Hochschule **Rosenheim**
University of Applied Sciences

Bachelorthesis Produktionstechnik

Andreas Bernatzky

Aufbau und regelungstechnischer Entwurf eines Lüfter Prüfstandes

Angefertigt am:	**Fraunhofer Institut für Bauphysik**
Betreuer am Institut:	**Dipl.-Ing. Markus Siede**
Erstkorrektor der HS Rosenheim:	**Prof. Dr.-Ing. Peter Zentgraf, M.Sc.**
Zweitkorrektor der HS Rosenheim:	**Prof. Dr.-Ing Martin Versen**
Abgegeben am:	**27.07.2017**

ERKLÄRUNG:

Ich versichere, dass ich diese Arbeit selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Rosenheim den, 27.07.2017

Andreas Bernatzky

Danksagung:

Besonders danken möchte ich meinem Erstkorrektor Prof. Dr.-Ing. Peter Zentgraf, M.Sc., welcher mit seiner Vorlesung und Begeisterung für das Aufgabengebiet der Regelungstechnik meine Begeisterung für diese Ingenieurswissenschaft geweckt hat und damit meinen weiteren beruflichen Schwerpunkt bestimmt hat.

Meinem Zweitkorrektor Herrn Prof. Dr.-Ing Martin Versen möchte ich für seine Anregungen und Anstöße bei der Entwicklung der GUI danken, da ohne diese Anregung einiges an Innovation nicht gegeben wäre.

Ebenso möchte ich meinem Betreuer Herrn Dipl.-Ing Markus Siede für die Bereitstellung des Themas danken. Meinen Kollegen Herrn Maximilian Kienberger und Frau Dipl.-Ing. Marie Pschirer möchte ich für das herzliche Arbeitsklima danken.

Zum Schluss würde ich gerne meiner Mutter Frau Sigrid Bernatzky danken, welche während meiner Bachelorarbeit für mein leibliches Wohl gesorgt hat und meinem Vater Herrn Klaus Bernatzky für die kritische Durchsicht meiner Arbeit. Meiner Schwester und meinen Brüdern Tanja, Julian und David Bernatzky will ich danken dafür das Sie mich während meiner Studienzeit unterstützt haben.

Kurzfassung:

Diese Bachelorarbeit wurde am Fraunhofer Institut für Bauphysik im Sommersemester 2017 angefertigt. Das Ziel dieser Bachelorarbeit war der Reglerentwurf und Aufbau eines einphasigen Lüfterprüfstandes, welcher über einen Mikrocontroller angesteuert wird. Die hierfür benötigte Streckenidentifikation und der dazugehörige Regler Entwurf wurden nach den praxisnahesten Identifikationsverfahren und Einstellregeln durchgeführt. Die Regelgröße war die Strömungsgeschwindigkeit der Luft in einer gerade durchströmten Rohrstrecke. Für die Rückführung der Regelgröße wurden einmal ein thermisch messender Sensor und einmal ein Differenzdrucksensor verwendet, für das System mit dem Differenzdrucksensor erfolgte eine Systemidentifikation und ein Reglerentwurf. Um die Messdaten in Echtzeit mit verfolgen zu können wurde in der Programmiersprache C# eine GUI geschrieben. Diese kann über die serielle Schnittstelle eines Laptops die Messdaten visualisieren, speichern und ermöglicht im laufenden Programm die Führungsgröße und die Regelparameter zu ändern. Um eine Kommunikation zwischen Mikrocontroller und Mess- Leistungselektronik zu ermöglichen wurde hier noch eine elektrische Schnittstelle geschaffen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Für eine Vielzahl von Raumklimauntersuchungen ist es notwendig, einen präzise geregelten und konstanten Luftstrom zu erzeugen. Hierfür soll ein Lüfter Prüfstand entwickelt werden, welcher über einen Mikrocontroller gesteuert und geregelt werden kann und gleichzeitig die Aufgabe der Messwerterfassung erfüllt. Hierzu soll eine Steuerelektronik entworfen werden, um eine Schnittstelle zwischen dem Mikrocontroller und der Leistungselektronik zu schaffen. Ebenso wird eine Schnittstelle zwischen Mikrocontroller und Messelektronik benötigt, welche entweder Digital über ein Busprotokoll realisiert werden kann, oder über die Verwendung von Messsensorik, welche mit analogen Signalen arbeitet. Beide Messsysteme bieten hierfür verschiedene Vorteile. Das analoge System überzeugt durch seine Einfachheit in der Handhabung, da hier das analoge Signal direkt vom Mikrocontroller aufgrund schon bestehender ADC-Wandler verarbeitet werden kann. Die digital arbeitende Messelektronik hingegen bietet die Möglichkeit, eine große Anzahl mit Busprotokoll arbeitender Sensoren flexibel in ein Messsystem zu integrieren. Um systematische Fehler durch den Einbau der Messsensorik am Versuchsstand möglichst gering zu halten, sollen im durchströmten Rohr fertig verbaute Messlanzen zum Einsatz kommen. Durch die Bestimmung des Differenzdrucks, welcher sich aus der Differenz von Gesamtdruck und statischen Druck ergibt, kann somit der dynamische Druck bestimmt werden, aus welchem sich die Regelgröße Strömungsgeschwindigkeit im Rohr ableiten lässt. Der hier zur Verwendung kommende Mikrocontroller soll auf der ATMEGA- oder SAM-Reihe der Firma Atmel basieren. Eine weitere Anforderung an den Prüfstand besteht darin, dass er möglichst einfach überall am Fraunhofer Institut für Bauphysik angeschlossen werden kann. Daher empfiehlt es sich einen einphasigen 230 Volt AC Motor zu verwenden, welcher mit entsprechender Leistungselektronik in der Drehzahl verstellbar ist. Da das so entstehende System eine hohe Güte bezüglich der Regelabweichung aufweisen soll, muss eine Systemidentifikation erfolgen, mit einem anschließenden Reglerentwurf und einer Gegenüberstellung der gängigsten und praktischsten Entwurfsverfahren.

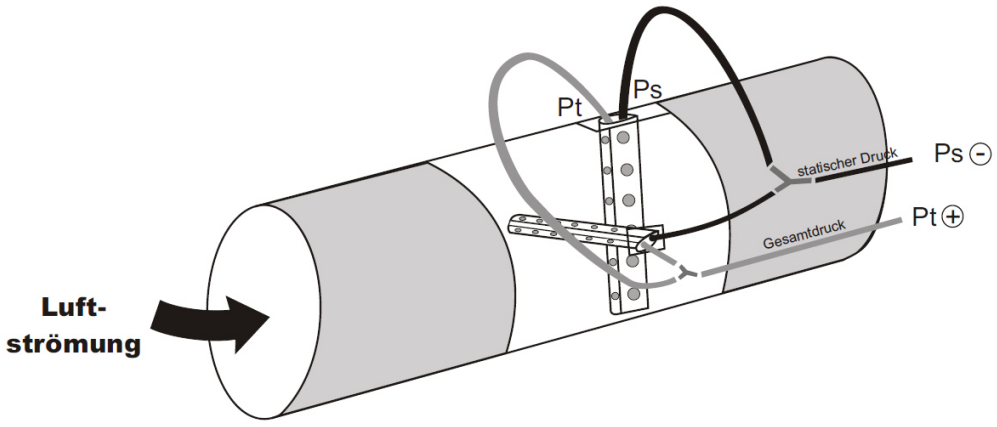


Abbildung 1 Messprinzip Differenzdrucksensor [MDUA GmbH & CO. KG,]

1.1 Stand der Technik

Der derzeitige Stand der Technik umfasst zwei Arten von Lüfter Prüfständen: Prüfstände, welche die Druckkennlinien von Lüftern aufnehmen sollen, um aus diesen Erkenntnisse über den Lüfter zu erhalten oder solche, welche einen konstanten Volumenstrom erzeugen sollen, um z.B. Erkenntnisse über ein anderes Objekt zu erhalten, z.B. die aerodynamische Beschaffenheit eines Autos. Für das Fraunhofer Institut für Bauphysik kommen Prüfstandsvarianten von der zweiten Sorte in Frage. Momentan werden Lüfter für groß angelegte Versuchsreihen über eine Beckhoff-CPU geregelt, welche aber ohne Systemidentifikation oder einen Regler Entwurf in Betrieb genommen werden. Der in dieser Bachelorarbeit verwendete Lüfter stammt aus einer ehemaligen Doktorarbeit und wurde per Hand über einen Lichtdimmer geregelt. Da nicht immer auf eine teure Beckhoff-CPU zurückgegriffen werden kann, aber gleichzeitig die Regelung per Hand oftmals zu unpraktisch und vor allem ungenau ist, soll ein Prüfstand entwickelt werden, welcher möglichst mobil ist und eine hohe Güte bezüglich der Regelabweichung aufweist.

2 Prüfstandskomponenten

Der Lüfter Prüfstand besteht in seiner Gesamtheit aus der verwendeten Leistungselektronik, dem Lüfter, der Sensorik und der Mess-Steuerplatine mit einem Mikrocontroller. Um mit dem Prüfstand interagieren zu können, ist außerdem ein Computer erforderlich.

2.1 Leistungselektronik

Um den einphasigen Lüfter Motor in seiner Drehzahl variieren zu können, bedarf es einer Leistungselektronik in Form eines Thyristorstellers oder Frequenzumrichters. Für den Prüfstand wurde ein einphasiger Thyristorsteller der Firma „*Chiemtronic“, „T-Drive 1 Phase Compact*“ verwendet. Der verwendete Thyristorsteller besitzt eine analoge 0-10 V-Schnittstelle, mit welcher die Drehzahl des Motors reguliert werden kann.

2.1.1 Thyristorsteller

Um die Drehzahlregulierung durch einen Thyristorsteller verstehen zu können ist es erforderlich, dessen Komponenten und deren Arbeitsweise zu verstehen. Für diese Bachelorarbeit wurde zuerst ein eigener Thyristorsteller nach einem Schaltplan [diy_bloke, 2012] entworfen, welcher direkt über einen Mikrocontroller angesteuert werden kann. Da für diesen aber kein ausreichendes Sicherheitskonzept geliefert werden konnte, wurde ein gekauftes Produkt verwendet. Dennoch soll anhand dieses Schaltplanes der Aufbau und die Arbeitsweise erklärt werden, da diese denen des gekauften Produkts ähnlich sind.

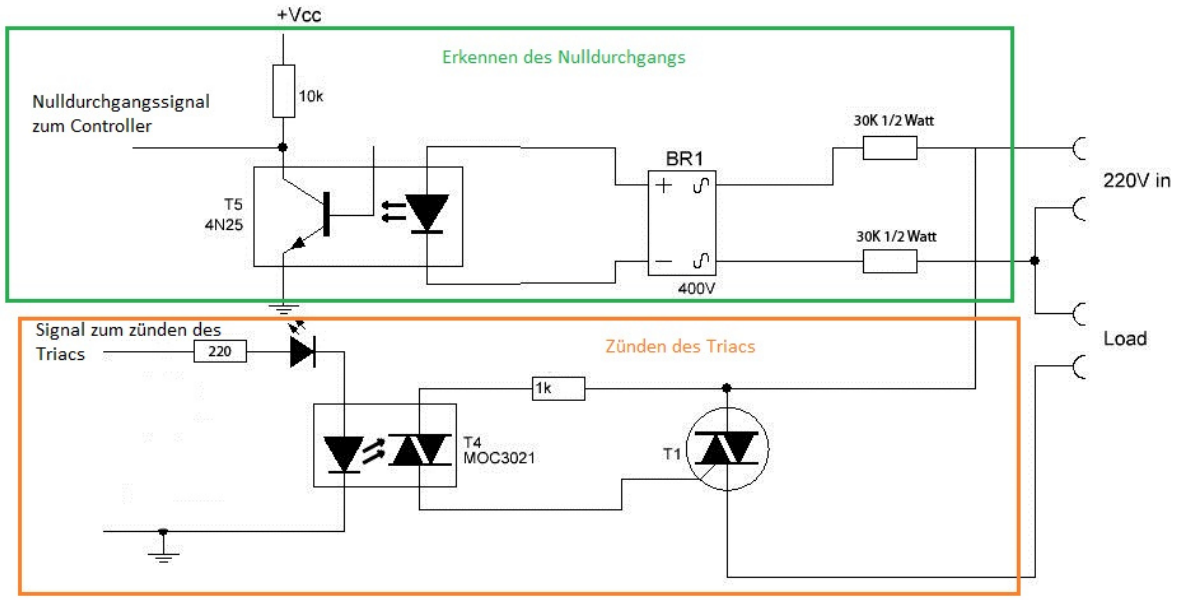


Abbildung 2 Thyristorsteller [diy_bloke, 2012]

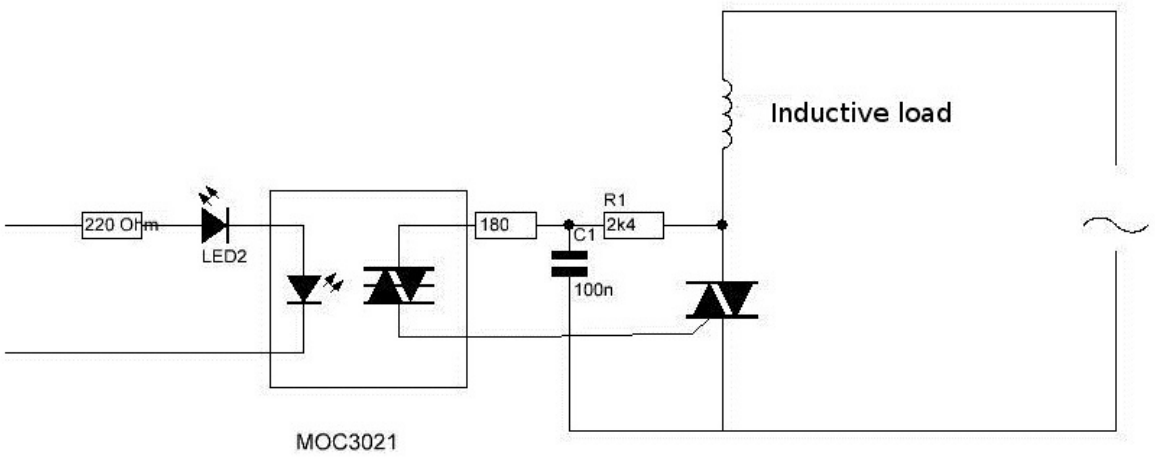


Abbildung 3 Thyristorsteller mit Snubber-Kondensator für induktive Lasten [diy_bloke, 2012]

Die Schaltung in Abbildung 2 wird an eine sinusförmige Wechselspannung mit einem Effektivwert von 230 V, und einer Frequenz von 50 Hz angeschlossen. Bei einer Netzfrequenz von 50 Hz dauert jede Periode 20ms.

$$\frac{1}{50\,\frac{1}{s}} = 20\,ms$$

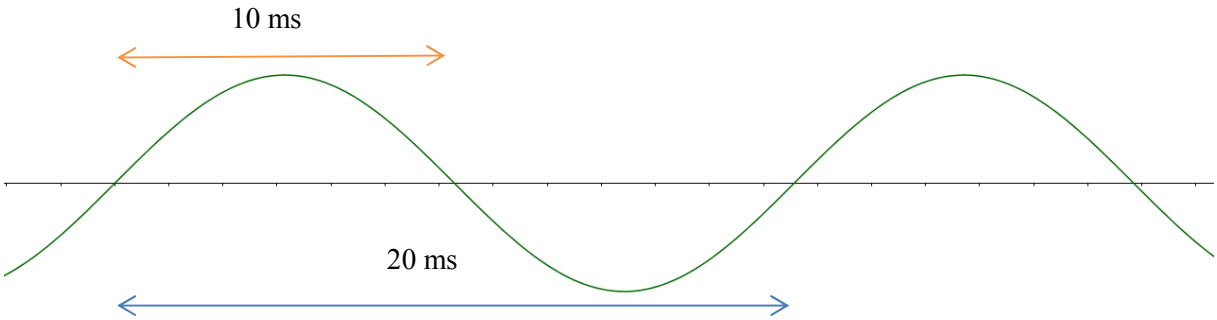


Abbildung 4 Sinusverlauf zeitlich

Zwischen jedem Nulldurchgang besteht daher eine Zeit von 10 ms. Durch den Brückengleichrichter BR1 (Abbildung 2) wird eine negative Halbwelle gleichgerichtet und „nach oben geklappt“ (vgl. Abbildung 5).

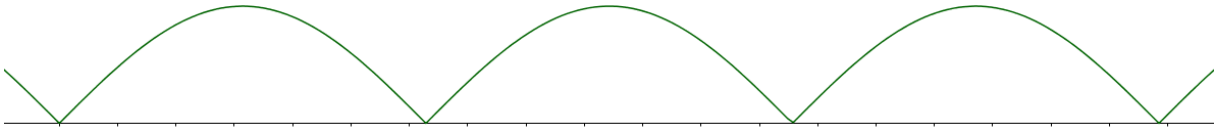


Abbildung 5 Sinus gleichgerichtet

Das gleichgerichtete Sinussignal schaltet somit den Transistor des Optokopplers durchgehend leitend, was einen Stromfluss gegen Masse bewirkt. Bei einem Nulldurchgang hingegen sperrt der Transistor, seine Kollektor-Emitter-Strecke, was den Stromfluss gegen einen analogen oder digitalen Eingang des Mikrocontrollers fließen lässt. Die Information des Nulldurchgangs ist für den Mikrocontroller essentiell, da ansonsten keine zeitgenaue Zündung des Triacs (T1) durch den Mikrocontroller erfolgen kann.

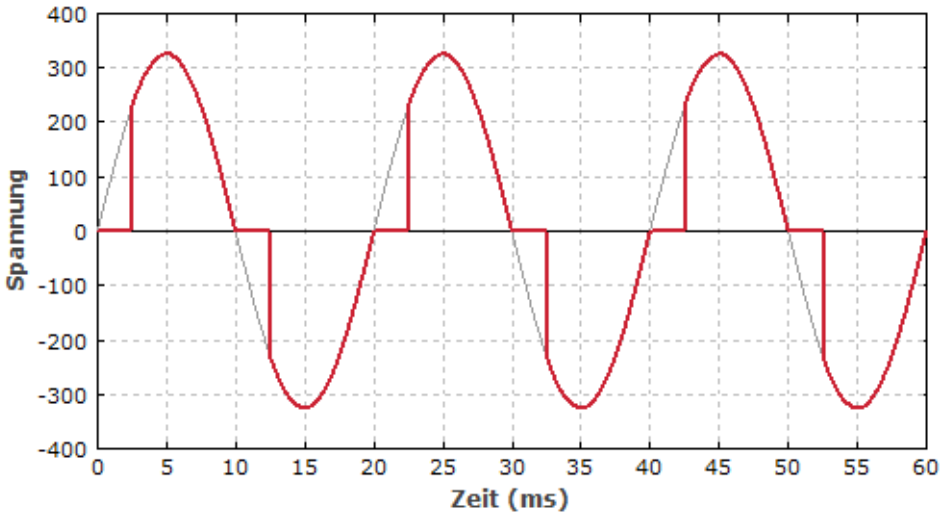


Abbildung 6 Phasenanschnittssteuerung [Paschotta, 2012]

Nach einer vorgegebenen Verweilzeit, beginnend ab dem Nulldurchgang, schaltet der Mikrocontroller den Triac durch (vgl. Abbildung 6). Man spricht hierbei von einer Phasenanschnittssteuerung. Die Veränderung der Drehzahl geschieht hierbei rein über die Änderung der Motorspannung und damit das Drehmoments des Motors und nicht über eine Änderung der Netzfrequenz, wie es ein Frequenzumrichter tun würde.

2.2 Lüfter

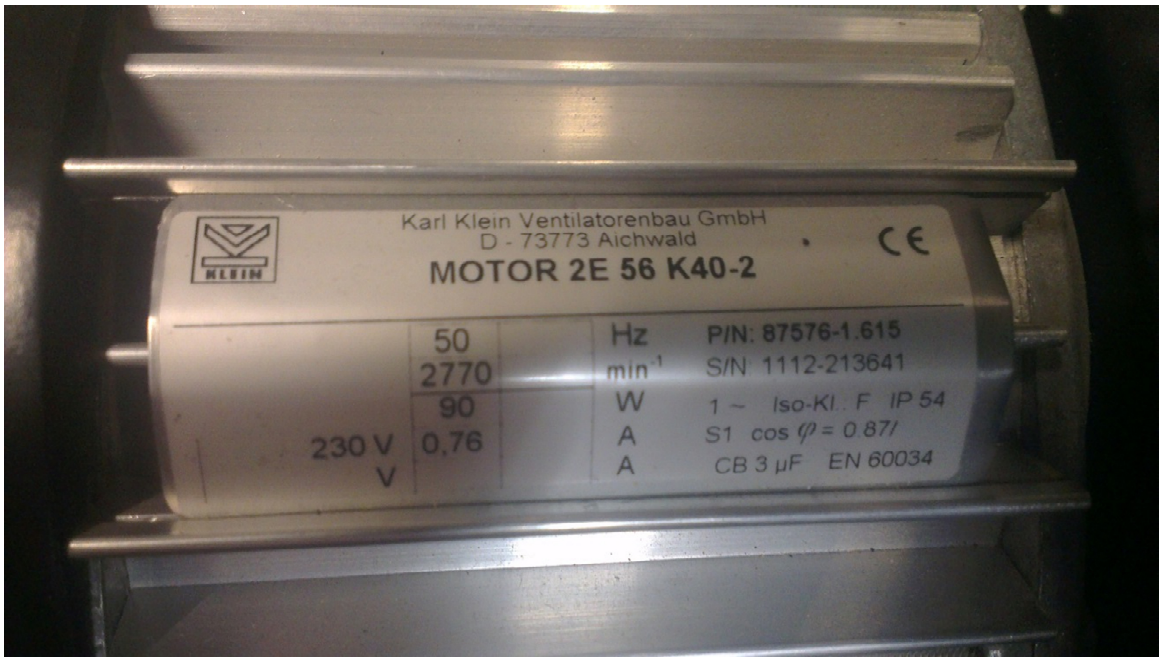


Abbildung 7 Motortypenschild

Bei dem verwendeten Lüfter für den Prüfstand handelt es sich um einen einphasigen Wechselstrommotor mit Betriebskondensator.

Der verwendete Motor gehört zu der Gruppe der Asynchronmotoren. Dies hat zur Folge, dass der Motoranker sich asynchron zu seinem elektrischen Drehfeld bewegt. Die Strömungsgeschwindigkeit im Rohr ist abhängig von der Drehzahl des Motors. Durch den Phasenanschnitt des Thyristorstellers wird nur ein Teil der Sinuswechselspannung genutzt. Durch das Beschneiden der Sinuskurve wird das Motordrehmoment verringert. Aufgrund seiner Motorkennlinie braucht der Lüfter für seine Nenndrehzahl nicht das maximale Motordrehmoment. Daher erreicht der verwendete Motor seine Nenndrehzahl schon bei 5 V Stellsignal am Thyristorsteller. Ein Stellsignal größer 5 V hat nur eine Erhöhung des verfügbaren Motordrehmoment zufolge, was wiederum für die Drehzahl des Motors und damit die maximale Strömungsgeschwindigkeit keine Rolle spielt.

2.2.1 Mathematisches Modell eines Lüftermotors

Das Verhalten der Regelgröße der Strömungsgeschwindigkeit im Rohr ist abhängig von der Drehzahl des Lüfter Motors. Um eine Aussage über die Regelstrecke zu erhalten, ist es erforderlich, diese mit einem möglichst dynamischen Testsignal anzuregen. Anhand der Sprungfunktion wird anschaulich sichtbar, dass die Änderung der Drehzahl des Motors und damit die Änderung der Strömungsgeschwindigkeit nicht zeitgleich mit dem Sprung erfolgen können, sondern eine Verzögerungszeit haben, um die Enddrehzahl n_{end} bzw. die Endgeschwindigkeit v_{end} zu erreichen. Der Verlauf der Strömungsgeschwindigkeit im Rohr wird im folgenden Bild gezeigt.

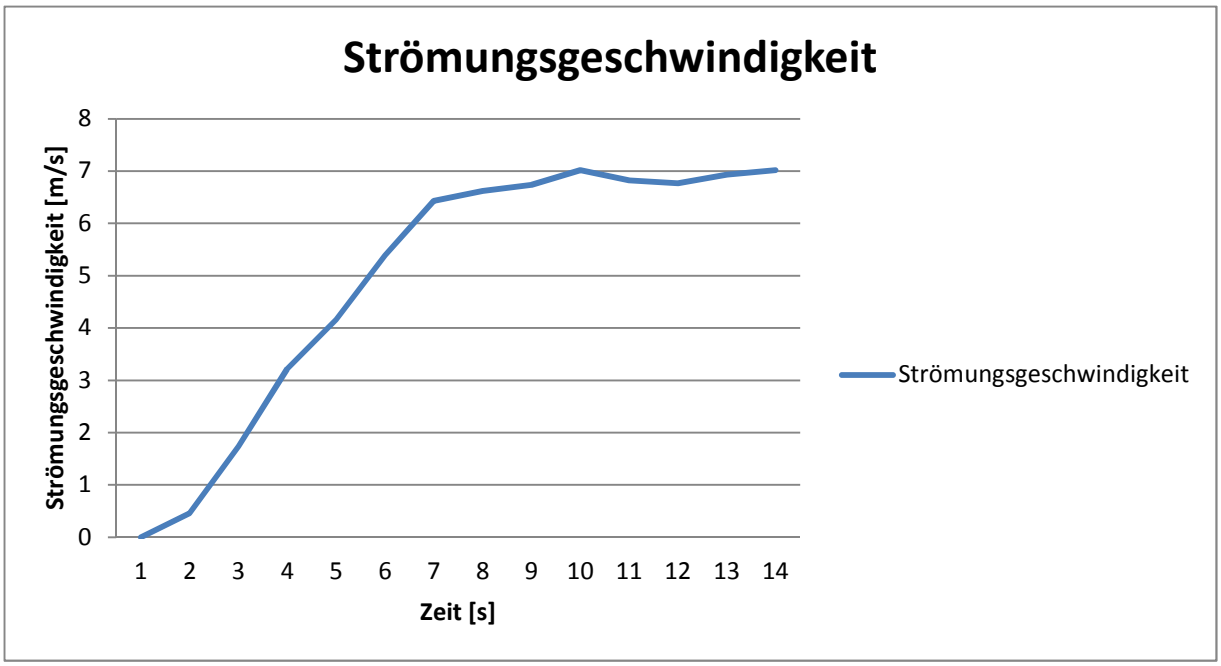


Abbildung 8 Sprungantwort des Lüfters gemessen mit Differenzdrucksensor

Mathematisch lässt sich der Verlauf der Strömungsgeschwindigkeit mit folgender Formel im Zeitbereich beschreiben. [Homann, 2016]

$$v(t) = v_{end} * \left(1 - e^{-\frac{t+T_t}{T_1}}\right)$$

Dieses Verhalten entspricht einem Proportionalglied mit einer Verzögerung 1. Ordnung. Zusätzlich besitzt die Sprungantwort noch ein Totzeitglied, welches das verzögerte Ansprechen der Sensorik und des Motors darstellt. Die Zeitkonstante T1 beschreibt die Zeit, nach der die Strömungsgeschwindigkeit 63% der Endströmungsgeschwindigkeit erreicht hat. [Homann, 2016]

2.3 Messtechnik zur Ermittlung der Strömungsgeschwindigkeit

Für eine funktionierende Regelung ist es notwendig, einen geschlossenen Regelkreis mit Rückführung der Regelgröße zu besitzen. Hierfür benötigt man Messtechnik, welche in der Lage ist, Strömungsgeschwindigkeiten im Rohr zu messen (vgl. Abbildung 9).

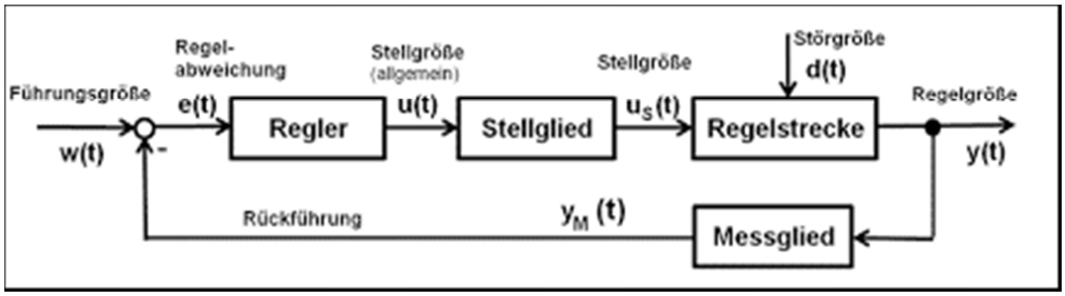


Abbildung 9 Geschlossener Regelkreis mit Messglied [Kümmeke]

Zur Verfügung stehen hier ein thermisches Messprinzip oder die Ermittlung der Strömungsgeschwindigkeit durch die Ermittlung des Differenzdrucks.

2.3.1 Thermische Anemometrie

Das thermische Messprinzip arbeitet nach dem Hitzedrahtprinzip. Das vorbeiströmende Medium kühlt den temperaturgeregelten Sensorkopf ab, welcher 40 K über der Temperatur des Luftmediums gehalten wird. Die benötigte Energiezufuhr zum Halten der Sensortemperatur ist das Maß für die Strömungsgeschwindigkeit des Mediums, welche auf ein 0-10 V-Signal umgesetzt wird. Vorteilhaft bei diesem Messprinzip ist die direkte Messung der tatsächlichen Luftgeschwindigkeit. [Schmidttechnology]

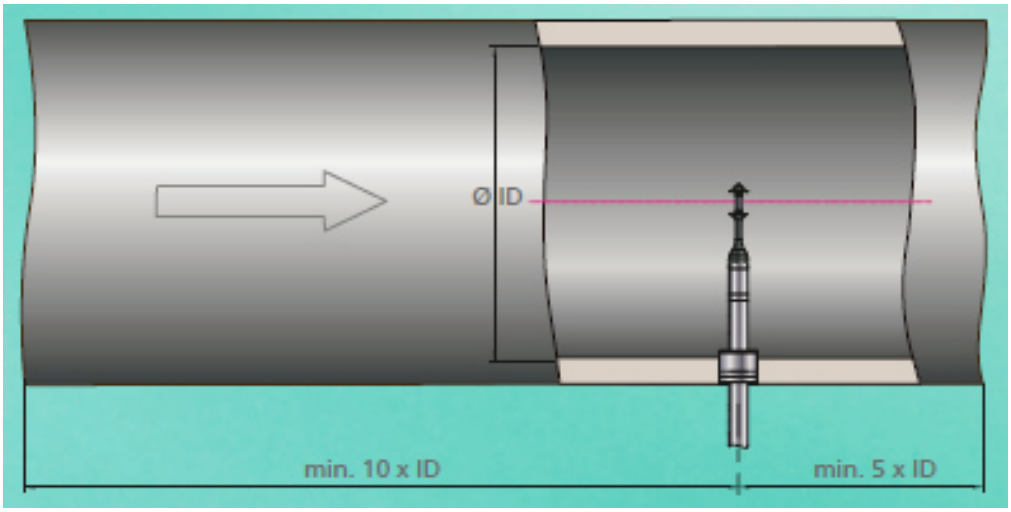


Abbildung 10 Hitzedrahtsonde verbaut im Rohr [Schmidttechnology]

Eine Hitzedrahtsonde besteht aus einem Heizwiderstand oder einer Heizfolie. In diesem Falle kommt das am Fraunhofer Institut für Bauphysik vorhandene Hitzedrahtanemometer, ein Fabrikat der Firma Schmidt GmbH, zum Einsatz. Diese Hitzedrahtsonden messen 360° axial, und müssen mittig im Rohrkanal (vgl. Abbildung 10) platziert werden. Sie verwenden das in Kapitel 2.3.1.2 beschriebenen Prinzip. Die thermische Anemometrie unterteilt sich in zwei Verfahren.

2.3.1.1 Thermische Anemometrie Abkühlverfahren(CCA)

Bei der thermischen Anemometrie, basierend auf dem Abkühlverfahren (engl. Constant-Current Anemometry), wird der Heizstrom eines Hitzedrahtes oder einer Heizfolie konstant gehalten. Gemessen wird die Temperaturabsenkung des Messwertaufnehmers durch die Strömung. Die Temperaturdifferenz durch die Abkühlung kann mithilfe der Messung des Widerstandes des Hitzedrahtes oder der Heizfolie bestimmt werden (vgl. Abbildung 11).

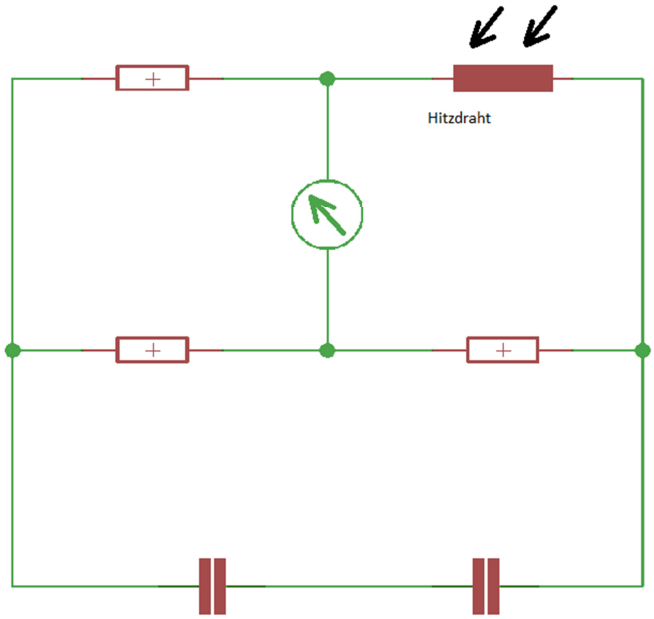


Abbildung 11 Abkühlverfahren

2.3.1.2 Thermische Anemometrie Konstanttemperaturverfahren (CTA)

Bei der thermischen Anemometrie, basierend auf dem Konstanttemperaturverfahren (engl. Constant-Temperature Anemometry), wird der Widerstand des Hitzedrahtes oder der Heißfolie durch Nachregeln der Heizspannung konstant gehalten. Heizspannung bzw. Heizstrom sind dann bei abgeglichener Brücke ein Maß für die Strömungsgeschwindigkeit. [Hoffman, 1999] [www.pressebox.de; Schmidttechnology]

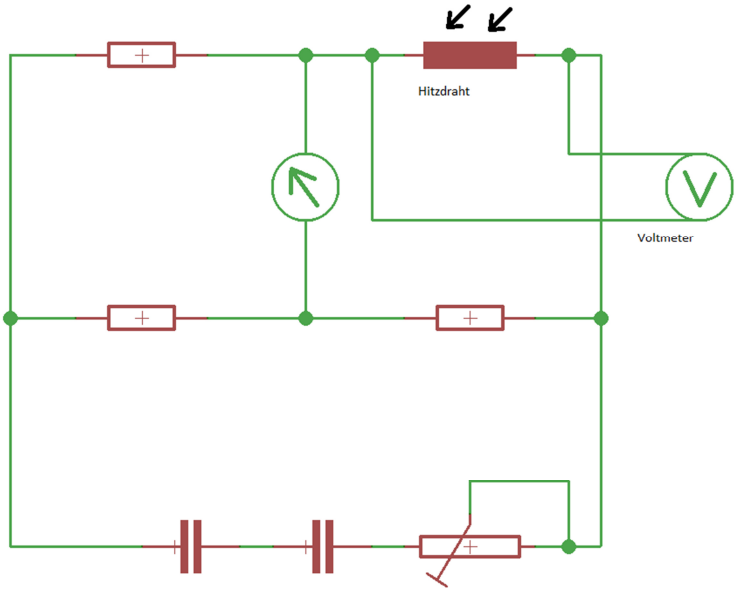


Abbildung 12 Konstanttemperaturverfahren

Die umgesetzte Leistung P des beheizten Messaufnehmers errechnet sich nach folgender Formel [Hoffmann, 1999].

$$P = (A + B * v^{0.5}) * (T_H - T_U)$$
$$I^2 * R_S = (A + B * v^{0.5}) * (T_H - T_U)$$

$T_H = Temperatur\ des\ Heizelements$
$T_U = Temperatur\ der\ strömenden\ Luft$
$R_s = ohmscher\ Widerstand\ des\ Drahts$
$A\&B = Konstanten, welche\ abhängig\ von\ der\ Messgeometrie\ sind$
$I = elektrischer\ Strom\ durch\ den\ Sensor$
$v = Strömungsgeschwindigkeit\ im\ Rohr$

2.3.2 Einfluss der Positionierung von Hitzedrahtanemometern auf das Messergebnis

Das zum Einsatz kommende Hitzedrahtanemometer vom Typ SS 20.500 soll auf seinen Messfehler bei nicht exakt mittiger Positionierung im Rohr untersucht werden. Hierzu musste zuerst die Überlegung angestellt werden, ob eine laminare oder turbulente Rohrströmung vorliegt. Bei laminarer Rohrströmung (Reynoldszahl < 2300) bildet sich über dem Rohrquerschnitt eine parabolische Geschwindigkeitsfunktion, welche in der Rohrmitte für r=0 am größten ist (vgl. Abbildung 13). [Gunt Hamburg]

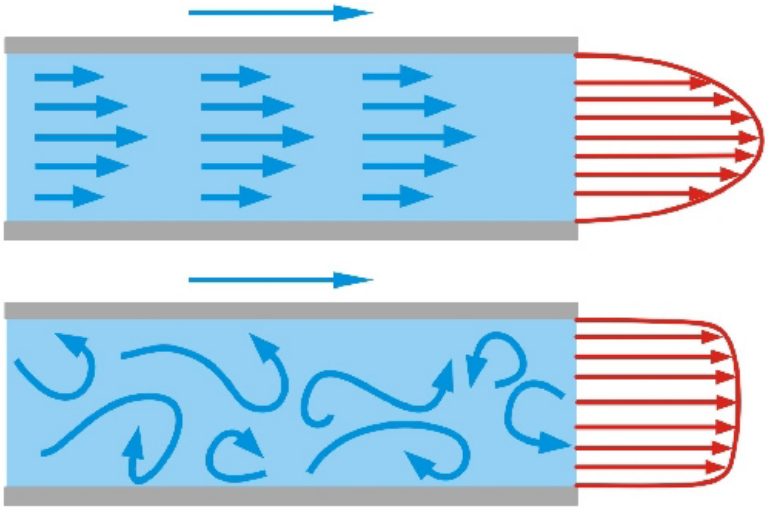


Abbildung 13 Geschwindigkeitsparaboloid bei laminarer Strömung (oben) und turbulenter Strömung (unten) [Gunt Hamburg]

Dieses Geschwindigkeitsprofil kann mit nachfolgender Formel berechnet werden:

$$v(r) = \frac{1}{4 * \theta} * \frac{\Delta p}{L} * (R^2 - r^2)$$

$v = Strömungsgeschwindigkeit$

$r = Radius$

$L = Länge\ des\ Rohrs$

$\theta = dynamische\ Viskosität$

$R = Rohrdurchmesser$

$\Delta p = Druckdifferenz\ zwischen\ Rohrbeginn\ und\ Rohrende$

Dieser Zusammenhang gilt jedoch nur bei laminarer Rohrströmung. Bei turbulenter Rohrströmung (Reynoldszahl >> 2300) hingegen entstehen in dem strömenden Medium diverse Wirbel, welche das Messergebnis beeinflussen können. Hierzu soll untersucht werden, welche Art von Strömung vorliegt. Mit einer zu erwartenden Strömungsgeschwindigkeit von 5 m/s bei 20 °C Lufttemperatur und einem Rohrradius von 50 mm soll folgende Berechnung angenommen werden [Junge, 2011]:

$$Re \approx \frac{4}{\pi} * \frac{\dot{V}}{\omega * d}$$

$$Re \approx \frac{4}{\pi} * \frac{5\frac{m}{s} * (0{,}05m)^2 * \pi}{15{,}2 * 10^{-6}\ \frac{m^2}{s} * 0{,}1m}$$

$$Re \approx 32894$$

Da die Reynoldszahl weitaus größer als 2300 ist, kann von einer turbulenten Strömung ausgegangen werden. Während bei einer laminaren Rohrströmung die Geschwindigkeit von außen nach innen stetig zunimmt, wächst bei der turbulenten Strömung die mittlere Strömungsgeschwindigkeit vom Rand zum Rohrinneren sehr rasch an, um dann über einen weiten Bereich des Innenraumes annähernd konstant zu bleiben. [Junge, 2011]

Diese Annahme soll mithilfe eines Versuches anhand einer geraden Rohrstrecke von 15 m überprüft werden. Hierzu soll die mittlere Strömungsgeschwindigkeit in der Rohrmitte gemessen werden. Anschließend soll das Hitzedrahtanemometer fortlaufend um 10 mm aus seiner Mittelposition nach oben wegbewegt werden.

Tabelle 1 Messfehler bei 2 V Ansteuersignal

Abweichung zur Mittelposition (Nulllage)[cm]	Ansteuersignal Thyristor [V]	Gemessene mittlere Strömungsgeschwindigkeit [m/s]	Differenz zur Nulllage [m/s]
0	2	2,79	0,00
1	2	2,73	0,06
2	2	2,61	0,18
3	2	2,54	0,24

Tabelle 2 Messfehler bei 4 V Ansteuersignal

Abweichung zur Mittelposition (Nulllage)[cm]	Ansteuersignal Thyristor [V]	Gemessene mittlere Strömungsgeschwindigkeit [m/s]	Differenz zur Nulllage [m/s]
0	4	7,96	0,00
1	4	7,80	0,16
2	4	7,46	0,50
3	4	7,22	0,75

Entgegen der Erwartung, dass die Strömungsgeschwindigkeit über dem Querschnitt konstant bleibt, hat sich herausgestellt, dass es speziell bei schnelleren Strömungsgeschwindigkeiten zu höheren Abweichungen durch einen Positionsfehler kommen kann. Dies ist darauf zurückzuführen, dass die sich bei turbulenter Strömung bildenden Wirbel in der Rohrmitte am schwächsten sind und dort das Messergebnis am wenigsten beeinflussen. [Schmidttechnology]

2.3.3 Differenzdruck Messprinzip

Bei der Ermittlung der Strömungsgeschwindigkeit aus dem gemessenen Differenzdruck besteht im Gegensatz zum Thermischen Messprinzip die Messvorrichtung aus zwei Komponenten.

2.3.3.1 Strömungsmessblende

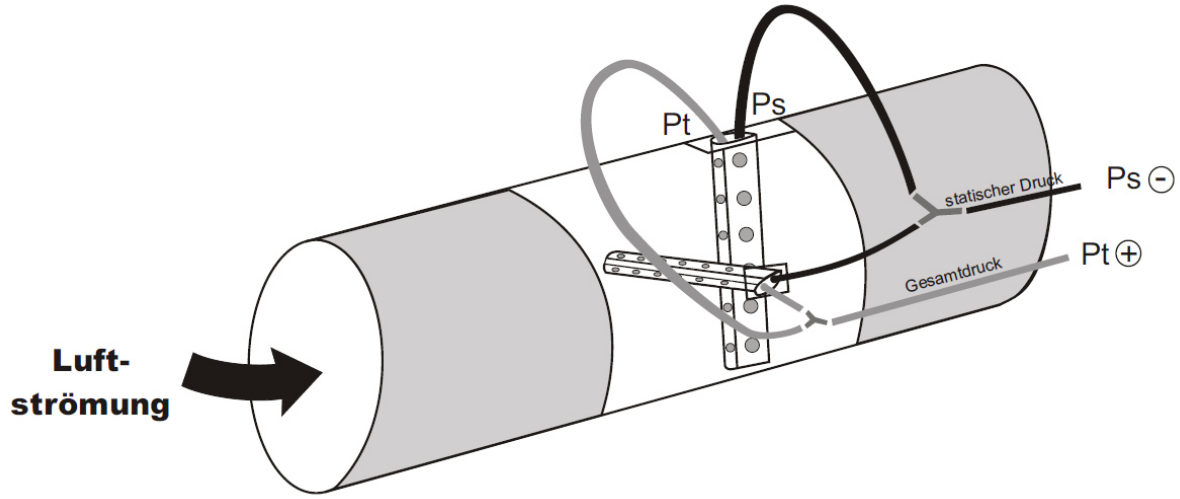


Abbildung 14 Strömungsmessblende [MDUA GmbH & CO. KG,]

Eine Strömungsmessblende wird in die bestehende Rohrstrecke verbaut und enthält zwei Messlanzen, welche entgegen der Strömungsrichtung den Gesamtdruck und senkrecht dazu den dazugehörigen statischen Druck erfassen.

Die Strömungsmessblende ist im Prinzip abgeleitet von einem klassischen Prandtlschen Staurohr, welches aus einer Kombination von einem Pitorohr zur Messung des Gesamtdrucks und einer Sonde zur Messung des statischen Drucks besteht [ELMTEC Ingenieursgesellschaft mbH]. Im Gegensatz zum Prandtlschen Staurohr wird hier eine komplette Messung über den gesamten Rohrquerschnitt vollführt. [Junge, 2011]

Ein weiterer Vorteil ist, dass im Gegensatz zur Hitzedrahtsonde oder einem einfachen Prandtlschen Staurohr keine systematischen Fehler durch den Einbau erfolgen können. Als Beispiel sei hier genannt, dass z.B. die Hitzedrahtsonde nicht exakt mittig im Rohr platziert ist, was einen Messfehler zur Folge hat (siehe Kapitel 2.3.2). Um einen Messfehler durch zu turbulente Strömungen zu vermeiden, ist es notwendig, eine Beruhigungsstrecke vor und hinter der Messblende vorzusehen. Vor der Strömungsmessblende ist daher eine Einlaufstrecke, welche mindestens das 5-fache des Rohrdurchmessers misst und eine Auslaufstrecke von mindestens dem 3-fachen des Rohrdurchmessers vorzusehen.

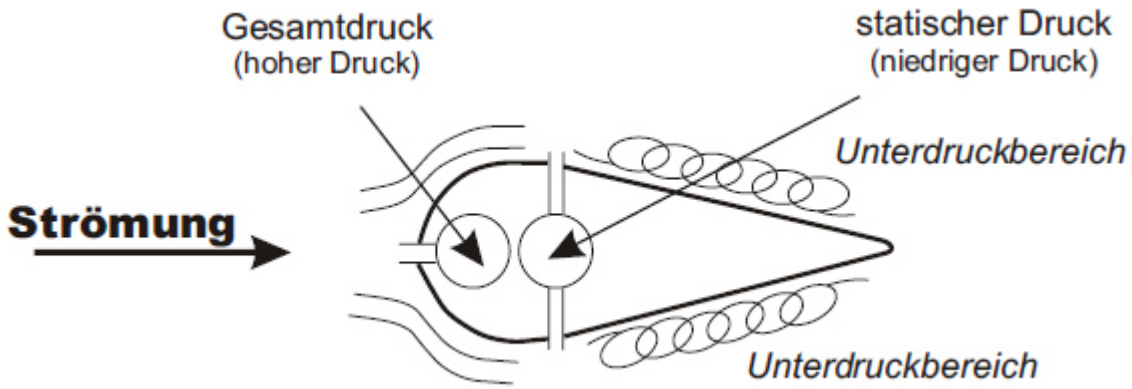


Abbildung 15 Umströmte Messlanze [MDUA GmbH & CO. KG,]

2.3.3.2 Differenzdrucksensor

Der verwendete Differenzdrucksensor der Firma Setra, Typ 267 MR6 arbeitet nach einem kapazitiven Druckmessprinzip. Der verwendete Sensor besitzt einen einstellbaren Messbereich, welcher linear auf ein 0-10 Volt- oder ein 0-5 Voltsignal umgesetzt werden kann. Der Sensor hat zwei Druckanschlüsse, wobei ein Anschluss den von der Strömungsmessblende erfassten Gesamtdruck misst und der andere Anschluss den statischen Druck. Aus den beiden mit druckbeaufschlagten Eingängen wird der Differenzdruck gebildet. Der so gemessene Differenzdruck oder dynamische Druck kann mit folgender Formel in eine Strömungsgeschwindigkeit umgerechnet werden [MDUA GmbH & CO. KG,]:

$$Luftgeschwindigkeit\ \left[\frac{m}{s}\right] = B_F * \sqrt{\frac{2 * p_d}{\rho}}$$

$B_F = Blendenfaktor\ 0{,}816$
$p_d = dynamischer\ Druck\ [pa]$
$\rho = Luftdichte\ [\frac{kg}{m^3}]$

2.4 Erfassung der Temperatur

Die in Kapitel 2.3.3.2 vorgestellte Formel zur Ermittlung der Strömungsgeschwindigkeit ist vom dynamischen Druck und der Luftdichte abhängig. Die Luftdichte ist wiederum abhängig vom statischen Druck und der Lufttemperatur. Folgende Formel gibt diesen Zusammenhang wieder:

$$\rho = \frac{p_s}{R_s * T}$$

Der statische Druck wird hier mit 93500 pa angenommen, was dem am Standort Holzkirchen vorherrschendem Luftdruck entspricht. Die spezifische Gaskonstante für trockene Luft wird mit $287{,}058\frac{J}{Kg*K}$ angenommen. Um eine Beeinflussung des Messergebnisses durch die Temperatur auszuschließen soll die Temperatur im Rohr durch einen vier Draht PT100 Messfühler erfasst werden.

Dieser wird an einem Meilhaus Messumformer vom Typ RED-MU-PT100-44K-V6 angeschlossen. Der Messumformer wird mit 24 V DC versorgt und setzt die Widerstandsänderung des PT100 Sensors in ein analoges 0-10 V Signal um.

2.4.1 Wahl des Messsystems

Um sich auf ein Messsystem festlegen zu können, bedarf es einer Abwägung der einzelnen Vor- und Nachteile der beiden Messprinzipien.

Tabelle 3 Punktebewertung des Messsystems

Kriterien	Messprinzip	
	Differenzdruck	Hitzedrahtsonde
Kosten	+	0
Gefahr durch systematischer Fehler	++	0
Messgenauigkeit	+	+
Trägheit des Sensors bei Änderung der Strömungsgeschwindigkeit	+	++
Länge der Beruhigungsstrecke	+	0
Anzahl der erfassbaren Messgrößen	+	++
Anzahl der Messbereiche	++	0
Untere Strömungsnachweisgrenze	+	++
Ergebnis gesamt	10	8
++=sehr gut(2), +=gut(1), 0=neutral(0)		

Schlussendlich wurde sich für die Verwendung des Messprinzips entschieden, welches die Strömungsgeschwindigkeit über den Differenzdruck ermittelt. Ausschlaggebend war hier vor allem die geringere Gefahr einen systematischen Fehler durch den Einbau zu begehen sowie die kürzere Beruhigungsstrecke. In Abbildung 16 ist ein Vergleich zwischen den zeitlichen Mittelwerten der beiden Sensortypen, bei verschiedenen Ansteuersignalen des Thyristorstellers zu sehen.

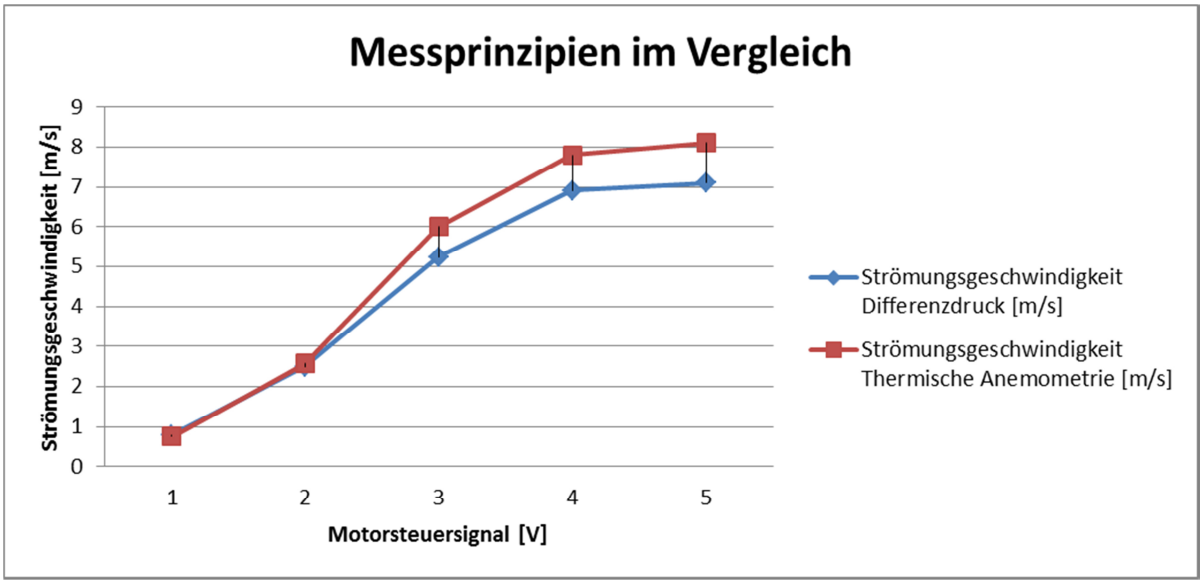


Abbildung 16 Vergleich der beiden Sensortypen bei jeweils gleichem Ansteuersignal

2.5 Mess-Steuerplatine

Um eine Kommunikation zwischen dem Mikrocontroller, der Leistungselektronik und der Sensorik zu ermöglichen bedarf es einer Schnittstelle. Obwohl die Sensorik des Differenzdrucksensors auf 0-5 V umgestellt werden kann und der Leistungssteller nur einen effektiven Ansteuerbereich von 0-5 V besitzt, soll der Prüfstand so ausgelegt werden, dass er der gängigen Industrieschnittstelle von 0-10 V entspricht. Dies ermöglicht unabhängiger von dem gerade verwendeten Lüfter zu sein. Eine einheitliche Schnittstelle ist hier sinnvoll, da der Lüfter so gegen einen größeren Lüfter, oder sogar theoretisch gegen einen Heizer ausgetauscht werden kann.

Um mit der Leistungselektronik kommunizieren zu können muss das pulsweiten modulierte Signal des Mikrocontrollers geglättet und mithilfe eines Operationsverstärkers auf 0-10 V verstärkt werden. Zugleich versorgt die Steuerplatine, die Sensorik und den Operationsverstärker mit 24 V Versorgungsspannung.

2.5.1 Einlesen der Sensorik

Das Einlesen des Sensorsignals ist in Abbildung 17 zu sehen. Über folgenden Zusammenhang kann das Sensorsignal in ein für den Mikrocontroller einlesbares Signal geteilt werden.

$$\frac{U_1}{U_2} = \frac{R_1}{R_2}$$

Der Spannungsteiler besitzt einen Spindeltrimmer um die Bauteil Toleranz der Widerstände R1 und R2 kompensieren zu können

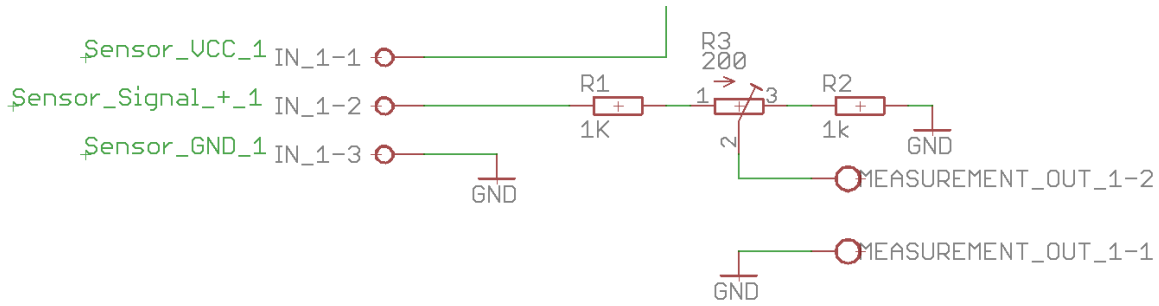


Abbildung 17 Spannungsteiler

Bevor die Platine verwendet werden kann, muss der Spannungsteiler mithilfe einer Präzisionsspannungsquelle abgeglichen werden.

2.5.2 Operationsverstäker

Um mit der Leistungselektronik (Thyristorsteller) kommunizieren zu können, muss das pulsweitenmodulierte Signal des Mikrocontrollers geglättet und mithilfe eines Operationsverstärkers auf 0-10 V verstärkt werden [Mikrocontroller.net, 2017]. Hier wird ein IC (Integrated Circuit) von der Firma Texas Instruments vom Typ LM358p verwendet. Der IC beinhaltet zwei Operationsverstärker, welche voneinander unabhängig beschalten werden können. Der Operationsverstärker wird als nichtinvertierender Verstärker mit einem Verstärkungsfaktor von 2 beschalten.

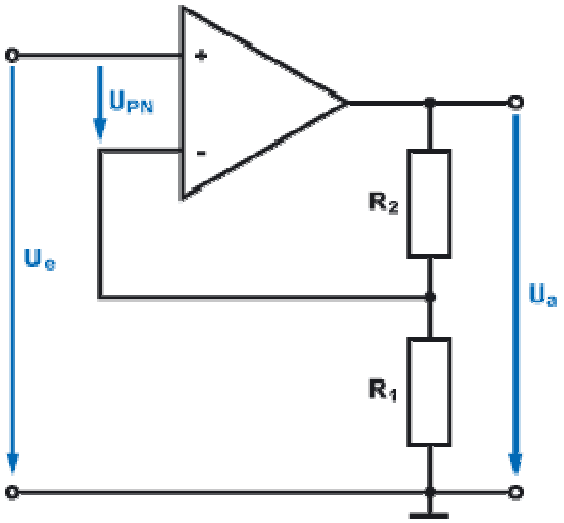


Abbildung 18 Nichtinvertierender Operationsverstärker (Schnabel)

Die Eingangsspannung U_e ist das geglättet PWM Signal vom Mikrocontroller kommend. Aufgrund des niederohmigen Ausgangsverhaltens des Operationsverstärkers, lässt sich diese Schaltung als Gleichspannungsquelle für das Steuersignal des Thyristorstellers benutzen [Schnabel]. Die Verstärkung errechnet sich über folgenden formeltechnischen Zusammenhang [Schnabel]:

$$V = 1 + \frac{R_2}{R_1}$$

2.5.2.1 Pulsweiten Modulation

*„Bei der **Pulsweitenmodulation** (engl. Pulse Width Modulation, abgekürzt **PWM**) wird das Verhältnis zwischen der Einschaltzeit und Periodendauer eines Rechtecksignals bei fester Grundfrequenz variiert. Das Verhältnis zwischen der Einschaltzeit t_{ein} und der Periodendauer $T = t_{ein} + t_{aus}$ wird als das Tastverhältnis p bezeichnet. (laut DIN IEC 60469-1: Tastgrad) (engl. Duty Cycle, meist abgekürzt DC, nicht zu verwechseln mit Direct Current = Gleichstrom)“* [Mikrocontroller.net, 2017]. Ein PWM-Signal besitzt immer dieselbe Frequenz, jedoch verändert sich das Verhältnis von Einschaltdauer zu der Summe aus Ein- und Ausschaltdauer.

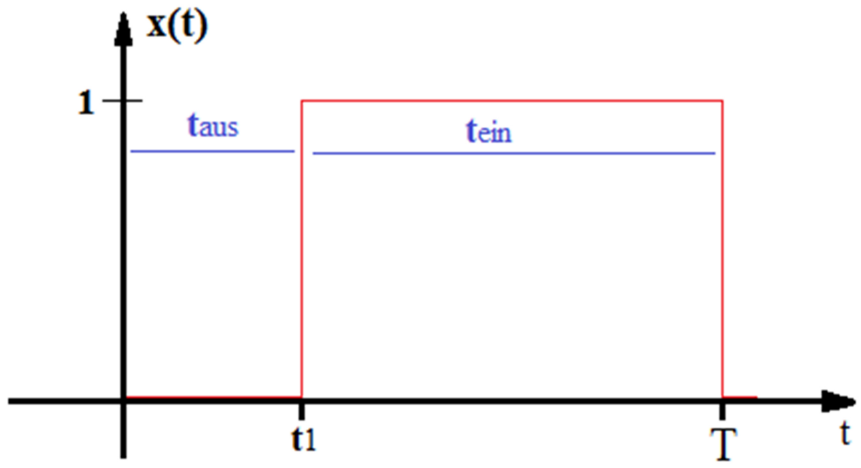


Abbildung 19 Verhältnis zwischen Einschaltzeit und Ausschaltzeit [Mikrocontroller.net, 2017]

Der verwendete Mikrocontroller besitzt für seine PWM Ausgänge eine Auflösung von 8 Bit (256 Stufen) und eine Amplitude von 5V. Abbildung 20 zeigt das PWM Signal des Mikrocontrollers, bei einem Tastverhältnis p von 50% (128 Stufen). Hieraus ergibt sich nach folgender Formel mit den abgelesenen Werten aus Abbildung 20 die gemittelte effektive Spannung [Mikrocontroller.net, 2017]:

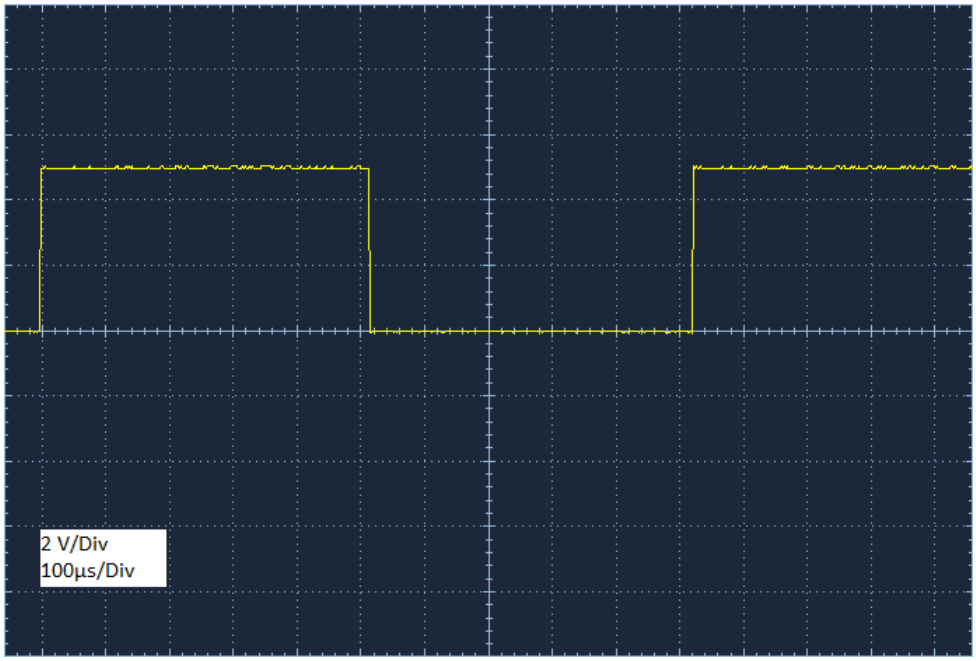


Abbildung 20 Pulsweiten Moduliertes Signal Tastgrad 50%

$$U_m = V_{CC} * \frac{t_{ein}}{t_{ein} + t_{aus}}$$

$$U_m = 5V * \frac{(520\mu s)}{520\mu s + 500\mu s}$$

$$U_m = 2{,}54V$$

2.5.2.2 Dimensionierung des RC-Filters

Um das pulsweiten modulierte Stellsignal des Mikrocontrollers in eine geglättete Gleichspannung zu wandeln, welche anschließend durch den Operationsverstärker verstärkt wird, bedarf es einer Glättung durch einen Tiefpassfilter (Abbildung 21). Bei der Dimensionierung eines RC-Filters zur Glättung des vom Mikrocontroller kommenden PWM-Signals, gilt es einen Kompromiss aus 2 Eigenschaften zu finden. Zum einen ist es wichtig, dass der Mittelwert des zu glättenden PWM-Signals möglichst schnell erreicht wird. Bis sich der Kondensator voll aufgeladen hat vergeht die Zeit $t_{neu} = 5 * R * C$. Dieser Zusammenhang wird aus der sich bildenden Zeitkonstante $\tau = R * C$ gebildet, da ein Kondensator eine Auflade Zeit von ungefähr $5 * \tau$ Sekunden besitzt.

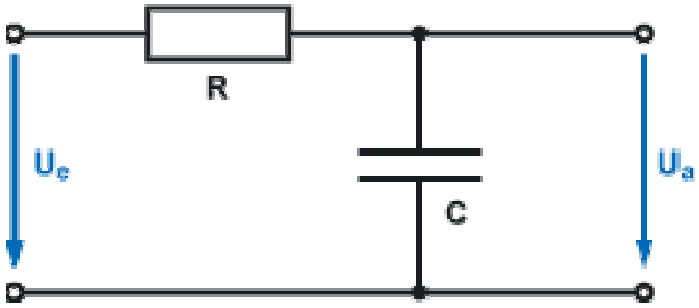


Abbildung 21 Tiefpassfilter [Schnabel]

Die zweite Eigenschaft die der RC Filter besitzen muss ist, dass das PWM Signal ausreichend geglättet wird und ein geringer Restripple V_{Ripple} auf der so entstehenden Gleichspannung bleibt. Hierzu muss bestimmt werden, was ein vertretbarer Restripple ist und wo dieser am stärksten auftritt. Der PWM Ausgang des Mikrocontroller besitzt eine Auflösung von 8 Bit, die PWM Frequenz lässt sich nach folgender Formel berechnen [Albers, 2010].

$$f_{PWM} = \frac{f_Q}{2^{res} * prescale}$$

$$f_{PWM} = \frac{8\ MHz}{(2^8 * 32)}$$

$$f_{PWM} = 976{,}56\ Hz$$

$f_q = Quarzfrequenz$
$res = Auflösung$
$prescale = Taktfrequenzteiler\ des\ Timers$

Die Angaben für f_q, res und $presccale$ können dem Datenblatt des verwendeten Mikrocontrollers entnommen werden [Atmel, 2015]. Bei einem PWM Tastverhältnis von 50 % tritt der stärkste Ripple auf, daher soll dieses Tastverhältnis für die Dimensionierung des RC-Glieds dienen [Texas Instruments, 1998] [Mikrocontroller.net, 2017]. Bei einer Auflösung von 8 Bit und 5 V PWM Amplitude löst der Mikrocontroller sein PWM Signal mit einer Feinheit von $\frac{5V}{256\ Stufen} = 19{,}5\ mV$ auf, der Restripple sollte daher unter 19,5 mV liegen. Mithilfe folgender Formel [Albers, 2010] wurde der Restripple berechnet:

$$\Delta U_{ss} = \frac{4 * U_0}{\pi * \sqrt{1 + C^2 * R^2 * \left(2 * \pi * f_{pwm}\right)^2}}$$

$$\Delta U_{ss} \frac{4 * 5V}{\pi * \sqrt{1 + (10\mu F)^2 * (5600\ \Omega)^2 * (2 * \pi * 976{,}56\ Hz)^2}}$$

$$\Delta U_{ss} = 18{,}5mV$$

Nach der obigen Formel $t_{neu} = 5 * R * C$ ergibt sich hier eine neue Aufladezeit von $t_{neu} = 0{,}28\ s$. Dieses Verhalten sollte am 0-10 V Ausgang mithilfe eines Oszilloskops überprüft werden. Die Platine ist hier im unbelasteten Zustand betrachtet worden (nicht am Thyristorsteller betrieben).

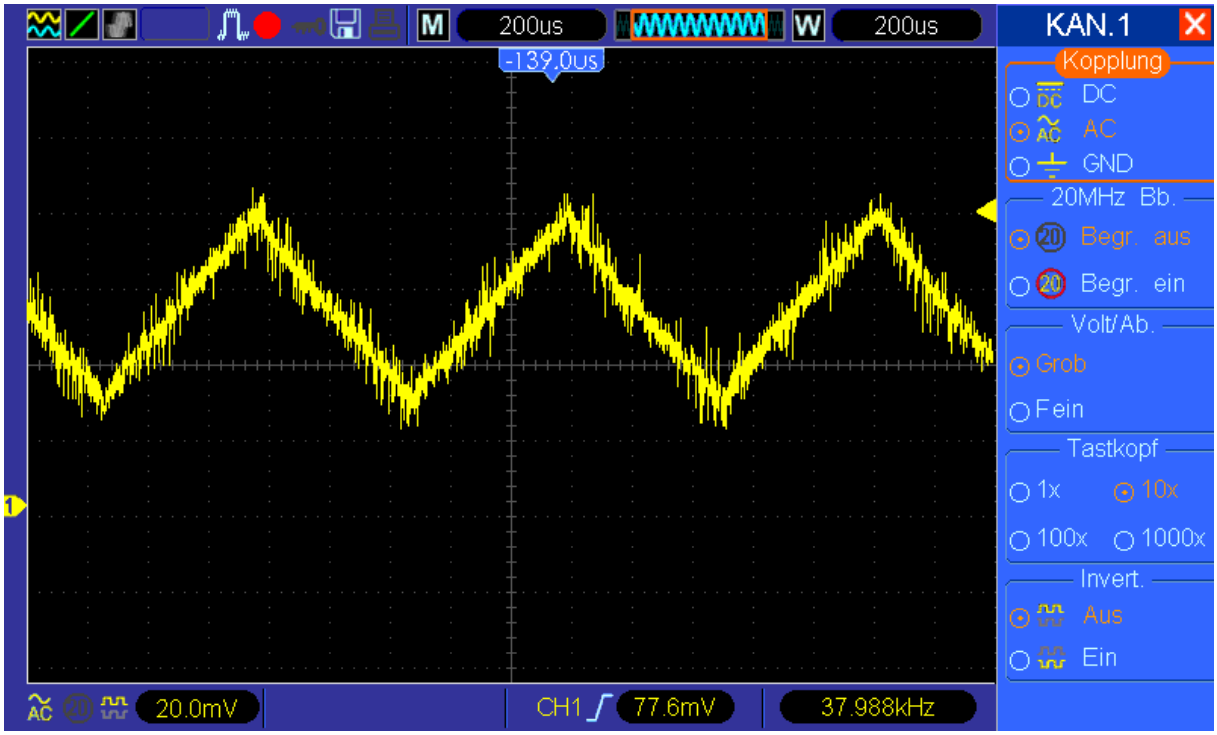


Abbildung 22 PWM-Signal nach der Glättung mit 10µF Kondensator

Das geglättete PWM-Signal lieferte ein unbefriedigendes Ergebnis, da ΔU_{ss} hier deutlich über 19,5 mV liegen. Der hier verbleibenden Restripple von ca. 60 mV hat zwar kein Einfluss auf die Ansteuerung des Thyristors, da der Motor ein zu träges System darstellt, dennoch wurde der Restripple nochmals überarbeitet. Hierzu wurde mit obiger Formel noch einmal eine Berechnung für einen 100µF Kondensator angestellt, was einen maximalen Restripple von 1,8 mV zu Folge haben sollte. Nach einer erneuten Überprüfung mit dem Oszilloskop wurde ein zufriedenstellendes Ergebnis erzielt, was jedoch nicht dem der Formel entspricht (Abbildung 23). Es konnte zwar kein exakter Zusammenhang zwischen Formel und Realität hergestellt werden, jedoch liegt der verbleibende Restripple deutlich unter 19,5 mV und soll daher beibehalten werden, da dieser irrelevant ist. Dass die Formel keine exakten Werte liefert wurde nicht weiter untersucht, es wird vermutet dass die Formel lediglich als grobe Abschätzung verwendet werden kann.

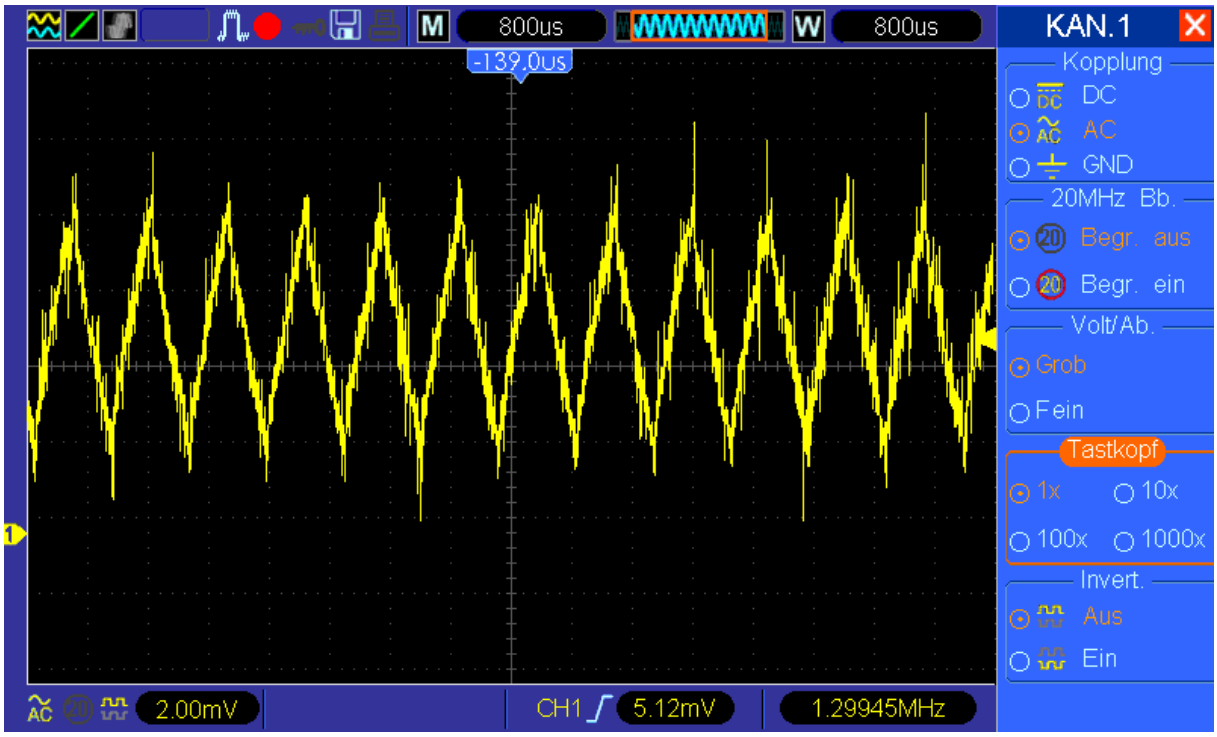


Abbildung 23 PWM-Signal nach der Glättung mit 100µF Kondensator

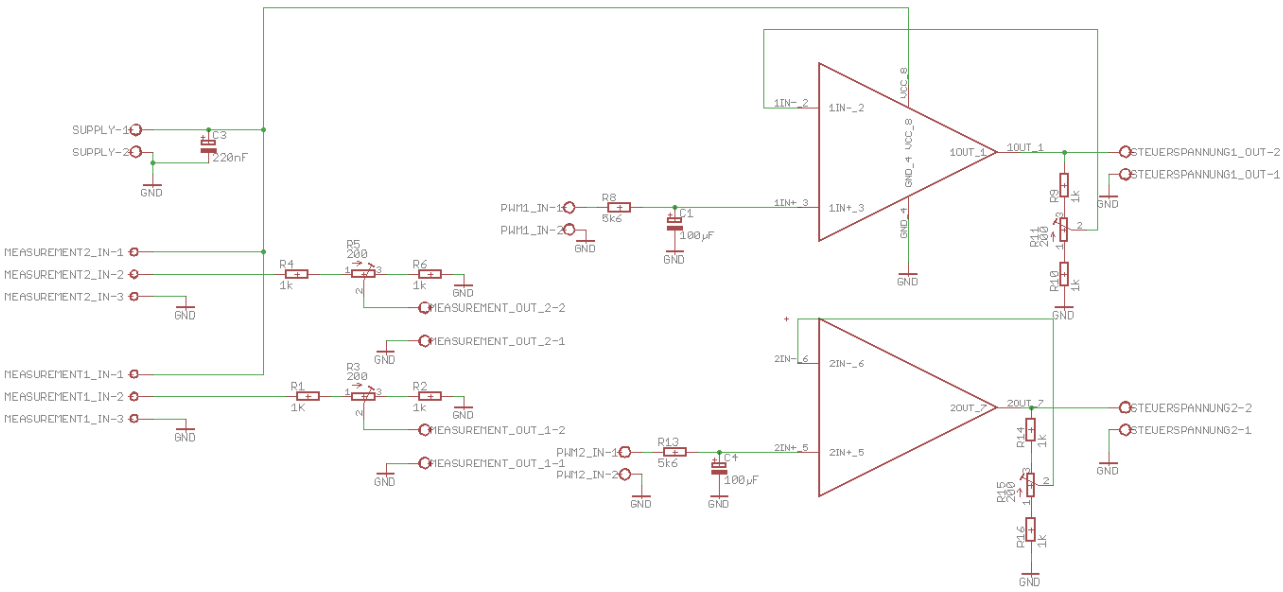


Abbildung 24 Fertiger Schaltplan der Platine

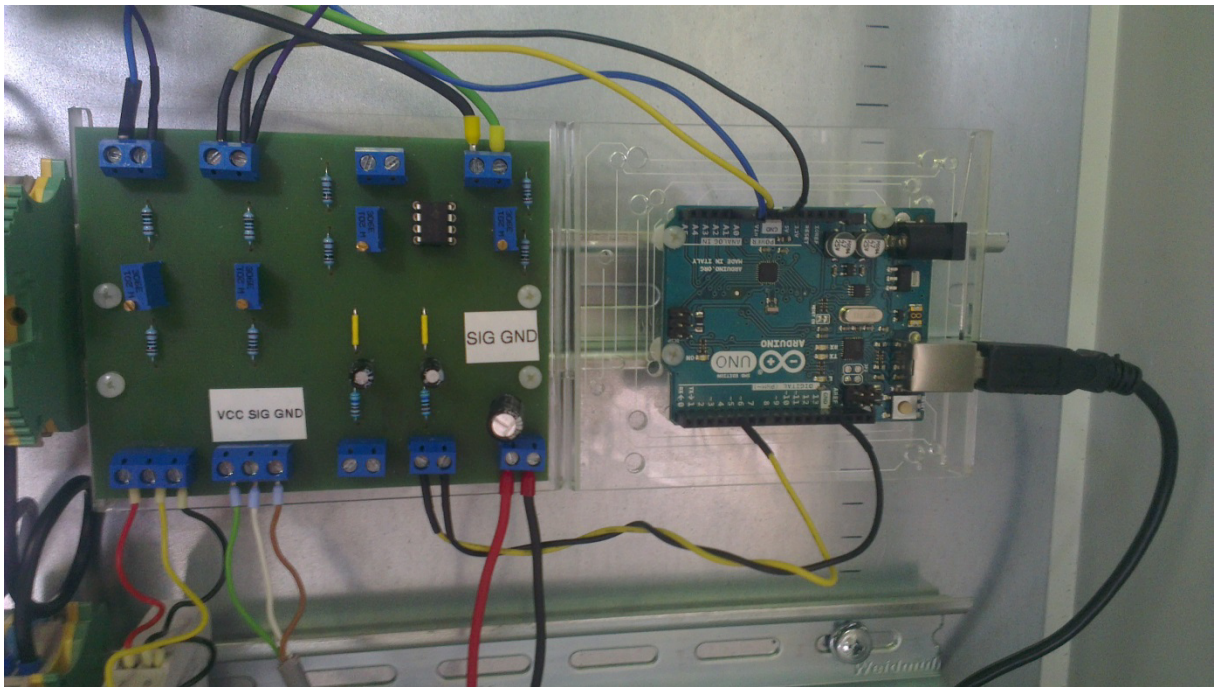


Abbildung 25 Angeschlossene Messsteuerplatine mit Arduino

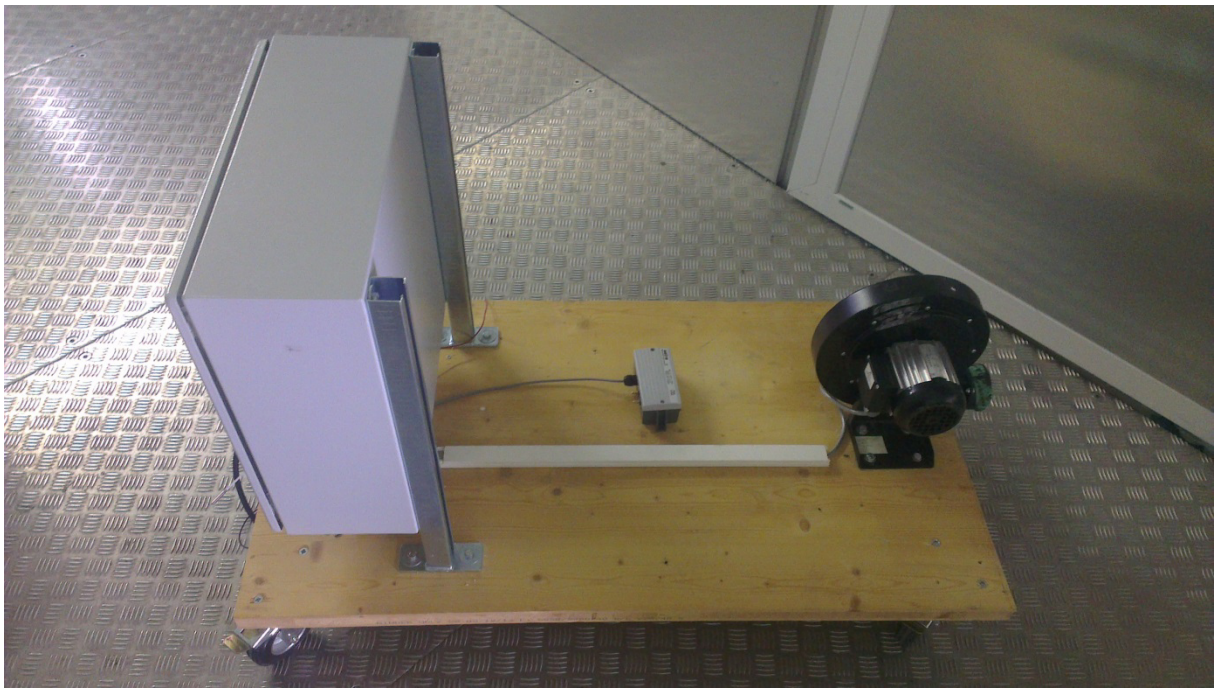


Abbildung 26 Lüfter mit Differenzdrucksensor und Schaltschrank

2.6 Mikrocontroller

Bei der Wahl des Mikrocontrollers galt es zu beachten, dass dieser über mindestens einen PWM Ausgang und einen Analogen Eingang verfügt und genügend Flash-Speicher besitzt. Da am Fraunhofer Institut für Bauphysik die Plattform Arduino schon bekannt war, sollte diese auch zum Einsatz kommen. Der Vorteil gegenüber einem eingebetteten System und der Programmierung in z.B. „Atmel Studio“ besteht darin, dass auch Personen mit wenig Programmiererfahrung leicht Änderungen am Prüfstand vornehmen können.

Der zur Verwendung kommende Arduino ist ein Arduino Uno, dieser basiert auf einem Atmega328p Mikrocontroller. Es wurde sich bewusst gegen einen viel leistungsstärkeren Arduino Due entschieden, da dieser nicht mit allen Programmierbefehlen und Bibliotheken, welche die Arduino IDE zur Verfügung stellt kompatibel ist.

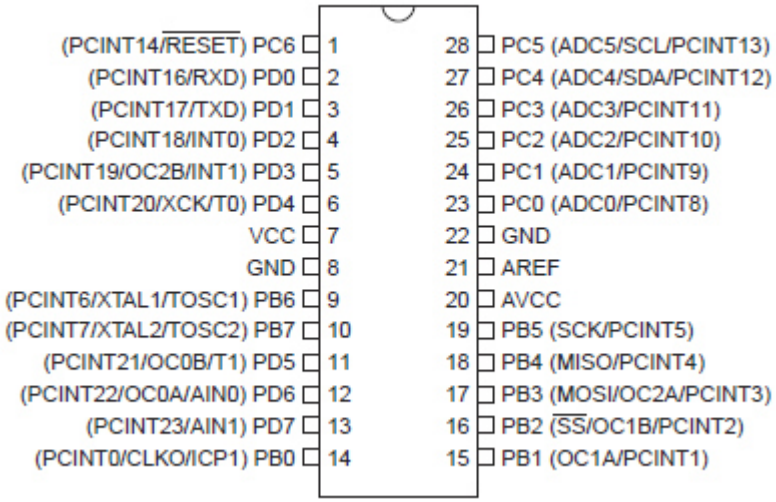


Abbildung 27 Atmega328p Pinbelegung [Atmel, 2015]

Der Atmega328p besitzt 6 Analoge Eingänge mit einer 10-Bit Auflösung und 14 I/O- Pins von denen 6 Stück zur Pulsweitenmodulation genutzt werden können. Der Atmega328p ist ein 8-Bit Mikrocontroller. Jeder Rechenschritt eines Mikrocontrollers wird durch einen Taktimpuls von einem externen oder internen Oszillator eingeleitet. Dieser bestimmt zu einem Großteil die Geschwindigkeit eines Mikrocontrollers. Der Atmega328p ist so zum Beispiel in der Lage mit einem Taktsignal eine Folge von 8 Bits (z.B. 10100101) zu verarbeiten. Bei einer verfügbaren Taktfrequenz von 16 MHz kann der Atmega328p einen Datensatz von 8-Bit 16 Millionen Mal pro Sekunde verarbeiten. Dies hängt jedoch auch stark von den verwendeten Befehlen ab und soll nur zum Grundverständnis dienen, da einige Befehle mehr als einen Taktzyklus in Anspruch nehmen. [Gaicher, 2015] [Atmel, 2015] [Arduino]

2.6.1 Speicher

Der Atmega328p besitzt drei verschiedene Speichertypen Flash, SRAM und EEPROM (Programm-, Statischer- und Festwertspeicher).

Der Flashspeicher enthält den Programmcode, welcher sequentiell aus dem Flashspeicher abgerufen und verarbeitet wird. Der Flashspeicher ist ein nichtflüchtiger Speicher, was bedeutet, dass er nach wegnehmen der Versorgungsspannung erhalten bleibt. Der Flashspeicher des Atmega328p ist in der Lage Programme bis zu einer Größe von 32kB zu speichern.

Ein SRAM (Static Random Access Memory) ist ein flüchtiger Speicher, welcher Variablenwerte enthält. Der SRAM besitzt eine Größe von 2kB.

Das EEPROM (Electrically Erasable Programmable Read Only Memory) ist ebenfalls ein nichtflüchtiger Speicher und dient dazu Grundeinstellungen zu speichern. Darüber hinaus wäre er theoretisch in der Lage, vom Prüfstand erfasste Messdaten bis zu einer Größe von 1kB zu speichern, welche nachher wieder ausgelesen werden könnten. [Atmel, 2015]

2.6.2 Schnittstellen

Der Arduino UNO bzw. der Atmega328p besitzt eine Vielzahl von Schnittstellen, welche für die unterschiedlichsten Aufgaben verwendet werden können. Als Beispiel sei hier die Kommunikation mit mehreren Temperatursensoren genannt, welche alle an einer gemeinsamen Datenbusleitung angeschlossen sind.

2.6.2.1 I²C

Die I²C (Inter-Integrated-Ciruit) Schnittstelle wurde von Philips Semiconductors (heutzutage NXP) 1980 entwickelt. Das Besondere an dieser Schnittstelle ist, dass Sie mit nur 2 bidirektionalen Kommunikationsleitungen auskommt (ohne Masse und Versorgung). Die erste Leitung wird als Serial Data (SDA) und die zweite Leitung als SCL (Serial Clock) bezeichnet [mborchers, 2008]. An diesen Bus können verschiedene Peripheriebausteine angeschlossen werden, wie z.B. Temperatursensoren, A/D-Wandler oder LCD-Treiber.

2.6.2.2 UART/USART

Die USART (Universal Synchronus/Asynchronous Receiver Transmitter) wurde bereits in den 60er Jahren entwickelt. Über die UART-Schnittstelle kann ein Mikrocontroller über die RS-232-Schnittstelle mit einem Computer, oder anderem Gerät kommunizieren. Der Arduino UNO besitzt einen integrierten UART/USB Converter, welcher eine Kommunikation über USB ermöglicht.

Bei der Übertragung via UART sendet der Mikrocontroller einen Datenrahmen bestehend aus einem Start-Bit, 5-9 Datenbits und einem Stop-Bit. Optional existiert noch ein Parity-Bit zum Erkennen von Übertragungsfehlern. [Gaicher, 2015]

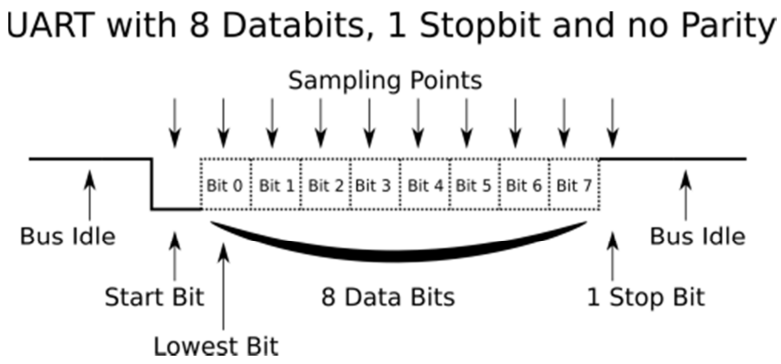


Abbildung 28 UART Kommunikation [myHDL, 2012]

2.6.2.3 Serial Peripheral Interface

Das Serial Peripheral Interface (SPI) ist ein keiner Norm unterliegendes Bus-System, welches von Motorola entwickelt wurde. Dieser Bus ermöglicht eine synchrone serielle Datenübertragung nach dem Master-Slave-Prinzip. Dieser Bus besteht im Wesentlichen aus den drei Leitungen MOSI (Master Out Slave In), MISO (Master In Slave Out) und SCLK (Serial Clock). Für jeden angeschlossenen Slave z.B. ein Temperatursensor wird noch zusätzlich eine Slave Select Leitung (CE) benötigt. Mit dieser Leitung kann der Master einen gewünschten Slave zur Kommunikation selektieren. Die Verschaltung von mehreren Slaves kann entweder in einer Sternverbindung, oder in einer Kaskadierung geschehen. Der Vorteil der Kaskadierung besteht darin, dass für jeden einzelnen Slave die Slave Select Leitung entfällt, jedoch ist dies mit erhöhtem Programmieraufwand und einem verlangsamten Datentransfer verbunden [Gaicher, 2015].

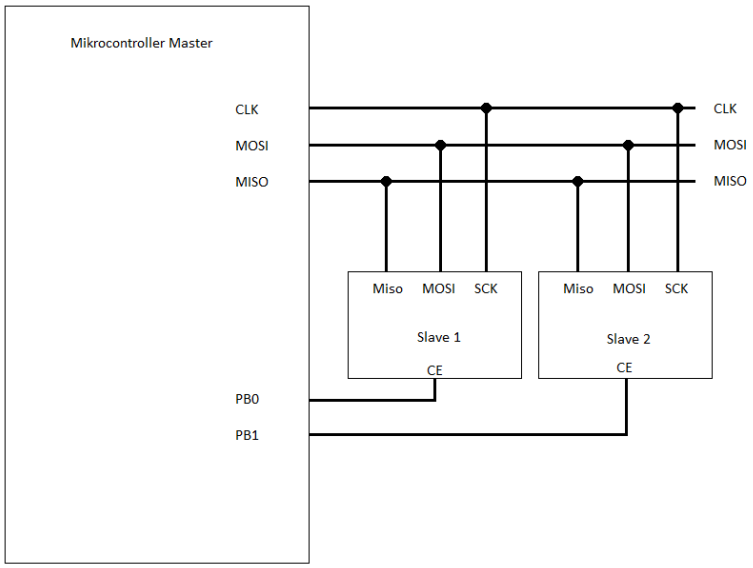


Abbildung 29 Sternverbindung mehrerer Slaves

3 Regelungstechnik

Technische Systeme sollen häufig so beeinflusst werden, dass bestimmte veränderliche Größen wie (z.B. Temperatur oder Drehzahl) ein vorgeschriebenes Verhalten aufweisen. In einem einfachen Fall wie es hier vorliegt soll eine technische Größe die Strömungsgeschwindigkeit im Rohr konstant gehalten werden, obwohl Störungen auf das System einwirken. Im Allgemeinen sind diese Aufgaben mit Steuerungen und Regelungen umsetzbar.

Im Falle einer Steuerung würde der Lüfter mit einer zuvor ermittelten Drehzahl drehen, um einen gewünschten Volumenstrom zu erzeugen. Würde hier aber beispielsweise auf die Systemstrecke bestehend aus dem Lüfter Prüfstand und der Messstrecke eine Störung einwirken, könnte diese nicht ausgeglichen werden. Eine Störung könnte beispielsweise die plötzliche Verengung einer Irisblende sein, welche zu einer Verkleinerung des Rohrquerschnitts führen würde und welche in einem Ansteigen der Luftgeschwindigkeit resultiert.

Eine Regelung hingegen ist in der Lage automatisch auf solche Störeinflüsse zu reagieren. Bei oben genanntem Beispiel würde eine Regelung die Drehzahl des Lüfters verringern, um die vorgegebene Luftgeschwindigkeit konstant zu halten. Eine Regelung besteht immer aus einer Messung des Istwertes, einem Vergleichen des Istwertes mit dem Sollwert und gegeben falls nachstellen um den Istwert dem Sollwert wieder anzunähern. Man spricht hierbei von einem geschlossenen Regelkreis vgl. Abbildung 9.

3.1 Systemidentifikation

„Nach der Methode der Modellgewinnung unterteilt man mathematische Modelle in theoretische oder analytische sowie empirische oder experimentelle Modelle. Analytische Modelle sind aus physikalischen Gesetzen abgeleitet, aus Messungen am Prozess werden experimentell gewonnene Modelle ermittelt. Parametrische Modelle sind in Form von Differenzial- oder Differenzengleichungssystemen oder als Übertragungsfunktion darzustellen, nicht parametrische Modelle können mit Kurven, Wertetabellen oder Antwortfunktionen wie zum Beispiel der Sprungantwortfunktion oder der Ortskurve der Frequenzgangfunktion, gebildet werden.“ [Lutz, 2014] Übertragungsfunktionen besitzen mindestens einen Signaleingang und mindestens einen Signalausgang. Diese Übertragungsfunktion bildet einen mathematischen Zusammenhang zwischen Ein- und Ausgangssignal eines Systems.

In dieser Bachelorarbeit wurde das Systemverhalten experimentell über das aufschalten eines Testsignal bestimmt. Durch Messungen des Ein- und Ausgangsverhalten wurde mithilfe der Software Matlab eine Übertragungsfunktion gewonnen. Als Testsignal wurde ein Sprung von definierter Höhe auf das System aufgebracht und anschließend das Ausgangsverhalten gemessen und mithilfe der Systemidentification Toolbox ein nichtlineares zeitinvariantes Modell gebildet. Bei einem zeitinvarianten Modelle ändern sich die Modellparameter nicht mit der Zeit. Ein Beispiel für ein zeitvariantes Modell wäre hierfür, dass sich die Lufttemperatur und damit die Dichte durch die Temperatur des Lüfter Motors ändern würde. Da dieser Einfluss so gering ist, ist er nicht relevant. Hierbei gebildete Modelle können zwei verschiedene Formen annehmen. Zum einen werden hier „Blackbox-Modelle“ unterschieden und zum anderen „Glassbox-Modelle“. Bei Blackbox-Modellen wird das Modell so entworfen, dass es lediglich das Systemverhalten nachahmt, ohne mit dessen mathematischen Hintergrund übereinzustimmen. Bei „Glasbox-Modellen“ wird das Modell so entworfen, dass die mathematische Systemstruktur mit all ihren Wirkungselementen auf das Modell

abgebildet wird [Ottens, 2008]. Zur Aufnahme der Ein- und Ausgangssignale wurde eine für den Prüfstand programmierte GUI (siehe Kapitel 5.2) benutzt, welche das Ein- und Ausgangsverhalten des Systems vom Mikrocontroller sekundenweise erfasst. Die Systemidentifikation wurde mit Sprungantwortverläufen durchgeführt, welche an einer Rohrstrecke von 15 m aufgenommen wurden

Wie bereits beschrieben muss ein System mit einem Testsignal angeregt werden, um das Ein- und Ausgangsverhalten bestimmen zu können. Hierzu bieten sich verschiedene Testsignale an.

Impulsantwort: Die Impulsantwort (Diracimpuls) eines Systems ist die, bei der als Eingangssignal ein Sprung zum Zeitpunkt t=0 auf das System gebracht wird. Für alle anderen Zeiten ist die Sprunghöhe gleich 0.

Sprungantwort: Die Sprungantwort eines Systems ist die, bei der als Eingangssignal ein Sprung von definierter Höhe zum Zeitpunkt t=0 auf das System gegeben wird, welcher auf dem System für Zeiten t > 0 verbleibt.

Frequenz Antwort: Bei der Frequenzantwort wird das System mit einem in Amplitude und Frequenz fest definierten, sinusförmigen Signal beaufschlagt. Die Systemantwort wird ebenso ein sinusförmiges Signal mit der gleichen Frequenz wie das Eingangssignal sein. Jedoch ist dieses nicht gleich in der Phase und der Amplitude wie das Eingangssignal. Die Frequenzantwort wird meist in zwei Diagrammen dargestellt, welche die Amplitudenänderung und die Phasenänderung eines Systems in Abhängigkeit von der Signalfrequenz zeigen. Dieses Diagramm ist als Bode-Diagramm bekannt. [Ljung, 2000]

3.1.1 Systemidentifikation nach Schwartz

Das Fraunhofer Institut für Bauphysik die Software Matlab nicht zur Verfügung steht, wurde zuerst versucht eine Systemidentifikation mithilfe eines nicht rechnergestützten Verfahrens durchzuführen. Hintergedanke war hierbei, dass für die Zukunft ein Verfahren zur Streckenidentifikation genutzt werden kann, welches ohne Softwareunterstützung auskommt. Das folgende Verfahren der sogenannten „Zeitprozentkennwerte“ kann für aperiodisch proportional wirkende Übertragungssysteme mit Verzögerung höherer Ordnung und gegebenfalls für Totzeitverhalten (Pt_nTt)-Systeme) genutzt werden. Das Verfahren nach Schwartz kann auch für (It_nTt) Systeme genutzt werden und stellt eine Verbesserung des Wendetangentenverfahrens dar. Da hier keine Einflüsse durch Zeichenungenauigkeit stattfinden. Mit dem Verfahren nach Schwartz sollten größtenteils alle Systeme am Fraunhofer Institut für Bauphysik abgedeckt werden können, da diese überwiegend aus Temperaturregelkreisen (Pt_nTt) bestehen.
Das Verfahren nach Schwartz nähert mithilfe der gemessenen Sprungantwort das System mithilfe einer Pt_n-Übertragungsfunktion an, welche n gleiche Zeitkonstanten T besitzt.

$$G(s) = \frac{Y(s)}{U(s)} = \frac{V}{(1 + s * T)^n}$$

Der Verstärkungsfaktor V wird nach der Beziehung $V = \frac{\Delta y(\infty)}{\Delta u}$ errechnet, wobei y die Ausgangscharakteristik und u die Eingangscharakteristik eines Systems beschreibt. Anschließend werden Zeitprozentkennwerte $t_{10}, t_{30}, t_{50}, t_{70}\ und\ t_{90}$ definiert dies sind die Zeitpunkte an denen die Sprungantwort m= 10%, 30%,… und 90% ihres stationären Endwertes erreicht [Ottens, 2008].

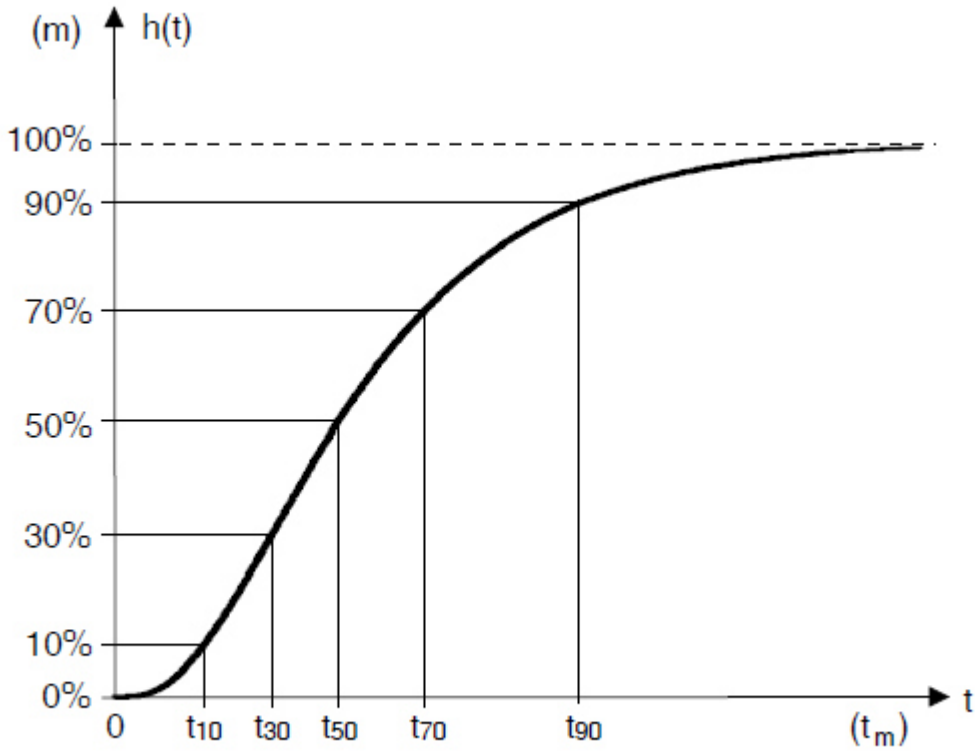


Abbildung 30 Zeitprozentkennwerte einer Sprungantwort

Durch Iteration mithilfe eines Tabellenwerkes (Tabelle 4 und 5) können die Quotienten der Zeitprozentverhältnisse gebildet werden. Diese geben mithilfe von Tabelle 5 Aufschluss über die Systemordnung. Mit anschließender Bildung der bezogenen Zeitprozentkennwerte (Tabelle 4) kann die Zeitkonstante des Systems gebildet werden. Die Zeitkonstante T ist die Zeit bei der die Sprungantwort eines Systems 63 % ihres Endwertes erreicht (siehe Kapitel 2.2.1).

Tabelle 4 Bezogene Zeitprozentkennwerte

n	t_{10} / T	t_{30} / T	t_{50} / T	t_{70} / T	t_{90} / T
1	0,11	0,36	0,69	1,20	2,30
2	0,53	1,10	1,68	2,44	3,89
3	1,10	1,91	2,67	3,62	5,32
4	1,74	2,76	3,67	4,76	6,68
5	2,43	3,63	4,67	5,89	7,99
6	3,15	4,52	5,67	7,01	9,27
7	3,89	5,41	6,67	8,11	10,5
8	4,66	6,31	7,67	9,21	11,8
9	5,43	7,22	8,67	10,3	13,0
10	6,22	8,13	9,67	11,4	14,2

Tabelle 5 Zeitprozentverhältnisse

n	t_{10} / t_{90}	t_{10} / t_{70}	t_{10} / t_{50}	t_{10} / t_{30}	t_{30} / t_{70}	t_{30} / t_{50}
1	0,05	0,09	0,15	0,30	0,30	0,52
2	0,14	0,22	0,32	0,48	0,45	0,65
3	0,21	0,31	0,41	0,58	0,53	0,72
4	0,26	0,37	0,48	0,63	0,58	0,75
5	0,30	0,42	0,52	0,67	0,62	0,78
6	0,34	0,45	0,56	0,70	0,65	0,80
7	0,37	0,48	0,58	0,72	0,67	0,81
8	0,40	0,51	0,61	0,74	0,69	0,82
9	0,42	0,53	0,63	0,75	0,70	0,83
10	0,44	0,55	0,65	0,76	0,71	0,84

Das Verfahren zur experimentellen Streckenidentifikation nach Schwartz soll anhand einer Sprungantwort von 2,5 V (entsprechend 65 Stufen) aufgezeigt werden.

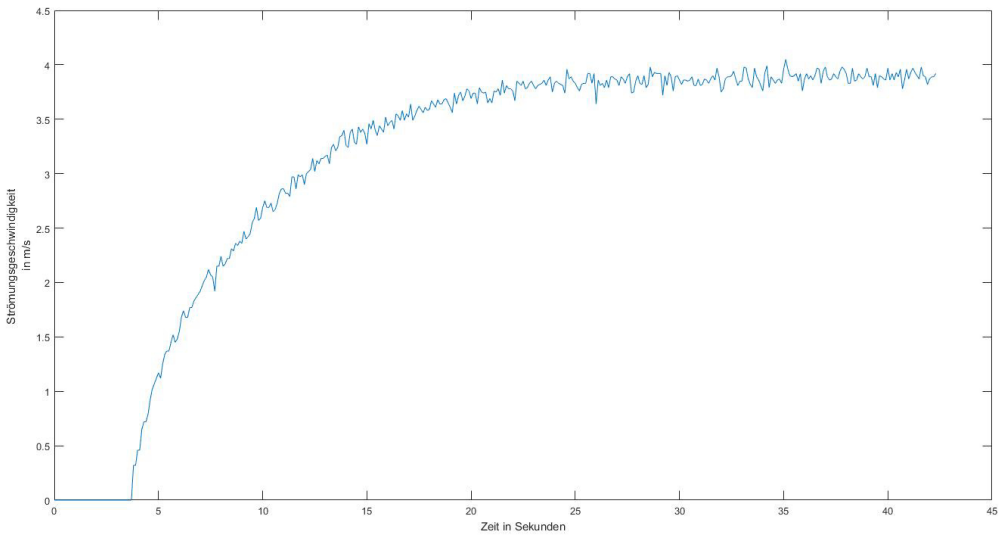


Abbildung 31 Sprungantwort 2,5 V

Die Verstärkung des Systems berechnet sich aus dem Quotient der Differenzen des Ein- und Ausgangsverhaltens des Systems nach folgender Formel.

$$V = \frac{\Delta y(\infty)}{\Delta u}$$

$$V = \frac{3{,}9\,\frac{m}{s} - 0\,\frac{m}{s}}{65\,Stufen - 0\,Stufen}$$

$$V = 0.06\,\frac{m/s}{Stufen}$$

Die einzelnen Zeitprozentkennwerte beziehen sich auf die abgeschlossene Sprungantwort und lassen sich einfach aus der Sprungantwort ablesen vgl. Abbildung 30. Der Zeitprozentkennwert von t_{10}.wird

über die Sprungantwort bestimmt. Hierzu muss im Diagramm die Zeit t bestimmt werden, wofür die Sprungantwort dem Wert $3{,}9 * 0{,}1 = 0{,}39$ entspricht. Die einzelnen Messwerte wurden in Abständen von 0,1 Sekunden aufgenommen.

$$t_{10} = 3{,}9\ Sekunden$$
$$t_{30} = 5{,}1\ Sekunden$$
$$t_{50} = 7{,}2\ Sekunden$$
$$t_{70} = 10{,}8\ Sekunden$$
$$t_{90} = 16{,}5\ Sekunden$$

Aus den so bestimmten Zeitprozentkennwerten, müssen nun die einzelnen Zeitprozentverhältnisse bestimmt werden. Mit den bestimmten Zeitprozentverhältnissen lässt sich aus Tabelle 5 die Systemordnung bestimmen. Theoretisch sollte jedes Zeitprozentverhältnis die gleiche Systemordnung ergeben, aber aufgrund von Ableseungenauigkeiten kann dies nicht immer gewährleistet werden. Daher sollte bei auftretenden Unstimmigkeiten die Systemordnung so gewählt werden, welche am häufigsten aufgetreten ist. Für jedes einzelne Zeitprozentverhältnis wird die Systemordnung n ausgewählt, welche die geringste Differenz zum Tabellenwert aufweist.

$$\frac{t_{10}}{t_{70}} = \frac{3{,}9}{10{,}8} = 0{,}36 \rightarrow n = 4$$
$$\frac{t_{10}}{t_{50}} = \frac{3{,}9}{7{,}2} = 0{,}54 \rightarrow n = 5$$
$$\frac{t_{10}}{t_{30}} = \frac{3{,}9}{5{,}3} = 0{,}74 \rightarrow n = 6$$
$$\frac{t_{10}}{t_{90}} = \frac{3{,}9}{16{,}5} = 0{,}23 \rightarrow n = 3$$
$$\frac{t_{30}}{t_{70}} = \frac{5{,}1}{10{,}8} = 0{,}47 \rightarrow n = 2$$
$$\frac{t_{30}}{t_{50}} = \frac{5{,}1}{7{,}2} = 0{,}71 \rightarrow n = 3$$

Die so bestimmte Systemordnung ist nach Schwartz vom Grad drei und gibt so ein PT_3-System vor.
Für das so entstandene System ist jetzt noch mithilfe von Tabelle 4, die Zeitkonstante zu bestimmen.
Hierfür werden die Zeitprozentkennwerte auf jeweils eine Zeitkonstante bezogen, welche zur Bestimmung der Systemzeitkonstante arithmetisch gemittelt werden.

$$\frac{t_{10}}{T} = 1{,}10 \rightarrow T_{10} = 3{,}54$$
$$\frac{t_{30}}{T} = 1{,}91 \rightarrow T_{30} = 2{,}77$$
$$\frac{t_{50}}{T} = 2{,}67 \rightarrow T_{50} = 2{,}7$$
$$\frac{t_{70}}{T} = 3{,}62 \rightarrow T_{70} = 2{,}98$$
$$\frac{t_{90}}{T} = 5{,}32 \rightarrow T_{90} = 3{,}1$$

$$T_1 = \frac{T_{10} + T_{30} + T_{50} + T_{70} + T_{90}}{5} = 3{,}02$$

Das so ermittelte System besitzt nach dem Systemidentifikationsverfahren nach Schwartz folgende Systemgleichung.

$$G_{SchwartzPT1}(s) = \frac{0{,}06}{(1 + s * 3{,}02)^3}$$

Wenn das bestimmte Modell eine niedrige Verzögerungsordnung hat ($n \approx 2$) existiert noch ein weiterer Modellansatz. Dieser Ansatz approximiert eine gegebene Sprungantwort durch ein PT_2-System, mit zwei verschiedenen Zeitkonstanten. Welches der beiden Modelle besser der Realität entspricht ist durch Vergleichen der Modellsprungantwort mit der Originalsprungantwort herauszufinden. Die Vorgehensweise ist hier fast analog zu dem Verfahren, welches mit einer Zeitkonstante auskommt. Die einfach bestimmte Zeitkonstante wird hier nur mit einem aus einem Tabellenwerk [Ottens, 2008] zu bestimmenden Faktor b ermittelt. Die anschließend resultierende

Tabelle 6 Bezogene Zeitprozentverhältnisse für Systeme mit zwei Zeitkonstanten

b	t_{10} / T	t_{30} / T	t_{50} / T	t_{70} / T	t_{90} / T
1	0,53	1,10	1,68	2,44	3,89
2	0,76	1,59	2,46	3,62	5,94
3	0,95	2,01	3,17	4,79	8,12
5	1,27	2,71	4,56	7,14	12,6
7	1,55	3,53	5,92	9,51	17,2
10	1,93	4,61	7,99	13,1	24,1
12,5	2,22	5,50	9,71	16,1	29,9
15	2,52	6,38	11,4	19,1	35,6
17,5	2,80	7,28	13,2	22,1	41,3
20	3,08	8,16	14,9	25,1	47,1

Übertragungsfunktion baut sich auf folgender Formel auf.

$$G(s) = \frac{Y(s)}{U(s)} = \frac{V}{(1 + s * T) * (1 + s * b * T)}$$

Nach Tabelle 6 ergeben sich folgende Zeitprozentverhältnise und der daraus resultierende Faktor b.

$$\frac{t_{10}}{t_{70}} = \frac{3,9}{10,8} = 0,36 \rightarrow b = 1$$
$$\frac{t_{10}}{t_{50}} = \frac{3,9}{7,2} = 0,54 \rightarrow b = 1$$
$$\frac{t_{10}}{t_{30}} = \frac{3,9}{5,3} = 0,74 \rightarrow b = 1$$
$$\frac{t_{10}}{t_{90}} = \frac{3,9}{16,5} = 0,23 \rightarrow b = 1$$
$$\frac{t_{30}}{t_{70}} = \frac{5,1}{10,8} = 0,47 \rightarrow b = 1$$
$$\frac{t_{30}}{t_{50}} = \frac{5,1}{7,2} = 0,71 \rightarrow b = 1$$

Tabelle 7 Zeitprozentverhältnisse für Systeme mit zwei Zeitkonstanten

b	t_{10} / t_{90}	t_{10} / t_{70}	t_{10} / t_{50}	t_{10} / t_{30}	t_{30} / t_{70}	t_{30} / t_{50}
1	0,14	0,22	0,32	0,48	0,45	0,65
2	0,13	0,21	0,31	0,48	0,44	0,65
3	0,12	0,20	0,30	0,47	0,42	0,63
5	0,10	0,18	0,28	0,45	0,39	0,61
7	0,09	0,16	0,26	0,44	0,37	0,60
10	0,08	0,15	0,24	0,42	0,35	0,58
12,5	0,07	0,14	0,23	0,40	0,34	0,57
15	0,07	0,13	0,22	0,40	0,33	0,56
17,5	0,07	0,13	0,21	0,39	0,33	0,55
20	0,06	0,12	0,21	0,38	0,32	0,55

Aus Tabelle 7 können hier die Zeitkonstanten mithilfe des zuvor ermittelten Faktor b aus den bezogenen Zeitprozentverhältnissen errechnet werden.

$$\frac{t_{10}}{T} = 0{,}53 \rightarrow T_{10} = 7{,}36$$
$$\frac{t_{30}}{T} = 1{,}10 \rightarrow T_{30} = 4{,}81$$
$$\frac{t_{50}}{T} = 1{,}68 \rightarrow T_{50} = 4{,}2$$
$$\frac{t_{70}}{T} = 2{,}44 \rightarrow T_{70} = 4{,}43$$
$$\frac{t_{90}}{T} = 3{,}89 \rightarrow T_{90} = 4{,}24$$

Aus den berechneten Zeitkonstanten ist wieder der Mittelwert zu bilden

$$T_1 = \frac{T_{10} + T_{30} + T_{50} + T_{70} + T_{90}}{5} = 5{,}0$$

Trotz dem Ansatz der Verwendung von zwei Zeitkonstanten wird das resultierende System zwar ein System zweiter Ordnung sein, jedoch ergbit sich für b=1 zweimal die selbe Zeitkonstante.
Das daraus resultierende identifizierte System besitzt folgende Systemgleichung

$$G_{Schwartz_{PT2}}(s) = \frac{0{,}0667}{(1 + s * 5{,}0)^1 * (1 + s * 5{,}0)^1}$$

3.1.2 Systemidentifikation mit Matlab

Die Systemidentifikation mit dem Programm Matlab ist ein rechnergestütztes Black-Box Verfahren und basiert auf verschiedenen Iterationsverfahren z.B. das Gauß-Newton-Verfahren. Die einzelnen Iterationsverfahren können in der Systemidentifcation Toolbox einzeln ausgewählt werden, wobei Matlab mit der Funktion „Auto“ das beste Verfahren für sich wählt. Zur System Identifikation in Matlab wird die System Identification Toolbox genutzt. Im Folgenden soll kurz die Vorgehensweise bei der Systemidentifikation mit Matlab gezeigt werden [Ljung, 2000]:

1. Aufnehmen des Ein- und Ausgangsverhaltens. Hierzu werden zweimalig Daten aufgenommen bestehend jeweils aus dem Signal mit dem das System angeregt wird und der Systemantwort. Der eine Datensatz wird zur Identifikation benötigt und der zweite Datensatz zur Validierung. Würde nachher das identifizierte Modell mit dem zur Identifikation genutzten Datensatz genutzt werden, würde eine zugute Übereinstimmung zustande kommen, welche nicht der Realität entspräche. Deshalb wird das identifizierte Modell mit einem Validierungsdatensatz verglichen, welcher nicht zur Identifikation benutzt wurde.

2. Untersuchung der Daten auf Messfehler und Wahl eines Bereiches, welcher z.B. nur die abgeschlossene Sprungantwort zeigt.

3. Auswahl des Modelltyps Transferfunktion, Polynomfunktion etc.

4. Auswahl der Modellstruktur (Pole und Nullstellen) und anschließende Iteration. Hier kann eine Auswahl durch Betrachten der Sprungantwort erfolgen. Dies erfordert ein wenig Grundkenntnis von den einzelnen Übertragungsfunktionen, welche in der Regelungstechnik auftreten.

5. Untersuchung des Modells und Abweichungen zur Realität (Best-fit). Ebenso sollten die Pole und Nullstelle des Systems betrachtet werden, um eine Aussage über die Stabilität des Systems treffen zu können.

6. Wenn eine gute Übereinstimmung zwischen Modellverhalten und realem Verhalten erreicht wurde kann die Systemidentifikation beendet werden. Ansonsten muss von Schritt 3 erneut begonnen werden und durch Ändern der Vorgabe der Pol- Nullstellen Konfiguration eine erneute Iteration unternommen werden.

   Nach durchlaufen der Schritte 1-6 liefert die System-Identification-Toolbox dem Nutzer eine fertige Übertragungsfunktion mit der anschließend ein Regler Entwurf geschehen kann. Um einen Einblick in das Iterationsverfahren der Systemidentification Toolbox zu gewinnen, soll im Folgenden das Gauß-Newton Verfahren anhand eines einfachen Beispiels vorgerechnet werden.

3.1.2.1 Gauß-Newton Verfahren

Die Systemidentifikation mithilfe von Matlab bedient sich verschiedener Mathematischer Verfahren, wie z.B. dem Levenberg-Marquardt Verfahren oder dem Gauß Newton Verfahren. Bei den verschiedenen Verfahren handelt es sich um verschiedene mathematische Iterationsprinzipien. Das Gauß-Newton Verfahren soll kurz erklärt werden um einen Einblick in die ungefähre Arbeitsweise der System Identification Toolbox zu gewähren.

Das Gauß-Newton Verfahren dient zur Lösung von Ausgleichsproblemen. Anwendungen sind wie hier in der Systemidentifikation vorgegebene Messdaten, welche aus Wertepaaren bestehen. Ziel ist es hier nicht eine Funktion zu finden, welche perfekt durch die einzelnen Werte verläuft, sondern Ziel ist es eine stetige Funktion f zu finden, welche diese Wertepaare bestmöglich nähert. Probleme hierbei sind das von der gesuchten Funktion die partiellen Ableitungen benötigt werden und das eine genügend gute Anfangswertbedingung benötigt wird, ansonsten konvergiert das einfache Newton-Verfahren nicht. [Nestler]

Eine Abhilfe schafft hier das Gauß-Newton-Verfahren, was nicht von diesen Problemen behaftet ist. Das Gauß-Newton-Verfahren besteht hierbei aus einer Kombination von linearer Ausgleichsrechnung und Newton-Verfahren.
Es sollen folgende Punkte durch den Gauß-Newton-Iterationsalgorithmus angenähert werden:

Tabelle 8 Stützpunkte

x_i	0	1	2
y_i	3	1	0,5

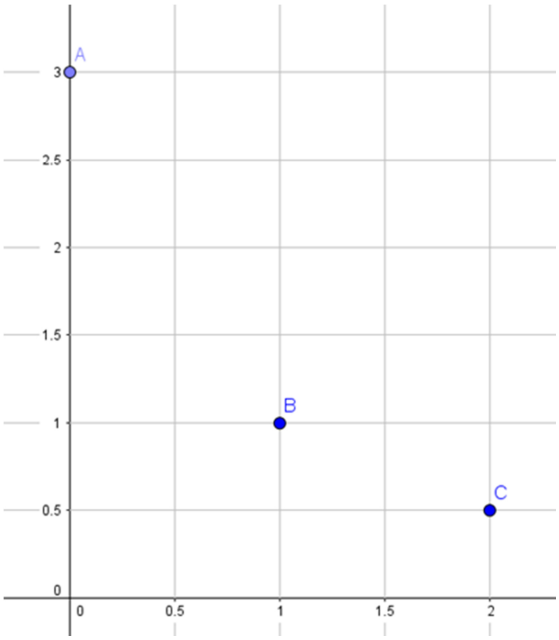


Abbildung 32 vorgegebene Stützpunkte

Der Algorithmus basiert auf einer Residuums Funktion und einem Residuums Vektor. Die Residuums Funktion ist die Funktion, welche für die Iteration angenommen wird. Ein Blick auf Abbildung 32 lässt einen Quadratischen Zusammenhang vermuten, daher soll als Residuums Funktion eine Funktion von quadratischer Form angenommen werden:

$$F_{Residuum}(x) = a_2x^2 + a_1x + a_0$$

Der Residuums Vektor besitzt folgende Form:

$$r(a) = y_i - f(x_{i,1}, \dots, x_{i,n})$$

Des angewendeten Residuums Vektor sieht folgendermaßen aus:

$$r(a) = \begin{pmatrix} 3 - a_0 \\ 1 - (a_2 + a_1 + a_0) \\ 0{,}5 - (4a_2 + 2_{a1} + a_0) \end{pmatrix}$$

Aus der Residuums Funktion muss eine Matrix gebildet werden, welche sämtliche erste partielle Ableitungen nach a_i enthält. Diese Ableitung ist auch als Funktional-Matrix, oder auch Jacobi-Matrix

bekannt. Die Größe der Jacobi-Matrix wird durch die Anzahl der zu bestimmenden Parameter vorgegeben. Für drei zu bestimmende Parameter hat die Jacobi-Matrix also drei Zeilen.

$$J = \begin{pmatrix} \frac{\delta r_1}{\delta a_0} & \frac{\delta r_1}{\delta a_1} & \frac{\delta r_1}{\delta a_3} \\ \frac{\delta r_2}{\delta a_0} & \frac{\delta r_2}{\delta a_1} & \frac{\delta r_2}{\delta a_3} \\ \frac{\delta r_3}{\delta a_0} & \frac{\delta r_3}{\delta a_1} & \frac{\delta r_3}{\delta a_3} \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 \\ -1 & -1 & -1 \\ -1 & -2 & -4 \end{pmatrix}$$

Der eigentliche Algorithmus arbeitet nach folgendem Schema, es wird als Anfangsbedingung $a = 1$ angenommen:

$$a^{j+1} = a^j - J^{-1} * r(a^j)$$

$$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} -1 & 0 & 0 \\ 1{,}5 & -2 & 0{,}5 \\ -0{,}5 & 1 & -0{,}5 \end{pmatrix} * \begin{pmatrix} 3-1 \\ 3-(1+1+1) \\ 0{,}5-(4+2+1) \end{pmatrix} = \begin{pmatrix} 3 \\ -2{,}75 \\ 0{,}75 \end{pmatrix}$$

Die Koeffizienten für a_0, a_1, a_2 wurden zu 3, -2,75 und 0,75 ermittelt. Eingesetzt in die angenommene Residuums Funktion ergibt sich $0{,}75x^2 - 2{,}75x + 3$ als beschreibendes Polynom.

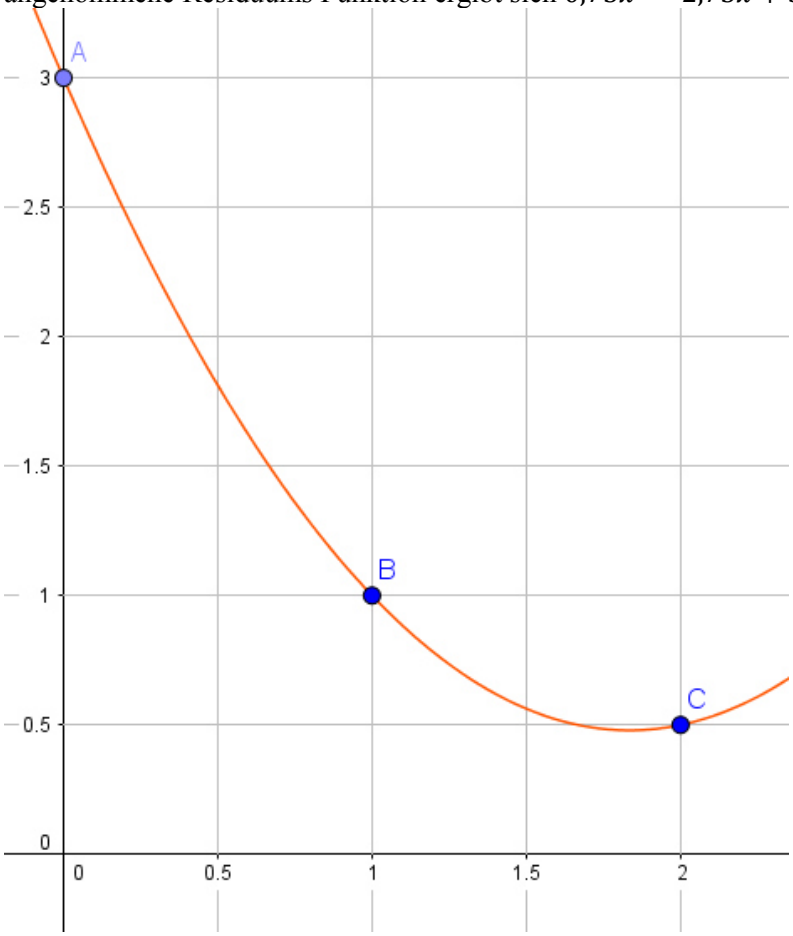


Abbildung 33 ermittelte Kurve

In diesem einfachen Beispiel konvergiert das Verfahren nach einem Durchlauf. Bei einem nicht konvergieren müsste die Jacobi-Matrix neu berechnet werden und die zuvor bestimmten Koeffizienten, wären die neuen Anfangsbedingungen.

3.1.3 System-Identification-Toolbox

Nach erfolgter Aufnahme der Sprungantwort müssen die als .txt vorliegenden Datensätze (Identifikation- und Validierungsdatensatz) in den Matlab Workspace importiert werden. Anhand des Identifikationsdatensatzes wird die Systemgleichung bestimmt, welche wiederum mit dem Validierungsdatensatz verglichen wird. Dort müssen Ein- und Ausgangsverhalten jeweils als Nx1 Matrix vorliegen. Anschließend können diese in der Toolbox durch drücken des Reiters „import" Data" und „Time Domain Data" importiert werden (Abbildung 34). Hier müssen nun die zusammengehörenden Nx1 Matrizen, welche Ein- und Ausgangsverhalten darstellen, noch so zugewiesen werden, dass Matlab erkennt welche Matrize das Ein- bzw. Ausgangsverhalten wiedergibt.

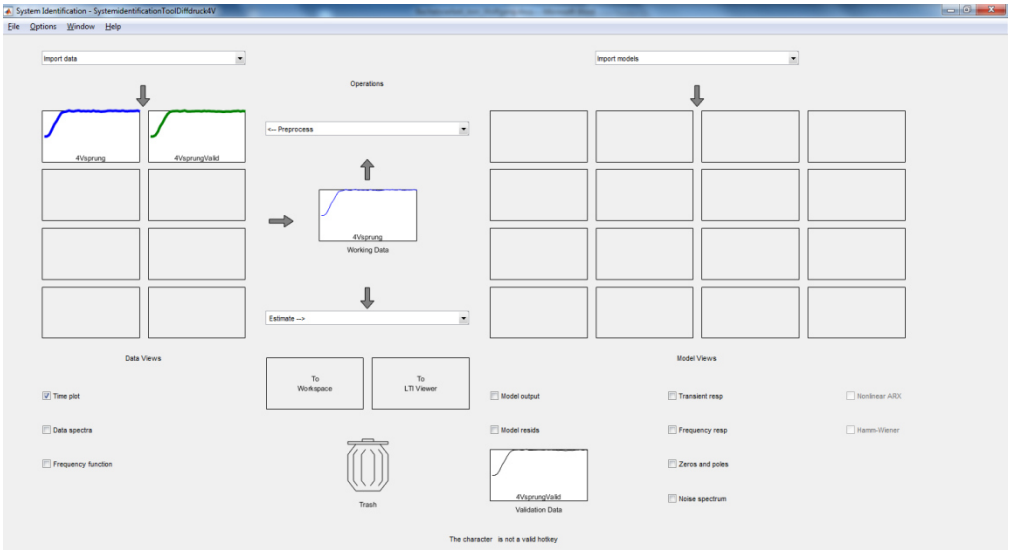


Abbildung 34 Importierte Datensätze

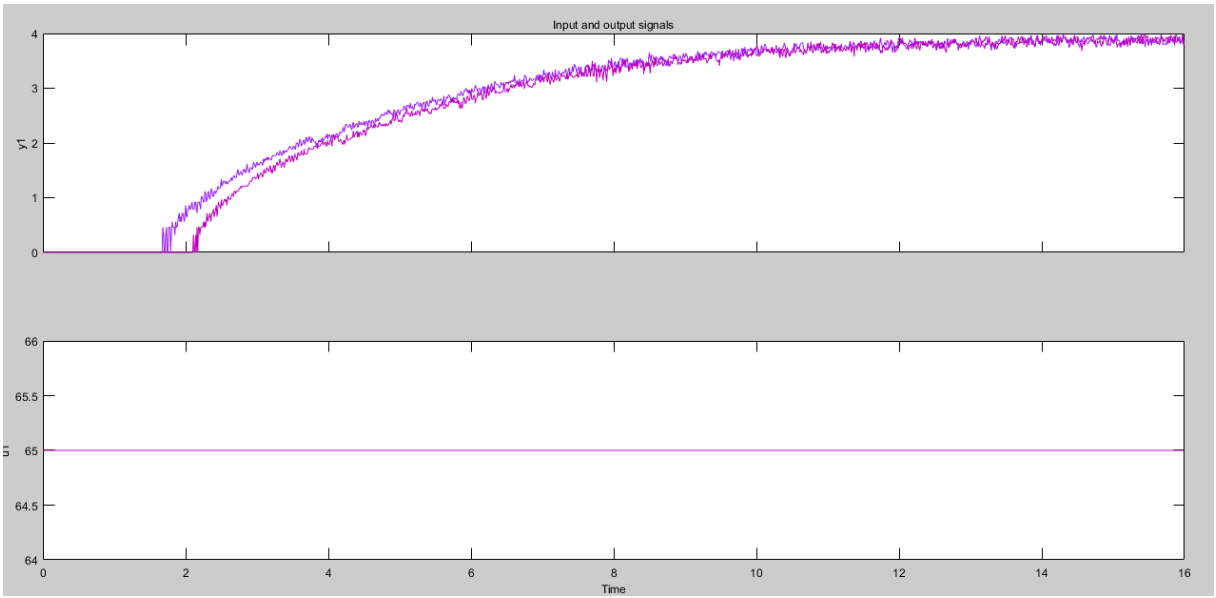


Abbildung 35 Ein- und Ausgangsverhalten

Anschließend können diese Daten durch Klicken auf den Reiter Preprocess gefiltert werden, oder es können Trends und Messreißer entfernt werden. Eine Auswahl der beiden Datensätze und setzen des Hakens bei Time Plot ermöglicht ein betrachten der beiden Datensätze (vgl. Abbildung 35).

Durch klicken auf den Reiter „Estimate“ und „Transfer Function Models“ kann die Auswahl der Iterationsmethode und die Anzahl der Pole und Nullstellen erfolgen. Die vorliegende Sprungantwort wurde als PT3 identifiziert. Dennoch wurden noch weitere Konstellationen verwendet, welche weniger gute Ergebnisse im Hinblick auf die Abweichung zur Realität lieferten (vgl. Abbildung 36).

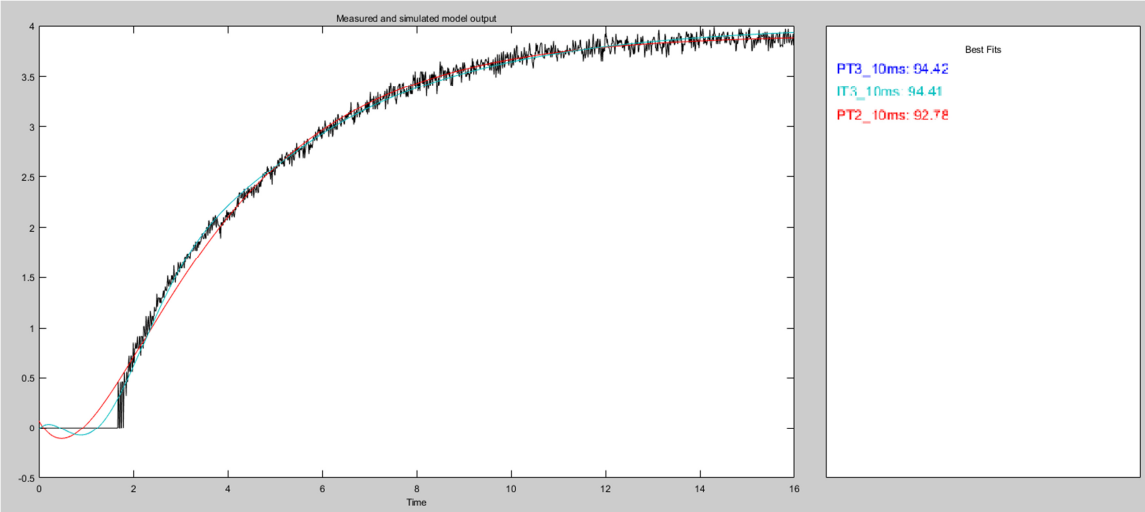


Abbildung 36 Best Fits Modelle zur Realität

Das betrachtete System konnte als PT3-System mit folgender Übertragungsfunktion identifiziert werden.

$$G_{PT3}(s) = \frac{0.05605}{s^3 + 2.032 * s^2 + 3.84 * s + 0.9094}$$

Abbildung 37 zeigt die Pole des identifizierten Systems, welches aus einer einfachen Polstelle und einer dominierenden doppelten Polstelle besteht. Da diese sich in der linken Halbebene befinden, geben diese Aufschluss darüber, dass das System stabil ist.

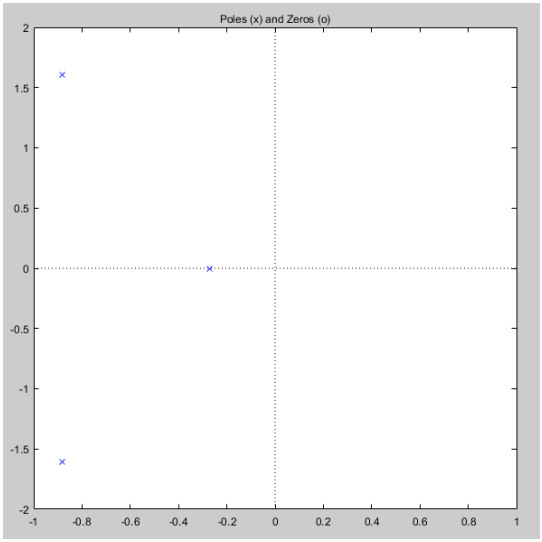


Abbildung 37 Polstellen des Identifizierten Systems

Für die Pol Konfiguration eines Systems gibt es drei Fälle die auftreten können:

1. Alle Pole sind negativ
   Die Sprungantwort strebt in Abhängigkeit vom Eingangssignal gegen einen Maximalwert (Stabil).
2. Ein Pol ist gleich null, alle anderen negativ
   Die Sprungantwort verläuft erst exponentiell, dann linear gegen unendlich (grenzstabil).
3. Mindestens ein Pol ist positiv
   Die Sprungantwort strebt exponentiell gegen unendlich (instabil). [wikibooks, 2015]

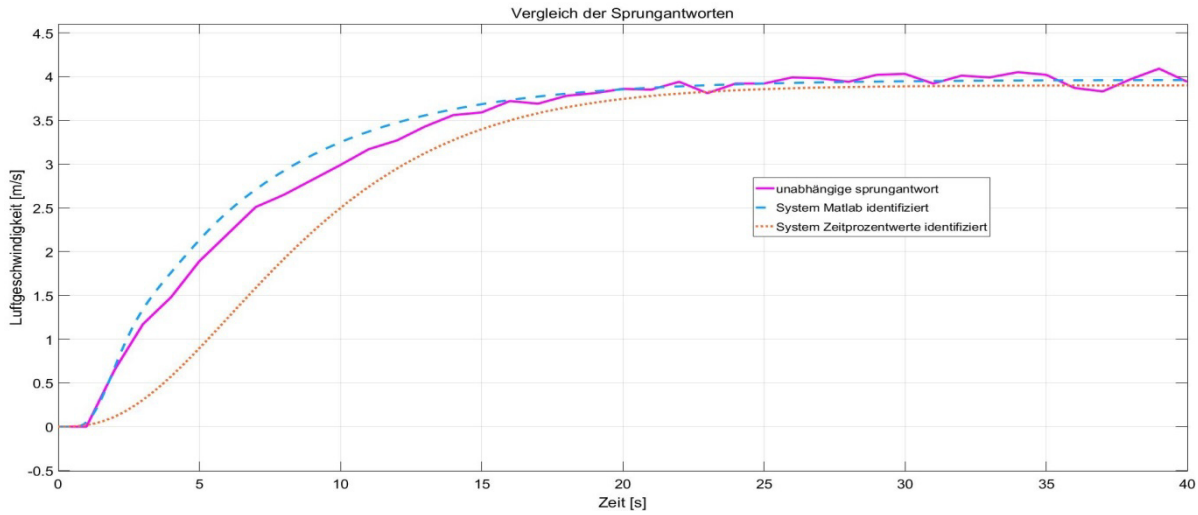


Abbildung 38 Vergleich der Sprungantworten

Abbildung 38 vergleicht eine reale unabhängige Sprungantwort (pink) mit der mithilfe von Matlab (blau) und dem Verfahren nach Schwartz (rot) identifizierten Systemgleichungen und deren Sprungantworten. Mit der unabhängigen Sprungantwort wird eine Sprungantwort bezeichnet, welche weder für die Identifikation mit dem Verfahren nach Schwartz oder der Identifikation mit Matlab verwendet wurde. Eine Verwendung einer abhängigen Sprungantwort würde ein Ergebnis erzielen, welches extrem zu Gunsten des jeweiligen Verfahrens ausfallen würde. Wie schon vermutet stellt die Systemgleichung, welche von Matlab zur Verfügung gestellt wird eine bessere Näherung zur realen Sprungantwort dar. Fortlaufend wird nun nur noch mit dem von Matlab identifizierten System ein Regler Entwurf durchgeführt.

4 Reglerentwurf

Mit dem mithilfe von Matlab erfolgreich identifizierten System kann anschließend ein Reglerentwurf erfolgen. Ziel des Reglerentwurfs ist es die drei Regelparameter P, I und D zu bestimmen, welche das Regelverhalten beeinflussen.

Proportional-Anteil: Der Proportional Anteil einer Regelung, oder auch P-Anteil beeinflusst die Stellgröße abhängig von ihrer Abweichung zum Sollwert. Bei einer hohen Regelabweichung wird die Stellgröße durch den Proportional-Anteil entsprechend nachgestellt. Bei der Wahl des P-Anteils ist darauf zu achten, dass dieser nicht zu groß gewählt wird. Bei einer Wahl über den kritischen Wert für den P-Anteil wird ein System destabilisiert und die Regelung ist unbrauchbar. Der kritische Wert für den P-Anteil wird für heuristische Einstellmethoden, wie das Verfahren von Ziegler-Nichols verwendet. Hier wird die Regler Verstärkung solange erhöht, bis das System grenzstabil wird. Dies ist daran zu erkennen, dass das System eine Dauerschwingung ausführt. Anhand dieses so bestimmten kritischen P-Anteils können mithilfe eines Tabellenwerkes die Einstellungen für die drei Regelparameter P, I und D geschehen. Ein reiner P-Regler kann die bleibende Regelabweichung eines Systems, welches keinen Integral Anteil besitzt nicht beseitigen.

Integrierender Anteil: Der Integrierende Anteil einer Regelung, oder auch I-Anteil beeinflusst die Regelabweichung. Der I-Anteil verändert solange die Stellgröße, bis die Regelabweichung verschwunden ist. Der I-Anteil dient also dazu die bleibende Regelabweichung zu eliminieren. Der I-Anteil muss ähnlich dem P-Regler mit Bedacht gewählt werden, da ein Regler mit falsch gewähltem I-Anteil zum Schwingen neigt.

Differenzierender Anteil: Der Differenzierende Anteil einer Regelung, oder auch D-Anteil betrachtet die Änderung der Regelabweichung. Der D-Anteil reagiert auf eine Änderung der Istwerte zueinander,

wenn die Änderung des Istwertes zum Zeitpunkt t=n zu t=n+1 besonders hoch ist greift dieser ein. Wenn ein D-Anteil für eine Regelung in Betracht gezogen wird ist darauf zu achten, dass die Rückführung durch das Messglied (der Istwert) einen geringen Rauschanteil besitzt. Bei schlecht geglätteten Signalen reagiert der D-Anteil über und verhindert eine effiziente Regelung.

An jeden Regler stehen unterschiedliche Anforderungen bezüglich Einschwingdauer, Überschwinghöhe und Regelabweichung. Beispielsweise ist für die Kühlwasserzufuhr eines Reaktorkerns eine schnelle Einschwingdauer des Reglers erforderlich. Hingegen für eine automatische Wasserzufuhr für ein Aquarium ist eine bleibende Regelabweichung mit langer Einschwingdauer zulässig, da dies keine kritische Anwendung ist. Da das Fraunhofer Institut für Bauphysik ausschließlich Langzeit Versuche durchführt, welche theoretisch hohe Einschwingdauern für Regler zulässt bestanden hier keine besonderen Vorgaben. Da es sich bei dem Lüfter auch nicht um ein kritisches System handelt, welches nur eine bestimmte Überschwinghöhe zulässt, wurde hier ein Überschwinger von maximal 20 % für zulässig erachtet. Bezüglich der Regelabweichung sollte der Prüfstand eine maximale Abweichung vom Sollwert von 0,1 $\frac{m}{s}$ zulassen. Auch wenn Einschwingdauer und Überschwinghöhe keine gravierende Rolle gespielt haben wurden diese berücksichtigt und so kurz wie möglich bzw. so gering wie möglich gehalten. Allgemein können die Anforderungen an einen Regler wie folgt formuliert werden [Berger, 2001]:

1. Der geschlossene Regelkreis ist stabil im kompletten Arbeitsbereich und besitzt ausreichend Stabilitätsreserven.
2. Im stationären Zustand geht die bleibende Regelabweichung gegen 0.
3. Der Regler ist robust gegenüber Parameterschwankungen der Regelstrecke.
4. Das der Regler ein gutes dynamischen Führungsverhalten und Störverhalten besitzt.
5. Der Regler besitzt eine kurze Anregelzeit und Ausregelzeit und ein geringes Überschwingen.

Für die Reglerentwürfe selbst wurden verschiedene Verfahren verwendet, wobei die Verfahren hier sich in zwei Varianten teilen. Die Verfahren welche ohne Systemgleichung auskommen und die Reglerparameter aus der Sprungantwort bestimmen. Und Verfahren welche eine zuvor bestimmte Systemgleichung verwenden und rechenaufwändiger sind.

4.1 Verfahren mit Systemidentifikation

Für den Reglerentwurf bieten sich verschiedene Verfahren, welche entweder auf rein heuristischen Verfahren der Parameterschätzung basieren und einen eher ungenauen, aber schnellen Reglerentwurf zulassen. Oder Verfahren welche die Systemgleichung zu Nutze ziehen und zeitaufwändiger sind, dafür aber bessere Ergebnisse liefern.

4.1.1 Ziegler-Nichols

Das Verfahren nach Ziegler-Nichols bestimmt die PID-Parameter über die Berechnung der kritischen Verstärkung und kritischen Zeitkonstante (Systemgrenze für den P-Anteil, dass das System eine Dauerschwingung ausführt). Im Folgenden soll der Regler Entwurf mithilfe von Ziegler Nichols anhand der mit der System-Identification Toolbox identifizierten Systemgleichung berechnet werden [Zentgraf, 2015]:

$$G_{PT3}(s) = \frac{0{,}05605}{s^3 + 2{,}032 * s^2 + 3{,}84 * s + 0{,}9094}$$

Bevor überhaupt ein Regler Entwurf vollzogen werden kann ist es wichtig das vorliegende System mithilfe des Hurwitz-Kriteriums auf Stabilität zu prüfen. Hierfür gilt es zwei Kriterien zu erfüllen.

1. Kriterium: Alle Polstellen a_i besitzen ein positives Vorzeichen
2. Kriterium: Das Hurwitz-Kriterium ist erfüllt

Das Hurwitz Kriterium betrachtet die Determinante des Nennerpolynoms. Für ein stabiles System ist die Determinante $D > 0$. Im Gegenteiligen Fall gilt das $D \leq 0$ ist und das Verfahren nach Ziegler-Nichols nicht angewendet werden kann. Bei einem Blick auf die Vorzeichen der Polstellen erhält man Aufschluss darüber, dass das 1. Kriterium erfüllt ist.

$$D = a_2 * a_1 - a_3 * a_0$$
$$D = 2{,}032 * 3{,}84 - 1 * 0{,}9094$$
$$D = 6{,}8934 \rightarrow D > 0$$

Nachfolgend wird der Bereich für die nicht kritische Verstärkung berechnet. Hierfür muss das System für den offenen Regelkreis auf seine obere und seine untere kritische Grenze geprüft werden. Die untere Grenze wird mithilfe des 1. Hurwitz-Kriteriums überprüft und die obere mit dem 2. Hurwitz-Kriterium.

$$1 + k_p * \frac{0{,}05605}{s^3 + 2{,}032 * s^2 + 3{,}84 * s + 0{,}9094} = 0$$
$$s^3 + 2{,}032s^2 + 3{,}84s + 0{,}9094 + k_p * 0{,}05605$$

Die Koeffizienten des geschlossenen Regelkreises ergeben sich wie folgt:

$$a_3 = s^3,\ a_2 = 2{,}032s^2,\ a_1 = 3{,}84s,\ a_0 = 0{,}9094 + kp * 0{,}05605$$

Für die untere kritische Grenze muss überprüft werden, ab wann das 1. Hurwitz-Kriterium verletzt wird.

$$0{,}9094 + kp * 0{,}05605 = 0$$
$$k_p > -16{,}22$$

Der unkritische Bereich für k_p kann so nur größer als -16,22 sein. Für die obere kritische Grenze muss überprüft werden, ab wann das 2. Hurwitz-Kriterium verletzt wird.

$$2{,}032 * 3{,}84 - 1 * (0{,}9094 + k_p * 0{,}05605) = 0$$
$$k_p > 122{,}98$$

Der unkritische Bereich für den P-Anteil kann mit $-14.86 < k_p < 122{,}98$ berechnet werden. Für die Auslegung eines reinen P-Reglers ist die Berechnung bis hier ausreichend. Soll aber ein etwas aufwendiger Regler wie ein PI- oder PID-Regler zur Verwendung kommen, ist es notwendig die kritische Periodendauer zu berechnen. Eine Berechnung der kritischen Periodendauer geschieht über die kritische Kreisfrequenz w_{krit} am offenen Regelkreis.

$$k_p * \frac{0{,}05605}{(jw)^3 + 2{,}032 * (jw)^2 + 3{,}84 * jw + 0{,}9094} = -1$$
$$k_p * 0{,}05605 = (jw)^3 + 2{,}032w^2 - 3{,}84jw - 0{,}9094$$
$$0 = -k_p * 0{,}05605 + (jw)^3 + 2{,}032w^2 - 3{,}84jw - 0{,}9094$$
$$0 = -k_p * 0{,}05605 + 2{,}032w^2 - 0{,}9094 + (jw)^3 - 3{,}84jw$$

Berechnung Realteil:

$$-k_p * 0{,}05605 + 2{,}032w^2 - 0{,}9094 = 0$$
$$w_{Realteil} = \sqrt{\frac{k_p * 0{,}05605 + 0{,}9094}{2{,}032}}$$
$$w_{Realteil} = 1{,}96$$

Berechnung Imaginärteil:

$$jw^3 - 3{,}84 = 0$$
$$jw * (w^2 - 3{,}84) = 0$$
$$w^2 - 3{,}84 = 0$$
$$w_{Imaginärteil} = 1{,}96$$

Bei einer korrekten Umformung und Berechnung ergibt sich für den Real- und Imaginärteil ein identisches Ergebnis.

Aus der kritischen Kreisfrequenz kann die kritische Periodendauer bestimmt werden:

$$T_{krit} = \frac{2\pi}{w_{krit}}$$
$$T_{krit} = 3{,}21$$

Mit der bestimmten kritischen Regler Verstärkung kp_{krit} und der kritischen Periodendauer T_{krit}, können nach nachfolgender Tabelle die einzelnen Parameter für den gewünschten Regler bestimmt werden. [Lutz, 2014]

Für die Auslegung der Regelparameter nach Ziegler-Nichols existieren mehrere Auslegungsregeln, welche unterschiedlich viel Überschwingen zulassen.

Tabelle 9 Regler Parameter für Ziegler-Nichols-Verfahren

Regler	Regler Parameter
P	Kp = 0,5* kp_{krit},
PI	Kp = 0,45* kp_{krit},Tn = 0,85*T_{krit}
PD	Kp = 0,55*kp_{krit},Tv = 0,15*T_{krit}
PID	Kp =0,6* kp_{krit}, Tn = 0,5*T_{krit}, Tv=0,125*T_{krit}

Für den Lüfter Prüfstand soll ein Regler gewählt werden, welcher nicht durch Rauschanteile der Messsignale beeinflusst wird und gleichzeitig keine bleibende Regelabweichung besitzt. Daher soll ein PI-Regler ausgewählt werden, der diesen geforderten Anforderungen nachkommen kann.

Die Regler Parameter errechnen sich für den P- und den I-Anteil wie folgt.

$$P = 0{,}45 * 122{,}98$$
$$P = 55{,}3$$

$$T_n = 0{,}85 * 3{,}21$$
$$T_n = 2{,}72$$

Die Angabe für den I-Anteil wird in Form der Nachstellzeit angegeben. Diese kann wie folgt in den I-Anteil überführt werden.

$$I = \frac{P}{T_n}$$
$$I = \frac{55{,}3}{2{,}72}$$
$$I = 20{,}33$$

4.1.2 Matlab PID-Tuning

Matlab bietet die Möglichkeit über die interne Simulationsumgebung Simulink den realen geschlossenen Regelkreis zu simulieren. Das Unterprogramm Simulink basiert auf einer auf Funktionsblöcken bestehenden Programmierumgebung. Mit diesen lassen sich physikalische Gegebenheiten ohne Programmierkenntnisse funktional nachstellen wie z.B. das Anlaufverhalten eines Motors, oder den Einfluss von Messrauschen auf ein Signal.

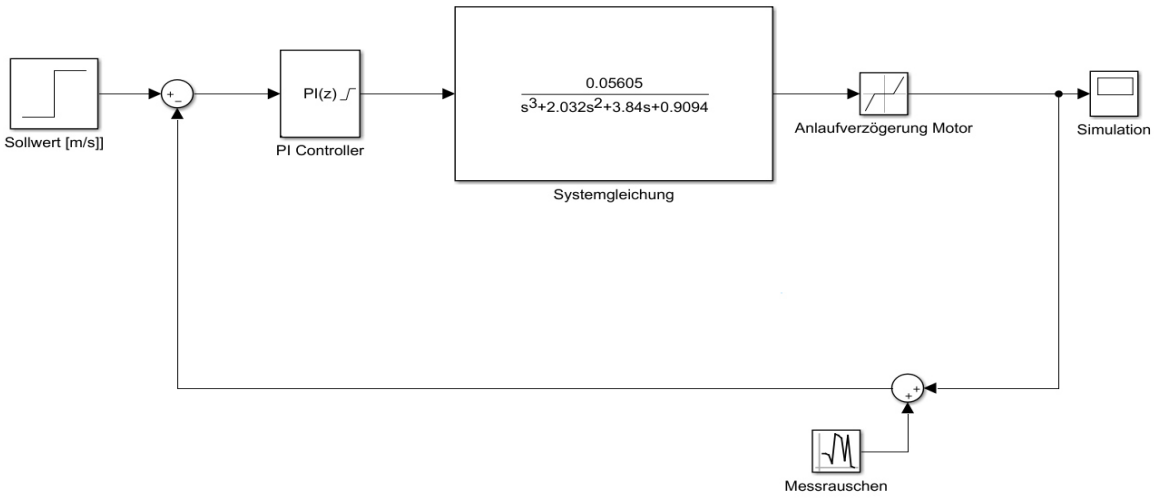
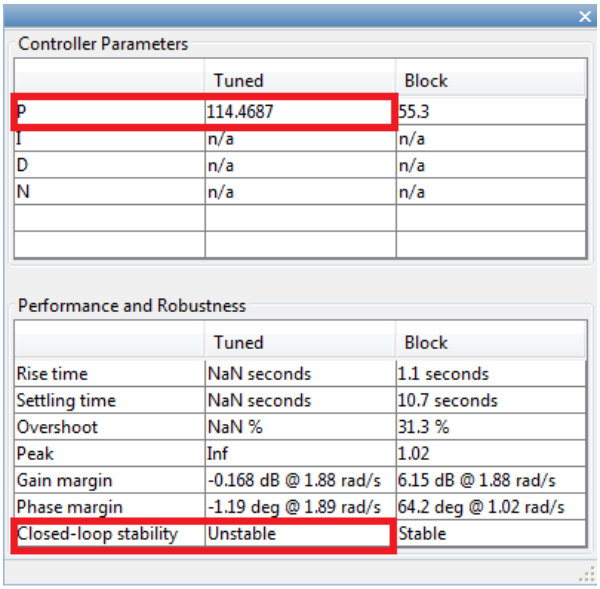


Abbildung 39 Simulation in Simulink

Für die Simulation des Regelkreises wurde ein zeitdiskreter Regler welcher in seiner Stellgröße beschränkt ist verwendet. Die Arduino Bibliothek welche den Regelungsalgorithmus enthält besitzt eine Sample Time von 100 ms. So wird die neue Stellgröße alle 0,1 Sekunden neu berechnet. In Simulink muss hier im PID Controller-Block ein zeitdiskreter Regler mit einer Sample Time von 0,1

Sekunde eingestellt werden. Um die Stellgrößenbeschränkung zu simulieren ist es erforderlich die untere und obere Stellgrößenbeschränkung auf 23 bzw. 128 Stufen einzustellen. Die obere Stellgrößenbeschränkung simuliert das unter 2.2 beschriebene Motorverhalten, da dieser bei einem Signal größer als 5V keine Erhöhung der Strömungsgeschwindigkeit mehr bringt. Die untere Stellgrößenbeschränkung simuliert das Überwinden des Festhaltemoments des Motors, da dieser erst bei einer Steuerspannung am Thyristorsteller von 0,9 Volt anläuft.

Durch ein öffnen des PID Systemblocks und der Auswahl der „Tune“ Schaltfläche, öffnet sich die PID-Tuner Toolbox. Hier kann durch verschieben eines Schiebereglers das Verhalten des Reglers bezüglich Ansprechzeit Störverhalten geändert werden. Für einen reinen P-Regler kann die zuvor in 4.1.1 berechnete kritische Regler Verstärkung überprüft werden. Durch öffnen der Schaltfläche „Show Parameters“ erhält man einiges an Informationen, wie zum Beispiel Einregelzeit und Overshoot des Reglers. Unter der Sektion Closed-loop stability ist zu sehen, ab wann das System instabil wird durch überschreiten der kritischen Regler Verstärkung. Die zuvor in Kapitel 4.1.1 berechnete kritische Verstärkung von $k_p < 122{,}98$ wird hier bei ungefähr einer Verstärkung von 114,5 erreicht. Zufriedenstellende Ergebnisse lieferte ein Regler für P=28,1 und I=4,7.



Controller Parameters

	Tuned	Block
P	114.4687	55.3
I	n/a	n/a
D	n/a	n/a
N	n/a	n/a

Performance and Robustness

	Tuned	Block
Rise time	NaN seconds	1.1 seconds
Settling time	NaN seconds	10.7 seconds
Overshoot	NaN %	31.3 %
Peak	Inf	1.02
Gain margin	-0.168 dB @ 1.88 rad/s	6.15 dB @ 1.88 rad/s
Phase margin	-1.19 deg @ 1.89 rad/s	64.2 deg @ 1.02 rad/s
Closed-loop stability	Unstable	Stable

Abbildung 40 Systeminstabilität nach Matlab

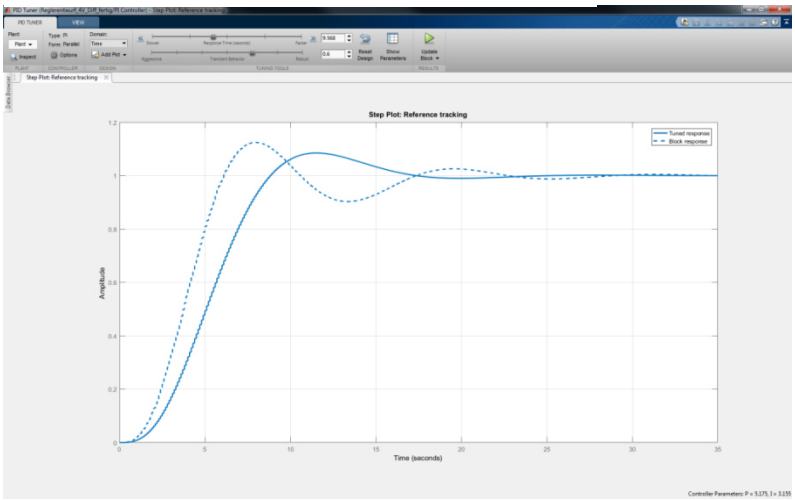


Abbildung 41 PID-Tuner Umgebung

4.1.3 T-Summen-Regel

Ein anderes Regler Entwurfsverfahren basiert auf die Auslegung des Reglers nach der Summe seiner Zeitkonstanten. Dieses Verfahren kann sowohl mit als auch ohne Systemgleichung angewendet werden. Eine Anwendung dieses Verfahrens ohne Systemgleichung basiert auf Einzeichnen zweier gleicher Flächen in die Sprungantwort und Ablesen der Summenzeitkonstante (vgl. Abbildung 42).

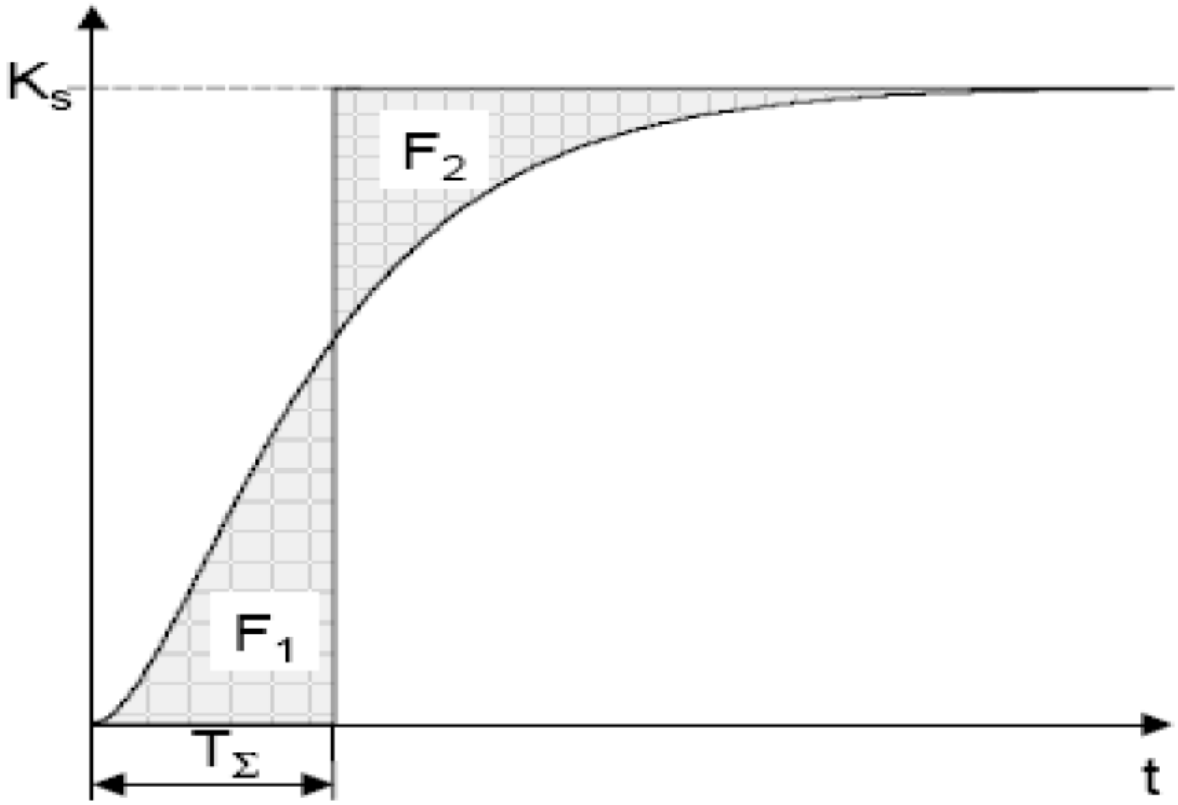


Abbildung 42 Zeichnerisches ermitteln der Summenzeitkonstante [Bate, WS2008/09]

Ein anderer Ansatz ist die Summe der Zeitkonstanten aus der Systemgleichung zu ermitteln.

$$G_{PT3}(s) = \frac{0{,}05605}{s^3 + 2{,}032 * s^2 + 3{,}84 * s + 0{,}9094}$$

Hierfür muss die in der Koeffizienten Darstellung vorliegende Systemgleichung zuerst in die Pol-Nullstellen Darstellung und anschließend in die Zeitkonstantendarstellung überführt werden.

Um die momentan in Koeffizienten Darstellung vorliegende Systemgleichung in die Pol-Nullstellen Darstellung zu überführen ist es notwendig die Nullstellen des Nenner Polynoms zu bestimmen. Dass die Übertragungsfunktion hier im Laplacen Bildbereich vorliegt spielt hierfür keine Rolle. Eine Lösung der Gleichung liefert Matlab oder ein Onlinekalkulationstool, wie es auf www.wolframalpha.com zu finden ist.

Die Lösung des Nenner Polynoms mit Matlab liefert mit dem Befehl „roots“ folgende Lösungen:

```
>> roots([1 2.032 3.84 0.9094])

ans =

  -0.8808 + 1.6087i
  -0.8808 - 1.6087i
  -0.2704 + 0.0000i
```

Abbildung 43 Matlab Lösung des Polynoms

Die Lösung des Nenner Polynoms beinhaltet also zwei imaginäre Lösungen und eine reelle Lösung. Mit dieser Information kann die Systemgleichung in die Pol-Nullstellengleichung überführt werden. Es ist darauf zu achten, dass der Begriff Nullstellen für das Nenner Polynom nicht missverstanden wird mit den Nullstellen wie sie eigentlich in der Regelungstechnik verstanden werden. In der Regelungstechnik wird der Begriff Nullstellen für die Nullstellen des Zählerpolynoms verwendet. Die Nullstellen des Nenner Polynoms werden als Polstellen bezeichnet. Im Folgenden die Pol-Nullstellendarstellung der Systemgleichung.

$$G_{PT3} = \frac{0{,}05605}{(s + 0{,}8808 + j1{,}6087) * (s + 0{,}8808 - j1{,}6087) * (s + 0{,}2704)}$$

Durch umformen der Produktterme kann die Systemgleichung in die Zeitkonstanten Darstellung überführt werden. Die doppelte Polstelle muss hierfür in ein Polynom überführt werden.

$$G_{PT3} = \frac{1}{0{,}2704 * 0{,}8808} * \frac{0{,}05605}{(s + 0{,}8808 + j1{,}6087) * (s + 0{,}8808 - j1{,}6087) * (s + 0{,}2704)}$$

$$G_{PT3} = \frac{0{,}061634}{(0{,}55s^2 + 0{,}52s + 1) * (1 + 3{,}70s)}$$

Die Zeitkonstanten besitzen die Einheit Sekunden und lassen sich als Faktoren für s bestimmen. Die Summenzeitkonstante errechnet sich folgendermaßen:

$$T_\Sigma = 0{,}55ek + 0{,}52sek + 3{,}70sek$$
$$T_\Sigma = 4{,}77$$

Die Regler-Einstellungen für die Summenzeitkonstanten unterteilen sich hier nochmal in zwei Unterschiedliche Tabellen. Zum einen die normalen Einstellregeln von Kuhn und in die schnellen Einstellregeln von Kuhn. [Lutz, 2014] [Bate, WS2008/09]

Tabelle 10 normale Einstellregeln T-Summe

Regler Parameter	Kp	Tn	Tv
P	1/Ks	-	-
PI	0,5/Ks	0,5*T_Σ	-
PD	1/Ks	-	0,33*T_Σ
PID	1/Ks	0,66*T_Σ	0,167*T_Σ

Tabelle 11 schnelle Einstellregeln T-Summe

Regler Parameter	Kp	Tn	Tv
PI	1/Ks	0,7*T_Σ	
PID	2/Ks	0,8*T_Σ	0,194*T_Σ

Die Regelparameter wurden hier nach den schnellen Einstellregeln eines PI-Reglers errechnet.

$$P = \frac{1}{K_s}$$
$$P = \frac{1}{0{,}05605}$$
$$P = 17{,}8$$

$$T_n = 0{,}7 * T_\Sigma$$
$$T_n = 0{,}7 * 4{,}77$$
$$T_n = 3{,}34$$

Analog zu Kapitel 4.1.1 muss hier die Nachstellzeit in einen I-Anteil überführt werden.

$$I = \frac{P}{T_n}$$
$$I = \frac{17{,}8}{3{,}34}$$
$$I = 5{,}33$$

4.2 Verfahren ohne Systemgleichung

Im Gegensatz zu den in Kapitel 4.1 genannten Rechenaufwendigen Verfahren existieren auch Reglerentwurfsverfahren, welche gänzlich ohne Systemgleichung und Rechnungen auskommen. Diese Verfahren können angewendet werden, wenn keine kritischen Anforderungen an die Regelung bestehen und wenn ein falsch dimensionierter Regler nicht zur Zerstörung der Regelstrecke oder Gefährdung von Personen führt.

4.2.1 Einstellregeln nach Takashi

Die Einstellregeln nach Takashi bestehen auf den Einstellregeln nach Ziegler Nichols (Kapitel 4.1.1). Die Einstellregeln nach Takashi berücksichtigen die Abtastrate des Reglers und errechnen die Regelparameter aus Zeiten, welche aus der Sprungantwort abgelesen werden können. [Lutz, 2014]

Tabelle 12 Einstellregeln nach Takashi

Regler	Kp	Tn	Tv
P-Regler	$\frac{T_g}{K_s * (T_u + T)}$	-	-
PI-Regler	$\frac{0{,}9 * T_g}{K_s * (T_u + \frac{T}{2})}$	$3{,}33 * (T_u + \frac{T}{2})$	-
PID-Regler	$\frac{1{,}2 * T_g}{K_s * (T_u + T)}$	$\frac{2 * (T_u + \frac{T}{2})^2}{T_u + T}$	$0{,}5 * (T_u + T)$

Aus der aufgenommenen Sprungantwort können Streckenverstärkung k_s, die Verzugszeit (Ersatztotzeit) T_u und die Ausgleichszeit T_g ermittelt werden. Die Totzeit T_t kann bei Fällen ohne Totzeitelement durch die Ersatztotzeit ersetzt werden.

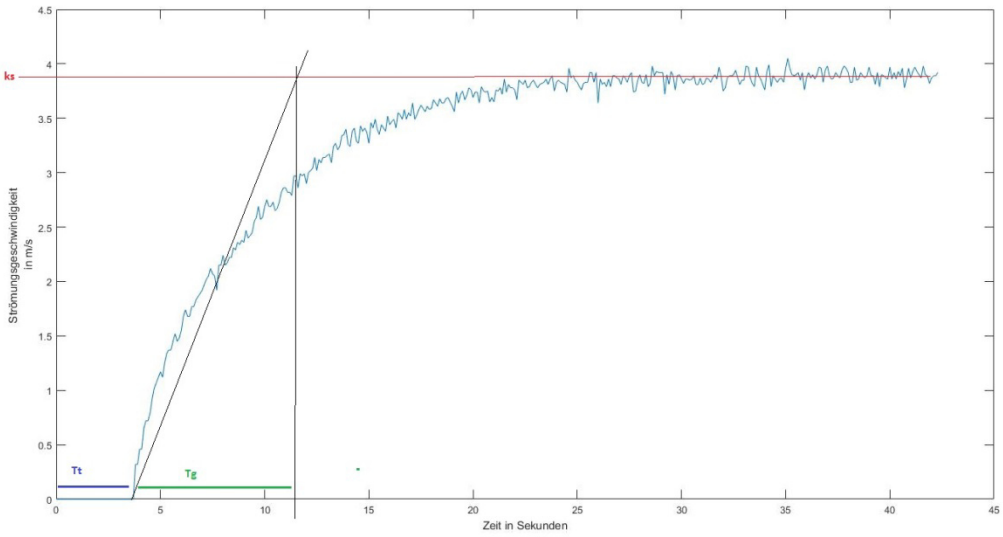


Abbildung 44 Sprungantwort mit eingezeichneten Streckenparametern

Die Regler Parameter für einen PI-Regler mit einer Abtastrate von 0,1 Sekunden ermitteln sich folgendermaßen:

$$P = \frac{0{,}9 * 7}{0{,}06 * (4 + \frac{0{,}1}{2})} = 25{,}93$$

$$T_n = 3{,}33 * \left(4 + \frac{0{,}1}{2}\right) = 13{,}49$$
$$I = \frac{25{,}39}{13{,}49} = 1{,}92$$

4.2.2 Einstellregeln nach Rosenberg

Die Einstellregeln nach Rosenberg sind ähnlich den Einstellregeln von Takashi, nur dass diese die Abtastrate des Reglers nicht berücksichtigen. [Bate, WS2008/09]

Tabelle 13 Einstellregeln nach Rosenberg

Regler	Kp	Tn	Tv
P	$\frac{1}{K_s} * \left(\frac{T_g}{T_u}\right)$	-	-
PI	$\left(\frac{0{,}91}{K_s}\right) * \left(\frac{T_g}{T_u}\right)$	$3{,}3 * T_u$	-
PID	$\left(\frac{1{,}2}{K_s}\right) * \left(\frac{T_g}{T_u}\right)$	$2 * T_u$	$0{,}44 * T_u$

Die Regler Parameter für einen PI-Regler ermitteln sich folgendermaßen:

$$P = \frac{1}{0{,}06} * \left(\frac{7}{4}\right) = 29{,}16$$

$$T_n = \left(\frac{0{,}91}{0{,}06}\right) * \left(\frac{7}{4}\right) = 26{,}54$$
$$I = \frac{29{,}16}{26{,}54} = 1{,}1$$

4.3 Simulation der Regelkreise

Bevor ein Regler an der realen Strecke getestet wird, sollte dieser zuerst in einer Simulation überprüft werden. Die Vorteile hierbei sind zum einen das es wesentlich schneller geht am Computer Regelkreise zu simulieren, anstelle von einzelnen Tests an der realen Strecke. Und zum anderen ist eine Zerstörung der Regelstrecke durch falsch errechnete Parameter ausgeschlossen.
Die einzelnen Einstellparameter wurden mit einem Simulinkmodell wie in Abbildung 39 simuliert:

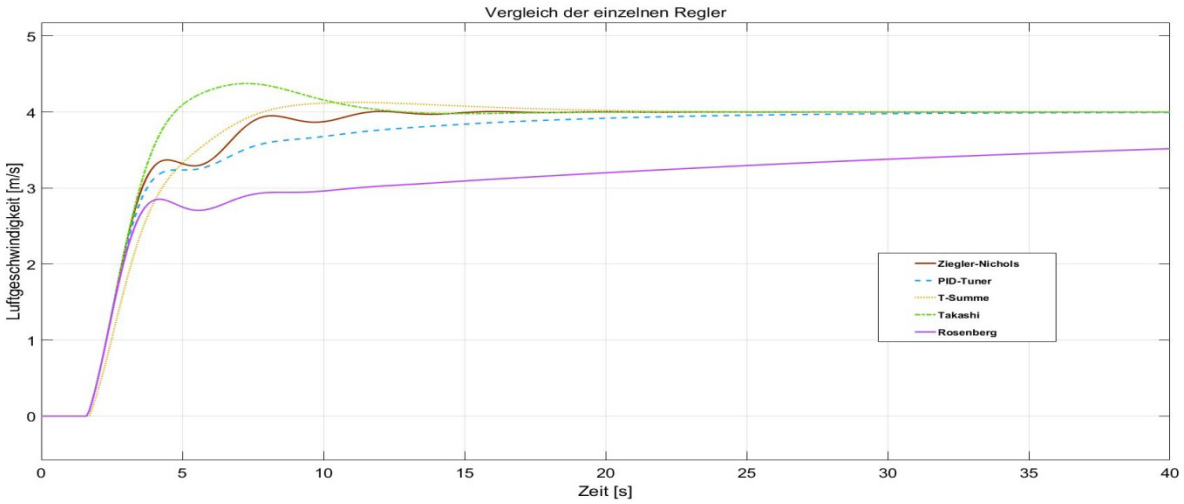


Abbildung 45 Regelparameter im Vergleich

Theoretisch ließe sich jeder Regler verwenden, aber es wurde sich für den Regler nach Ziegler-Nichols entschieden (rot). Der Grund hierfür lag das dieser bezüglich Einregeldauer, Überschwinghöhe, und Anregelzeit den besten Kompromiss darstellt. Auch interessant ist, dass das Verfahren nach Takashi (grün) ebenfalls eine gute theoretische Alternative wäre, obwohl dieses mit ähnlich wenig Rechenaufwand wie das Verfahren nach Rosenberg (violett) auskommt, welches eher bezüglich Einregeldauer zu wünschen übrig lässt.

Um die Simulation zu validieren wurde ein Test an der realen Strecke durchgeführt, Abbildung 46 zeigt einen Vergleich der PID-Tuner Simulation und der realen Strecke:

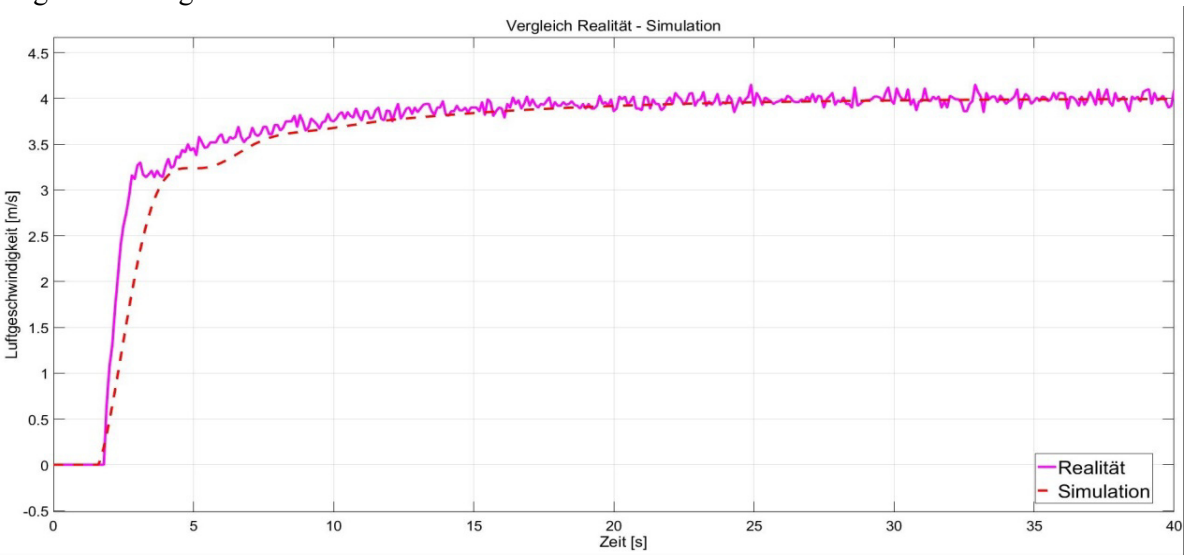


Abbildung 46 Vergleich zwischen Simulation und realer Regelung

5 Programmierung

Für den Lüfter Prüfstand galt es zweierlei Programmierungen vorzunehmen. Zum einen die Programmierung auf den Arduino, welche die Berechnung des Regelungsalgorithmus übernimmt und die Messwerterfassung durchführt. Und zum anderen die Programmierung einer GUI, welche es ermöglicht unabhängig von dem Endgerät über die Serielle Schnittstelle zu kommunizieren. Die Unabhängigkeit vom Endgerät ist insoweit gegeben, dass die GUI mit jeder Seriellen Schnittstelle eines Endgerätes kommunizieren kann, sofern diese bezüglich Baudrate und Trennzeichenfolge übereinstimmen. Es wurde besonders darauf Wert gelegt, dass die GUI nur als Visualisierung und als Mensch-Maschineschnittstelle fungiert. Das heißt das Sie für keinerlei zeitkritische Anwendungen, wie die Durchführung des Regelungsalgorithmus zuständig ist.

5.1 Arduino Programm

Das Programm auf dem Arduino wurde in der Arduino IDE 1.6.9 (Integrated Development Environment) programmiert. Da der Arduino auf einem ATMEGA328p basiert, hätte dieser auch über ATMEL Studios programmiert werden können. Der genaue Programmcode kann im Anhang A gefunden werden.

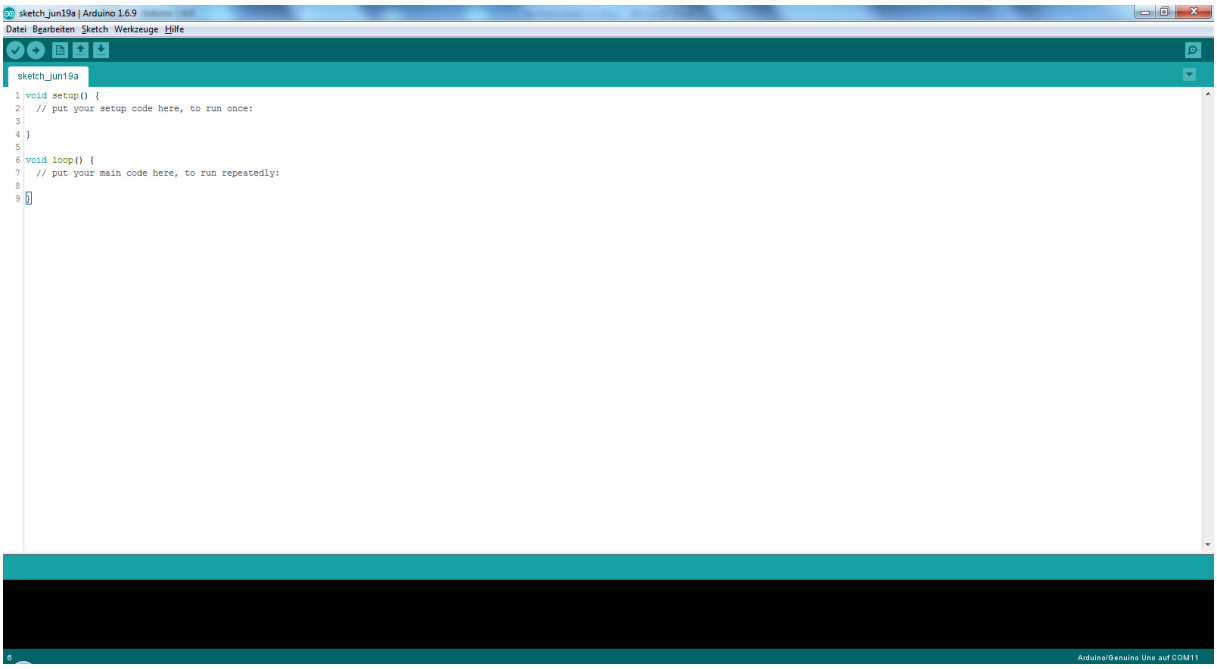


Abbildung 47 Arduino Entwicklungsumgebung

Die Programmiersprache welche in der Arduino IDE verwendet wird ist eine Mischung aus Ansi-C und C#. Die Programmiersprache selbst besticht darin, dass diese mit vielen Beispielen gut dokumentiert ist und darüber hinaus noch eine stetig wachsende Community besitzt. Das Programm teilt sich hier in zwei Schleifen. Zum einen die Setup-Schleife und zum anderen die Loop-Schleife. In der Setup-Schleife werden einmalige Einstellungen vorgenommen, wie das deklarieren eines Analogen Eingangs oder das Festlegen von Grundeinstellungen. In der Loop-Schleife steht der Programmcode, welcher zyklisch durchlaufen wird. Hier befinden sich der Regelungsalgorithmus [Beauregard] und das Senden der Messwerte an die GUI, da dies immer wieder ausgeführt werden muss. Abbildung 48 zeigt einen Programmablaufplan des Arduino Programms.

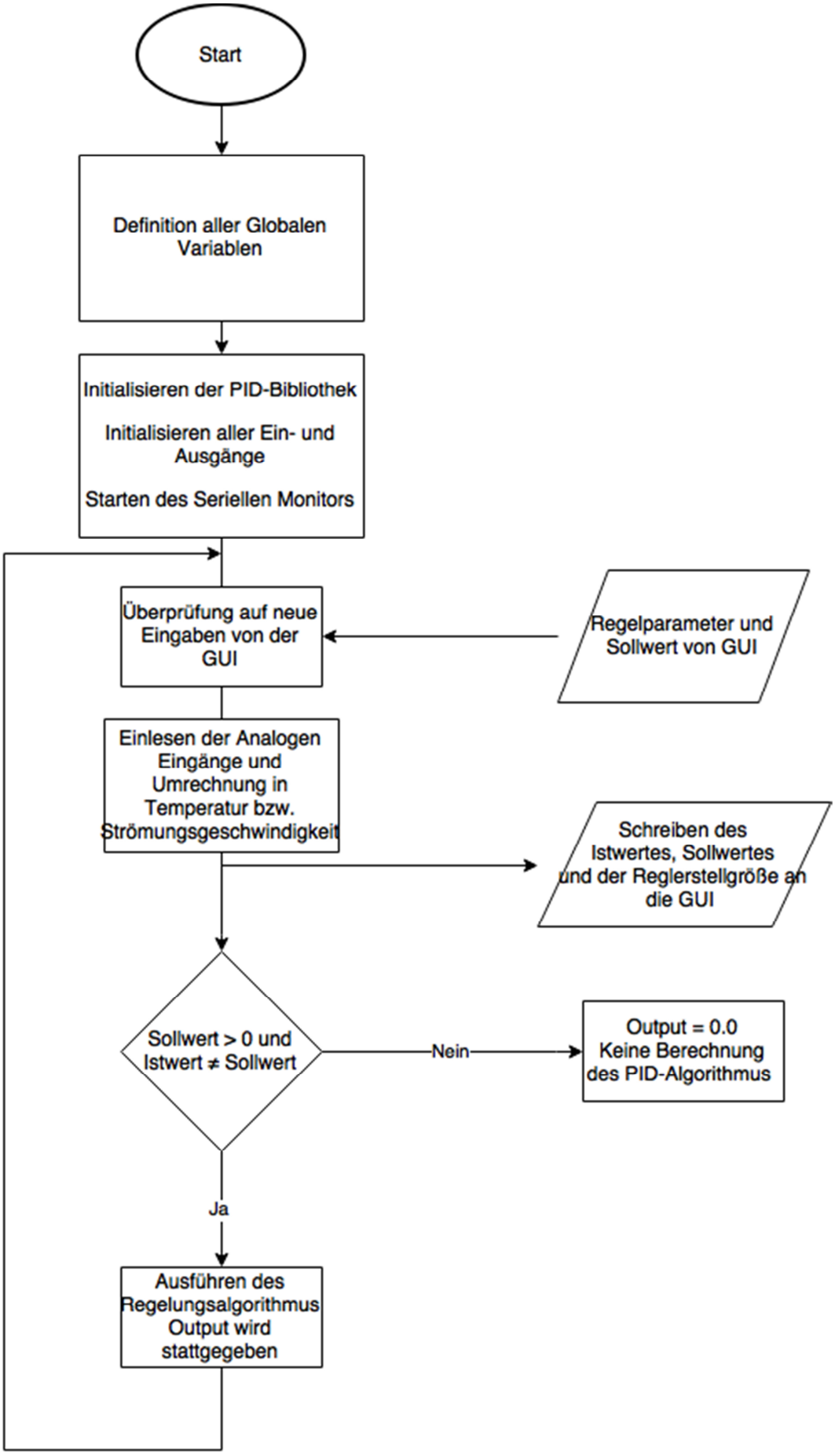


Abbildung 48 Programmablaufplan Arduinoprogramm

5.2 GUI

Um eine Kommunikation zwischen Mensch und Prüfstand zu ermöglichen wurde in der Programmiersprache C# in Visual Studio 2017 eine GUI geschrieben. Über die GUI können die Regelparameter und der Sollwert festgelegt werden.

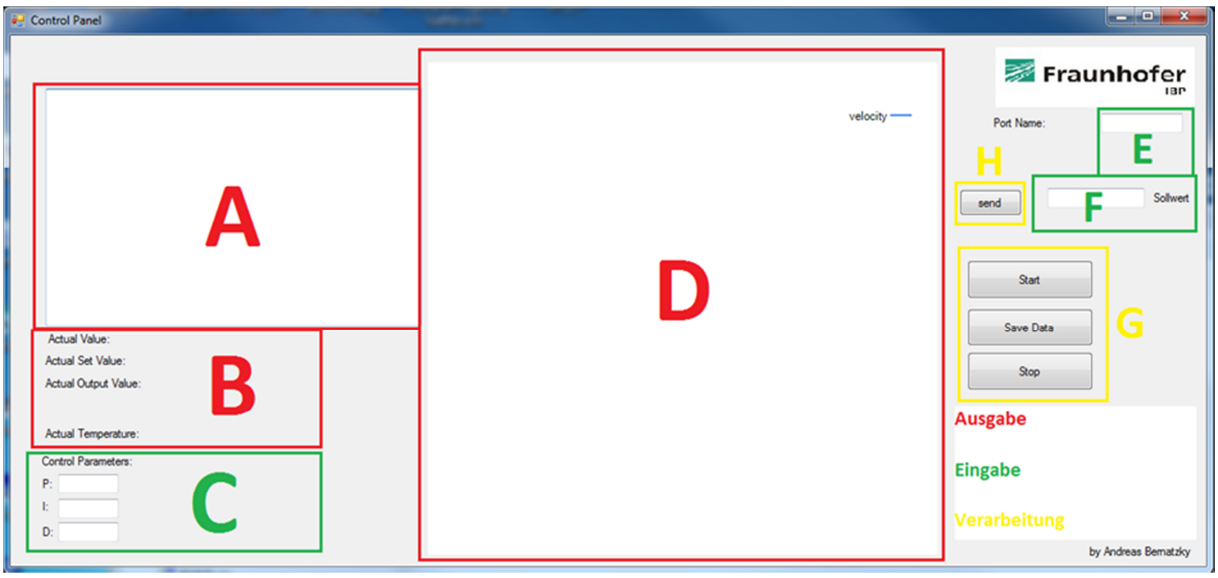


Abbildung 49 GUI unterteilt in Arbeitsbereiche

Die GUI lässt sich in drei Bereiche unterteilen, welchen jeweils die Aufgaben von Eingabe, Ausgabe und Verarbeitung übernehmen (vgl. Abbildung 49). Die Ausgabefelder bestehen hierbei aus drei Feldern. Zum einen wird hier die momentane Regelgröße als Graph gezeichnet (D) und die einzelnen Größen wie die Regelgröße und die Temperatur werden vereinzelt dargestellt (B). Diese Darstellung erfolgt einzeln einmal wird jeder Wert für sich angezeigt und zum anderen werden die aktuellen und vergangenen Werte in einer TextBox mit Zeitstempel dargestellt (A). Die Eingabefelder bestehen aus der Zuweisung des aktuellen Ports (E), an dem der Arduino angeschlossen ist. Die Eingabe der PID-Parameter (C) erfolgt unter der Rubrik „Control Parameters“ die Auswahl des aktuellen Sollwertes (F) neben der TextBox mit der Sollwert Beschriftung. Für die Verarbeitung wurden 4 Felder (G&H) vorgesehen. Send-, Start-,Save und Stop-Button. Der Start-Button dient zum Aktivieren der Seriellen Kommunikation zwischen Arduino und Computer. Wichtig ist hierbei das die Kommunikationsrate von Arduino und GUI identisch ist. Sowohl für den Arduino als auch die GUI wurde eine Baudrate von 9600 festgelegt. Der Save Button speichert den Inhalt der Ausgabe TextBox in eine Txt-Datei. Wenn eine komplett neue Messung begonnen werden soll ist darauf zu achten, dass die davor gespeicherte Measurement.txt Datei umbenannt wird, ansonsten wird diese Überspeichert. Durch klicken des Stop-Buttons wird die Serielle Kommunikation zwischen GUI und Arduino getrennt. Der „Send-Button“ (H) dient zum übergeben eines neuen Sollwertes und der Übergabe der eventuell neuen Regelparameter. Die GUI erhält neue Messwerte im Sekundentakt vom Arduino, welche von der GUI mit einem Zeitstempel bestehend auf der aktuellen Systemzeit versehen werden. Das Zeichnen des Graphen wird durch eine Anwendungsbibliothek übernommen (Chmielewski, 2016). Der Programmcode kann in Anhang B gefunden werden.

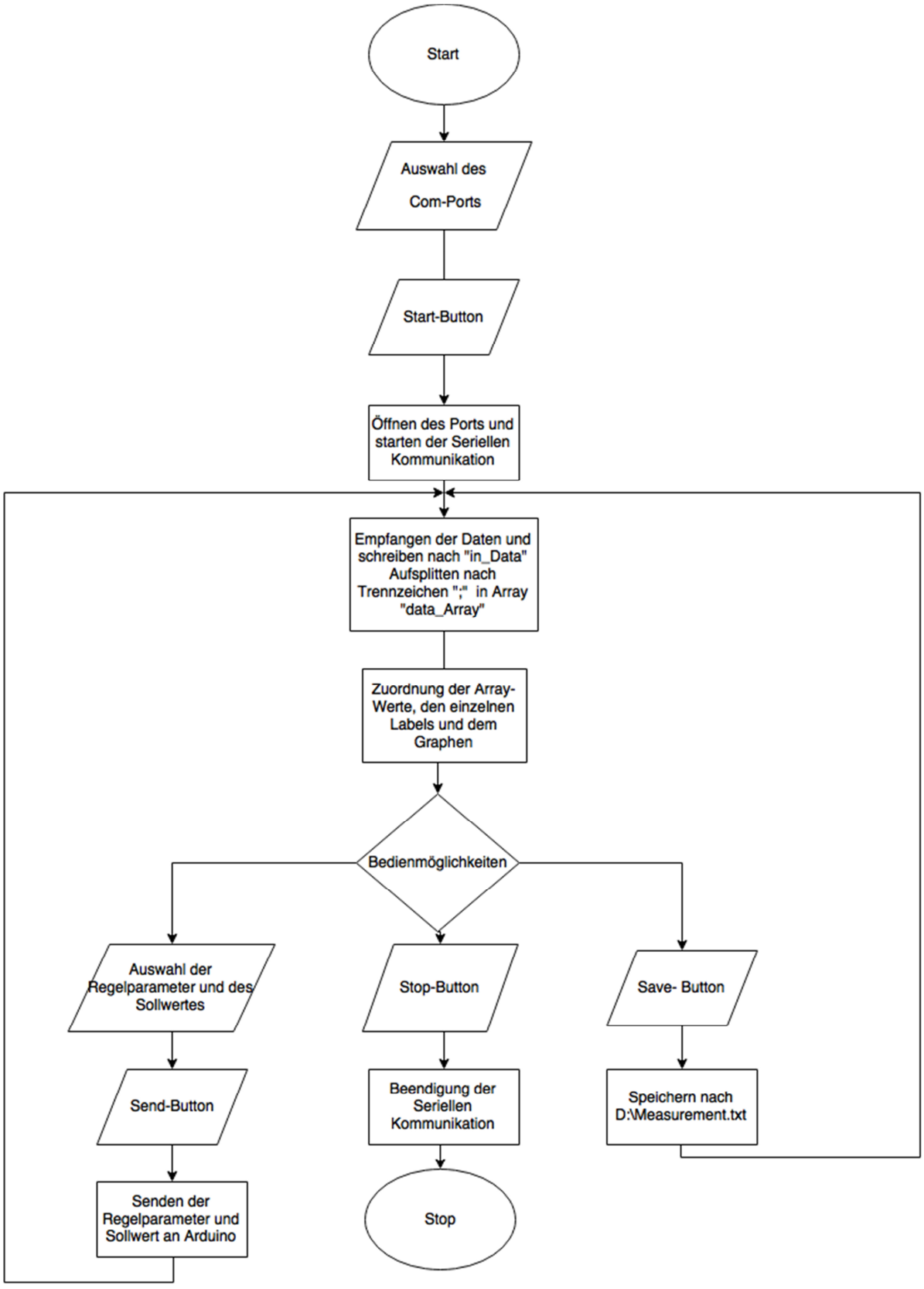


Abbildung 50 Programmablaufplan GUI

6 Bedienungsanleitung des Prüfstandes

Der Lüfter Prüfstand kann an jeder haushaltsüblichen 230 V AC Steckdose angeschlossen werden. Jedoch sollte diese Steckdose mit einem FI-Schutzschalter vom TYP B / B+ abgesichert sein. Ein Anschluss an einer Steckdose ohne einen FI-Schutzschalter von diesem Typ, bedarf eine Absprache mit den Sicherheitsbeauftragten Herrn Martin Lebschy. Bei Änderungen am elektrischen Aufbau des Prüfstandes bedarf es einer Einhaltung der Produkt-Normen: DIN EN 61439-3, Din EN 60670-24 und DIN VDE 0603-1, welche die Richtlinien für Installationsverteiler umfassen, welche von nicht Elektro-Fachkräften bedient werden können sollen [Callondann, 2017]. Nach einer Änderung bedarf es einer erneuten Abnahme von Herrn Martin Lebschy.

Nach Anschluss des Prüfstandes muss der Schaltschrank geöffnet werden und die Sicherung durch umlegen des Schalters ein gesichert werden. Ein aufleuchten der blauen Kontrollleuchte des Thyristorstellers zeigt die Betriebsbereitschaft des Thyristorstellers an.

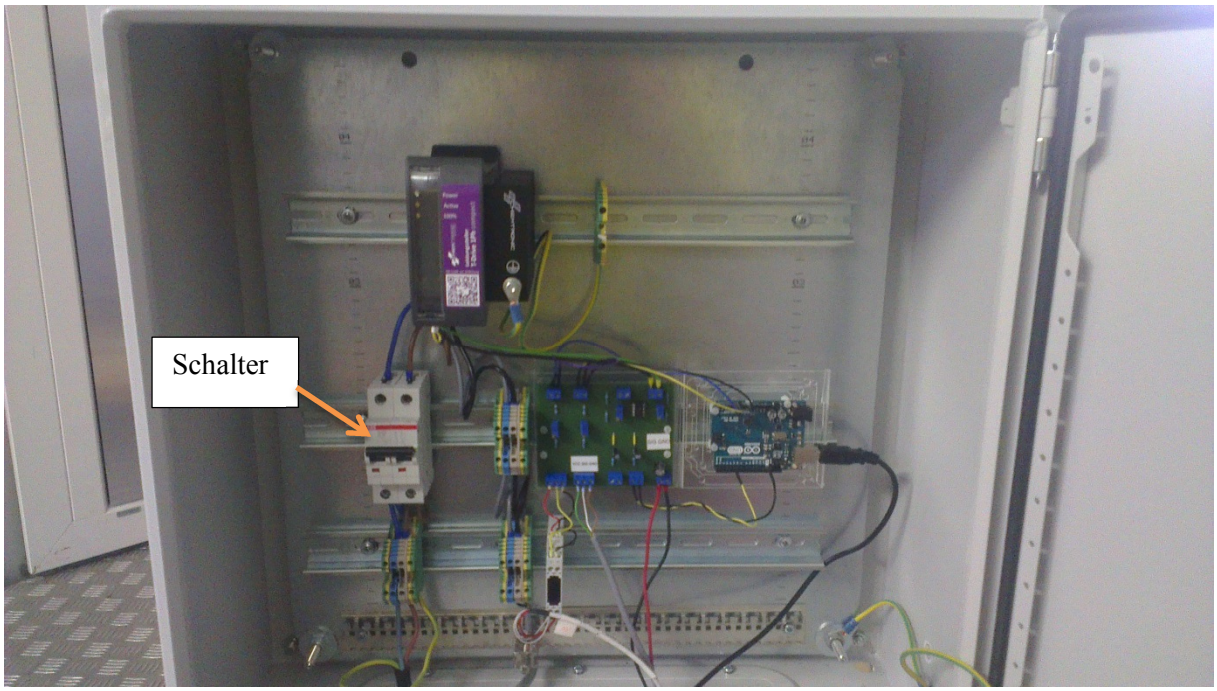


Abbildung 51 Schaltschrank

Für den weiteren Betrieb muss der Schaltschrank wieder geschlossen werden. Zur Kommunikation mit dem Prüfstand wird ein Laptop oder Computer benötigt, welcher mindestens eine USB-Schnittstelle aufweist und über eine nicht schreibgeschützte Partition D:\ verfügt. Diese kann nun mit dem am Prüfstand sich befindenden USB-B Kabel verbunden werden.

Die GUI ArduinoSaveReceive.exe muss sich im selben Ordner mit der Anwendungsbibliothek kayChart.dll befinden. Durch einen Doppelklick auf die ArduinoSaveReceive.exe öffnet diese sich und erwartet nun eine Zuweisung des entsprechenden COM-Ports auf dem der Arduino angeschlossen ist. Dieser muss dem Geräte-Manager unter Windows entnommen werden (Abbildung 52).

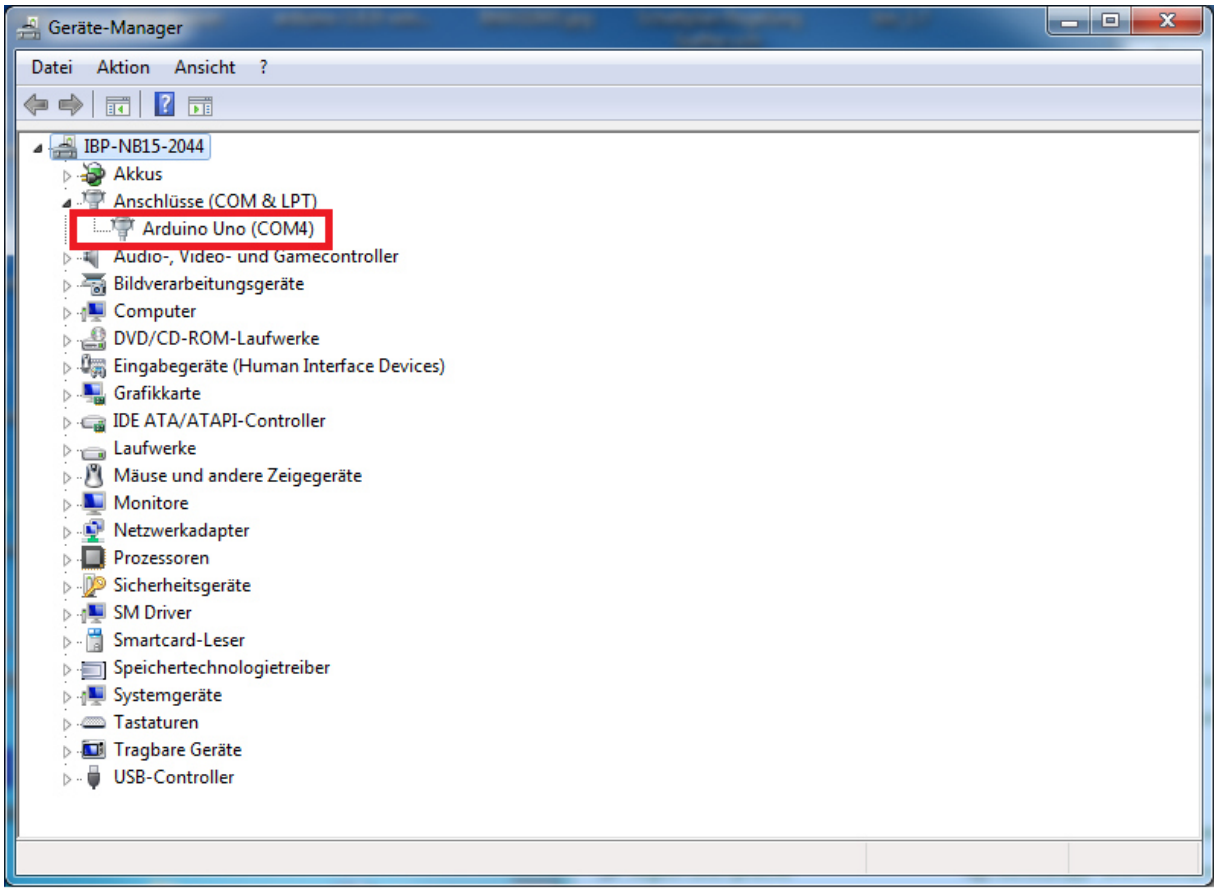


Abbildung 52 Gerätemanager Ansicht der Arduino Uno ist hier COM4 zugewiesen

„COM4“ ist hier jetzt in die GUI in das Feld „Port Name:“ einzutragen. Durch ein klicken auf den Button „Start“ wird die Serielle Kommunikation zwischen Arduino und GUI gestartet. Abschließend können die Regelparameter und ein Sollwert festgelegt werden, welche beide mit dem Button „Send“ an den Arduino übergeben werden. Zum Ausschalten des Prüfstandes muss ein Sollwert von 0,0 m/s übergeben werden. Anschließend kann der Prüfstand durch Aussichern des Schalters und trennen der Versorgungsspannung außer Betrieb genommen werden.

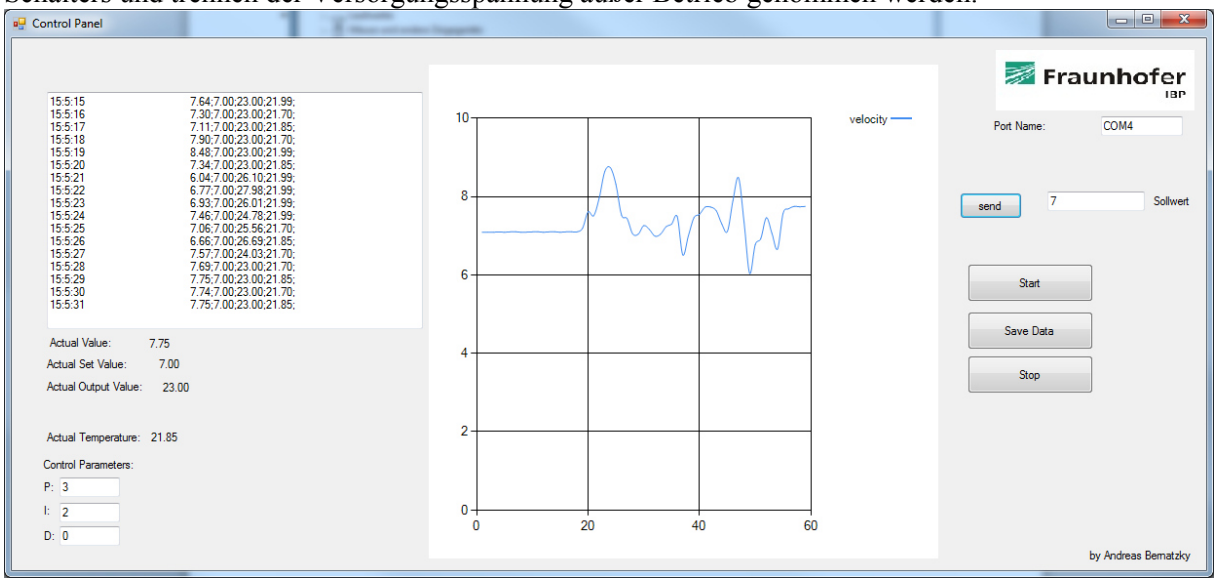


Abbildung 53 GUI im Betrieb

Die Bedienung und Inbetriebnahme des Prüfstandes in Stichpunkten:

1. Herstellen der Netzversorgung durch Einstecken des Schukosteckers.
2. Einsichern des sich im Schaltschrank befindenden Schalters.
3. Verbindung Arduino-Computer herstellen über USB-Kabel.
4. Ausführen von ArduinoSaveReceive.exe.
5. Auswahl des COM-Ports über den Gerätemanager.
6. Starten der Seriellen Kommunikation über den „Start-Button“.
7. Eintragen des Sollwertes und der Regelparameter.
8. Übergeben des Sollwertes und der Regelparameter über den „Send-Button“.
9. (Optional) Speichern der Messwerte über den „Save-Button“.

Die Außerbetriebnahme des Prüfstandes in Stichpunkten:

1. Übergabe des Sollwertes 0,0 m/s mit „Send-Button“.
2. (Optional) Speichern der Messwerte über den „Save-Button“.
3. Aussichern des sich im Schaltschrank befindenden Schalters.
4. Trennen der Netzversorgung durch Ausstecken des Schukosteckers.

7 Diskussion

Ziel der Bachelorarbeit war es, einen mobilen Prüfstand zu entwickeln welcher seine Drehzahl und die daraus resultierende Strömungsgeschwindigkeit im Rohr möglichst präzise regeln kann. Hierfür sollten verschiedene Ansätze zur Systemidentifikation und Reglerentwürfe überprüft und miteinander verglichen werden. Dass dies zu einem guten Regler geführt hat, welcher eine geringe Regelabweichung und gleichzeitig gutes dynamisches Verhalten aufweist, zeigt Abbildung 54.

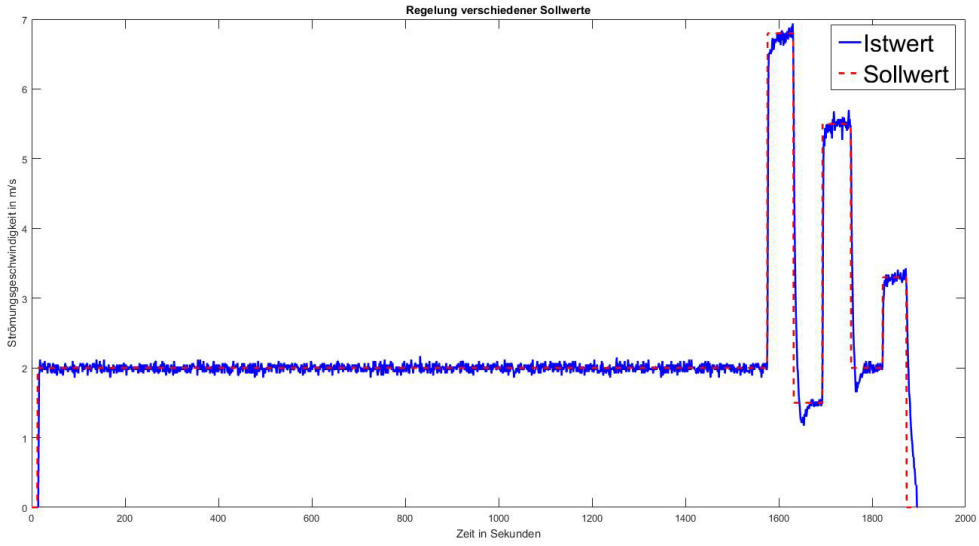


Abbildung 54 Test der Ziegler-Nichols Regelung mit verschiedenen Sollwerten

8 Ausblick

Der Differenzdrucksensor des Lüfter Prüfstands soll in Zukunft durch einen digitalen Differenzdrucksensor ersetzt werden, der über einen RS485-Wandler an den Arduino angebunden werden soll. Zusätzlich soll im Schaltschrank noch ein 230V AC/24V DC Wandler verbaut werden, welcher dafür sorgt dass kein zusätzliches 24 V DC Netzteil für die Messsensorik mehr benötigt wird. Um den Prüfstand mit der GUI von überall am Institut aus bedienen zu können soll noch ein USB-Server zur Verwendung kommen. Dieser integriert ein USB Gerät über TCP/IP so als wäre es direkt am Computer angeschlossen, was für die serielle Arbeitsweise der GUI notwendig ist. So wird ermöglicht, dass der Prüfstand innerhalb des Intranets also von überall am Institut bedient werden kann (vgl. Abbildung 55).

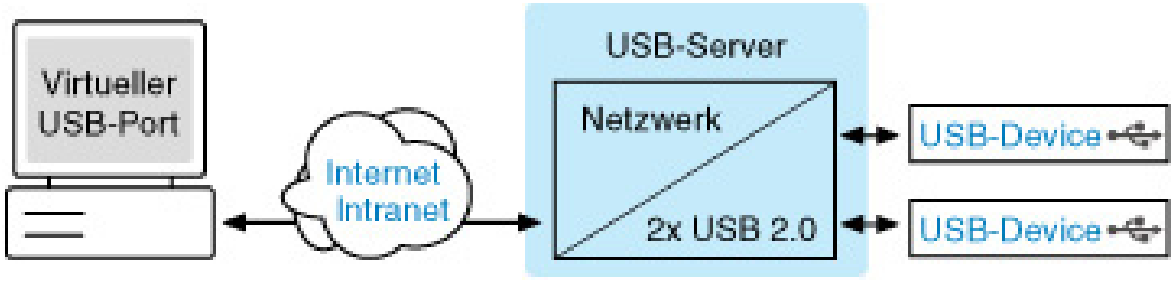


Abbildung 55 Arbeitsweise des USB-Servers [W&T]

Anhang A Arduino Programm

```
/**********************************ArduinoSaveReceiveMainProgram**********************
************************************

 * Firma: Fraunhofer IBP

 * Autor: Andreas Bernatzky

 * Programm: ArduinoSaveReceiveMain1.7

 * Datum: 24.06.17

 * Beschreibung: Regelung des Lüfterprüfstands(siehe Bachelorarbeit Andreas Bernatzky)

 *               Dieses Programm funktioniert nur im Zusammenhang mit der
dazugehörigen

 *               GUI "ArduinoSaveReceive.exe".

 *               Das Programm ist in der Lage den Lüfterprüfstand auf einen
gewünschten Sollwert zu regeln.

 *               Da die PID Parameter für das in der Bachelorarbeit behandelte System
(siehe Bachelorarbeit Andreas Bernatzky) bestimmt wurden, ist es eher unwahrscheinlich

 *               das diese Parameter für andere Systeme ebenfalls funktionieren.
Hierfür muss eine erneute Systemidentifikation mit anschließender Reglerauslegung

 *               erfolgen, was in der Bachelorarbeit erklärt wird.

 *               Die Reglerparameter können jedoch theoretisch auch empirisch bestimmt
werden.

 * Hardware:     Das Programm ArduinoSaveReceive1.7 wurde für den Arduino UNO
geschrieben und gedebuggt.

 *               Da die Kommunikation mit der GUI rein Seriell funktioniert sollte
diese für jeden Arduino Typ funktionieren,

 *               welcher über mindestens eine UART verfügt. Bei Arduino Modellen mit
mehr als einer UART (Mega) muss darauf

 *               geachtet werden, das die Messwerte an die Haupt UART (UART 0)
übergeben wird.

 *               1 x Arduino UNO oder Mega

 *               1 x USB Typ B Kabel

 *               1 x Regelsensor Schmidsonde oder Differenzdruck Sensor (das Programm
muss hierfür angepasst werden unter der Sektion "Sensor"

 *

 * Software:     Auf dem Arduino muss sich dieses Programm ArduinoSaveReceiveMain1.7
befinden.
```

```
 *               Für eine erfolgreiche Kommunikation mit dem Programm muss sich auf
dem Computer das Programm "ArduinoSaveReceive.exe" befinden

 *               Eine Bedienung der GUI kann der Dokumentation entnommen werden.

 *

 *

 * Version: 1.7

**********************************************************************************
**********************************/

/*/**********************************PID_v1_OWN**********************************
********************

 *

 *Die Bibliothek PID_v1_OWN.h wurde so abgeändert, das diese anstelle mit dem
Variablentyp double mit Float arbeitet. Ansonsten kommt es zu Komplikationen mit dem
Befehl ToFloat().

 *Die Bibliothek PID_v1.h würde nämliche eine Variable vom Typ double erwarten.

 */



#include <PID_v1_OWN.h> // Bibliothek welche den Regelungsalgorithmus enthält

float dichte;

float Temp;

float pressure ;

float Setpoint, Input, Output;

float consKp=0.0, consKi=0.0, consKd=0.0;  // Festlegung der PID Parameter (Werden
durch die GUI übergeben)

String inString = "";    // Empfangsstring


long previousTime = 0;

long interval = 1000; // Festlegung der Messintervalle 1000 entspricht pro Sekunde ein
Messwert, 2500 entspräche alle 2,5 sekunden ein messwert etc.


//Deklarierung der PID Variablen

PID myPID(&Input, &Output, &Setpoint, consKp, consKi, consKd, DIRECT);
```

```
 /*********************************Setup
Schleife*******************************************************************************
*********

 *

 * Die Setup Schleife wird beim Programmstart einmalig durchlaufen.

 * Festlegung der GPIO-Pins

 * INPUT pins müssen für analoge Signale vom Typ Ax am Arduino sein. Für digitale
Signale kann jeder beliebige PIN verwendet werden.

 * Ausgespart werden sollten immer PIN 0 (RX), PIN1 (TX) und PIN 13 (in Reihe
geschaltene LED am Arduino)

 *

*******************************************************************************
*************************************************/

void setup() {


pinMode(5,OUTPUT); // Deklarierung PIN 5 als Ausgang

pinMode(A0,INPUT);  // Deklarierung PIN A0= als Eingang

myPID.SetMode(AUTOMATIC);

myPID.SetOutputLimits(23, 128); // maximales Stellsignal für Motor ist 5 V


Serial.begin(9600); // Initialisiere Seriellen Monitor

 }

/*********************************Loop
Schleife*******************************************************************************
*********

 *

 * Die Loop-Schleife wird kontinuierlich sich immer wiederholend durchlaufen. Bei
Änderungen sei hierauf zu achten, dass das Programm

 * sequenziell arbeitet ( von oben nach unten ).

 *

 *

*******************************************************************************
*************************************************/
```

```
void loop() {


/**********************************Zustandsvariable
speichern*********************************************************************************

 *

 *Der Befehl Serial.readStringUntil(;) liest empfangene Daten vom Seriellen Monitor
bis zum auftreten eines ";" als Trennzeichen aus und schreibt diese in einen String.

 *Mit dem Befehl toFloat() wird dieser String in eine Variable vom Typ Float
umgewandelt, welche weitere Berechnungen ermöglicht

 *

*********************************************************************************
************************************************/



    while (Serial.available() > 0) {

     int inChar = Serial.read();


     if (inChar != '\n') {


    consKp = Serial.readStringUntil(';').toFloat();

    consKi = Serial.readStringUntil(';').toFloat();

    consKd = Serial.readStringUntil(';').toFloat();

    Setpoint = Serial.readStringUntil(';').toFloat();

    myPID.SetTunings(consKp,consKi,consKd);

     }


     else {



    inString = "";

     }
```

```
  }



 /*********************************Sensor
Rückführung*************************************************************************
*********

 Diese Sektion entspricht Regelungstechnisch dem Istwert und gleichzeitig der
Reglerrückführung.

 Bei Verwendung verschiedener Sensoren ist darauf zu achten hier Änderungen
vorzunehmen

 Beispiel für analogen Schmidsensor: Input= (analogRead(A0)/1023.0)*20.0*0.74

 Die 20.0 steht für die maximal messbare Luftgeschwindigkeit des verwendeten Sensors.

 Die 0.74 steht für den Profilfaktor, dieser muss für jeden Rohrquerschnitt korrekt
angepasst werden und kann der Bedienungsanleitung

 von jedem Schmidsensor entnommen werden. 0.74 sei hier beispielsweise für ein Rohr
mit einem 100mm Innendurchmesser verwendet


*************************************************************************************
************************************************/

 int zaehl=0;          // Zählervariable für for schleife

 int mwleng = 2;   // Gibt vor wie oft gemessen werden soll

 float adcAverage; // Hier werden die 2 Messwerte aufsummiert

 float adcVal;     // Enthält den gemittelten Wert

 float adcValue;   // Misste den aktuellen Wert, welcher anschließend aufsummiert wird

 float adcAverage1;

 float adcVal1;

 float adcValue1;

 for(zaehl=0; zaehl <mwleng; zaehl++){

   adcValue=analogRead(A0);

   adcValue1=analogRead(A1);

   adcAverage += adcValue;

   adcAverage1 +=adcValue1;

 }
```

```
adcVal = adcAverage/ mwleng;

adcVal1 = adcAverage1/mwleng;

Temp= (adcVal1 * (300/1023.0)-100.0);

dichte = 101600/(287.058*(273.15+Temp)); // Berechnung der momentanen Luftdichte

                                         // 287.058 spezifische Gaskonstante trockene
Luft

                                         // 101600 pa Luftdruck Holzkirchen

 pressure= (((adcVal)/1023.0)*200);

 Input = (sqrt((2*pressure)/dichte)*0.816); // Sensorsignal ist regelungstechnisch
mein momentaner Istwert

                                         // Input entspricht meiner momentanen
Strömungsgeschwindigkeit

 /*********************************Kommunikation mit
GUI*************************************************************************

 *Die GUI kommuniziert mit dem Arduino über eine Serielle Schnittstelle(UART) diese
kann dem Gerätemanager von Windows entnommen werden

 *und entspricht für die GUI, dem Comport wo der Arduino angeschlossen ist. Wichtig
ist das zwischen jeder an die GUI übergebene Variable ein

 *";" als Trennzeichen erfolgt

 *Die if abfrage dient dazu um den code nicht mit einem delay bremsen zu müssen
(Sekunden messwerte) Ein delay hätte zu folge das der komplette code

 *praktisch "einfriert" für eine sekunde. Die Berechnung der Regelparameter würde
hierbei jedoch auch eingeforen und nur noch sekündlich ausgeführt,

 * werden. Die Regelparamerter werden bedingt durch die Bibliothek mit einer
SampleTime von 100ms berechnet.

 *

 *

**********************************************************************************
*************************************************/

unsigned long currentTime = millis();

if(currentTime - previousTime > interval) {

  previousTime = currentTime;

 Serial.print(Input); //Regelgröße
 Serial.print(";");
 Serial.print(Setpoint); //Sollwert
 Serial.print(";");
 Serial.print(Output); //Momentanes PWM Ausgangssignal 0-255
 Serial.print(";");
```

```
 Serial.print(Temp); // Momentane Temperatur
 Serial.println(";");

}

 /*********************************Berechnung
Stellgröße*************************************************************************
*

 * Hier wird die Stellgröße des Arduinos bei jedem Loop-Schleifen durchlauf neu
berechnet

*****************************************************************************
*****************************************************/


if(Setpoint == 0.00){

  Output=0.00;

  analogWrite(5,Output);

}

else

{

  myPID.Compute();

  analogWrite(5,Output);

}

}
```

Anhang B Visual Studio C # GUI-Programm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using rtChart;
using System.Diagnostics;
using System.IO.Ports;



namespace ArduinoSaveReceive
{
    public partial class Form1 : Form
    {

        private SerialPort myport;
        private DateTime datetime;
        private string in_data;    // davor private string
        String[] dataArray;  // beinhaltet alle messwerte für die übergabe
        public Form1()
        {
            InitializeComponent();
        }

        kayChart serialDataChart;
        private void start_btn_click(object sender, EventArgs e)
        {

            myport = new SerialPort();
            myport.BaudRate = 9600;
            myport.PortName = port_name_tb.Text;
            myport.Parity = Parity.None;
            myport.DataBits = 8;
            myport.StopBits = StopBits.One;
            myport.DataReceived += myport_DataReceived;
            myport.DataReceived += new
SerialDataReceivedEventHandler(serialDataReceivedEventHandler);
            try {

                myport.Open();
                data_tb.Text = " ";
            }
                catch (Exception ex){
                MessageBox.Show(ex.Message, "Error Wrong COM Port selected");
            }

        }

        void myport_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {


            in_data = myport.ReadLine(); // Schreibe empfangene Daten in Variable
in_data
```

```
             dataArray = in_data.Split(new[] { ";" },
StringSplitOptions.RemoveEmptyEntries);  //Splittet eingehendes vom Serial monitor

// in ein Array mit Trennzeichen ;
             /*  for (int i = 0; i < dataArray.Length; i++)
               {
                   dataArray = dataArray[i];
               }
           */


            this.Invoke(new EventHandler(displaydata_event));

        }



        /// <summary>
        ///
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void serialDataReceivedEventHandler(object sender,
SerialDataReceivedEventArgs e)
        {


            // initialization of chart update
            double data;  // übergabevariable an Graph
            bool result = Double.TryParse(dataArray[0], out data);
            if (result)
            {
                serialDataChart.TriggeredUpdate(data/100); // Schreibe in Diagramm
/100 um z.B. 273.15 nicht als 27315 zu interpretieren

            }
        } // Uebergeben des Werts an Diagramm

        private void Form1_Load(object sender, EventArgs e) // Zeichne neues Diagramm
        {
            serialDataChart = new kayChart(chart1, 60);
            serialDataChart.serieName = "velocity";
        }
//

/// <summary>
/// //////////
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>

private void displaydata_event(object sender, EventArgs e)   //Darstellung in textbox
        {

            datetime = DateTime.Now;
            string time = datetime.Hour + ":" + datetime.Minute + ":" +
datetime.Second;
            data_tb.AppendText (time + "\t\t\t" + in_data  +"\n"); //"\t\t\t"  //
Multilineversion
```

```
            // int data_value = Convert.ToInt32(dataArray[0]);  // Grafische
Darstellung der Progressbar macht Mucken bei mehr als einer Übergabe vom Serialport
            // value_pb.Value = data_value;   // Uebergabe an progressbar
             val_lbl.Text = dataArray[0];   // Seperate Anzeige des Momentanen Wertes
angeordnet direkt ueber Graph
             set_lbl.Text = dataArray[1];  // Seperate Anzeige des Momentanen Set
Values
             out_lbl.Text = dataArray[2];  // Separate Anzeige des Momentanen Outputs
             temp_lbl.Text = dataArray[3]; // Separate Anzeige für die Momentane
Temperatur

        private void send_btn_Click(object sender, EventArgs e)
        {
            string text = send_tb.Text;
            string textp = P_tb.Text;
            string texti = I_tb.Text;
            string textd = D_tb.Text;

            myport.WriteLine(";");
            myport.WriteLine(textp);
            myport.WriteLine(";");
            myport.WriteLine(texti);
            myport.WriteLine(";");
            myport.WriteLine(textd);
            myport.WriteLine(";");
            myport.WriteLine(text);

        }

        private void save_btn_Click(object sender, EventArgs e)
        {
            try
            {
                string pathfile = @"D:\";     // Verzeichnis -> so anpassen das für
jeden Computer gleich
                string filename = "Measurement  " + datetime.Day + "." +
datetime.Month + "." + datetime.Year + ".txt";                          // Dateiname
                System.IO.File.WriteAllText(pathfile + filename, data_tb.Text);
//Zugriffsbefehl auf Windows
                MessageBox.Show("Data has been saved to" + pathfile, "Save File");
            }
            catch (Exception ex3)
            {
                MessageBox.Show(ex3.Message, "Error");
            }
        }

        private void stop_btn_Click(object sender, EventArgs e)
        {
            try
            {
                myport.Close();  // Schließt den Seriellen Port ->
                MessageBox.Show("->!!SPEICHERN NICHT VERGESSEN!!<-");
            }
            catch (Exception ex2)
            {
                MessageBox.Show(ex2.Message, "Error");
            }
        }
    }
```

9 Literaturverzeichnis

Albers, K. (14. Juni 2010). *Mikrcontroller.net.* Abgerufen am 8. Mai 2017 von https://www.mikrocontroller.net/attachment/79923/PWM_Tiefpass_Restwelligkeit.pdf

Arduino. (kein Datum). Abgerufen am 15. 5 2017 von https://www.arduino.cc/en/Main/arduinoBoardUno

Atmel. (November 2015). *www.atmel.com.* Abgerufen am 8. Mai 2017 von http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf

Bate, B. (WS2008/09). *FH Dortmund*. Abgerufen am 5. Mai 2017 von http://www.kahlert.com/web/download/pid_einstellregeln.pdf

Beauregard, B. (kein Datum). *Github*. Abgerufen am 4. März 2017 von https://github.com/br3ttb/Arduino-PID-Library

Berger, P. M. (2001). *Grundkurs der Regelungstechnik.* Pullach: Libri.

Callondann, K. (16. Juni 2017). CE-Kennzeichnung an Unterverteilung fehlt. *de Das Elektrohandwerk*, S. 21.

Chmielewski, S. (30. 11 2016). *nuget*. Abgerufen am 18. April 2017 von https://www.nuget.org/packages/kayChart.dll/

diy_bloke. (30. Oktober 2012). *www.instructables.com*. Abgerufen am 15. März 2017 von http://www.instructables.com/id/Arduino-controlled-light-dimmer-The-circuit/

ELMTEC Ingenieursgesellschaft mbH. (kein Datum). *http://www.elmtec.de/produkte/pdf/staurohr%20fest.pdf.* Abgerufen am 14. April 2017 von http://www.elmtec.de/produkte/pdf/staurohr%20fest.pdf

Gaicher, H. (2015). *AVR-Mikrocontroller.* tredition GmbH.

Gunt Hamburg. (kein Datum). *Gunt Hamburg*. Abgerufen am 19. 07 2017 von http://www.gunt.de/index.php?option=com_gunt&task=gunt.list.category&product_id=548&lang=de

Hoffman, J. (1999). *Handbuch der Messtechnik.* München, Wien: Carl Hansen.

Hoffmann, J. (1999). *Handbuch der Meßtechnik.* München,Wien: Carl Hanser.

Homann, D. I. (21. März 2016). *Modellbildung, Identifikation und Reglerentwurf für eine pneumatische Lageregelung.* Abgerufen am 23. April 2017 von www.ifr.ing.tu-bs.de: https://www.ifr.ing.tu-bs.de/static/files/lehre/labore/rtp1/Skript_Pneumatische_Lageregelung.pdf

Junge, G. (2011). Einführung in die Technische Strömungslehre. München: Carl Hanser.

Kümmeke, H. (kein Datum). *www.wikipedia.de*. Abgerufen am 24. April 2017 von https://de.wikipedia.org/wiki/Regelungstechnik

Ljung, L. (November 2000). *mathworks.* Abgerufen am 30. Mai 2017 von http://www.mathworks.com

Lutz, W. (2014). Taschenbuch der Regelungstechnik. In W. Lutz, *Taschenbuch der Regelungstechnik* (S. 357). Europa-Lehrmittel.

mborchers. (8. April 2008). *http://www.elektronik-magazin.de/page/der-i2c-bus-was-ist-das-21*. Abgerufen am 15. Mai 2017 von http://www.elektronik-magazin.de/page/der-i2c-bus-was-ist-das-21

MDUA GmbH & CO. KG. (.). *www.mdua-messtechnik.de.* Abgerufen am 14. April 2017 von mdua-messtechnik: http://www.mdua-messtechnik.de/html/content/06-staurohre-prandtl-messkreuze/DEBIMO-Luftstroemungs-Messlanzen.pdf

Mikrocontroller.net. (12. März 2017). Abgerufen am 8. Mai 2017 von https://www.mikrocontroller.net/articles/Pulsweitenmodulation#DA-Wandlung_mit_PWM

myHDL. (17. Mai 2012). Abgerufen am 15. Mai 2017 von http://old.myhdl.org/doku.php/projects:uart_rs232_receiver_transmitter

Nestler, B. (kein Datum). *Hochschule Karlsruhe.* Abgerufen am 2017. Juni 6 von http://www.iwi.hs-karlsruhe.de/~nebr0001/ice/nestler/fileadmin/files/ModSim/Notes/ModSim-Skript3.pdf

Ottens, M. (Sommersemester 2008). *www.yumpu.com.* Abgerufen am 20. Mai 2017 von https://www.yumpu.com/de/document/view/2091502/praktische-verfahren-zur-experimentellen-systemidentifikation

Paschotta, D. R. (7. Dezember 2012). *www.energie-lexikon.info*. Abgerufen am 8. April 2017 von https://www.energie-lexikon.info/phasenanschnittsteuerung.html

Schmidttechnology. (kein Datum). *www.schmidttechnology.de.* Abgerufen am 17. April 2017 von http://schmidttechnology.de/de/sensorik/download/Schmidt_sensorik_DE.pdf

Schnabel, P. (kein Datum). *https://www.elektronik-kompendium.de/sites/slt/0210151.htm*. Abgerufen am 8. Mai 2017 von https://www.elektronik-kompendium.de/sites/slt/0210151.htm

Setra. (kein Datum). *www.setra.com.* Abgerufen am 3. Mai 2017 von https://www.setra.com/products/pressure/model-267mr-multi-configurable-low-differential-pressure-transducer-0

Texas Instruments. (November 1998). *www.ti.com.* Abgerufen am 8. Mai 2017 von http://www.ti.com/lit/an/spra490/spra490.pdf

W&T. (kein Datum). *www.Wut.de*. Abgerufen am 1. Juli 2017 von https://www.wut.de/e-53642-ww-dade-000.php

Wagner, A. (kein Datum). *www.phillippi-trust.de*. Abgerufen am 16. Mai 2017 von http://www.philippi-trust.de/hendrik/braunschweig/wirbeldoku/hagen_poi.html

wikibooks. (27. Juli 2015). *https://de.wikibooks.org/wiki/Einf%C3%BChrung_in_die_Systemtheorie/_%C3%9Cbertragungsfunktion#Zeitkonstantendarstellung*. Abgerufen am 8. Juni 2017 von https://de.wikibooks.org/wiki/Einf%C3%BChrung_in_die_Systemtheorie/_%C3%9Cbertragungsfunktion#Zeitkonstantendarstellung

www.pressebox.de. (kein Datum). Abgerufen am 2. Mai 2017 von https://www.pressebox.de/pressemitteilung/schmidt-technology-gmbh/Thermische-Anemometrie/boxid/385059

Zentgraf, P. D. (2015). Abgerufen am 7. Juni 2017