkurve der Simulation und der realen Messung relevant für die Bewertung. Eine Anpassung der Regelparameter kann jederzeit vorgenommen werden, allerdings muss ermittelt werden, ob die Simulation künftig hierzu verwendet werden kann. Hierzu wird die Führungsgröße in der Simulation bei 0,48 s auf 1 gesetzt, um so einen nahezu zeitgleichen Beginn der Ventilschaltung zu erreichen. Folglich können dadurch die beiden Stromkurven in einem Diagramm direkt miteinander verglichen werden.

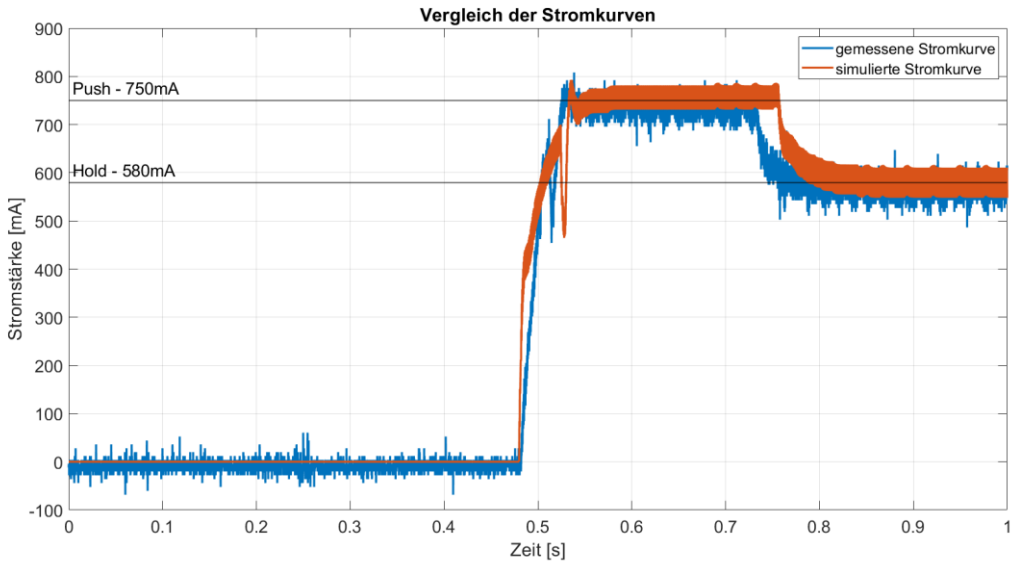


Abbildung 56: Vergleich der simulierten Stromkurve mit der am Oszilloskop gemessenen Stromkurve

Durch die Abbildung 56 werden die Problematiken, welche bei der ersten Ventilschaltung aufgetreten sind, erneut deutlich. Zu Beginn zeigt sich bei der realen Messung ein schwächerer Anstieg, der auf die Trägheit der Hardware zurückzuführen ist. Gleichzeitig zeigt sich dadurch aber auch ein geeigneterer Verlauf der Stromkurve. Während die reale Messung gleichmäßig bis zum Peak ansteigt, zeigt sich in der Simulation zunächst der sehr abrupte und steile Anstieg und ab einem Strom von 400 mA eine erste Verringerung der Steigung. Durch den zunächst steilen Anstieg und die darauffolgende Reduzierung der Steigung, zeigt sich eine langsamere Einregelung im Vergleich zur Realität. Durch die Anpassung der Widerstandsänderung beim Schalten des Ventiles, welche im vorherigen Kapitel vorgenommen wurde, zeigt sich ein sehr ähnliches Verhalten beim Stromabfall. Im Einschwingverhalten ist zu erkennen, dass die Simulation einen kürzeren Überschwinger mit einem darauffolgenden Unterschwinger besitzt. Hier ist somit ebenfalls das direktere Verhalten der Simulation zu erkennen, welche für eine abruptere Regelung sorgt. Während die Trägheit der Hardware ein Nachteil in der Einstellung der Regelparameter darstellt, zeigt sich ein allgemein glatteres Einregelverhalten. Während der Beharrung auf dem *Push-Strom* zeigt sich kein großer Unterschied zwischen den beiden Stromkurven. Durch den Unterschwinger in der Simulation jedoch wird erst ab einem späteren Zeitpunkt der *Push-Strom* gehalten, weshalb auch erst zu einem späteren Zeitpunkt der Sollstrom auf den *Hold-Strom* gesetzt wird. Bei der

Reduzierung der Stromstärke zeigt sich in beiden Kurven ein ähnliches Verhalten. Die Simulation besitzt erneut einen leicht steileren Abfall zu Beginn, allerdings nimmt dies auch schnell wieder ab und die Kurven zeigen beide ein langsames Einregeln auf den *Hold-Strom* ohne Unterschwinger.

Für die allgemeine Bewertung der Regelung aus einem MBD kann zu Beginn festgestellt werden, dass die grundlegende Funktion erfüllt wurde. Es ist möglich einen Regler aus einem MBD in einen C-Code zu konvertieren und in die Software zu integrieren. Die Verwendung der Simulation zur Einstellung der Regelparameter kann nicht direkt nach dem ersten Versuch bestätigt werden. Allerdings kann die Simulation in einem iterativen Prozess weiter an die Realität angepasst werden. Das bereits durch die Entwicklung überarbeitete Modell kann bereits unter einigen Berücksichtigungen verwendet werden. Ein wichtiger Aspekt, der jedoch künftig integriert werden muss, ist die Trägheit des Systems. Die zu Beginn erwähnten Anforderungen konnten nicht vollständig erfüllt werden, dennoch zeigt sich eine stabile Einregelung auf die vorgegebenen Ströme. Weiterhin muss dazu gesagt werden, dass die Anforderungen bereits bei der überarbeiteten Simulation nicht mehr erreicht wurden. Der allgemeine Aspekt dieser Arbeit, eine Parametrisierung des Reglers über eine Simulation in MBD, kann auf Grund des ähnlichen Verhaltens somit dennoch als erfüllt angesehen werden.

6 Vergleich der Methoden

Nach der finalen Bewertung der Regelung wird in diesem Kapitel ein allgemeiner Vergleich zwischen der Entwicklung des Reglers als reiner C-Code und als MBD gezogen. Hierbei werden drei Aspekte näher betrachtet. Zu Beginn wird auf die Komplexität bei der Entwicklung eingegangen. Dabei geht es sowohl um den ersten Aufbau als auch die allgemeine Verständlichkeit des Reglers für weitere Mitarbeiter:innen. Der nächste Aspekt ist die Anpassungsfähigkeit. In diesem Kapitel wird die schnelle Adaption auf neue Systeme analysiert. Dabei wird unter anderem die Flexibilität der Regelung und die Auslegung der Regelparameter betrachtet. Abschließend wird die allgemeine Nutzbarkeit von MBD in der Software-Entwicklung analysiert.

6.1 Komplexität

In dieser Arbeit wurden viele Schritte aufgebaut, um final einen Regler aus *Simulink* in der Software zu verwenden und darüber ein Ventil zu schalten. Dabei müssen allerdings einige Dinge berücksichtigt werden. Zunächst kann dennoch gesagt werden, es benötigt insgesamt mehr Prozesse, um den Regler in der Software zu integrieren. Beim direkten Entwickeln des C-Codes, kann dieser von Beginn an, angepasst an die Software-Architektur, aufgebaut und ohne Umwege integriert werden.

Der erste Punkt ist der grundlegende Aufbau. Durch die Verwendung von fertigen Funktionsblöcken in einem MBD kann der Regler schneller aufgebaut werden. Hinter jedem Funktionsblock steckt ein C-Code, welcher von Hand geschrieben deutlich mehr Entwicklungszeit in Anspruch nehmen würde. Der zweite Aspekt ist die Übersichtlichkeit. Sowohl im C-Code als auch in *Simulink* kann die Definition der Variablen vom funktionalen Aufbau getrennt werden. Der funktionale Aufbau des Reglers in einer graphischen Umgebung ist allerdings deutlich übersichtlicher. Eine zweite Person kann somit die Verarbeitung der Signale schneller nachvollziehen. Außerdem können Fehler leichter entdeckt werden.

Der nächste Punkt ist die Einschränkung der Programmierung durch Randbedingungen. Beide Methoden werden durch die Softwareumgebung gleich eingeschränkt. Im MBD existieren durch die Code-Generierung mit *TargetLink* weitere Einschränkungen. Es gibt zum einen eine aktuell deutliche Einschränkung in der Verwendung der Funktionsblöcke. Hierzu legt *TargetLink* eine eigene Bibliothek fest, in der viele Blöcke aus *Simulink* übernommen werden, allerdings auch viele weiterhin fehlen. Des Weiteren muss ein klarer Rahmen über das *TargetLink* Subsystem geschaffen werden. Dadurch wird allerdings keine Einschränkung bei der Modell-Entwicklung selbst vorgenommen. Zu guter Letzt müssen die Variablen konkreter definiert werden. Während *Simulink* bereits eine eigene Vordefinierung für den Datentypen festlegt, muss für die Code-Generierung sowohl der Datentyp als auch die Variablenklasse definiert werden. Dadurch zeigt sich außerdem ein unflexibleres Verhalten in der Verwendung von unterschiedlichen Datentypen. Beim direkten Coding zeigen sich jedoch ebenfalls gewisse Vorgaben. In

der Automobilbranche werden die Richtlinien durch den C-Programmierstandard MISRA-C definiert. Dieser Standard begrenzt den Sprachumfang der C-Sprache auf eine Teilmenge, welche zum Schutz vor Sprachaspekten, die die Sicherheit des Gesamtsystems gefährden kann, definiert wird. Die Abkürzung *MISRA* steht dabei für die *Motor Industry Software Reliability Association*, welche die Richtlinien festlegt. Im Insgesamten zeigt sich dennoch eine freiere Gestaltung in der Entwicklung des C-Codes bei der direkten Entwicklung. Dadurch mag zwar der Entwickler im Einzelnen schneller arbeiten können, allerdings sorgt die dadurch auftretende Vielfalt in der einzelnen Code-Architektur für eine generelle Unübersichtlichkeit. Durch die automatisierte Code-Generierung von *TargetLink* wird der Software-Code immer nach dem gleichen Schema aufgebaut werden. Dies schafft eine allgemeine Ordnung und dadurch ein besseres Verständnis.

Zur Komplexität kann somit im Insgesamten gesagt werden, dass beide Methoden ihre Vor- und Nachteile aufweisen. Für die direkte Entwicklung des Reglers durch eine einzelne Person erscheint die direkte Programmierung des C-Codes als die einfachere Methode. Die Funktionsblöcke im MBD sorgen zwar für einen schnellen Aufbau der Regelung, doch durch die einzelnen Prozesse, vom Aufbau des Modelles, bis hin zur Einbettung in einem *TargetLink* Subsystem, Definierung der Variablen, Anpassung einiger Blöcke und der finalen Code-Generierung können in jedem Schritt leicht Fehler auftreten. Für eine einfache Funktion ist daher die direkte Entwicklung des C-Codes die leichtere Methode. Für komplexere Systeme zeigen sich allerdings durch die deutlich bessere Übersichtlichkeit in der graphischen Programmierung klare Vorteile im MBD.

6.2 Anpassungsfähigkeit

Bei der Anpassungsfähigkeit liegt ein bisher unerwähnter Vorteil beim MBD vor. Durch die zusätzliche Entwicklung der Simulation, kann der Regler direkt am PC getestet und optimiert werden. Durch die zusätzliche *Toolbox Simscape* kann ein direktes physikalisches System aufgebaut werden, wodurch die mathematische Bestimmung der Regelstrecke keine Notwendigkeit darstellt. Die direkte Darstellung der Stromkurve und die Einstellung der Parameter über ein separates Skript sorgen dafür, dass sich jeder Benutzer schnell einarbeiten und den Regler für ein neues Ventil auslegen kann. Zusätzlich zur iterativen Anpassung kann die Simulation zur Auswertung der Sprungantwort und damit schnelleren Ermittlung geeigneter Regelparameter verwendet werden. Hierzu muss dennoch berücksichtigt werden, dass somit zunächst eine Anpassung der Hardware-Parameter zur Simulation, eine darauffolgende Anpassung der Regelparameter in *Simulink*, eine Anpassung der Werte in der *TargetLink* Datenbank und eine erneute Code-Generierung stattfinden muss. Dennoch ist es von Vorteil, dass der grundlegende Aufbau der Simulation immer gleich bleiben kann, während sich die Hardware am direkten Prüfstand in jedem Projekt ändert.

In der ursprünglichen Methode können die Parameter im C-Code direkt überarbeitet werden. Allerdings muss separat eine Bestimmung der neuen Regelparameter durchgeführt werden. Eine Alternative dazu ist ein iterativer Prozess, wie bei der Simulation, allerdings an der direkten Hardware. Hierbei kann der *Debug-Port* verwendet werden, um direkt auf die Software zuzugreifen und die Werte anzupassen. Somit kann über die Stromkurve am Oszilloskop die Einstellung der Regelparameter stattfinden. Es kann allerdings durch vorzeitiges Eingreifen von Sicherheitsbedingungen dazu kommen, dass die Steuerung abgebrochen und somit kein verwendbares Ergebnis am Oszilloskop gezeigt wird.

Somit kann gesagt werden, dass das MBD besser geeignet ist für die schnelle Anpassung des Reglers. Dabei mag die Simulation, wie die Entwicklung in dieser Arbeit gezeigt hat, nicht direkt beim ersten Aufbau ideal zur Reglerdimensionierung geeignet sein, doch durch eine iterative Anpassung der Simulation an die Hardware, kann das Modell künftig besser für weitere Ventilregelungen verwendet werden.

6.3 Allgemeine Nutzbarkeit

Die Frage der allgemeinen Nutzbarkeit bezieht sich vor allem auf das MBD. Die allgemeine Programmierung des C-Codes kann selbstverständlich immer verwendet werden. Bei der neuen Methode müssen allerdings zwei Fragen geklärt werden. Zunächst geht es um die Verwendung der Code-Generierung bei unterschiedlichen Steuereinheiten. Durch den allgemeinen Prozess der Software-Generierung zeigt sich jedoch kein Problem. Für den Aufbau wird ein einfacher C-Code benötigt, welcher über *TargetLink* generiert werden kann. Da die Regelung auf Applikationsebene stattfindet und durch die Software-Architektur im AUTOSAR-Standard über die BSW von der Hardware entkoppelt wird, kann der Prozess für alle Steuereinheiten verwendet werden. Des Weiteren kann durch die verschiedenen Vorlagen in *TargetLink* die Datenbank direkt an verschiedene Architektur-Standards angepasst werden. Zur Verwendung von *TargetLink* in komplett anderen Systemen muss berücksichtigt werden, dass nur reiner Produktionscode in C oder C++ generiert werden kann. Andere Sprachen werden von *TargetLink* nicht unterstützt.

Ein weiterer Aspekt ist die allgemeine Verwendung von MBD. Es muss berücksichtigt werden, dass in dieser Arbeit viele Programme verwendet werden, um den Regler zu erzeugen, welche nicht frei zugänglich sind und somit einen hohen Kostenfaktor für eine Firma darstellen können. Grundlegend muss eine Lizenz für *Matlab* und *Simulink* erworben werden. Des Weiteren wird eine Lizenz von *DSpace* zur Nutzung von *TargetLink* benötigt. Zusätzlich werden in dieser Arbeit die Toolboxen *Stateflow* und *Simscape* von *Simulink* verwendet. Diese werden allerdings nicht zwingend benötigt. Für einen reinen mathematischen Aufbau des Reglers und der Simulation sind die grundlegenden Funktionsblöcke von *Simulink* ausreichend.

Resultierend kann gesagt werden, dass die Verwendung von MBD in der Software-Entwicklung für verschiedene Systeme geeignet ist. Je nach Anwendungszweck und Effizienz sollte dabei jedoch der Kostenfaktor durch zusätzliche Programme nicht unterschätzt werden.

7 Fazit und Ausblick

Zusammenfassend kann festgestellt werden, dass die entwickelte Stromregelung zusammen mit der Simulation des Magnetventils für die allgemeine Programmierung des Reglers und der Ansteuerung der Ventile im Luftfedersystem geeignet ist. Durch den grundlegenden Aufbau des Magnetventils über *Simscape* kann die Regelstrecke schnell und einfach über die Definition der Hardware-Spezifikationen im Skript an weitere Ventile angepasst werden. Dabei muss der Signalfluss und die verwendeten Softwarekomponenten bei der Schaltung der Ventile berücksichtigt werden. In der verwendeten Software werden die Magnetventile für die Höhenregelung der vier Luftfedern, den Kompressor, das Reservoir, den Boost und das Auslassventil über ein Bussignal angesteuert. Folglich können alle diese Ventile auch über die neue Regelung angesteuert werden. Einige Ventile werden allerdings auch über eine externe Stromregelung gesteuert, bei der eine Anpassung schwieriger werden kann.

Die zusätzliche Dimensionierung des Reglers über die Sprungantwort kann aktuell zwar nur mit Anpassungen des Integrierbeiwertes optimal verwendet werden, sollte jedoch durch eine Anpassung der Systemträgheit in der Regelstrecke künftig optimiert werden können. Hierzu sollte ein Faktor bei der Induktivität der Spule integriert werden. Durch eine höhere Induktivität kann die Trägheit verstärkt und somit an das reale Verhalten angepasst werden. Des Weiteren muss das Verhalten des Stromabfalles bei Öffnung des Ventiles weiter analysiert werden. In dieser Arbeit wurde bereits eine Anpassung an das Verhalten des Auslassventiles vorgenommen, wodurch die Simulation optimiert werden konnte. Künftig muss jedoch der Stromabfall bei weiteren Ventilen aufgenommen werden und ein Zusammenhang zwischen der Stärke des Abfalles und den Spezifikationen der Spule und den Ausgangsbedingungen hergestellt werden.

Zur Integration des Reglers in die Software sollte in Zukunft eine neue Architektur aufgebaut werden. Da zur Validierung in dieser Arbeit eine vollständig funktionierende Software mit einem passenden Prüfstand benötigt wurde, musste eine einfache Übergangslösung gefunden werden. Da jedoch durch die Regelung Funktionen der Softwarekomponenten *SIO* und *PwmICtrlr* ersetzt werden können, wird hier ein insgesamt neuer Aufbau dieser Komponenten und die Integration einer zusätzlichen Komponente für den Regler benötigt. Dadurch kann allgemein die Stromregelung, welche momentan über den *PwmICtrlr* innerhalb der BSW stattfindet, in die Anwendungsschicht verschoben werden und somit ein leichterer Zugriff geschaffen werden.

8 Literaturverzeichnis

AUTOSAR | Vector. (2024, 18. Novembera). Verfügbar unter: https://www.vector.com/de/de/know-how/autosar/

Balakrishnan, M. & Kumar N, N. (2015). *Detection of Plunger Movement in DC Solenoids*, Texas Instruments. Verfügbar unter: https://www.ti.com/lit/pdf/ssiy001

Beier, T. & Wurl, P. (2013). *Regelungstechnik. Basiswissen, Grundlagen, Anwendungsbeispiele*. München: Carl Hanser Verlag.

DIN, 19 226. *Regelungstechnik und Steuerungstechnik, Begriffe und Benennungen*.

Gieck, K. & Gieck, R. (2013). *Technische Formelsammlung* (33 neu bearbeitet). München: Carl Hanser Verlag.

Grote, K.-H. & Feldhusen, J. (Hrsg.). (2014). *Dubbel. Taschenbuch für den Maschinenbau* (24. Aufl.). Berlin, Heidelberg: Springer-Verlag.

Horn.. swp0021.dvi. Verfügbar unter: https://www.tugraz.at/fileadmin/user_upload/Institute/IRT/Aufgabensammlung/Regelungstechnik/Regelungstechnik.pdf

Hüning, F. (2016). *Sensoren und Sensorschnittstellen*. Berlin, Boston: De Gruyter Oldenbourg.

Isermann, R. (2006). *Fahrdynamik-Regelung. Modellbildung, Fahrerassistenzsysteme, Mechatronik* (1. Aufl.). mit 28 Tabellen. Wiesbaden: Vieweg.

Kindel, O. & Friedrich, M. (2009). *Softwareentwicklung mit AUTOSAR. Grundlagen, Engineering, Management in der Praxis* (1. Aufl.). Heidelberg: dpunkt.Verlag GmbH.

Lunze, J. (2020). *Regelungstechnik 1. Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen* (12. Aufl.). Berin: Springer-Verlag.

Neue, R. (2024, 18. November). *AUTomotive Open System ARchitecture. Eine Einführung*. Verfügbar unter: https://ess.cs.tu-dortmund.de/Teaching/PGs/autolab/ausarbeitungen/Neue-AUTOSAR-Ausarbeitung.pdf

PWM - Pulsweitenmodulation (PDM PLM PBM). (2024, 19. Novemberb). Verfügbar unter: https://www.elektronik-kompendium.de/sites/kom/0401111.htm

Rost, J. (2024, 18. November). *embeddeers GmbH » Grundlagen der AUTOSAR Architektur*. Verfügbar unter: https://www.embeddeers.com/knowledge-area/grundlagen-der-autosar-architektur/

Simscape. (2024, 19. Novemberc). Verfügbar unter: https://de.mathworks.com/products/simscape.html

Solver Configuration. (2024, 19. Novemberd). Verfügbar unter: https://de.mathworks.com/help/simscape/ref/solverconfiguration.html

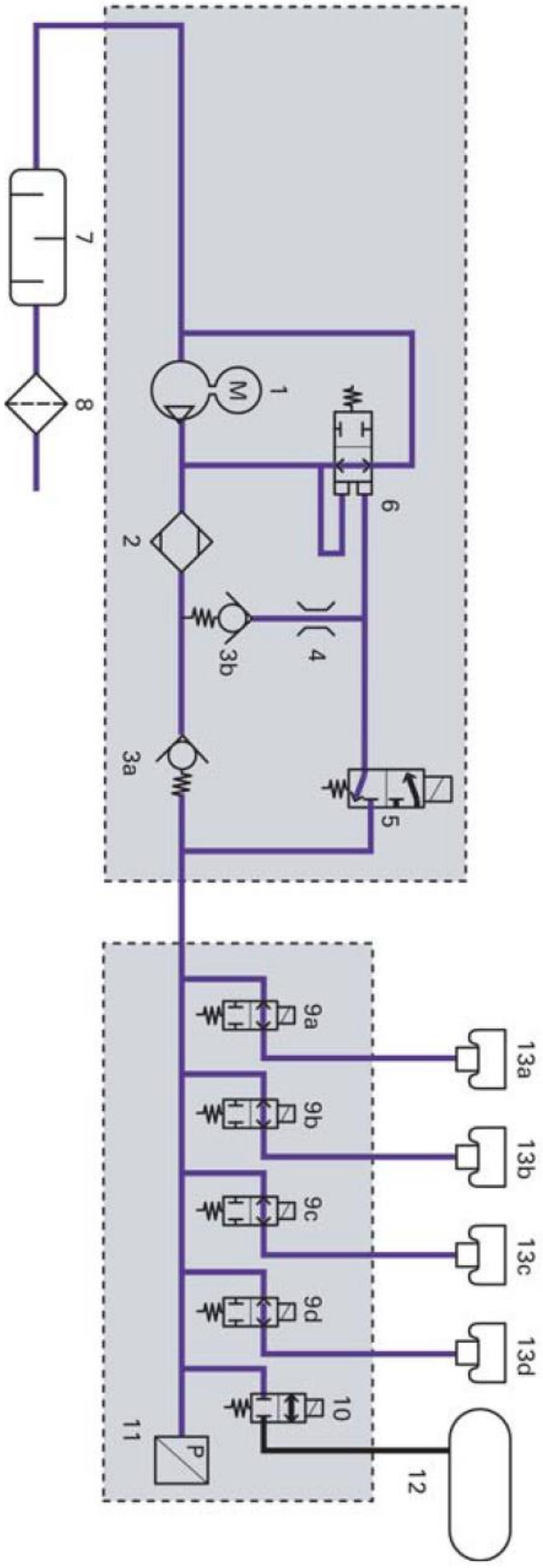
Storr, W. (2018). *Tiefpassfilter - Passives RC Filter Tutorial*. Verfügbar unter: https://www.electronics-tutorials.ws/de/filtern/passiver-tiefpassfilter.html

TargetLink. (2024, 19. Novembere). Verfügbar unter: https://www.dspace.com/de/gmb/home/products/sw/pcgs/targetlink.cfm

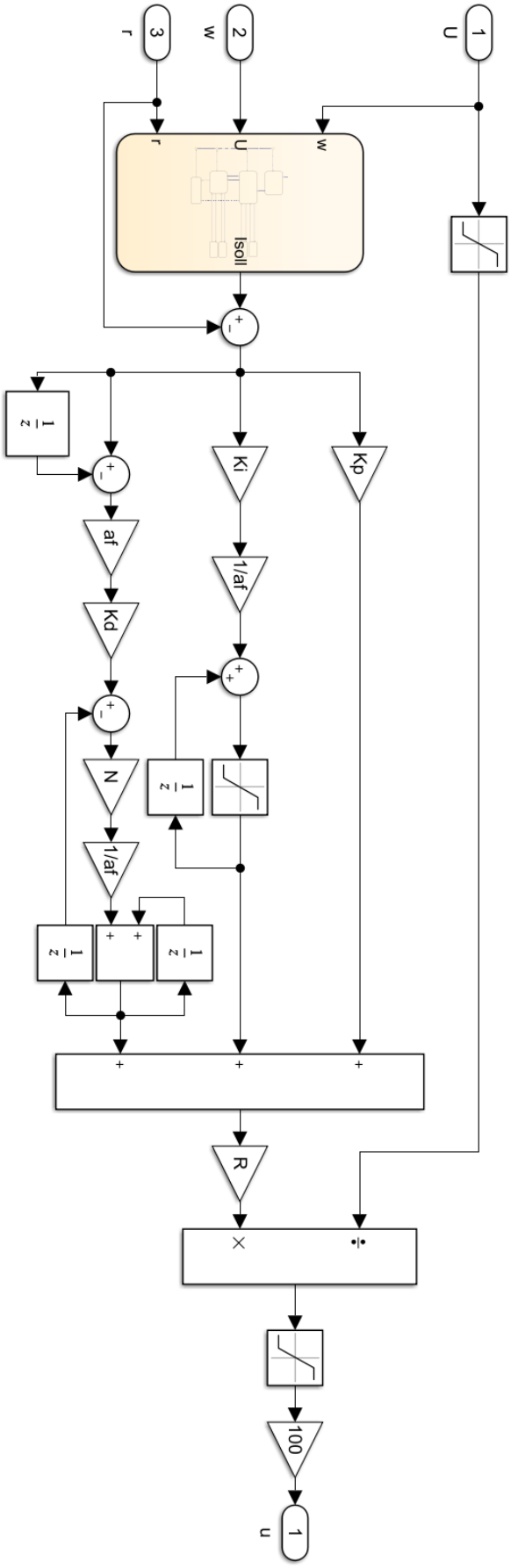
Warum sollten Sie Model-Based Design einführen? (2024, 21. Novemberf). Verfügbar unter: https://de.mathworks.com/campaigns/offers/next/why-adopt-model-based-design.html

Anhang

I. Systemübersicht des *Four-Corner-Luftfedersystems* bei Druckabbau



II. Gesamtdarstellung des PID-Reglers als MBD



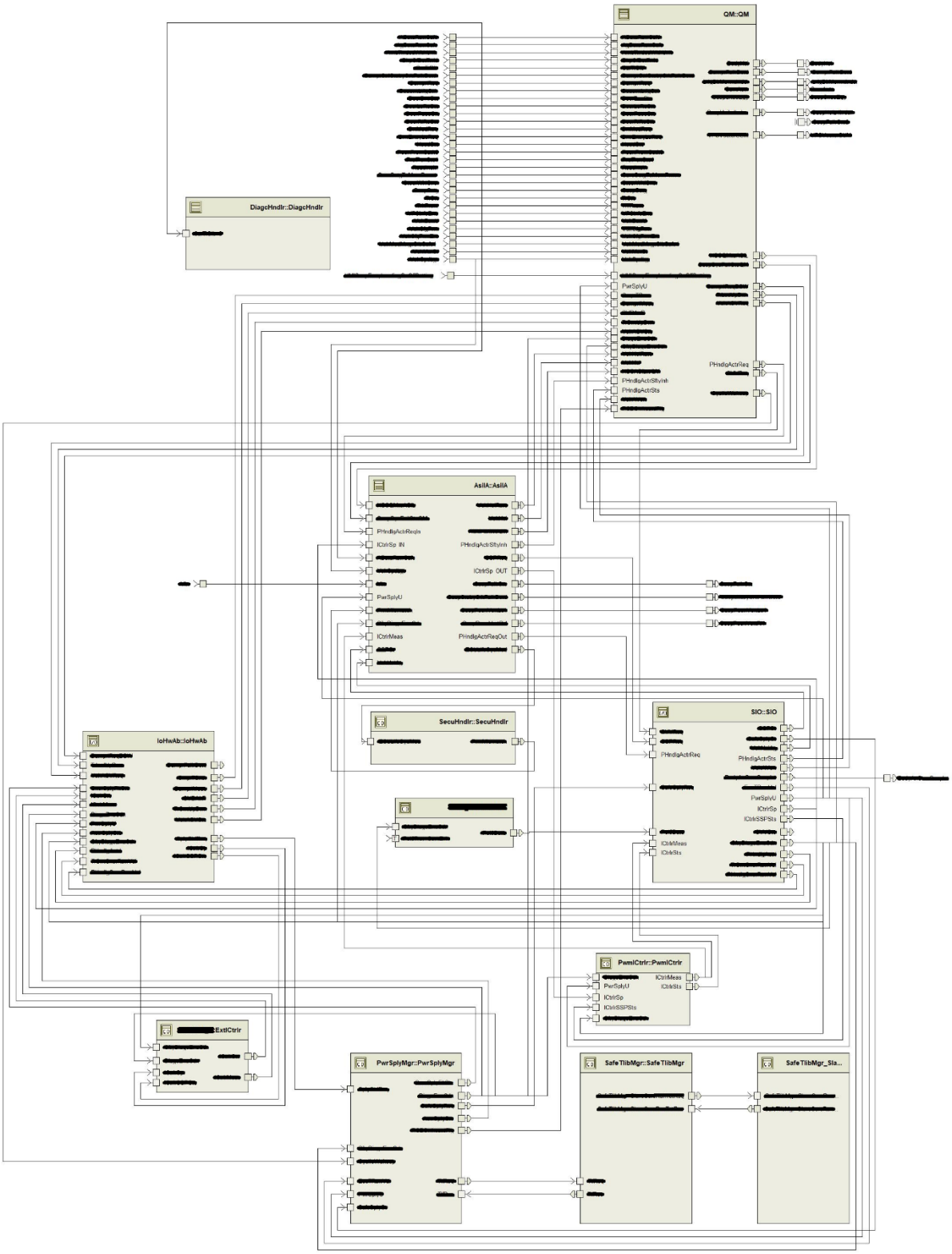
III. Skript zur Ableitung der PT1-Übertragungsfunktion aus der Sprungantwort

```
n=numel(Mess_Strom.Time);
x=Mess_Strom.Time;
y=Mess_Strom.Data;
ym=y(round(n*3/4):n);
K=mean(ym);
KT = 0.632*K;
for i=1:1:n
   if y(i)>=KT
      T=x(i);
      break
   end
end
t=0:0.001:0.05;
PT=K*(1-exp(-t./T));
ys=smoothdata(y,'gaussian',200);
yt=diff(ys);
[my,ny]=max(yt);
wy=ys(ny+1);
my=my/SampleT;
TU=x(ny+1)-wy/my;
Tg=K/my;
ty=0:0.01:(Tg+TU);
wt=my*ty-my*TU;
```

IV. Skript zur Ermittlung der Summenzeitkonstante

```
n=numel(Mess_Strom.Time);
x=Mess_Strom.Time;
y=Mess_Strom.Data;
ym=y(round(n*3/4):n);
Ks=mean(ym)/Step*1000;
for i=1:1:n
   A1=sum(y(1:i));
   A2=mean(ym)*(n-i)-sum(y(i:n));
   if A1>=A2
      Tsum=x(i);
      break
   end
end
```

V. Gesamtentwurf der Software-Architektur mit zensierten Signalen



Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit mit dem Titel:

Entwicklung und Implementierung eines modellbasierten Reglers in eine C-Code-basierte Software im Automotive Kontext

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Datum	Unterschrift