Bachelorthesis im Studiengang Produktionstechnik

Entwicklung eines grafisch-orientierten Analyse-Werkzeugs zur Auswirkung von Pol-/Nullstellenkonfiguration auf regelungstechnische Kenngrößen

Fabian Tutschka

12. März 2012

Prüfer
Prof. Dr.-Ing. Peter Zentgraf M.Sc.

Zweitprüfer
Prof. Dr.-Ing. Franz Plötz

Erklärung

Erklärung des Verfassers der Bachelor Arbeit gemäß § 35,7 RaPo zum Thema:

„Entwicklung eines grafisch-orientierten Analyse-Werkzeugs zur Auswirkung von Pol-/Nullstellenkonfiguration auf regelungstechnische Kenngrößen“

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum	Unterschrift

Zusammenfassung - Im Rahmen der vorliegenden Arbeit wird die Entwicklung einer Matlab Bedienoberfläche beschrieben, welche die Analyse eines dynamischen Systems ermöglicht, sowie Möglichkeiten zum Reglerentwurf bietet.
Vor der Reglerauslegung spielt die Systemanalyse einer Regelstrecke eine wichtige Rolle. Dabei stehen die Systemdynamik, die Stabilität, oder auch das Verhalten bei $t \to \infty$ im Vordergrund. Diese Eigenschaften können mit den regelungstechnischen Kenngrößen: Bode Diagramm, Ortskurve, Sprungantwort sowie Impulsantwort abgeschätzt werden. Das in dieser Arbeit entwickelte Tool ermöglicht eine sofortige Aktualisierung und Neudarstellung aller vier genannten Größen bei Änderung der Übertragungsfunktion in der komplexen Ebene.
Im zweiten Abschnitt der Bachelorarbeit werden die zur Verfügung stehenden Möglichkeiten der Reglerauslegung erläutert. Hier kann zwischen einem Regelkreis mit oder ohne Stellsignallimitierung ausgewählt werden. Die möglichen Eingangssignale für die Führungs- und Störgröße sind die Sprungfunktion, Impulsfunktion, eine beliebig wählbare zeitabhängige Funktion sowie ein sich im Matlab Workspace befindliches Signal. Für das Signal der Störung steht zusätzliches Rauschen zur Verfügung. Die Berechnung der Reglerübertragungsfunktion erfolgt entweder direkt aus der Strecken- und Führungsübertragungsfunktion oder über Reglerentwurfsverfahren nach Ziegler und Nichols sowie nach Chien, Hrones und Redwick. Ein bestmöglicher PID Regler kann über die Methode der kleinsten Fehlerqudadrate berechnet werden.

Abstract - The work presented here describes the development of a Graphical User Interface in Matlab, which allows the analysis of a dynamic system as well as offers options for controller design.
Before the controller is developed, the analysis of the plant controlled system plays a major role. Here, the main focus is on system dynamics, stability as well as the behaviour at $t \to \infty$. These attributes can be evaluated with the help of key metrics of control engineering like the Bode diagramm, the Nyquist plot, step resopnse and impulse response. The tool developed in the scope of this work allows immediate updating and depiction of all four previously mentioned key metrics at change of the transfer function on a complex level.
The second part of the bachelor thesis addresses the available options for the development of a controller. Here, it is possible to choose between a control circuit with or without control signal limitation. The input signals for the reference and disturbande value are the jump function, the impulse function, an optional time-dependent function, as well as a workspace signal. For the disturbance signal, additional noise is available. The calculation of the control transmission function is effected either directly from the plant controller- and reference transfer function or via controller design processes according to Ziegler and Nichols as well as Chien, Hrones and Reswick. A best possible PID controller can be calculated by using the method of lowest error squares.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die ersten Erkenntnisse aus der Regelungstechnik gehen weit in die Vergangenheit zurück. So wurden bereits mehrere Jahrhunderte vor Christus von den Mechanikern Ktesibios und Philon erste Schwimmregler entwickelt [5]. Der Durchbruch in der Regelungstechnik vollzog sich jedoch erst im 18. Jahrhundert nach Christus. Die aus dieser Zeit bekanntesten neuen Regler waren der Dampfdruckregler und der Fliehkraftregler, der von James Watt als Drehzahlregler für Dampfmaschinen eingesetzt wurde. Im 20. Jahrhundert entwickelte sich die Regelungstechnik, die bis dato größtenteils Anwendung im Kraftmaschinenbau fand, schließlich zu einer eigenständigen Disziplin in der Ingenieurwissenschaft. Die mathematische Darstellung mit Hilfe der Laplace Transformation, sowie die Definition des Regelkreises, welcher symbolisch im Blockschema dargestellt wird, entstanden zu dieser Zeit [8].
Seit 1950 übernehmen Automatisierungsgeräte verstärkt Kontroll- und Überwachungsfunktionen, die vorher vom Menschen getätigt wurden. Die rasche Weiterentwicklung der Mikroprozesstechnik in den 1970er Jahren beschleunigte die Automatisierung, sodass heutzutage in den verschiedensten Bereichen Regelungen zu finden sind. Anwendungen gibt es sowohl in technischen, als auch in nichttechnischen Gebieten [1]:

- Gebäudeautomatisierung, z.B. Raumtemperaturregelung
- Prozessregelung in der Verfahrenstechnik
- Regelung von Fahrzeugen und Flugkörpern
- Regelung von Energiesystemen
- Biologische Regelkreise

Im Zuge der kommenden Energiewende wird die Regelung von Systemen einen immer größeren Stellenwert erlangen. Durch effiziente Regler kann der Energieverbrauch von Verbrauchern reduziert werden. Des Weiteren sind heutige Produktionsprozesse bereits stark automatisiert. Gründe hierfür sind das ständige Streben nach Leistungs- und Qualitätssteigerung sowie die Kostenersparnis durch Rationalisierung. Zur Umsetzung der Automatisierung sind Regelungs- und Steuerungseinrichtungen notwendig.
Der Grundgedanke der Regelungstechnik ist, ein dynamisches System von außen so zu beeinflussen, dass ein gewünschter Zielwert erreicht wird. Die Stellgröße hängt dabei

immer davon ab, inwieweit der Zielwert bereits erreicht wurde, und wird somit im laufenden Prozess ständig variiert. Ein Beispiel ist die Regelung der Wassertemperatur eines Schwimmbeckens mittels Zweipunktregler. Wird die notwendige Energie zur Erwärmung des Wassers durch Solarthermie bereitgestellt, so muss geregelt werden, dass die Temperatur ein vorgegebenes Intervall nicht unter bzw. überschreitet. Becken- und Absorbertemperatur werden jeweils von einem Temperaturfühler gemessen. Liegt die Differenz der gemessenen Temperaturen oberhalb eines bestimmten Grenzwertes, wird die Pumpe angeschaltet, sodass Wasser durch den Absorber läuft und somit erwärmt wird [9].

1.2 Aufgabenstellung und Zielsetzung

Um effiziente Regelsysteme, wie in Kapitel 1.1 beschrieben, zu entwickeln, brauchen Entwickler umfangreiches regelungstechnisches Hintergrundwissen. Das in dieser Arbeit zu entwickelnde Matlab Tool soll den Benutzer unterstützen, grundlegende Sachverhalte und Zusammenhänge, z.B. Stabilität von Systemen, zu verstehen und später anwenden zu können.
Das Ein/Ausgangsverhalten linearer Systeme lässt sich durch eine Übertragungsfunktion beschreiben, die wiederum durch Pole, Nullstellen und dem Verstärkungsfaktor eindeutig definiert ist. In Kapitel 2 wird diese Thematik näher erläutert. Die Aufgabenstellung dieser Arbeit ist in zwei Abschnitte unterteilt. Im ersten Teil soll eine Matlab Bedienoberfläche (GUI[1]) entwickelt werden, welche die Auswirkung der Lage der Pole und Nullstellen in der komplexen Ebene auf bestimmte regelungstechnische Kenngrößen im Zeitbereich sowie im Frequenzbereich veranschaulicht. Die GUI baut auf dem bereits existierenden Matlab m-File *pz2curve* auf. Dieses Programm besteht aus zwei Matlab Graphik Fenstern, die in Abbildung 1.1 gezeigt sind. Das linke Fenster beinhaltet eine komplexe Ebene mit einem komplexen Polpaar, welches per Mausklick verschoben werden kann. Im rechten Fenster sind das Bode Diagramm, die Ortskurve und die Sprungantwort platziert. Ändern die Pole ihre Lage in der komplexen Ebene werden die Größen im rechten Graphikfenster sofort aktualisiert.
Das zu entwickelnde GUI soll erweitert werden und folgende Mindestanforderungen erfüllen:

- Die darzustellenden regelungstechnischen Kenngrößen sind: Bode Diagramm, Ortskurve, Sprungantwort und Impulsantwort.
- Pole und Nullstellen können in der komplexen Ebene bei gedrückter, linker Maustaste verschoben werden. Dabei kann der Benutzer wählen, ob die Pole oder Nullstellen:

[1] Graphical User Interface

  - in der komplexen Ebene frei beweglich sind.
  - bei konstantem Radius im Kreis bewegt werden sollen.
  - bei konstantem Winkel der Verbindungslinie von Pol zu Ursprung auf einer Linie bewegt werden sollen.
  - bei konstantem Realteil nur im y-Wertebereich beweglich sind.
  - bei konstantem Imaginärteil nur im x-Wertebereich beweglich sind.

- Zu jeder Verschiebung der Pole bzw. Nullstellen erfolgt permanent eine Neuberechnung der Kenngrößen, die sofort graphisch angezeigt wird.
- Pole und Nullstellen können durch einen Klick mit der rechten Maustaste hinzugefügt oder gelöscht werden.
- Bei Verschiebung von Polen oder Nullstellen über den Wertebereich der Achsskalierung erfolgt automatisch eine Erweiterung der Skalierung.
- Wahlweise können alle Kenngrößen auf einen Blick oder nur einzelne Kenngrößen angezeigt werden.
- Durch Auswahl des Benutzers wird die momentane Übertragungsfunktion in Polynomform, in Pol/Nullstellendarstellung oder in Zeitkonstantendarstellung angezeigt.
- Eine im Matlab Workspace[2] befindliche Übertragungsfunktion kann jederzeit in die GUI geladen werden.
- Relevante Systemeigenschaften wie Überschwingweite, Anstiegszeit, Überschwingzeit und Beruhigungszeit werden in Echtzeit angezeigt.
- Der Verstärkungsfaktor kann beliebig eingestellt werden, die stationäre Verstärkung kann auf eins normiert werden.
- Es kann zusätzlich das approximierte Bode Diagramm angezeigt werden.

[2]Der Workspace ist ein Fenster, welches alle Variablen mit deren Namen und Werten beinhaltet.

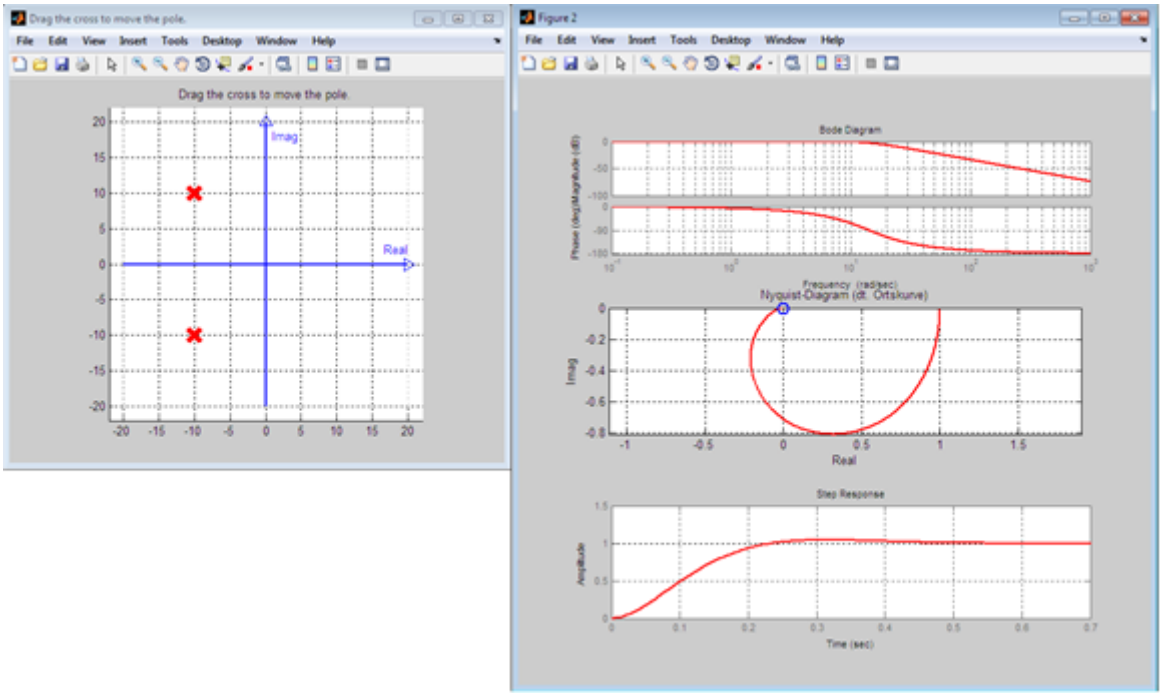

Abb. 1.1: Graphikfenster von *pz2curve*

Im zweiten Abschnitt gilt es die GUI so zu erweitern, dass diese beim Reglerentwurf Unterstützung bietet.

- Über ein Menü kann entweder die Systemanalyse oder der Reglerentwurf ausgewählt werden. Durch einen weiteren Menüpunkt können mögliche Einstellungen in einem separaten GUI vorgenommen werden.
- Je nach Auswahl stellt die aus der komplexen Ebene resultierende Übertragungsfunktion die Führungsübertragungsfunktion $G_W(s)$, die Streckenübertragungsfunktion $G_S(s)$ oder die Reglerübertragungsfunktion $G_R(s)$dar.
- Wird $G_S(s)$ ausgewählt, so kann sich der Nutzer entweder $G_W(s)$ oder $G_R(s)$ neu berechnen lassen. Ansonsten wird stets $G_R(s)$ berechnet
- $G_S(s)$ und $G_R(s)$ können über Edit Felder eingegeben werden.
- Beide Übertragungsfunktionen können wahlweise in Polynomform, Pol/Nullstellendarstellung oder in der Zeitkonstantendarstellung angezeigt werden.
- Bei Auslegung eines PID-Reglers kann dieser als additiver oder mulitplikativer PID-Regler ausgegeben werden.
- Neben den bisherigen regelungstechnischen Kenngrößen kann das Verhalten des Stellsignals und die aktuelle Regelabweichung angezeigt werden.

- Das Stellsignal kann mittels Eingabe der Grenzen durch den Benutzer limitiert werden.
- Bei Bedarf kann am Streckenausgang eine Störung beaufschlagt werden.
- Für die Führungsgröße w und die Störgröße d stehen folgende Signale zur Verfügung:
  - Einheitssprung.
  - Dirac-Impuls mit frei wählbarer Impulsbreite τ.
  - Eine beliebige mathematische Funktion, die über ein Edit Feld eingegeben wird. Dabei müssen die für Matlab typischen Regeln bezüglich der Eingabe beachtet werden.

2 Grundlagen

Zum Verständnis der nachfolgenden Kapitel wird in diesem Abschnitt eine Einführung in die Grundlagen der Regelungstechnik vorgenommen. Es werden nur die Größen behandelt, die für diese Arbeit relevant sind.

2.1 Dynamische Systeme

Das Ziel einer Regelung ist es, ein technisches System so zu beeinflussen, dass es bestimmten Forderungen nachgeht. Ein derartiges System kann beispielsweise ein Elektromotor sein, dessen Drehzahl zu regeln ist. Solch ein System wird als *dynamisches System* bezeichnet, da deren wichtigsten Größen zeitabhängig sind. Die zeitlichen Veränderungen eines dynamischen Systems werden dabei von Eingangsgrößen hervorgerufen. Diese Eingangsgrößen werden im System verarbeitet. Das System gibt eine oder mehrere Ausgangsgrößen aus, die das Verhalten des Systems als Reaktion auf das Eingangssignal beschreiben [1]. Abbildung 2.1 zeigt das Blockschaltbild eines dynamischen Systems mit Eingangs-, Ausgangs- und Störgröße, welches auch dem Prinzip einer Steuerung entspricht. Das System wird in der Regelungstechnik Regelstrecke genannt, die Eingangsgröße wird auch als Stellgröße bezeichnet. Die Regelstrecke kann mathematisch durch eine Differentialgleichung dargestellt werden, welche die zeitveränderlichen Größen miteinander verknüpft [2]. Für die Reglerauslegung ist es wichtig, so viel wie möglich über die Regelstrecke zu wissen, um das System später beeinflussen zu können. Relevante Informationen sind z.B. Zeitkonstanten, Dämpfungsgrad, Resonanzfrequenz, Totzeit usw.

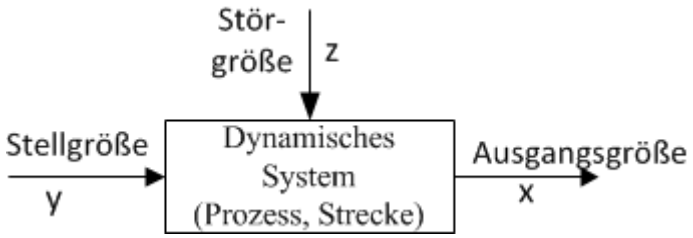

Abb. 2.1: Blockbild eines dynamischen Systems aus [2]

2.2 Der Regelkreis

Abbildung 2.2 stellt die einfachste Grundstruktur eines Regelkreises dar. Auf eine erweiterte Struktur des Regelkreises, die zusätzlich die Dynamik der Messglieder berücksichtigt, wird hier nicht weiter eingegangen.

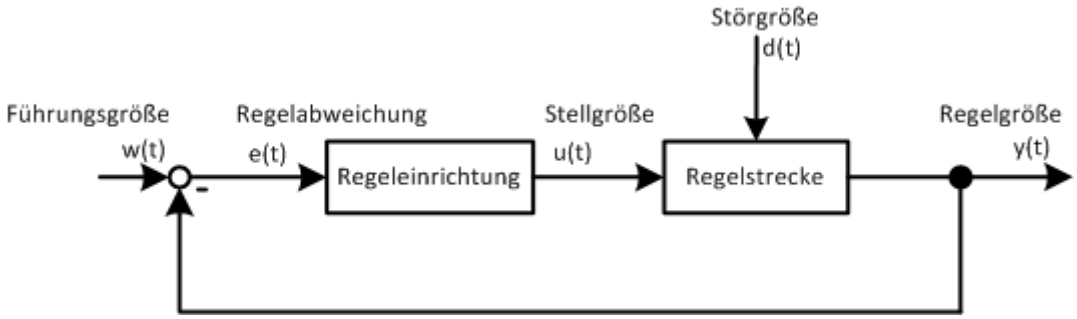

Abb. 2.2: Blockschaltbild eines Regelkreises nach [1]

2.2.1 Regelungsaufgabe

Im Folgenden werden die einzelnen Komponenten der in Abbildung 2.2 dargestellten Regelung erläutert.

Regelstrecke Die Regelstrecke ist ein dynamisches System und spiegelt das Verhalten der zu regelnden Anlage wieder. Sie beeinflusst die Regelgröße maßgeblich [3].

Regeleinrichtung Die Regeleinrichtung stellt den Regler dar. Der Regler ist ein eigenständiges System, welches nach Eingabe einer Regelabweichung *e(t)* die Stellgröße *u(t)* ausgibt. Die Aufgabe der Regeleinrichtung ist die ständige Minderung der Differenz zwischen Führungs- und Regelgröße [3].

Regelgröße *y(t)* Die Regelgröße *y(t)* (Istwert) ist Ausgangssignal der Regelstrecke und somit die zu regelnde Größe.

Störgröße *d(t)* Die Störgröße *d(t)* ist ein Eingangssignal der Regelstrecke und hat dadurch auch Einfluss auf die Regelgröße. Sie wirkt auf die Regelgröße ein, sodass diese von dem gewünschten Verhalten abweicht.

Führungsgröße *w(t)* Die Führungsgröße *w(t)* (Sollwert) stellt den Zielwert der Regelung dar, den die Regelgröße bei endlicher Zeit erreichen soll.

Stellgröße *u(t)* Die Stellgröße *u(t)* ist das berechnete Signal des Reglers auf die Regelstrecke, zur Beeinflussung der Regelgröße.

Regelabweichung *e(t)* Die Regelabweichung *e(t)* ist gemäß (2.1) die Differenz aus der Führungsgröße *w(t)* und der Regelgröße *y(t)*.

$$e(t) = w(t) - y(t). \tag{2.1}$$

Nach der Einführung dieser Komponenten des Regelkreises, kann die Regelungsaufgabe erläutert werden:

Bei gegebener Regelstrecke wird eine Regeleinrichtung gesucht, die laufend ein Stellsignal ausgibt, welches die Regelgröße bestmöglich der Führungsgröße folgen lässt. Ebenso soll die Störgröße die Regelgröße geringstmöglich beeinflussen. Idealerweise gilt $w(t) = y(t)$ für alle Zeitpunkte t [1].

2.2.2 Wirkungsweise

Um die im vorigen Kapitel erläuterte Regelungsaufgabe erfüllen zu können bedarf es nach Lunze J. drei wichtige Schritte, welche die Wirkungsweise einer Regelung beschreiben [1].

Messen Zur laufenden Aktualisierung der Stellgröße durch den Regler muss im ersten Schritt die Regelgröße gemessen werden. Messbare Größen werden über eine Messeinrichtung aufgezeichnet. Bei nicht messbaren Größen, z.B. Qualitätskennwerten erfolgt eine Berechnung aus weiteren Messgrößen.

Vergleichen Die Regelabweichung aus Gleichung (2.1) ist das Ergebnis aus dem Vergleich der Führungsgröße mit der Regelgröße. Im Blockschaltbild des Regelkreises (Abbildung 2.2) wird der Vergleich durch die Rückführung bewirkt.

Stellen Der Regler nutzt die ermittelte Regelabweichung zur Berechnung der Stellgröße. Diese wird an die Regelstrecke weitergeleitet und bewirkt das Erreichen der Regelungsaufgabe.

2.3 Mathematische Beschreibung

Wie bereits in Kapitel 2.1 erwähnt, wird der Zusammenhang zwischen Eingangssignal *u(t)* und Ausgangssignal *y(t)* eines dynamischen Systems häufig durch Differentialglei-

chungen dargestellt. Eine lineare Differentialgleichung n-ter Ordnung hat die Form

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1}}{dt^{n-1}} + ... + a_1 \frac{dy}{dt} + a_0 y(t) = b_q \frac{d^q u}{dt^q} + b_{q-1} \frac{d^{q-1}}{dt^{q-1}} + ... + b_1 \frac{du}{dt} + b_0 u(t). \quad (2.2)$$

Dabei sind a_i und b_i Koeffizienten, die sich aus den physikalischen Zusammenhängen herleiten lassen.

Ein in der Literatur häufig beschriebenes Beispiel für ein derartige System ist der Feder-Masse-Schwinger, siehe Abbildung 2.3. Dieses System beinhaltet eine Feder, die auf der einen Seite an einem Körper, welcher zugleich das Eingangssignal *u(t)* darstellt, befestigt ist. Am anderen Ende der Feder befindet sich ein Dämpfungsglied, das wiederum mit einer Masse verbunden ist. Die Position der Masse beschreibt das Ausgangssignal *y(t)*. Eingangssignal sowie Ausgangssignal legen somit die Position der Systemkomponenten fest. Wenn alle Anfangsbedingungen $u(0), \dot{u}(0), \ddot{u}(0), y(0), \dot{y}(0), \ddot{y}(0)$ gleich Null sind, bedeutet dies, dass sich Eingangs- und Ausgangssignal bei $t = 0$ in der Anfangsposition befinden und dass die Geschwindigkeit und Beschleunigung beider Signale bei $t = 0$ ebenfalls Null sind. Das System ist in Ruhelage.

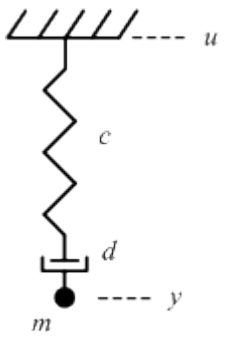

Abb. 2.3: Feder-Masse-Schwinger

2.4 Die Übertragungsfunktion

Diese Kapitel erfordert Kenntnisse der Laplace Transformation.

Die Übertragungsfunktion wird definiert als der Quotient der Laplace Transformierten des Ausgangssignals zu der des Eingangssignals [4]:

$$G(s) = \frac{\mathfrak{L}\{y(t)\}}{\mathfrak{L}\{u(t)\}} = \frac{Y(s)}{U(s)}. \quad (2.3)$$

Die Bedingung für die Gültigkeit der Gleichung (2.3) ist, dass sämtliche Anfangsbedingungen, also $u(0), \dot{u}(0), \ddot{u}(0), ..., y(0), \dot{y}(0), \ddot{y}(0), ...$ gleich Null sind. Die Übertragungsfunktion beschreibt das dynamische Übertragungsverhalten zwischen Eingangs- und Ausgangsgröße [5].

Es gibt mehrere Möglichkeiten die Übertragungsfunktion zu beschreiben. Da die unterschiedliche Darstellung eine wichtige Anforderung an das Matlab Programm ist, behandeln die nächsten Kapitel die verschiedenen Darstellungsformen.

2.4.1 Die Polynomform

Die Polynomform der Übertragungsfunktion kann direkt aus der Differentialgleichung abgeleitet werden. Nach Transformation beider Seiten der Gleichung (2.2) in den Laplace Bereich und anschließender Umformung entsteht:

$$Y(s)(a_n s^n + ... + a_1 s + a_0) = U(s)(b_q s^q + ... + b_1 s + b_0). \tag{2.4}$$

Der Quotient aus Ausgangssignal $Y(s)$ und Eingangssignal $U(s)$ entspricht der Übertragungsfunktion in Polynomform:

$$G(s) = \frac{b_q s^q + b_{q-1} s^{q-1} + ... + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + ... + a_1 s + a_0} \tag{2.5}$$

Nach Division durch a_n entsteht die Gleichung:

$$G(s) = \frac{\frac{b_q}{a_n} s^q + \frac{b_{q-1}}{a_n} s^{q-1} + ... + \frac{b_1}{a_n} s + \frac{b_0}{a_n}}{s^n + \frac{a_{n-1}}{a_n} s^{n-1} + ... + \frac{a_1}{a_n} s + \frac{a_0}{a_n}} \tag{2.6}$$

Der Quotient $\frac{b_q}{a_n}$ ist als der Verstärkungsfaktor k definiert.

2.4.2 Pol-Nullstellen-Form

Zähler und Nenner der Gleichung (2.5) sind Polynome, die in deren Linearfaktoren zerlegt werden können. Somit ergibt sich die Übertragungsfunktion in der Pol-Nullstellen-

Form:

$$G(s) = k\frac{\prod_{i=1}^{q}(s - s_{0i})}{\prod_{i=1}^{n}(s - s_i)} \tag{2.7}$$

Die Lösungen s_{0i}, für die der Zähler der Gleichung (2.7) gleich Null wird heißen *Nullstellen* der Übertragungsfunktion. Bei $s = s_{0i}$ wird $G(s) = 0$. Die Lösungen s_i des Nenners sind die *Pole* der Übertragungsfunktion. Für $s = s_i$ konvergiert die Übertragungsfunktion gegen unendlich. Sowohl Pole als auch Nullstellen resultieren aus Polynomen mit reellen Koeffizienten, weshalb diese nur als konjugiert komplexes Paar, oder reell auftreten[1]. In der komplexen Ebene werden Pole durch ein „x“ und Nullstellen durch einen Kreis „o“ dargestellt. Abbildung 2.4 zeigt ein komplexes Polpaar sowie eine auf der positiven Realachse liegende Nullstelle einer Übertragungsfunktion. Liegen Pole und Nullstellen auf dem selben Punkt können diese gekürzt werden.

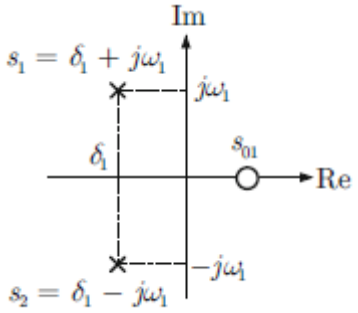

Abb. 2.4: Pole und Nullstellen in der komplexen Ebene[1]

2.4.3 Zeitkonstantenform

Eine häufig verwendete Möglichkeit zur Beschreibung der Übertragungsform ist die Zeitkonstantenform. Die Zeitkonstante T_i ist definiert als:

$$T_i = \frac{1}{|s_i|} \tag{2.8}$$

Bei negativ reellen Polen nimmt ein Eigenvorgang $exp(-\frac{t}{T_i})$ in der Zeit T_i den 1/e ten

Faktor des Anfangswertes an [1]. Bei einem komplexen Polpaar ist T_i:

$$T_i = \frac{1}{\sqrt{\delta_i^2 + \omega_i^2}}. \tag{2.9}$$

Für Nullstellen gilt analog:

$$T_{0i} = \frac{1}{|s_{0i}|} \tag{2.10}$$

Die Übertragungsfunktion in der Zeitkonstantenform ist folgendermaßen definiert:

$$G(s) = \frac{k_s}{s^r} \frac{(T_0 i s + 1)...(T_{0j}^2 s^2 + 2d_{0j} T_{0j} s + 1)}{(T_i s + 1)...(T_j^2 s^2 + 2d_j T_j s + 1)} \tag{2.11}$$

r gibt die Anzahl der verschwindenden Pole ($s_i = 0$) an [5]. Auf die stationäre Verstärkung k_s wird im Kapitel 2.4.4 eingegangen. d wird als Dämpfungsfaktor bezeichnet und ergibt

$$d_i = -\frac{\delta_i}{\sqrt{\delta_i^2 + \omega_i^2}}. \tag{2.12}$$

2.4.4 Die stationäre Verstärkung k_s

Die stationäre Verstärkung k_s bezeichnet den Wert, den das System im eingeschwungenen Zustand ($t \to \infty$) anstrebt. Abbildung 2.5 zeigt die Sprungantwort eines PT2-Systems mit der Zeitkonstante $T = 0.1s$, mit dem Dämpfungsfaktor $d = 0.433$ und mit der stationären Verstärkung $k_s = 1$. Die Sprungantwort konvergiert für $t \to \infty$ gegen den Wert der stationären Verstärkung.

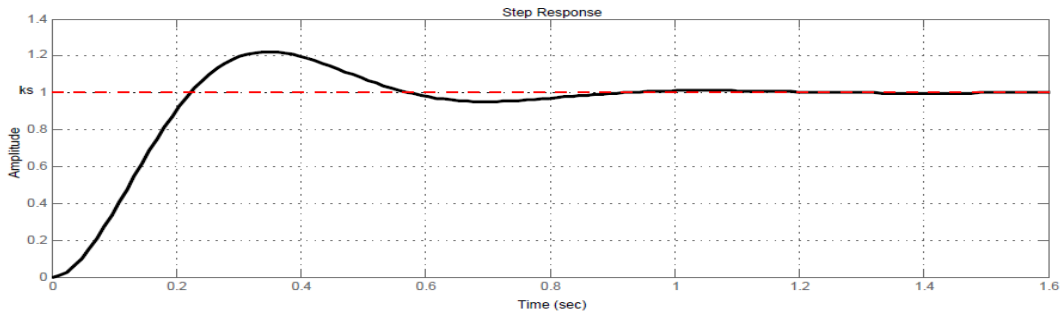

Abb. 2.5: Sprungantwort eines Systems 2-ter Ordnung mit $k_s = 1$

Bei Kenntnis einer Bildfunktion $Y(s)$ lässt sich k_s über den Endwertsatz ermitteln. Nach Föllinger O. lautet dieser:

Strebt y(t) für $t \to \infty$ einem endlichen Grenzwert zu, so gilt $\lim_{t\to\infty} y(t) = \lim_{s\to 0} sY(s)$ [2].

2.4.5 Umrechnung stationärer Verstärkung und Verstärkungsfaktor

Wie in Kapitel 1.2 beschrieben ist die Eingabe eines beliebigen Verstärkungsfaktors k, sowie die Möglichkeit die stationäre Verstärkung k_s auf Eins zu normieren, eine Anforderung an das Matlab GUI. Zur Weiterverarbeitung einer Übertragungsfunktion benötigt die *System Control Toolbox* von Matlab entweder die Koeffizienten der Polynome des Zählers sowie des Nenners bei Polynomform oder die Lage der Pole und Nullstellen sowie den Verstärkungsfaktor k bei Kenntnis der Pol/Nullstellenform. Deshalb muss der Verstärkungsfaktor zu jeder Zeit aus der stationären Verstärkung berechnet werden können. Im Folgenden wird die Berechnung an einem System dritter Ordnung erläutert.

Gegeben ist die Übertragungsfunktion

$$G(s) = k_s \frac{(T_{0i}s + 1)(T_{0j}^2 s^2 + 2d_{0j}T_{0j}s + 1)}{(T_i s + 1)(T_j^2 s^2 + 2d_j T_j s + 1)}.$$

Durch Herausziehen der Zeitkonstanten aus den Klammern ergibt sich die Gleichung:

$$G(s) = \underbrace{k_s \frac{T_{0i} T_{0j}^2}{T_i T_j^2}}_{k} \frac{(s - (-\frac{1}{T_{0i}}))(s^2 + \frac{2d_{0j}}{T_{0j}} s + \frac{1}{T_{0j}^2})}{(s - (-\frac{1}{T_j}))(s^2 + \frac{2d_j}{T_j} s + \frac{1}{T_j^2})}.$$

Der Term des ersten Bruchstrichs ergibt den Verstärkungsfaktor k. Es zeigt sich, dass die Zeitkonstanten der komplexen Polpaare im Gegensatz zu den Zeitkonstanten der auf der Realachse liegenden Pole quadratisch in die Berechnung von k eingehen. Folglich wird das Vorzeichen von k nur durch die Zeitkonstanten der realen Pole beeinflusst. Pole und Nullstellen, die im Ursprung liegen haben keinen Einfluss auf die Berechnung von k. Nach Lunze J. werden Zeitkonstanten nur bei stabilen Systemen verwendet. Deren Pole s_i haben einen negativen Realteil, die zugehörigen Zeitkonstanten sind positiv. Um auch instabile Systeme mit positivem Realteil betrachten zu können, werden dafür negative Zeitkonstanten verwendet ($T_i = -\frac{1}{s_i}$ für $s_i < 0$)[1]. Befindet sich beispielsweise ein Pol auf der positiven Realachse, so berechnet sich k zu einem negativen Wert. Gleiches gilt für Nullstellen.

Die allgemeine Berechnung des Verstärkungsfaktors für ein System nach Gleichung(2.11) mit

- n Polen auf der Realachse
- m Nullstellen auf der Realachse
- k komplexen Polpaaren
- l komplexen Nullstellenpaaren

lautet

$$k = k_s \frac{\prod_{i=1}^{m} T_{0i} \prod_{j=1}^{l} T_{0j}^2}{\prod_{i=1}^{n} T_i \prod_{j=1}^{k} T_j^2}. \tag{2.13}$$

2.5 Bedeutung und Lage der Pole und Nullstellen in der s-Ebene

Ein PT2 System mit $0 < d < 1$ (Abb. 2.6) hat die Pole:

$$s_{1,2} = -\omega_0 d \pm j\omega_0 \sqrt{1 - d^2}.$$

Dabei ist $-\omega_0 d$ der Realteil und $j\omega_0\sqrt{1-d^2}$ der Imaginärteil. Der Abstand der Pole zum Ursprung entspricht dem Betrag der komplexen Zahl:

$$|s_{1,2}| = \sqrt{\mathfrak{Re}\{s_{1,2}\}^2 + \mathfrak{Im}\{s_{1,2}\}^2} = \omega_0 \tag{2.14}$$

Der Abstand der Pole vom Ursprung ist folglich die Eigenfrequenz w_0 des Systems. Je weiter die Pole vom Ursprung entfernt sind, desto schneller reagiert das System ($w \sim \frac{1}{T}$).
Als Nächstes soll der Einfluss des Winkels Φ_d, der zwischen der Realachse und der Verbindungslinie von Pol zum Ursprung liegt, untersucht werden(2.15):

$$tan\Phi_d = \frac{\omega_0\sqrt{1-d^2}}{\omega_0 d} = \frac{\sqrt{1-d^2}}{d} \tag{2.15}$$

Nach Umstellen von (2.15) ergibt sich für den Dämpfungsfaktor

$$d(\Phi_d) = \sqrt{\frac{1}{1+tan\Phi_d}}. \tag{2.16}$$

Der Winkel Φ_d beeinflusst nach Gleichung (2.16) die Dämfpung des Systems. Deshalb wird auch von dem Dämpfungswinkel Φ_d gesprochen. Gleichung (2.17) zeigt zwei Spezialfälle:

$$|d(\Phi_d)| = \begin{cases} 1, & \text{für } \Phi_d = 0°, \\ 0 & \text{für } \Phi \to 90°. \end{cases} \tag{2.17}$$

Im ersten Fall liegt das Polpaar auf dem gleichen Punkt auf der negativen Realachse. Das System ist mit $d = 1$ gedämpft. Ein Dämpfungsfaktor > 1 tritt bei Polen auf, die beide auf der negativen Realachse liegen, jedoch nicht in ein und demselben Punkt. Das nicht schwingungsfähige PT2-Glied verhält sich dann wie zwei in Serie geschaltete PT1-Glieder [5].
In Fall 2 befinden sich die Pole auf der Imaginärachse mit jeweils unterschiedlichem Vorzeichen. Das System unterliegt einer ungedämpften, harmonischen Dauerschwingung [6]. Sobald der Imaginärteil der Pole nicht Null ist beginnt das System zu schwingen.

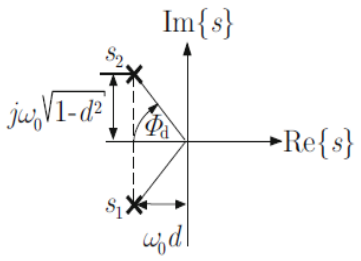

Abb. 2.6: komplexes Polpaar eines PT2-Systems[1]

Die Lage der Pole in der komplexen Ebene hat noch einen weiteren Einfluss auf das Systemverhalten: Der Realteil der Pole findet sich im Exponenten der Exponentialfunktionen $e^{s_i t}$ wieder und bewirkt somit das Auf- bzw. Absteigen der Antwortfunktion $y(t)$. Es wird je nach Vorzeichen des Realteils unterschieden:

- Bei negativem Realteil ergibt sich eine exponentiell fallende Funktion. Sind die Realteile sämtlicher Pole des Systems negativ, klingt die Eigenbewegung ab. Das System ist stabil.
- Ein positiver Realteil bewirkt eine exponentiell steigende Funktion. Befindet sich ein Pol im I. oder IV. Quadranten der komplexen Ebene, so steigt die Übergangsfunktion an. Das System ist instabil.

2.6 Der Frequenzgang

Der Frequenzgang ist eine gebrochen rationale komplexe Funktion, die von ω abhängt [5]. Sie lässt sich aus der Übertragungsfunktion $G(s)$ durch $s \rightarrow jw$ herleiten. Der Frequenzgang beschreibt das Ein- und Ausgangsverhalten von Übertragungsgliedern, bei einem sinusförmigen Eingangssignal(2.18) und Ausgangssignal(2.19)

$$u(t) = \hat{U} sin(\omega t) = \hat{U} e^{j\omega t} \tag{2.18}$$

$$y(t) = \hat{Y} sin(\omega t + \rho) = \hat{Y} e^{j\omega t} e^{\rho} \tag{2.19}$$

Diese Sinusschwingungen können als Radiusvektoren mit Betrag $\hat{U}$,$\hat{Y}$, die mit der Winkelgeschwindigkeit ω rotieren, interpretiert werden. Die Phase ρ ist die Phasenverschiebung des Ausgangssignals zum Eingangssignal.

Durch Ableiten des Eingangs- und Ausgangssignal und Einsetzen in die Differentialgleichung (2.2) ergibt sich der Frequenzgang

$$G(jw) = \frac{Y(jw)}{U(jw)} = \frac{b_q(j\omega)^q + b_{q-1}(j\omega)^{q-1} + ... + b_1(j\omega) + b_0}{a_n(j\omega)^n + a_{n-1}(j\omega)^{n-1} + ... + a_1(j\omega) + a_0}. \tag{2.20}$$

2.6.1 Ortskurve

Die Ortskurve ist eine graphische Darstellungsmöglichkeit des Frequenzgangs mit der Frequenz $\omega = -\infty...\infty$. Meist wird ω auf $0...\infty$ beschränkt. Die Ortskurve beruht darauf, dass der Frequenzgang eine komplexe Variable ist, die sich in Real und Imaginärteil aufspalten lässt. Werden Real- und Imaginärteil in der komplexen Ebene für ω von 0 bis ∞ aufgetragen, ergibt sich der positive Teil der Ortskurve [4]. Abbildung 2.7 zeigt die Ortskurve eines Tiefpasses mit dem Frequenzgang $G(j\omega) = \frac{1}{1+j\omega RC}$. Daraus lässt sich das Funktionsprinzip des Tiefpasses ableiten: Bei $\omega = 0$ liegt der Frequenzgang auf der positiven Realachse. Der Betrag hat hier ein Maximum, die Phase $\rho(\omega)$ geht gegen Null. Das Ausgangssignal ist dem Eingangssignal also nicht phasenverschoben und besitzt die gleiche Amplitude.
Bei hohen Frequenzen ($\omega \to \infty$) nähert sich die Ortskurve der Imaginärachse an und hat somit einen verringerten Betrag und eine ansteigende Phasenverschiebung ($\rho_{max} = 90°$). Die Amplitude des Ausgangssignals ist nun sehr viel kleiner als die des Eingangssignals. Der Tiefpass lässt folglich Signale mit niedrigen Frequenzen passieren, Signale mit hohen Frequenzen werden in deren Amplitude abgeschwächt und zusätzlich zeitlich verzögert.

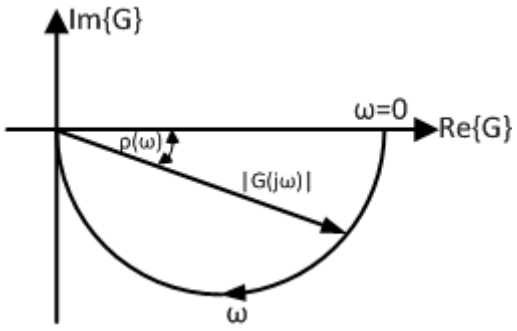

Abb. 2.7: Ortskurve eines Tiefpasses (PT1-System) mit Zeitkonstante RC nach [1]

2.6.2 Bode Diagramm

Das Bode[1]-Diagramm, auch Frequenzkennliniendiagramm genannt, stellt die Amplitude und Phase des Frequenzgangs getrennt voneinander, in Abhängigkeit von der Frequenz ω, dar. Somit beinhaltet das Bode Diagramm einen Amplitudengang und einen Phasengang. Nach Zerlegung des Frequenzgangs (2.20) in seine Amplitude und Phase ergeben sich folgende Gleichungen:

$$|G(jw)| = \sqrt{\mathfrak{Re}\{G(j\omega)\}^2 + \mathfrak{Im}\{G(j\omega)\}^2} \tag{2.21}$$

$$\rho(\omega) = \arctan \frac{\mathfrak{Im}\{G(j\omega)\}}{\mathfrak{Re}\{G(j\omega)\}} \tag{2.22}$$

In der Praxis ist sowohl für die Frequenz, als auch für die Amplitude ein großer Bereich interessant, weshalb ω und $|G(j\omega)|$ in logarithmische Maßstäbe umgerechnet werden. Der Amplitudengang $|G|_{dB}$ wird in Dezibel $[dB]$ angegeben und errechnet sich aus dem dimensionslosen Betrag des Frequenzgangs zu[1]:

$$|G|_{dB} = 20lg|G| \tag{2.23}$$

In Abbildung 2.8 ist das qualitative Bode Diagramm eines Tiefpasses abgebildet. Das obere Diagramm stellt den Amplitudengang dar, das untere den Phasengang. $\omega_e = \frac{1}{T}$ wird nach Lunze J. als Knickfrequenz bezeichnet[1]. Die schwarz gestrichelte Linie gibt jeweils den approximierten Verlauf des Amplitudengangs sowie des Phasengangs an. Auffallend ist, dass die Approximation des Amplitudengangs bei $\omega = \omega_e$ gegenüber dem wahren Verlauf mit $3dB$ maximal abweicht. Die Approximation des Phasengangs entspricht hingegen bei $\omega = \omega_e$ exakt dem wahren Verlauf. Die maximalen Abweichungen des angenäherten Phasengang gegenüber dem wahren Verlauf liegen bei $\omega = \frac{0.1}{T}$ und bei $\omega = \frac{10}{T}$.
Es lassen sich aus dem Bode Diagramm die gleichen Schlüsse über die Funktionsweise des Tiefpasses ziehen, wie bei der Ortskurve aus Kapitel 2.6.1.
Sowohl das Bode-Diagramm, als auch die Ortskurve eines offenen Regelkreises liefern konkrete Informationen über die Stabilität bzw. Stabilitätsreserve des geschlossenen Systems.

[1]Bode, Hendrik Wade *24.12.1905 Madison, USA †1982, Forschungsingenieur, Prof.systems engineering Harvard University[1]

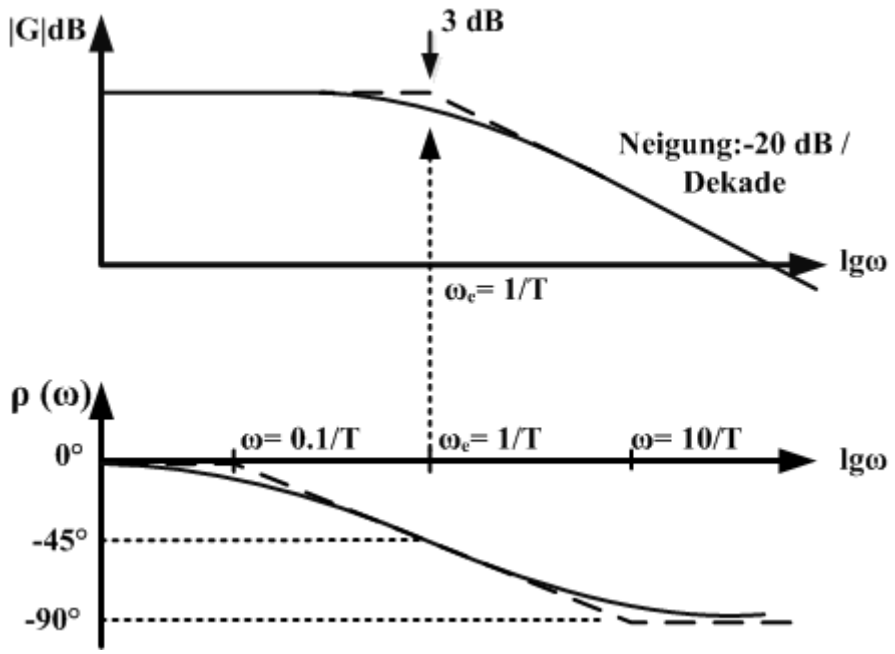

Abb. 2.8: Bode Diagramm eines Tiefpasses (PT1-System) nach [1]

2.7 Sprungantwort

Die Sprungantwort ist die Antwort eines Systems auf einen Sprung als Eingangssignal. Die Ausgangsfunktion wird in der Regelungstechnik auch als *Übergangsfunktion* bezeichnet. Die Eingangsgröße wird aus dem Einheitssprung

$$\sigma(t) = \begin{cases} 0 & \text{für } t < 0, \\ 1 & \text{für } t \geq 0 \end{cases} \tag{2.24}$$

und der Sprunghöhe u_0 zu

$$u(t) = u_0 \sigma(t) \tag{2.25}$$

zusammengesetzt[1].

In Abbildung 2.9 ist die charakteristische Sprungantwort eines PT1-Systems dargestellt. Die gestrichelte Linie ist das Eingangssignal mit Sprunghöhe u_0.

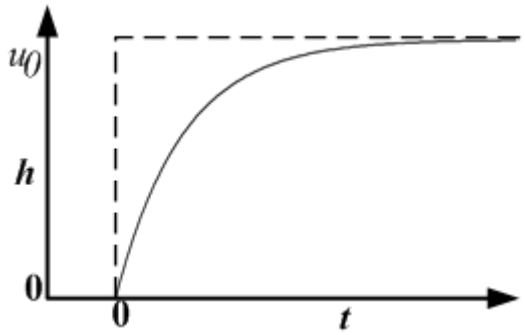

Abb. 2.9: Sprungantwort eines PT1-Systems

Für $t \to \infty$ konvergiert die Sprungantwort gegen u_0. In diesem Fall entspricht die Sprunghöhe auch der statischen Verstärkung k_s (Kapitel 2.4.4).

2.8 Impulsantwort

Die Impulsantwort, auch *Gewichtsfunktion* $g(t)$ genannt, ist die Reaktion des Systems auf einen Dirac-Impuls $\delta(t)$ als Eingangsgröße. Ein Dirac-Impuls ist in Abbildung 2.10 dargestellt und folgendermaßen definiert:

$$\delta(t) = \lim_{\tau \to 0} \begin{cases} \frac{1}{\tau} & \text{für } 0 \leq t \leq \tau, \\ 0 & \text{für } t < 0 \text{ und } t > \tau \end{cases} \tag{2.26}$$

Der Dirac-Impuls ist folglich unendlich hoch und unendlich kurz[1]. Die Fläche, die der Dirac-Impuls mit der Zeitachse erschließt ist immer 1, da: $\frac{1}{\tau} \cdot \tau = 1$

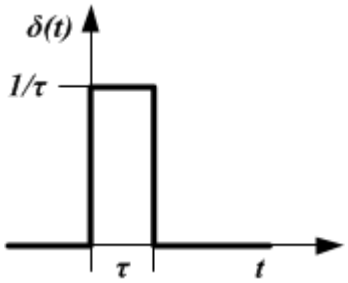

Abb. 2.10: Dirac-Impuls

Die Gewichtsfunktion steht mit der Übergangsfunktion in Zusammenhang. Die Umrechnung der Übergangsfunktion $h(t)$ und der Gewichtsnfunktion $g(t)$ aus deren Bildfunktionen $H(s)$ und $G(s)$ erfolgt folgendermaßen:

$$h(t) \circ\!\!-\!\!\bullet\, H(s) = G(s) \cdot \frac{1}{s} \tag{2.27}$$

$$g(t) \circ\!\!-\!\!\bullet\, G(s) = H(s) \cdot s \tag{2.28}$$

Durch Anwendung der Laplace Transformation Rechenregel *Transformation der Ableitung* [2] ergibt sich der Zusammenhang[6]:

$$g(t) = \frac{d}{dt}(h(t)) \tag{2.29}$$

Abbildung 2.11 zeigt die Impulsantwort eines PT1-Systems. Die gestrichelte Linie stellt das Eingangssignal, den Dirac-Impuls, dar. Betrachtet man die Sprungantwort (Abbildung 2.9) und die Impulsantwort, so zeigt sich, dass die Gewichtsfunktion die Ableitung der Übergangsfunktion ist , siehe (2.29).

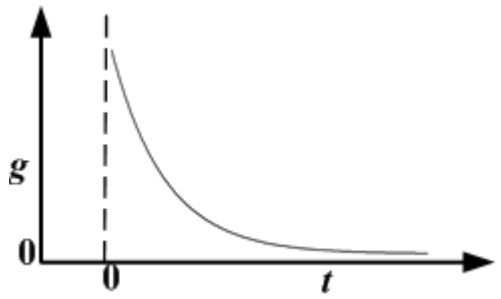

Abb. 2.11: Impulsantwort eines PT1-Systems

[2] Transformation der Ableitung: $\mathfrak{L}\{f'(t)\} = s\mathfrak{L}\{f(t)\} - f(0+)$

3 Matlab/Simulink

Die Software Matlab/Simulink von Mathworks dient Ingenieuren der Regelungstechnik als Werkzeug zur rechnergestützten Simulation und Analyse von Regelsystemen. Die Stärken von Matlab liegen in der Matrizen- und Vektorrechnung.
In Simulink wird hingegen mit graphischen Funktionsblöcken gearbeitet. Für die Simulation dynamischer Systeme hat sich Simulink bereits auf dem Markt etabliert.
Dieses Kapitel soll wichtige Einzelheiten zur Umsetzung der gewünschten Forderungen darlegen, jedoch nicht tiefer die Programmierung der Software erläutern.

3.1 Matlab

3.1.1 Datentypen

Wird eine neue Variable im Workspace angelegt, ist diese immer einem konkreten Datentyp zugrunde gelegt. Viele Matlab Funktionen benötigen zur korrekten Verarbeitung der Eingabeparameter einen bestimmten, festgelegten Datentyp. Deshalb wird nun auf die wichtigsten Typen und deren mögliche Verschachtelungen eingegangen.

double Der Datentpy `double` (64 Bit) gehört den Gleitkommazahlen an und wird in Matlab für Rechenoperationen verwendet.

int, uint Die Datentypen `int8, int16, int32` berücksichtigen im Gegensatz zu `uint8, uint16, unit32` das Vorzeichen. Beide Typen sind Festkommazahlen.

char Ein weiterer, sehr wichtiger Variablentyp zur Verwendung von Zeichenketten(Strings) ist `char` (16 Bit).

Struktur Um eine Vielzahl von Daten übersichtlich zu verwalten, können diese grundlegenden Variablentypen über Strukturen miteinander verknüpft werden. Eine Struktur enthält immer einen String, dem eine frei wählbar, Variable zugeordnet ist und wird folgendermaßen erzeugt:

```
struct('name1',wert1,'name2',wert2,...)
```

Beispielsweise könnten die Informationen eines Studenten in einer Struktur abgespeichert sein. Der Code

```
student = struct('Name','Mustermann Max','Alter',24)
```

erzeugt eine Struktur namens *student* und ergibt folgendes Resultat:

```
Name: 'Mustermann Max'
Alter: 24
```

Die gleiche Struktur kann auch über den Befehl

```
student.name = 'Mustermann Max'
student.alter = 24
```

erzielt werden.

Cell Array Eine weitere Kombination von Variablen stellt das Cell Array dar. Im Vergleich zur Struktur ist das Cell Array noch allgemeiner gehalten, da in jeder Zelle unterschiedliche Daten stehen können [10]. Ein leeres 1x2 Cell Array mit dem Namen `zelle` wird folgendermaßen erzeugt:

```
zelle = cell(1, 2)
```

Um *zelle* mit Daten zu füllen kann beispielsweise

```
zelle{1, 1} = 'Dies ist ein String';
zelle{1, 2} = 1;
```

eingegeben werden.

3.1.2 Graphic Handles

Um in Matlab grafische Objekte zu manipulieren, wird das *Graphic System* genutzt. Abbildung 3.1 zeigt die relevaten Grafikobjekte hierarchisch geordnet.

Der Code

```
t=0:0.01:2*pi;
h=plot(t,sin(t));
```

öffnet ein *figure* und ein *axes* Objekt und plottet eine Sinusfunktion über der Zeitachse t. Daraus wird ersichtlich, dass die übergeordneten Graphik Objekte *figure* sowie *axes* ohne separaten Befehl kreiert werden, sobald ein hierarchisch tiefer angeordnetes Objekt entsteht. h ist nun das Handle des geplotteten Line-Objekts. Dieses Handle kann

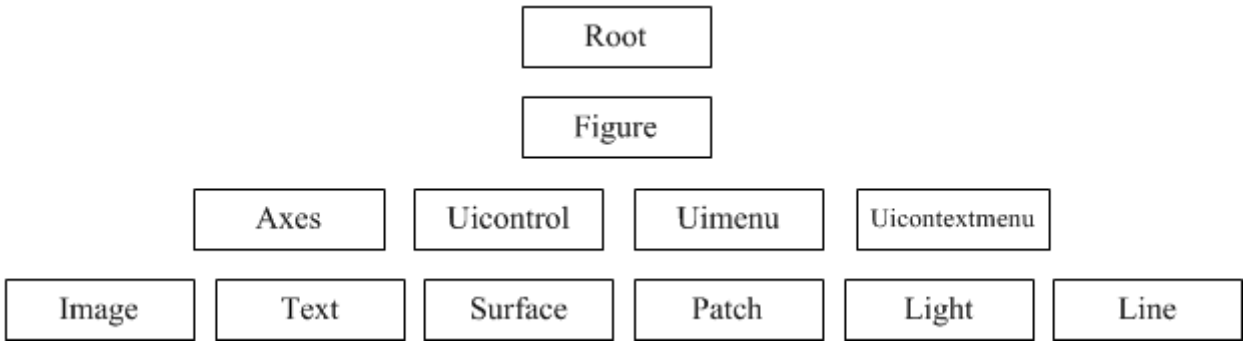

Abb. 3.1: Hierarchie der Grafikobjekte aus [11]

genutzt werden um die Eigenschaften des *line*-Objekts, in diesem Fall die Sinuskurve, zu beeinflussen. Die durch den Befehl

```
s=get(h)
```

entstandene Variable s ist eine Struktur mit sämtlichen Eigenschaften des *line*-Objekts. Mit

```
get(h, 'property');
set(h,'property', value);
```

kann der Wert einer Eigenschaft betrachtet oder auch geändert werden.
Im Folgenden werden die wichtigsten Befehle aufgeführt, um die Handles aktiver Objekte zu finden:

- `gcf`: Spricht das Handle des aktiven *figure* Objekts an.
- `gca`: Spricht das Handle des aktiven *axes* Objekts an.
- `findobj(h,'propertie',value1)`: Spricht das hierarchisch untergeordnete Objekt im Handle h, mit Wert *value1* der Eigenschaft *propertie* an.
- `gcbo`: Spricht das Handle des Objekts, dessen Callback aktuell durchlaufen wird, an.

Werden Handles innerhalb einer Matlab Funktion kreiert, so stehen diese auch nur in der jeweiligen Funktion zur Verfügung. Möglichkeiten zur Übergabe der Handles an andere Funktionen werden in Abschnitt 3.1.3 erläutert.

3.1.3 Graphical User Interface

GUI ist die Abkürzung für *Graphical User Interface*, zu deutsch: Bedienoberfläche. Über eine GUI kann der Nutzer mit dem Computer, bzw. hier mit Matlab, kommunizieren,

ohne Ausdrücke in das Command Window eingeben zu müssen. Die Bedienoberfläche besteht aus *user interface control* (uicontrol) sowie aus *user interface menu* (uimenu) Elementen, die beliebig miteinander verknüpft werden können. Zur einfacheren Anordnung dieser Objekte gibt es in Matlab den GUI Builder, auch GUIDE genannt. Die zur Auswahl stehenden uicontrols sind:

- Push Button
- Radio Button
- Edit Text
- Pop-up Menu
- Toggle Button
- Axes
- Button Group
- Slider
- Check Box
- Static Text
- Listbox
- Table
- Panel

Die *Callback* Eigenschaft der uicontrol, sowie uimenu Elemente speichert einen String, der den Namen der Callback Funktion beinhaltet. Wird wie in Abbildung 3.2 ein Push Button in der Bedienoberfläche angelegt, entsteht automatisch die zugehörige Callback Funktion. Diese ist in Listing A.1 dargestellt. Die Callback Funktion kann vom Nutzer beliebig festgelegt werden und wird immer durchlaufen, wenn das Objekt vom Nutzer aktiviert wird. Der Callback eines Push Buttons wird beispielsweise durch einfaches Klicken auf den Button ausgelöst. Zur Verknüpfung der eingesetzten Objekte kann das vom GUI Builder kreierte Matlab m-File genutzt werden. Dieses beinhaltet mehrere Funktionen, die die Kommunikation untereinander bestimmen.

Wie in Kapitel 3.1.2 beschrieben, können Handles nicht ohne weiteres in unterschiedlichen Matlab Funktionen genutzt werden. Für die Programmierung einer GUI bieten sich zwei Möglichkeiten zur Übergabe der Handles an:

1. Mit dem Befehl

   ```
   setappdata(h,'name',value)
   ```

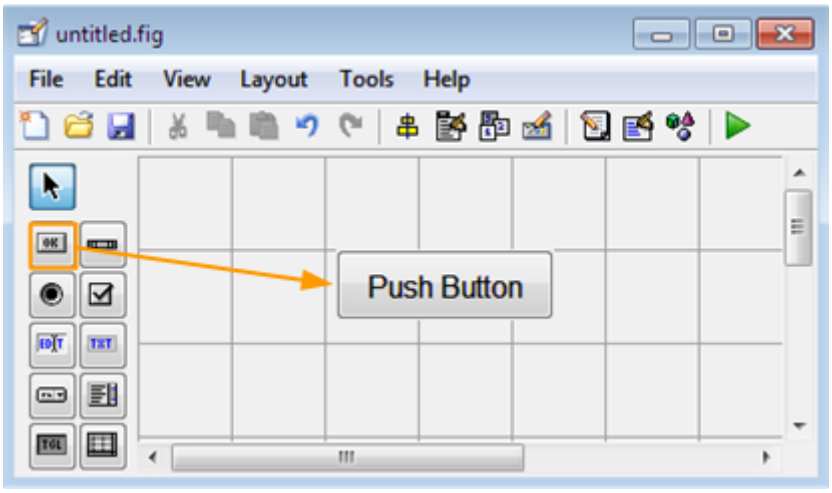

Abb. 3.2: Hinzufügen eines Push Buttons

wird der Wert *value* mit dem Namen *name* im Handle *h* des zugehörigen Objekts abgespeichert.

```
neuername=getappdata(h,'name')
```

weist der Variablen *neuername* den Wert von *name* zu. Somit können die Handles an beliebigen Stellen verwendet werden.

2. Ein Handle kann ebenso in der Handles Struktur abgespeichert werden. Wird das Handle eines neuen Objekts, z.b. eines Plots *neueshandle* genannt, so wird dieses der Struktur folgendermaßen angehängt:

   ```
   handles.neueshandle=plot(x,y);
   ```

   Wichtig ist, dass die Handles Struktur am Ende der Funktion mit `guidata(hObject, handles)` neu abgespeichert wird. Dadurch kann das Handle an eine andere Funktion über die Eingabeparameter `hObject, handles` übergeben werden. Das abgespeicherte Handle steht nun unter dem Namen `handles.neueshandle` zur Verfügung.

3.2 Simulink

Wie bereits erwähnt wird in Simulink mit Funktionsblöcken gearbeitet, die passend miteinander verknüpft werden müssen. In Abbildung 3.3 ist ein in Simulink konstruierter, einschleifiger Regelkreis sowie die geplotteten Ergebnisse im *Scope*, dargestellt. Für den Sprung sowie die zwei Übertragungsfunktionen $G_R(s)$ und $G_S(s)$ gibt es separate Blöcke, die jeweils mit konkreten Informationen gefüllt werden müssen. Der *Step*

Block benötigt beispielsweise die Startzeit, den Anfangswert, den Endwert und das Zeitintervall zwischen Anfangs- und Endwert. Für den *transfer-fcn* Block werden die Koeffizienten des Zählers sowie des Nenners der Übertragungsfunktion als Information benötigt. Die Blöcke wurden entsprechend der Abbildung 3.3 verknüpft.

Für die zu entwickelnde Matlab GUI werden Simulink Modelle benötigt um das nichtlineare Verhalten, welches durch den Einsatz der Stellgliedlimitierung entsteht, trotzdem simulieren zu können. Die Modelle müssen also, wenn notwendig, automatisch gestartet werden und sollten bestenfalls im Hintergrund, ohne zusätzliche Fenster, durchlaufen werden. Die aktuellen Parameter der Übertragungsfunktionen oder der Eingangssignale müssen vor dem Start der Simulation von einer Matlab Funktion an den passenden Simulink Block übergeben werden. Für die Eingangssignale wird hierfür der *simin* Block verwendet. Dieser holt sich die dem passenden Namen entsprechende Strukturdatei aus dem *base* Workspace. Für die Koeffizienten der Übertragungsfunktionen wird ebenfalls der Zähler und Nenner mit einem konkreten Namen in den *base* eingetragen. Diese Namen müssen in den Simulink Blöcken in die zu editierenden Felder geschrieben werden.
Ebenso müssen die durch Simulink berechneten Werte wieder in die GUI übertragen werden. In dem Regelkreis aus Abbildung 3.3 wären dies die beiden Verläufe, welche im *Scope* dargestellt sind. Zur Übergabe dieser Werte wird der *Out* Block verwendet. Wird dieser Block in einem Modell platziert, so beinhaltet die Matrix y, nach Eingabe des Befehls

```
[t, x, y]=sim('modellname')
```

die berechnete Werte. Der Vektor t enthält die verwendeten Zeitwerte.
Mit dem `sim` Befehl wird die Simulation gestartet und durchlaufen, ohne dass ein separates Fenster geöffnet werden muss.

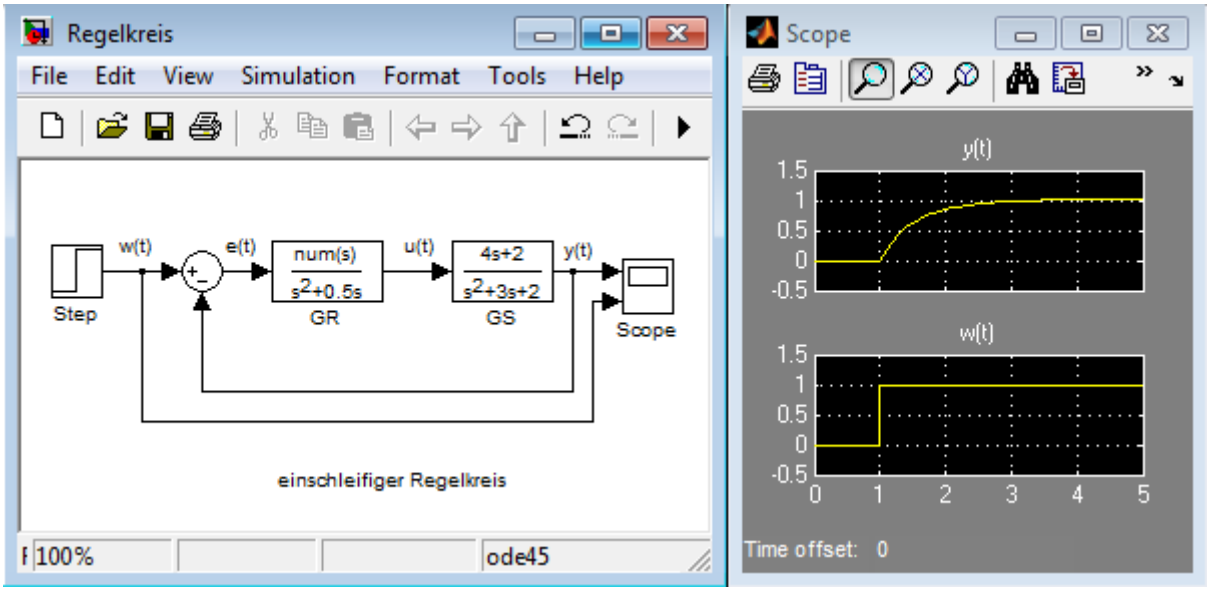

Abb. 3.3: Regelkreis in Simulink und Scope

4 Durchführung

Diese Kapitel beschreibt die Durchführung der wichtigsten Arbeitsschritte sowie die entstandenen Schwierigkeiten. Vereinbarungen werden im Folgenden nur für die Polstellen angegeben. Selbiges gilt ohne weitere Anmerkung auch für alle Nullstellen.

4.1 Systemanalyse

4.1.1 Verschieben, Hinzufügen oder Löschen von Polstellen

4.1.1.1 Verschieben von Polstellen

Das Verschieben der Pole ist eine der Hauptanforderungen an das Matlab Tool, welche bereits in dem Matlab m-file *pz2curve* existiert. In diesem Unterkapitel soll eine Vorlage für die Interaktion eines Nutzers mit einem Plot vorgestellt und erläutert werden. Das Listing A.2 im Anhang beinhaltet eine Funktion namens *plotdisplacement*. Die Ausführung dieser Funktion plottet ein Kreuz, welches anschließend durch Drücken und Halten der linken Maustaste verschoben werden kann.
Die Funktion ist durch eine *if-else Bedingung* in zwei Abschnitte unterteilt. Bei erstmaligem Starten der Funktion wird aktuell keine Callback Funktion durchlaufen. Das handle *gcbo* ist somit leer. Deshalb wird der Code zunächst ab Zeile 15 ausgeführt. Die Eigenschaft *userdata* des *figure* Objekts wird in Zeile 16 mit dem Wert 0 belegt.
Die *WindowButtonDownFCn* wird immer dann ausgeführt, wenn der Benutzer eine beliebige Maustaste drückt, wobei der Mauszeiger innerhalb des *figure* Objekts liegen muss. Die Funktion *WindowButtonUpFcn* wird analog nur durchlaufen, wenn die Maustaste losgelassen wird. Folglich wird in Zeile 17 festgelegt, dass bei einem Klick der Maustaste der Wert von *userdata* auf Eins gesetzt wird. Die nachfolgende Zeile ordnet *userdata* beim Loslassen einer Maustaste wieder den Wert Null zu. Für die Funktionalität ist wichtig, dass *userdata* beim Loslassen der Maus immer den Wert Null zugeordnet bekommt. Die Zeilen 19-25 bewirken den Plot des Kreuzes im Ursprung.
Die *ButtonDownFcn* Funktion eines *line* Objekts wird immer dann ausgeführt, wenn der Benutzer eine Maustaste drückt und der Mauszeiger dabei auf dem *line* Objekt liegt. Der *ButtonDownFcn* des Handles h, welches dem Plot des Kreuzes zugeordnet wurde, wird mit dem Wert *mfilename* besetzt. Das bedeutet, dass bei einem Klick auf das Kreuz die gesamte Funktion *plotdisplacement* erneut von Beginn ausgeführt wird. Das

Programm springt folglich wieder in Zeile 3. Dadurch, dass das Programm in diesem Moment eine Callback Funktion ausführt, ist die *if-Bedingung aus* Zeile 3 erfüllt und das Programm springt in Zeile 4. Der Inhalt der folgenden *while Schleife* wird augeführt, wenn der Ausdruck *get(gcf,'userdata')* den Wert Eins hat. Dies ist bekanntlich nur der Fall, wenn eine Maustaste gedrückt, oder gehalten wird. Die Koordinaten der aktuellen Mausposition werden in den folgenden Zeilen eingelesen und zur Weiterverwendung umgestellt. In Zeile 11 wird dem Kreuz der neue x- und y-Wert zugewiesen. Das Kreuz bewegt sich. Der Befehl *drawnow* aus Zeile 12 bewirkt ein Update des aktuellen *figure* Objekts.

4.1.1.2 Auswahl bei Mausklick

Bei Rechtsklick in dem *axes* Objekt der komplexen Ebene soll eine Auswahlmöglichkeit zum Hinzufügen, beziehungsweise Löschen von Polen oder Nullstellen entstehen. Dafür wird die *ButtonDownFcn*, die Eigenschaft des *axes* Objekts ist, verwendet. Dem Wert dieser Funktion wird durch den `@` Operator das *Function Handle @rechtsklick* zugewiesen. Wird also die *ButtonDownFcn* durch einen Mausklick auf das entsprechende *axes* Objekt aufgerufen, so wird ebenso die Funktion *rechtsklick* durchlaufen. Ein Klick mit dem rechten Mauszeiger bewirkt, dass die Zeilen 5 bis 10 des Listings A.3 durchlaufen werden. In Zeile 5 und 6 wird jeweils die aktuelle Mausposition bezogen auf das *figure*, bzw. das *axes* Objekt ausgegeben. In Zeile 8 wird die Variable *cp2* definiert, deren zweiter Wert um 50 units geringer ist als der der Varialbe *cp*. Diese Verschiebung des y-Werts wird benötigt, um die Listbox immer direkt auf der Höhe der Mausposition anzuordnen. Zeile 10 legt die Position der Listbox fest und macht diese sichtbar.

4.1.1.3 Polstellen hinzufügen

Wie im vorherigen Kapitel beschrieben, erscheint bei einem Rechtsklick mit der Maus eine Listbox. Hier kann der Nutzer entscheiden, ob Pole hinzugefügt oder gelöscht werden sollen. Je nach Mausposition entstehen die in Tabelle 4.1 angegebenen Plotmöglichkeiten. Der x- bzw. y-Wert der aktuellen Position der Maus wird mit m_x und m_y angegeben. In Spalte eins und zwei stehen die verschiedenen Varianten der Mausposition. In den Spalten 3 und 4 werden die daraus resultierenden Plots der Pole angegeben. Die Variable a ist die Zahl, die 1/10 des aktuellen x- bzw. y-Wertebereichs beträgt.
In Fall 2 und Fall 4 wird ein konjugiert komplexes Paar geplottet. Treten Möglichkeit 1 und 3 ein, wird jeweils ein Pol dargestellt, da sich das Objekt auf der reellen Achse befindet.

Mausposition		**Plot**	
x-Wert	**y-Wert**	**x-Wert**	**y-Wert**
$\|m_x\| > 2$	$\|m_y\| < 2$	m_x	0
$\|m_x\| < 2$	$\|m_y\| > 2$	0	$\pm m_y$
$\|m_x\| < 2$	$\|m_y\| < 2$	0	0
$\|m_x\| > 2$	$\|m_y\| > 2$	m_x	$\pm m_y$

Tab. 4.1: Plotmöglichkeiten in Abhängigkeit von der Mausposition

In Listing A.4 ist der ausführende Code dargestellt. Damit nach Klicken auf einen Pol die Hauptfunktion erneut durchlaufen wird, müssen die Eigenschaften *ButtonDownFcn* der Handles der *plot* Objekte mit *mfilename* besetzt sein. Dies ermöglicht das Verschieben der Kreuze bzw. der Kreise.

4.1.1.4 Polstellen löschen

Entscheidet sich der Nutzer dafür Pole zu löschen, werden automatisch die am nächsten an der aktuellen Mausposition liegenden Pole gelöscht. Dies wird im Listing A.5 im Anhang umgesetzt.
In der ersten Zeile werden die Handles sämtlicher Pole in den Spaltenvektor *po* eingelesen. In der darauffolgenden *for-Schleife* werden zunächst die x- und y-Werte der einzelnen Pole abgefragt. Im gleichen Schleifendurchlauf erfolgt die Abstandsberechnung des jeweiligen Pols zu der aktuellen Position der Maus. Nach der *for-Schleife* wird der am nächsten an der Mausposition befindliche Pol über die `min()` Funktion ausfindig gemacht, welcher anschließend gelöscht wird. In den beiden letzten Zeilen des Listings wird überprüft, ob es sich bei dem ausgewählten Pol um ein Polpaar handelt. Ist dies der Fall, wird das Polpaar gelöscht. Liegt der am kürzesten entfernte Pol auf der Realachse, so wird nur dieser eine Pol gelöscht.

Sowohl beim Hinzufügen, als auch beim Löschen von Polen muss immer die Kausalität des Systems beachtet werden. Ein System ist kausal und somit auch technisch realisierbar, wenn mehr oder genauso viele Pole wie Nullstellen existieren. Würde das Hinzufügen oder Löschen eines Pols oder einer Nullstelle eine Akausalität des Systems bewirken, so wird der Plotvorgang unterbrochen und der Nutzer auf die Kausalität hingewiesen. Dieser Hinweis ist in Abbildung 4.1 zu sehen.

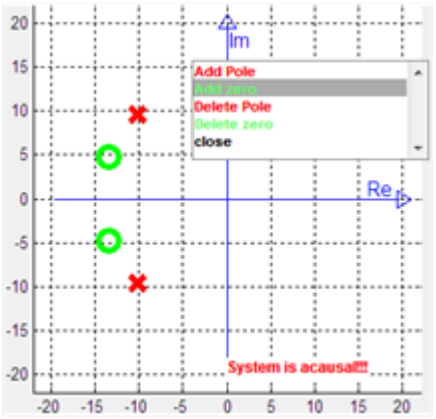

Abb. 4.1: Hinweis auf Akausalität

4.1.2 Skalierung der komplexen Ebene anpassen

4.1.2.1 Erweiterung der Skalierung

Die Vergrößerung der Achsenskalierung der komplexen Ebene hat den Zweck auch während der Verschiebung von Polen den aktuellen x- und y-Wertebereich zu verlassen. Die Skalierung wird automatisch erweitert, wenn die folgenden zwei Voraussetzungen erfüllt sind:

1. Die Pole werden zu diesem Zeitpunkt verschoben, die linke Maustaste ist folglich ständig gedrückt.
2. Die Position des Mauszeigers befindet sich außerhalb der in Abbildung 4.2 gezeigten Umrandung der komplexen Ebene. Die Tabelle 4.2 zeigt die verschiedenen Mauspositionen, die eine Vergrößerung bewirken. $xLim$ und $yLim$ sind Variablen, die zur Berechnung des Limits der Achsskalierung verwendet werden. Wird der Wert einer der beiden Variablen erhöht, so vergrößert sich auch die Skalierung der jeweiligen Achse.

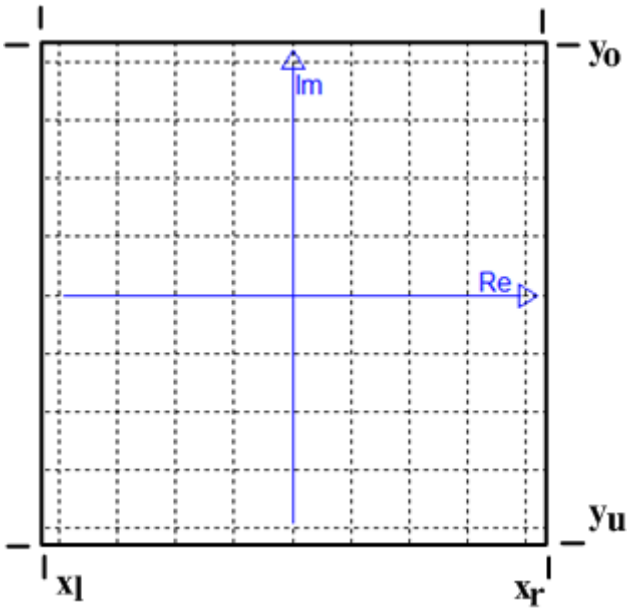

Abb. 4.2: Komplexe Ebene

Mausposition		**Plot**	
x-Wert	**y-Wert**	**xLim**	**yLim**
$m_x < x_l$		$+3$	$+0$
$m_x > x_r$		$+3$	$+0$
$m_y < y_u$		$+0$	$+3$
$m_y > y_o$		$+0$	$+3$

Tab. 4.2: Erweiterung Achsskalierung

Die Achsenskalierung für die x-Achse wird mit dem Befehl `xlim([a b])` folgendermaßen abgeändert:

```
xlim([-1 1] * xLim)
```

Bei einem Schleifendurchlauf ändert sich die Achsenskalierung folglich um den Wert sechs.

4.1.2.2 Anpassen der Skalierung

Befinden sich mehrere Pole oder Nullstellen in der komplexen Ebene, kann es durch eine beliebige Anwendung des Nutzers passieren, dass einzelne Pole nicht mehr in der Ebene zu sehen sind. Deshalb ist es notwendig, eine Neuanordnung der Skalierung des

axes Objekts der komplexen Ebene durchzuführen, in der alle Pole einzusehen sind. Dies wird dem Nutzer durch einen *Pushbutton*, namens *resize* ermöglicht. Dabei sucht das Programm den am weitesten vom Ursprung entfernten Pol. Der x- und y- Wert dieses Pols dient als neuer Wert für die Limitierung der beiden Achsen.

4.1.3 Polverschiebung in Polarkoordinaten

In den Programmeinstellungen (siehe Abbildung A.1) kann ausgewählt werden, ob die Pole frei, in Polarkoordinaten oder in kartesischen Koordinaten in der komplexen Ebene bewegt werden. Die Bewegung der Pole in Polarkoordinaten, speziell auf einer Kreisbahn, ist Thema dieses Unterkapitels. Dafür muss unter *settings/pole displacement/polar/circle* ein Häckchen gesetzt werden.
Die Bewegung auf einer Kreisbahn bedeutet, dass die Entfernung eines Poles vom Ursprung während der Polverschiebung nicht geändert werden kann. Durch diese Einstellung wird der Einfluss des Dämpfungswinkels (siehe Kapitel 2.5) auf das Systemverhalten veranschaulicht.
Die folgenden Berechnungen beziehen sich auf die Abbildung 4.3. Zur Berechnung der neuen x- und y-Position des Pols werden zwei Gleichungen benötigt:

$$r = \sqrt{x_{alt}^2 + y_{alt}^2} = \sqrt{x_{neu}^2 + y_{neu}^2} \tag{4.1}$$

$$\tan(\alpha) = \frac{y_{maus}}{x_{maus}} = \frac{y_{neu}}{x_{neu}} \tag{4.2}$$

x_{alt}: vorheriger x-Wert des Pols, zur Berechnung von r.
y_{alt}: vorheriger y-Wert des Pols, zur Berechnung von r.
x_{maus}: x-Wert des Mauszeigers, zur Berechnung von $\tan\alpha$.
y_{maus}: y-Wert des Mauszeigers, zur Berechnung von $\tan\alpha$.
x_{neu}: x-Wert der neuen Position des Pols.
y_{neu}: y-Wert der neuen Position des Pols.

Nach Auflösung der Gleichungen 4.1 und 4.2 nach y_{neu} und x_{neu} ergeben sich die Formeln:

$$y_{neu} = \sqrt{\frac{\tan^2(\alpha) r^2}{1 + \tan^2(\alpha)}} \tag{4.3}$$

$$x_{neu} = \frac{y_{neu}}{\tan(\alpha)} \tag{4.4}$$

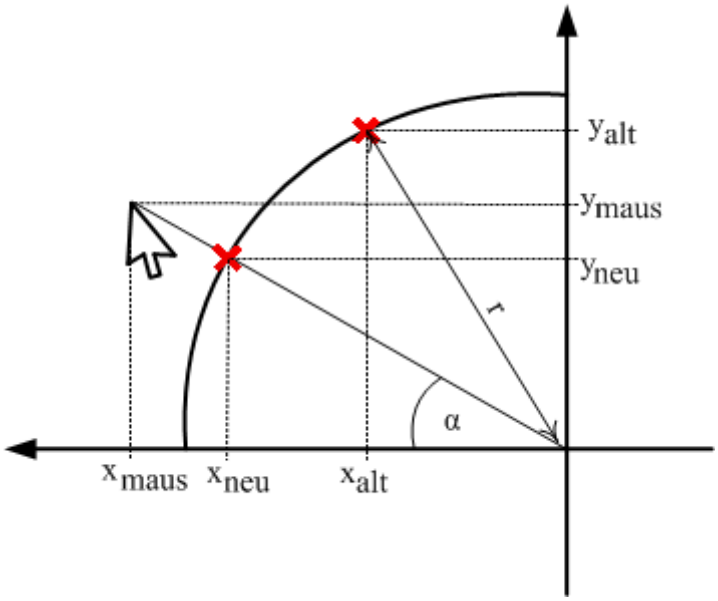

Abb. 4.3: Kreisbewegung eines Pols

Die Nützlichkeit dieser Funktion ist in Abbildung 4.4 dargestellt. Die Graphik zeigt ein Polpaar mit drei verschiedenen Dämpfungswinkeln Φ_d sowie die dazugehörigen Bode Diagramme und Sprungantworten. In Abbildung 4.4a) liegt das Polpaar auf der Realachse, dies bedeutet eine Dämpfung von Eins, siehe (2.17). Die Sprungantwort zeigt hier keinerlei Überschwingen auf. In Bild 4.4b) wurde das komplexe Polpaar auf dem Kreis verschoben bis der Dämpfungswinkel 45° erreicht hat. Die Dämpfung ist hier kleiner Eins, die Sprungantwort schwingt über. Ein Dämpfungswinkel von 90° ist in Abbildung 4.4c) dargestellt. Hier ist die Dämpfung gleich Null. Dies ist an der Dauerschwingung der Sprungantwort und an dem Ausschlag des Amplitudengangs zu erkennen.
Des Weiteren verdeutlicht Abbildung 4.4 den Zusammenhang zwischen Systemdynamik und Pollage. Dadurch, dass sich die Pole in allen drei Konfigurationen im gleichen Abstand zum Ursprung befinden, ist die Eigenfrequenz stets konstant, siehe Gleichung (2.14). Dies wird durch das rote Kreuz in den Amplitudengängen ersichtlich, welches sich immer bei der gleichen Frequenz, der Eigenfrequenz, befindet.

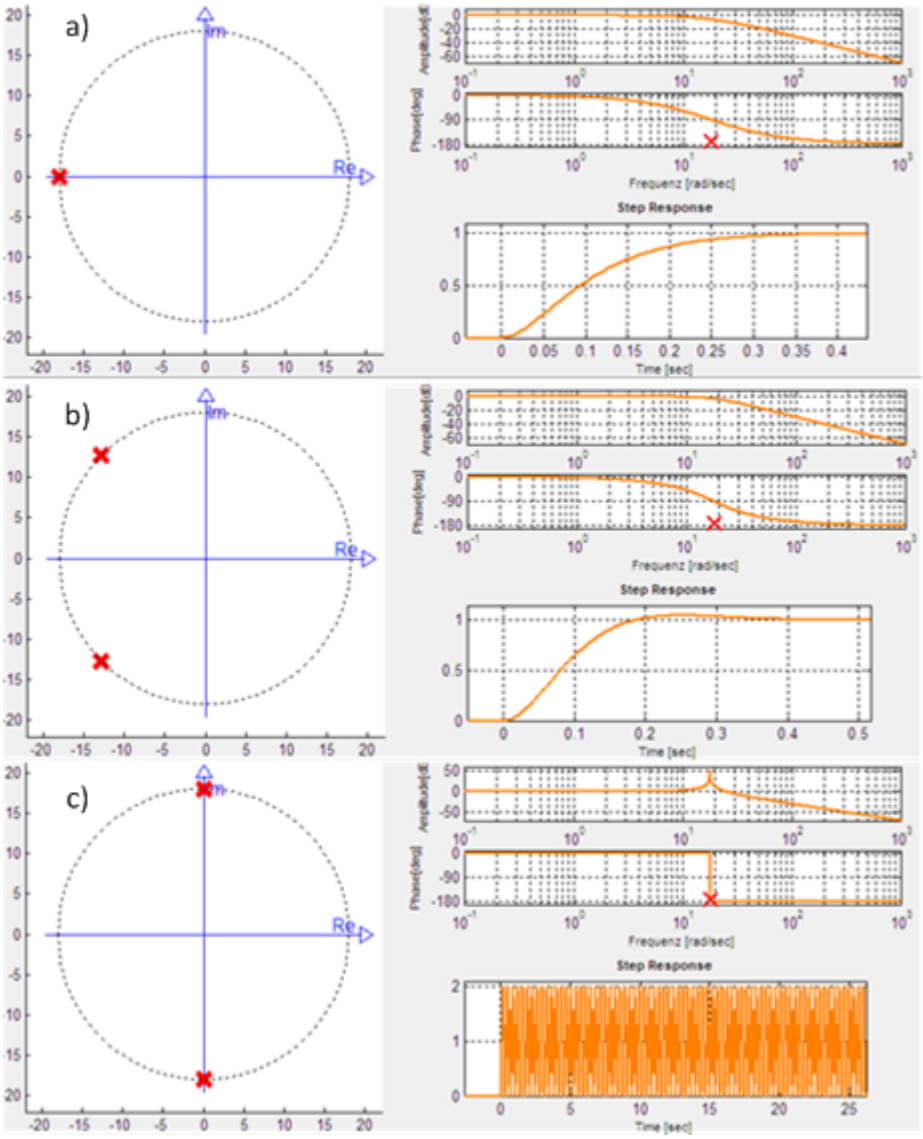

Abb. 4.4: Einfluss unterschiedlicher Dämpfungswinkel, a) $\Phi_d = 0°$, b) $\Phi_d = 45°$, c) $\Phi_d = 90°$

4.1.4 Bode Diagramm in Matlab

In Matlab gibt es bereits mehrere Möglichkeiten das Bode Diagramm einer Übertragungsfunktion graphisch darzustellen. Beide Möglichkeiten, die im Folgenden aufgezählt werden haben ihre Schwachstellen:

bode(sys) Die Funktion `bode(sys)` plottet das Bode Diagramm eines dynamischen Systems *sys*. Der Frequenzbereich wird dabei automatisch festgelegt. Bei Übertra-

gungsfunktionen höherer Ordnung entstehen hierbei ungewünschte Phasensprünge. Abbildung 4.5 zeigt einen auf `bode(sys)` basierenden Phasengang eines Systems 4.ter Ordnung (keine Nullstellen, vier Pole).

freqresp(sys,w) Die Funktion `h=freqresp(sys, w)` berechnet den Frequenzgang des dynamischen Systems *sys* mit dem Frequenzvektor *w*. Hierbei muss der Frequenzgang noch in den Amplitudengang und Phasengang zerlegt werden. Bei Verwendung dieser Methode wird der Phasengang abhängig von der Anzahl der Pole nach oben verschoben. Die Lage des Phasengangs ist folglich nicht korrekt. Als Beispiel ist in Abbildung 4.6der Phasengang eines Allpassgliedes 1. Ordnung mit `freqresp(sys, w)` dargestellt. Der Phasengang beginnt bei 360°, müsste jedoch bei 0° starten. Somit ergibt sich eine Verschiebung nach oben um 360°.

Zur Sicherstellung eines korrekten Bode Diagramms für sämtliche Übertragungsfunktionen wurde die Funktion aus Listing A.6 geschrieben. Dabei wird zunächst der Frequenzgang mit `freqresp(sys, w)` errechnet, welcher den korrekten Verlauf, jedoch den falschen Startpunkt bei $\omega = 0,1$ aufweist. Mit `bode(sys)` wird anschließend der korrekte Wert des Phasengangs bei der Startfrequenz, hier: $\omega = 0,1$, ausgegeben und mit dem zuvor berechneten Wert des Phasengangs verglichen. Die Differenz wird mit dem korrekten Verlauf addiert.

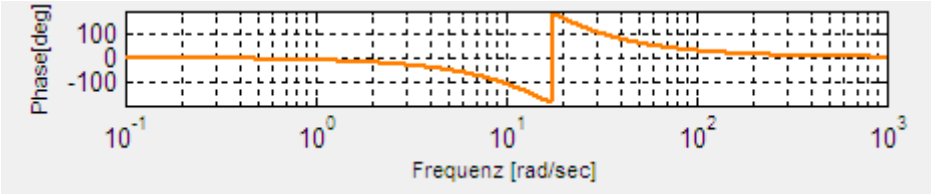

Abb. 4.5: Bode Plot mit `bode(sys)`

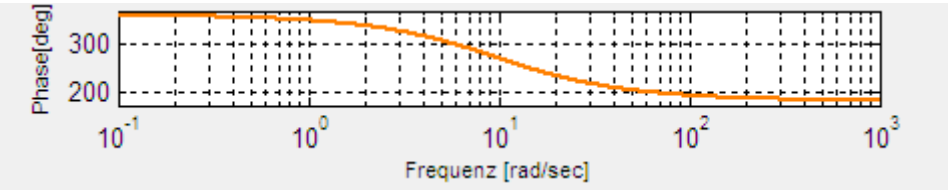

Abb. 4.6: Bode Plot eines Allpassgliedes 1.Ordnung mit `freqresp(sys, w)`

4.2 Reglerentwurfsaufgabe

Dieses Kapitel beschreibt die Durchführung des zweiten wichtigen Abschnitts des Matlab GUIs, den Reglerentwurf.

4.2.1 Berechnung der Übertragungsfunktionen

Wie in Kapitel 1.2 erwähnt, besteht der zweite Abschnitt dieser Bachelorarbeit darin, das Tool so zu erweitern, dass es für den Reglerentwurf verwendet werden kann. Um einen einschleifigen Regelkreis wie in Abbildung 2.2 zu konstruieren, sind die folgenden Übertragungsfunktionen relevant:

- Führungsübertragungsfunktion
- Streckenübertragungsfunktion
- Reglerübertragungsfunktion

Bei Kenntnis zweier Übertragungsfunktionen kann die fehlende berechnet werden. Die Führungsübertragungsfunktion, die auch Wunschübertragungsfunktion oder Übertragungsfunktion des geschlossenen Regelkreises genannt wird, ist folgendermaßen definiert:

$$G_W(s) = \frac{G_0(s)}{1 + G_0(s)},\ mit\ G_0(s) = G_R(s)G_S(s) \tag{4.5}$$

In der Realität ist meist die Streckenübertragungsfunktion $G_S(s)$, sowie die Führungsübertragungsfunktion $G_W(s)$ bekannt, die Reglerübertragungsfunktion wird gesucht. Die direkte Berechnung von $G_R(s)$ aus $G_W(s)$ und $G_S(s)$ ist:

$$G_R(s) = \frac{G_W(s)}{G_S(s)(1 - G_W(s))} \tag{4.6}$$

Diese Berechnung ist erforderlich, wenn die Führungsübertragungsfunktion in der komplexen Ebene angezeigt wird. Außerdem ist es möglich, die Streckenübertragungsfunktion, oder die Reglerübertragungsfunktion in der komplexen Ebene zu sehen. Ist die Reglerübertragungsfunktion ausgewählt erfolgt bei Änderung der Übertragungsfunktion eine Neuberechnung von $G_W(s)$, nach 4.5.
Bei Darstellung von $G_S(s)$ besteht die Möglichkeit $G_R(s)$, oder $G_W(s)$ neu berechnen zu lassen.

4.2.2 Der PID-Regler

Der PID Regler ist der am häufigsten verwendete Regler [8]. Er setzt sich aus dem P-Anteil, dem I-Anteil und dem D-Anteil zusammen.

P-Anteil Der P-Anteil hat die Übertragungsfunktion k_P und stellt somit eine Verstärkung dar.

I-Anteil Der I-Anteil, oder auch integraler Anteil genannt, ist durch die Übertragungsfunktion $\frac{k_I}{s}$ gekennzeichnet. Dieser Teil des PID Regler gibt solange ein Stellsignal aus, bis die Regelabweichung verschwunden ist. Befindet sich bereits ein integraler Anteil in der Übertragungsfunktion der Regelstrecke, genügt ein P-Regler um eine bleibende Regelabweichung zu verhindern [5].

D-Anteil Der D-Anteil, oder auch differenzierender Anteil genannt, hat die Übertragungsfunktion $k_D s$. Die Stärke des von dem D-Anteil ausgegebenen Stellsignals hängt immer von der Änderung der Regelabweichung ab. Je stärker die Regelabweichung schwankt, desto intensiver greift die Regelung ein[5].

Je nachdem ob diese Anteile in Reihe oder parallel zueiander liegen, wird zwischen dem PID Regler in *multiplikativer* und *additiver* Form unterschieden:

Additiver PID Regler Der ideale additive PID Regler wird durch die Gleichung

$$G_R(s) = k_P + \frac{k_i}{s} + k_D s = k_P(1 + \frac{1}{T_I s} + T_D s) = \frac{k_P s + k_i + k_D s^2}{s} \tag{4.7}$$

beschrieben. Der Bruch auf der rechten Seite von (4.7) zeigt, dass das System aufgrund der höheren Zählerordnung technisch nicht realisierbar ist. Durch eine Verzögerung in Form eines PT1-Glieds, welches dem D-Anteil in Reihe geschalten wird, entsteht der reale PID Regler in additiver Form. Dieser ist kausal und somit technisch realisierbar, siehe Gleichung (4.8):

$$G_R(s) = k_P + \frac{k_i}{s} + \frac{k_D s}{1 + T_1 s} = k_P(1 + \frac{1}{T_I s} + \frac{T_D s}{1 + T_1 s}) = \tag{4.8}$$

$$\frac{(k_P T_1 + k_D)s^2 + (k_I T_1 + k_P)s + k_I}{T_1 s^2 + s}$$

Multiplikativer PID Regler In Gleichung (4.10) ist bereits die technisch realisierbare Form des multiplikativen PID Reglers dargestellt. Es handelt sich dabei um eine

andere Übertragungsfunktion als beim additiven PID-Regler.

$$G_R(s) = \frac{k_P(1 + T_I s)(1 + T_D s)}{(T_I s)(1 + T_1 s)} \tag{4.9}$$

Eine wichtige Anmerkung ist, dass die additive und multiplikative Form des PID Reglers nicht die gleichen Übertragungsfunktionen darstellen. Gleiche Werte für k_P, k_I und k_D liefern folglich unterschiedliche Reglerverhalten [5]. Der additive PID-Regler wird für den Entwurf im Zeitbereich verwendet. Der multiplikative PID-Regler wird für die Auslegung im Frequenzbereich angewandt.

4.2.3 Anwendung des PID Reglers

Im entwickelten Programm gibt es verschiedene Möglichkeiten einen PID Regler zu realisieren. Die einfachste Variante ist, durch gezielte Änderung der Parameter k_P, k_I und k_D und gleichzeitiger Simulation der Regelgröße, das optimale Regelverhalten zu erlangen. Des Weiteren können die Reglerentwurfsverfahren nach Ziegler und Nichols sowie nach Chien, Hrones und Reswick angewandt werden. Zuletzt wird noch eine bestmögliche PID-Berechnung nach der Methode der kleinsten Fehlerquadrate angeboten. In diesem Kapitel wird durch gezielte Parameteränderung die Funktionsweise der PID-Anteile erläuert.

Eine Regelstrecke mit der Übertragungsfunktion

$$G_S(s) = \frac{180}{s^2 + 7s + 180} \tag{4.10}$$

soll mittels eines PID-Reglers geregelt werden. Die Strecke weist typisches PT2-Verhalten auf, siehe Abbildung 4.7.

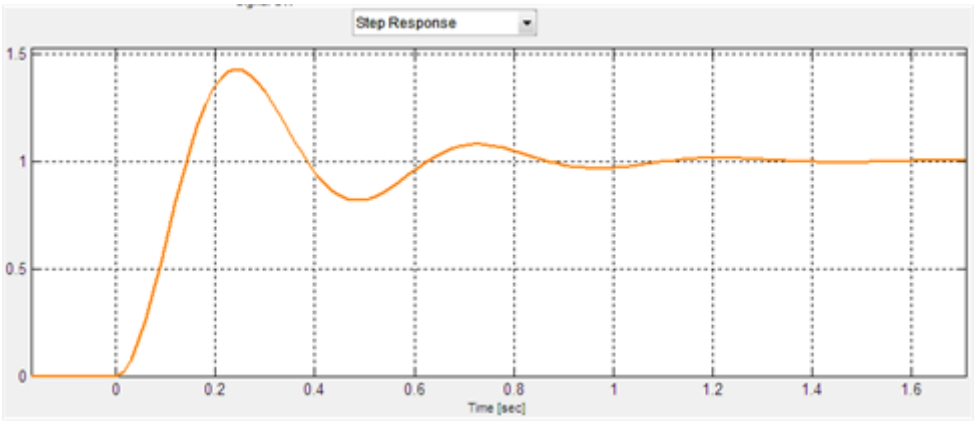

Abb. 4.7: Sprungantwort der Regelstrecke

Im ersten Schritt wird nun ein reiner P-Regler mit unterschiedlichen Werten für k_P verwendet. Der geregelte Verlauf ist in Abbildung 4.8 gekennzeichnet. Es wird deutlich, dass bei beiden Proportionalreglern eine Regelabweichung vorhanden ist. Dies liegt an dem fehlenden I-Anteil in der Übertragungsfunktion des Reglers, bzw. der Strecke.

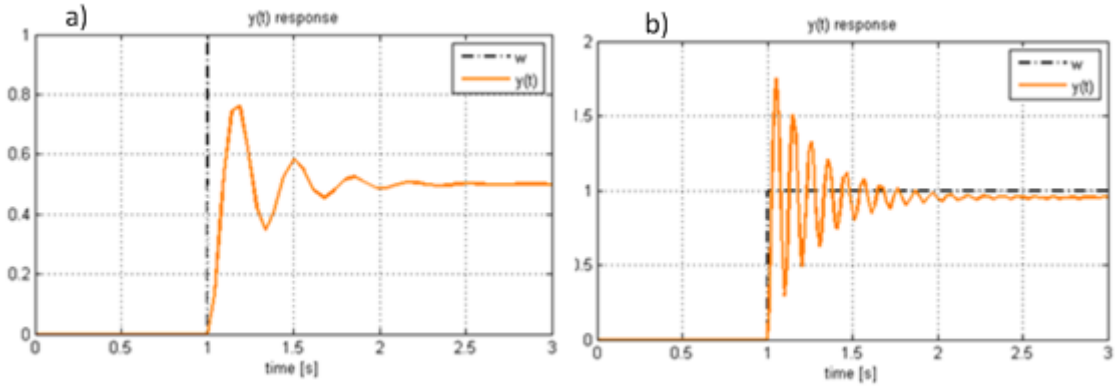

Abb. 4.8: P-Regler mit a) $k_P = 1$, b) $k_P = 20$

Als Nächstes wird das Regelverhalten bei Wahl eines PI-Reglers und eines PD-Reglers analysiert. In Abbildung 4.9 ist das Regelverhalten eines PI-Regler mit $k_P = 2$ und $k_I = 3$ dem eines PD-Regler mit $k_P = 3$ und $k_D = 3$ gegenübergestellt. Die Parmeter wurden frei gewählt.
Die Wirkungsweise beider Regler wird dabei deutlich: Während der PI-Regler kurz nach dem Sprung ein träges Verhalten aufweist, reagiert der PD-Regler sofort nach der Sollwert-Änderung sehr intensiv. Dies äußert sich in starkem Überschwingen. Aufgrund des I-Anteils kann der PI-Regler die Regelabweichung etwa zwei Sekunden nach dem Sprung ausgleichen. Der PD-Regler hat eine bleibende Regelabweichung.

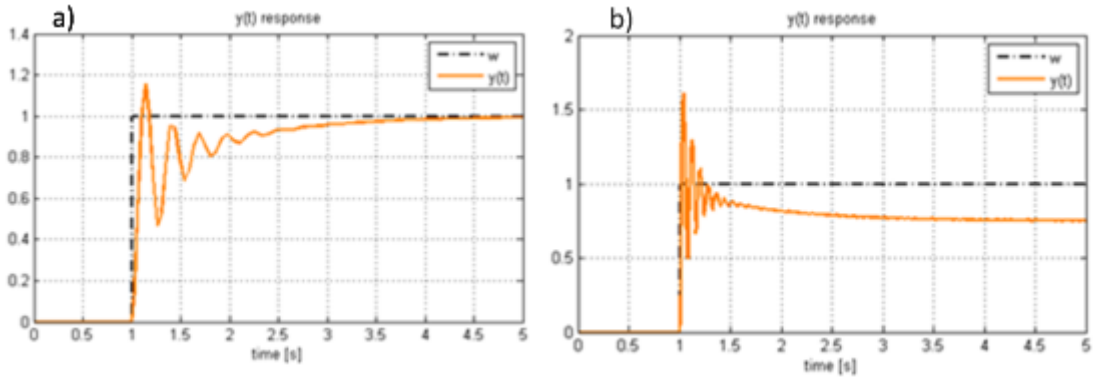

Abb. 4.9: a) PI-Relger mit $k_P = 2$ und $k_I = 3$, b) PD-Regler mit $k_P = 3$, $k_D = 3$

Der PID-Regler kombiniert die Stärken des PI und PD-Reglers. Abbildung 4.10 zeigt die Simulation der Regelgröße für den PID Regler: $2,3 + \frac{10}{s} + \frac{0,13s}{1+0,1s}$. Die Regelgröße reagiert schnell auf den Sprung der Führungsgröße und erreicht den Sollwert nach ungefähr einer Sekunde.

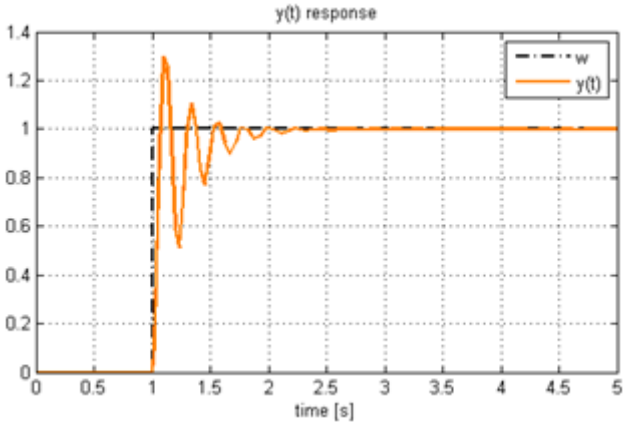

Abb. 4.10: PID Regler mit $k_P = 2.3$, $k_I = 10$, $k_D = 0.13$

Dieses Beispiel zeigt, dass mit dieser Matlab GUI das Verständnis der Reglerverhalten an konkreten Beispielen geschult werden kann. In Abbildung 4.11 sind die Einstellungen für den entworfenen PID-Regler dargestellt.

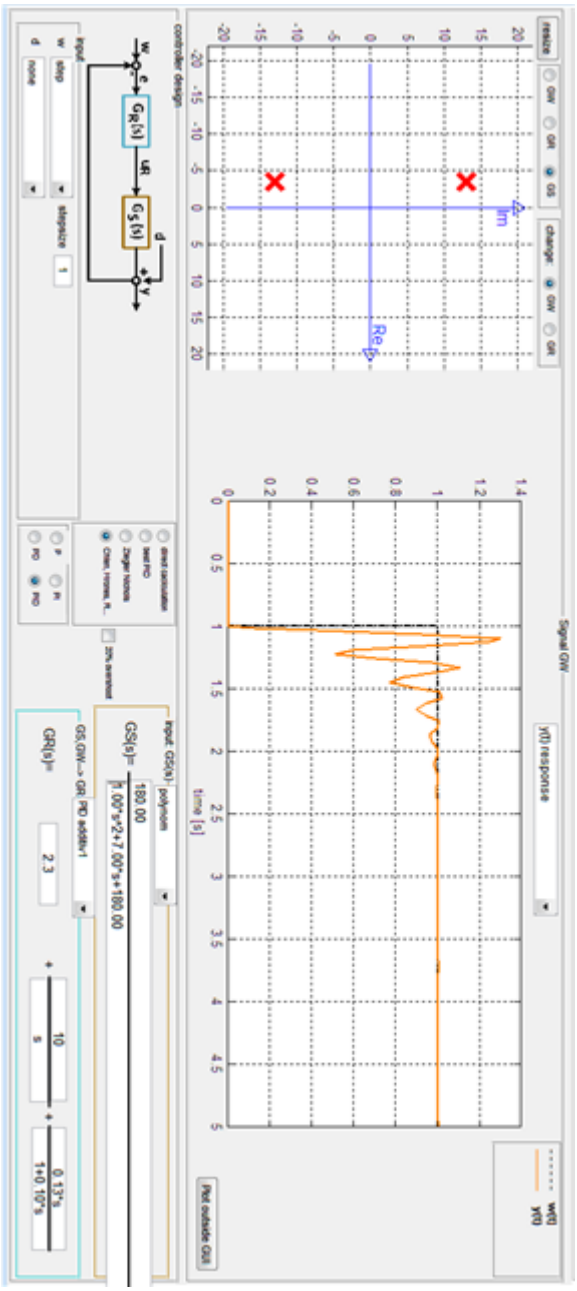

Abb. 4.11: Einstellungen für PID-Reglerauslegung

4.2.4 Simulation des geschlossenen Regelkreises

Neben der Neuberechnung ausgewählter Übertragungsfunktionen ist die Simulation des geschlossenen Regelkreises eine wichtige Anforderung an das Programm. Dabei

kann ausgewählt werden zwischen:

- Regelgröße $y(t)$
- Amplitudengang des geschlossenen Systems
- Phasengang des geschlossenen Systems
- Nyquist Diagramm des geschlossenen Systems
- Regelabweichung $e(t)$

Bei Kenntnis von $G_R(s)$ und $G_S(s)$ oder nur von $G_W(s)$ ist ein Regelkreis, wie in Abbildung 2.2, mit Matlab einfach zu simulieren. Der Amplitudengang, Phasengang sowie das Nyquist-Diagramm berechnen sich analog wie in Kapitel 4.1 mit den Matlab Funktionen `bode()` und `freqresp()`. Für $y(t)$ und $e(t)$ wird die Matlab Funktion `lsim(sys, u, t)` verwendet. Diese Funktion berechnet die zeitabhängige Antwort eines dynamischen Systems mit den folgenden Eingabeparametern:

sys: Übertragungsfunktion in `zpk`-, oder `tf`-Form
u: Vektor Eingangssignal
t: Vektor Zeitsignal

Die Werte des Zeitvektors müssen arithmetisch ansteigen. Dies bedeutet beispielsweise, dass ein Einheitssprung nicht exakt wiedergegeben werden kann. Ein für `lsim(sys, u, t)` verwendeter Einheitssprung würde wie in Abbildung 4.12 dargestellt aussehen. Je kleiner der Abstand dt, desto realistischer wird die Sprungfunktion dargestellt. Wird ein Dirac Impuls als Eingangssignal benötigt, so kann dieser näherungsweise wie in Abbildung 4.13 realisiert werden. Die Impulsbreite wird dabei in eine bestimmte Anzahl an Abschnitten unterteilt. Die Impulsbreite wird in zehn Bereiche aufgeteilt.

Abbildung 4.15 und 4.16 zeigen die plotbaren Signale, bei Auswahl der Regelgröße oder Regelabweichung. Im Folgenden wird die Bedeutung der gezeigten Verläufe beschrieben:

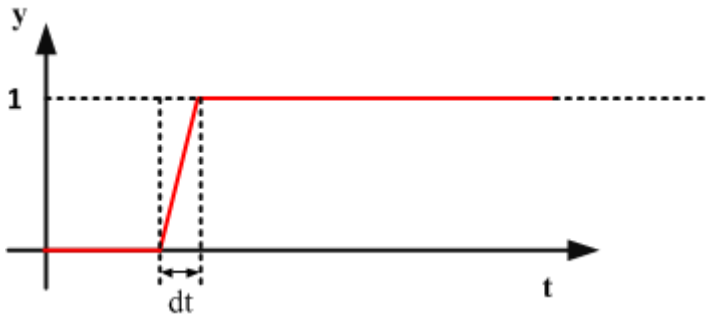

Abb. 4.12: Einheitssprung zur Verwendung für `lsim(sys, u, t)`

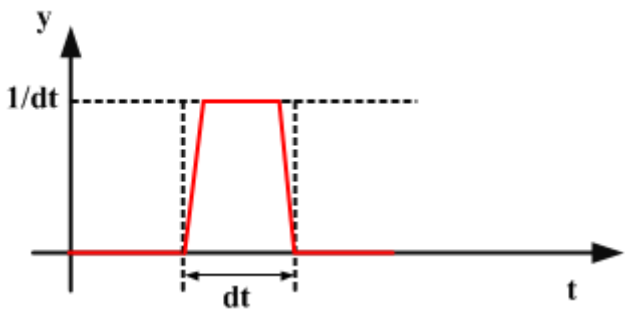

Abb. 4.13: Impulsfunktion zur Verwendung für `lsim(sys, u, t)`

schwarz gestrichtelt Diese Linie stellt das Signal der Führungsgröße dar.

magenta gestrichelt Diese Linie stellt das Signal der Störgröße dar und wird nur bei Auswahl einer Störung geplottet.

blau gestrichelt Der blau gestrichelte Plot bezeichnet $y(t)$ oder $e(t)$ ohne Stellsignallimitierung. Die Funktionswerte werden mit der Funktion `lsim(sys, u, t)` berechnet. Aufgrund der eben erläuterten Annäherungen der Eingangssignale stellen die berechneten Ausgangssignale keine exakte Lösung dar. Dies ist in Abbildung 4.14 zu erkennen. Die orange Linie fällt erst bei Ende des Impulses ab. Die blau gestrichelte Linie fällt hingegen bereits 0,1 Sekunden früher ab.

orange Der orangefarbige Graph ist $y(t)$ oder $e(t)$ mit Stellsignallimitierung. Die hierfür berechneten Funktionswerte stammen aus den Ausgabewerten des Simulink Modells.

Die Limitierung des Stellsignals ist eine wichtige Komponente bei der Simulation eines Regelkreises. In Abbildung 4.17 ist ein Regelkreis mit Stellsignallimitierung dargestellt. Der Steller oder auch Aktuator genannt kann beispielsweise ein Elektromotor sein. Dieser erhält von der Regeleinrichtung, welche durch ein Steuergerät realisiert werden kann, ein Stellsignal. Motoren haben immer einen maximalen Stromwert, der verarbeitet werden kann. Stromsignale oberhalb dieses Wertes können zu Schäden im Motor führen. Deshalb muss das Stellsignal limitiert werden. Die Limitierung kann negative Auswirkungen auf die Regelgröße haben. Das Erreichen der Führungsgröße wird hinausgezögert, was auch in den Abbildungen 4.15 und 4.16 zu erkennen ist.
Die Limitierung des Stellglieds bewirkt nichtlineares Verhalten des Systems. Deshalb ist die Simulation einer Limitierung in Matlab nicht möglich. Folglich muss diese über eine numerische Berechnung in Simulink erfolgen. Abbildung 4.18 zeigt das verwendete Simulink Modell.
Die *Multiport Switch* Blöcke werden eingesetzt, um nicht für jedes Eingangssignal der Führungs- oder Störgröße ein separates Simulink Modell aufbauen zu müssen. Die Entscheidung, welche Verzweigung aktiv ist, erfolgt über einen separaten Eingang.

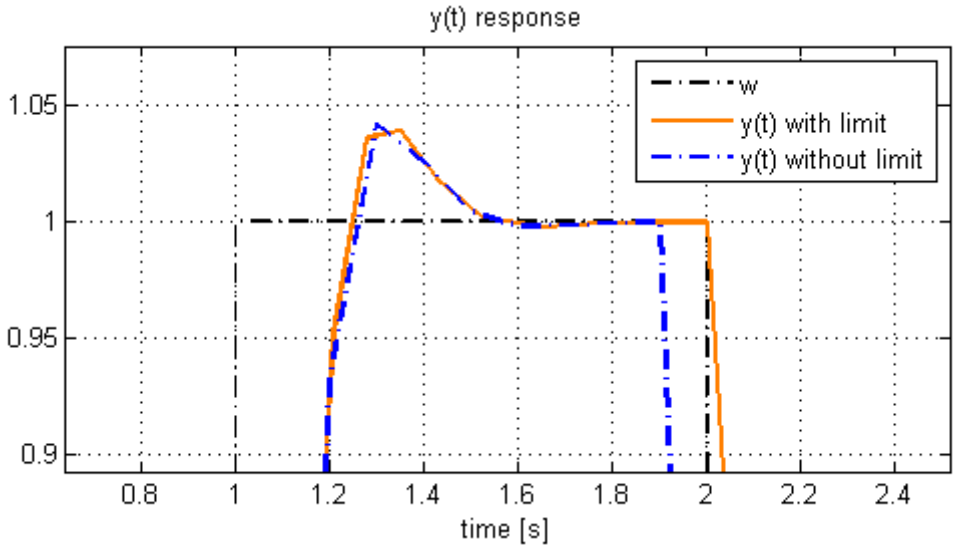

Abb. 4.14: Impuls

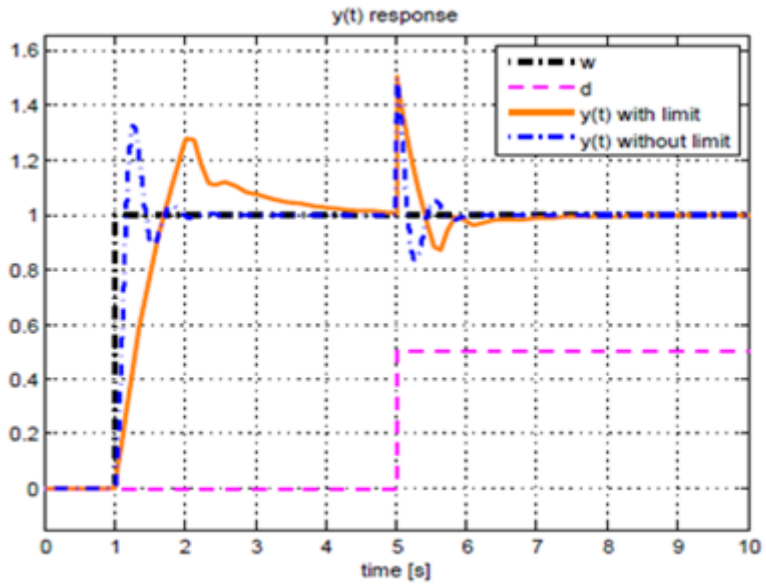

Abb. 4.15: Regelgröße $y(t)$

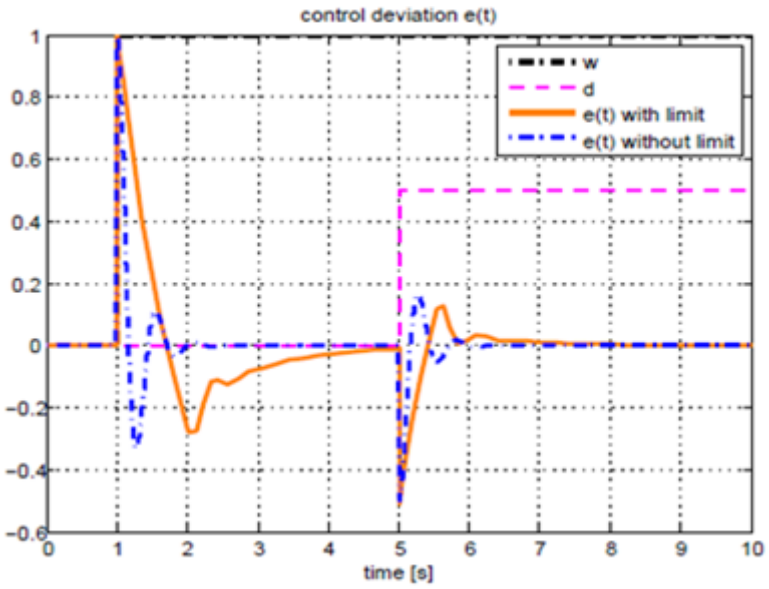

Abb. 4.16: Regelabweichung $e(t)$

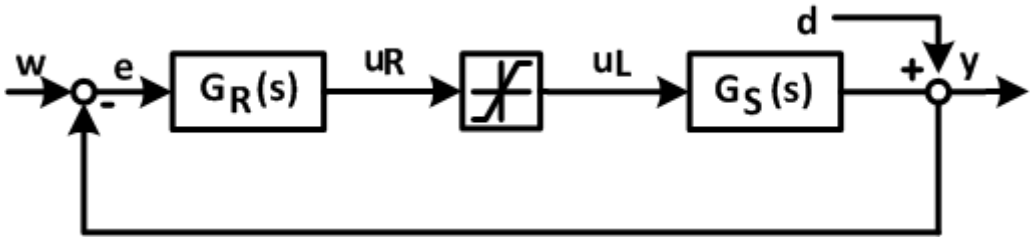

Abb. 4.17: Regelkreis mit Stellsignallimitierung im Frequenzbereich

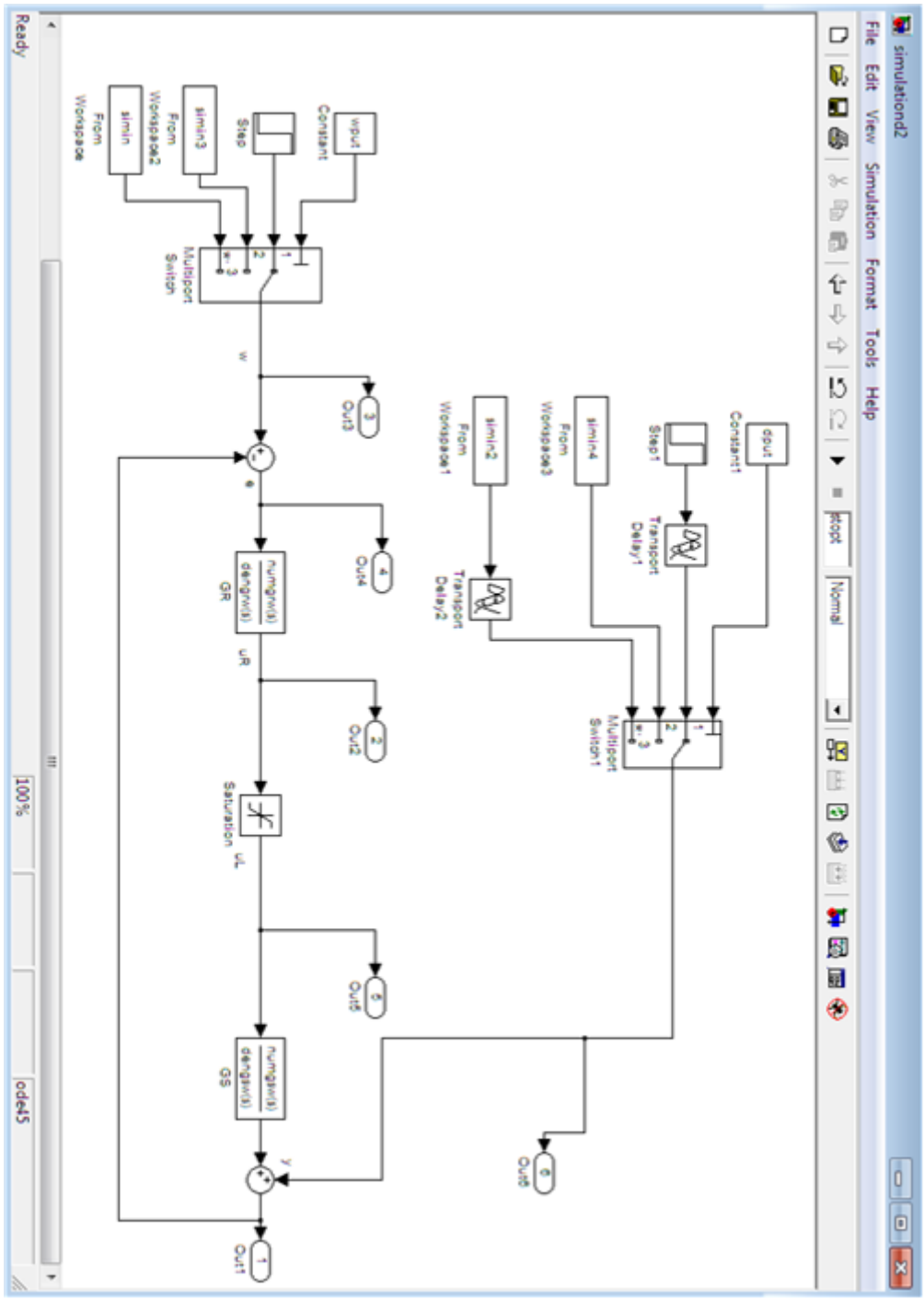

Abb. 4.18: Simulink Modell

4.2.5 Eingangssignale

Die wählbaren Eingangssignale für die Führungsgröße bzw. die Störgröße werden in diesem Unterkapitel vorgestellt.

Die Abbildung 4.19 zeigt die möglichen Eingangssignale für eine Störung. Diese sind:

Sprung Die Sprungfunktion wurde bereits in Kapitel 2.7 behandelt. Hier kann zusätzlich noch die Sprunghöhe und bei einem Sprung als Störgröße die Startzeit frei gewählt werden.

Impuls Die Impulsfunktion wurde bereits in Kapitel 2.8 behandelt. Wird der Impuls als Störgröße gewählt, so kann auch hier die Startzeit eingestellt werden.

function input Hier kann über ein Edit-Feld eine zeitabhängige Funktion eingegeben werden. Die Variable für den Zeitwert muss mit t angegeben werden. Für eine exponentiell absteigende Sinusfunktion am Eingang kann folglich $sin(1.5 * t). * exp(-0.1 * t) + 0.5$ eingegeben werden. Dadurch entsteht das Störgrößensignal aus Abbildung 4.20.

from workspace Über diese Auswahlmöglichkeit können im Workspace befindliche Signale in die GUI eingebettet werden. Das Signal muss im Workspace als eine nx2 (n>10) Matrix angegeben sein. In der ersten Spalte steht dabei immer der Zeitvektor. In Spalte 2 sind die Funktionswerte angeordnet. Bei einem Klick auf diesen Button werden im Workspace alle Variablen durchsucht, die solch eine Matrix darstellen. Wurde beispielsweise eine Variable namens *signal* gefunden, wird der Nutzer im Command Window folgendermaßen angesprochen:

```
Do you want to use the variable 'signal' for the input d?
y/n
```

Der Bediener hat die Möglichkeit mit der Eingabetaste dieses Signal auszuwählen oder durch n + Eingabetaste den Workspace weiter zu durchsuchen.
Wird das folgende Signal aus dem Workspace geladen, so entsteht das Eingangssignal aus Abbildung 4.21.

```
0     0
1     0
2     0
3     1
4     2
5     3
6     4
7     5
```

```
 8        5
 9        5
10        5
```

noise Bei dieser Auswahl wird ein Rauschen simuliert. Als zusätzliche Parameter zur Bestimmung des Rauschens müssen der Mittelwert, sowie die Standardabweichung des Signals angegeben werden. Der Abstand im x-Wertebereich der einzelnen Punkte kann unter *settings/delta t* eingestellt werden. In Abbdilung 4.22 ist ein Rauschen zu sehen mit Mittelwert = 2, Standardabweichung = 0,1, delta t = 0,2. Dieses Signal steht nur für die Störgröße zur Verfügung.

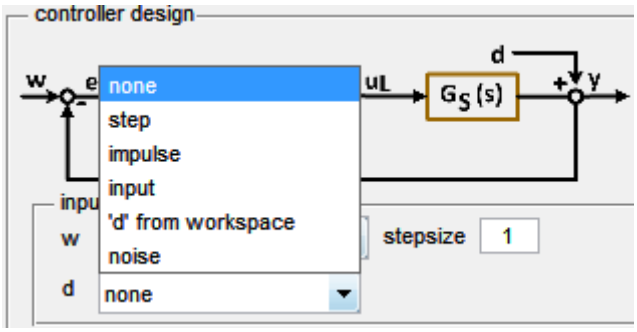

Abb. 4.19: Eingangssignale für die Störgröße

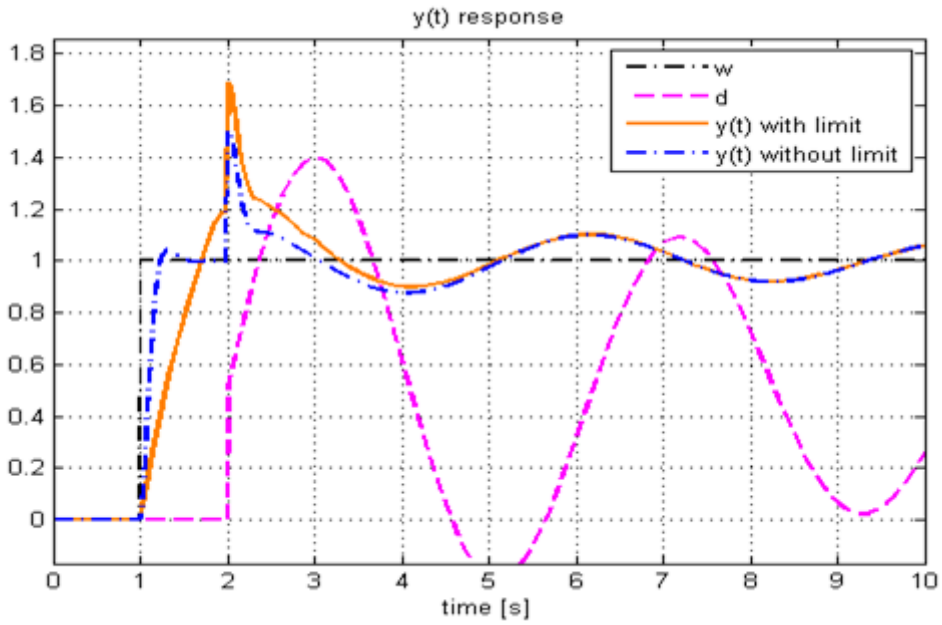

Abb. 4.20: Exponentiell fallendes Sinussignal als Störgröße

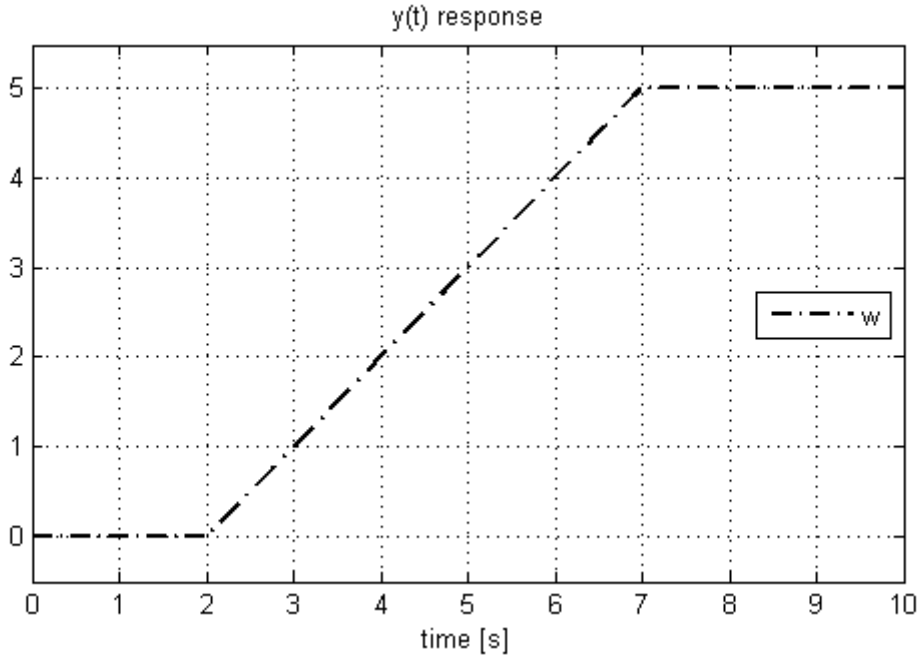

Abb. 4.21: Signal vom Workspace

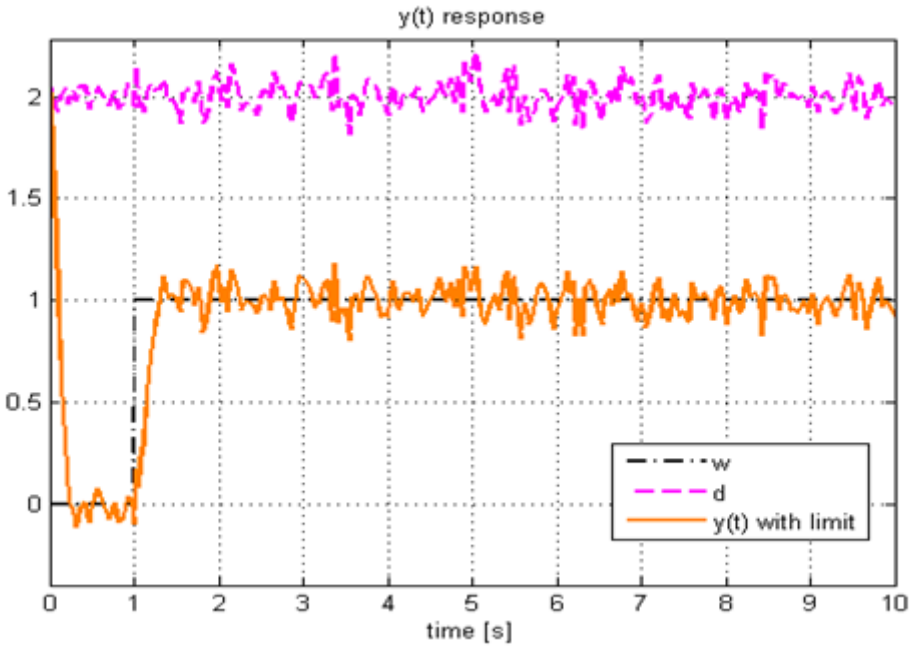

Abb. 4.22: Rauschen mit Mittelwert = 2, Standardabweichung = 0,1, delta t = 0,2

4.3 Animationen

Die Animationen sollen den Verlauf der Sprungantwort an mehreren Beispielen veranschaulichen. Sowohl bei der Systemanalyse, als auch beim Reglerentwurf sind die folgenden Animationen wählbar:

- Feder Masse Animation
- Drehzahl Animation
- Nachlauf Animation
- Magnetschwebekörper Animation

Die drei zuletzt genannten Animationen wurden gewählt, um einen Bezug zu dem Regelungstechnik Praktikum des Studiengangs Produktionstechnik an der Hochschule Rosenheim zu schaffen.

In Abbildung 4.23 ist das Programmfenster bei Auswahl der Magnetschwebekörper Animation gezeigt. In diesem Fall schwingt sich die Kugel wie die gezeigte Sprungantwort zwischen den zwei Balken ein.
Das Listing im Anhang A.7 beinhaltet die Matlab Funktion zur Magnetschwebekörper Animation. Diese kann als Vorlage für beliebige Animationen verwendet werden, in welchen eine translatorische Bewegung dargestellt werden soll. Die Funktion besteht aus zwei Hauptabschnitten. Zeile 1 bis Zeile 40 beinhalten das erste Plotten der Kugel sowie der beiden Platten. In den restlichen Zeilen wird eine *for-Schleife* durchlaufen. Die Anzahl der Schleifendurchläufe ist genauso groß wie die Anzahl der Elemente des zuvor aus der Übertragungsfunktion *sys* berechneten Vektors *bew*. Somit werden den Handles der Plots bei jedem Schleifendurchlauf neue x-, y-, z-Werte zugeordnet. Dadurch beginnt sich die Kugel zu bewegen.

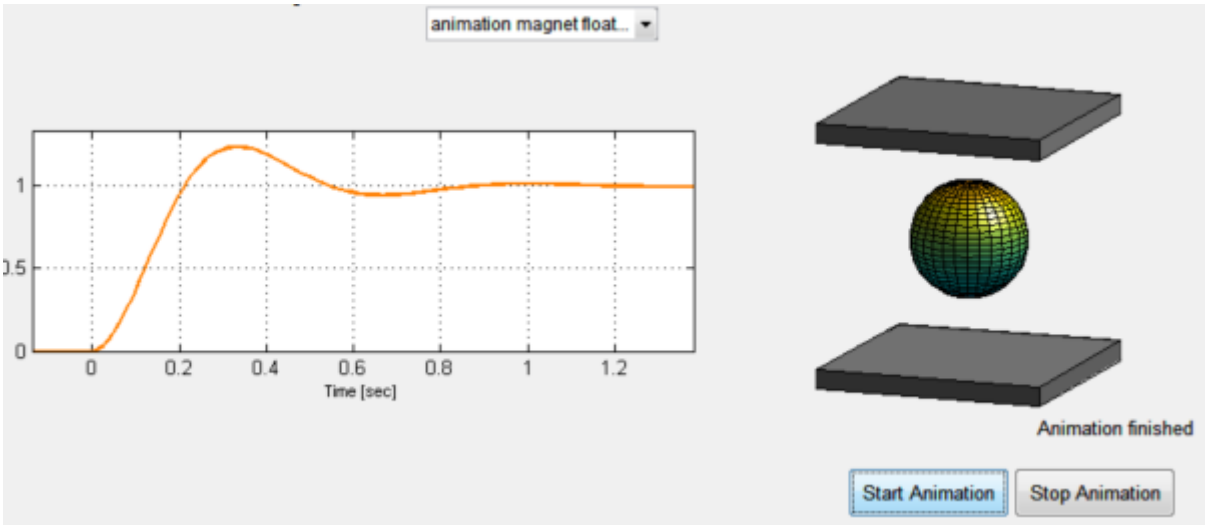

Abb. 4.23: Magnetschwebekörper Animation

4.4 Laufzeitoptimierung

Kurze Matlab Funktionen werden meist sehr schnell ausgeführt, sodass eine explizite Laufzeitoptimierung nicht notwendig ist. Trotzdem können auch einzelne Matlab Befehle eine lange Rechendauer in Anspruch nehmen. Deshalb bietet es sich immer an, zur Optimierung eines Programms eine von Matlab angebotene Hilfestellung wahrzunehmen. Hierfür werden die Befehle `profile on`, sowie `profile report` verwendet. Vor dem Starten eines Programms wird der erste der beiden Befehle ins Matlab Command Window eingetippt. Nun wird bei sämtlichen Codes, die anschließend durchlaufen werden, die benötigte Zeit gespeichert. Tippt man im Command Window den zweiten genannten Befehl ein, so ist das Aufzeichnen der Rechenzeiten beendet. So kann genau herausgefunden werden, welche Codes die meiste Rechenleistung in Anspruch nehmen. Dadurch kann gezielt versucht werden die Laufzeit zu verkürzen. Vor allem bei komplexen Programmen, in welchen mehrere Funktionen hintereinander durchlaufen werden, macht diese Vorgehensweise Sinn, um unnütze Codeverwendungen zu entdecken.
In Abbildung 4.24 ist der *profile report*, der während einer Polverschiebung aufgezeichnet wurde, dargestellt. In hierarchischer Anordnung werden sämtliche Funktionen angezeigt, die einen Teil zur Laufzeit beitragen. Nun kann man Funktionen auswählen und einzeln untersuchen. Abbildung 4.25 zeigt die Unterfunktionen in der Funktion *plotting*. Dabei zeigt sich, dass das Aktivieren eines *axes* Objekts mittels `axes(handles.axes)` ein Drittel der Laufzeit von *plotting* ausmacht. Mit diesem Hintergrundwissen kann beispielsweise versucht werden bei immer wiederkehrenden Plots nicht jedes mal einen neuen Plotbefehl zu verwenden. In diesem Fall bietet es sich an die x- und y-Werte des

handles mit dem Befehl `set(handle,'xdata', value1,'ydata',value2)` zu aktualisieren. Somit entfällt ebenfalls der `axes(handles.axes)` Befehl.

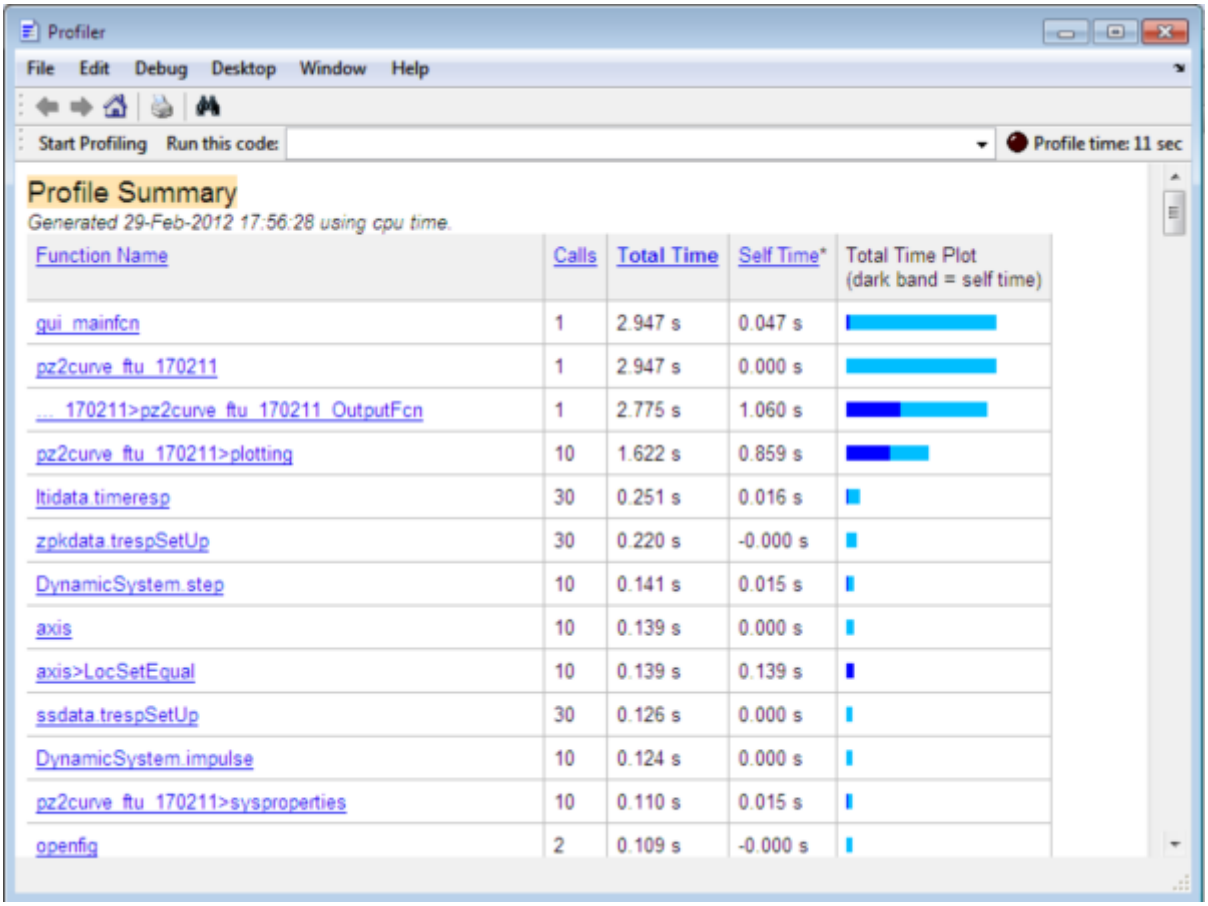

Abb. 4.24: profile report

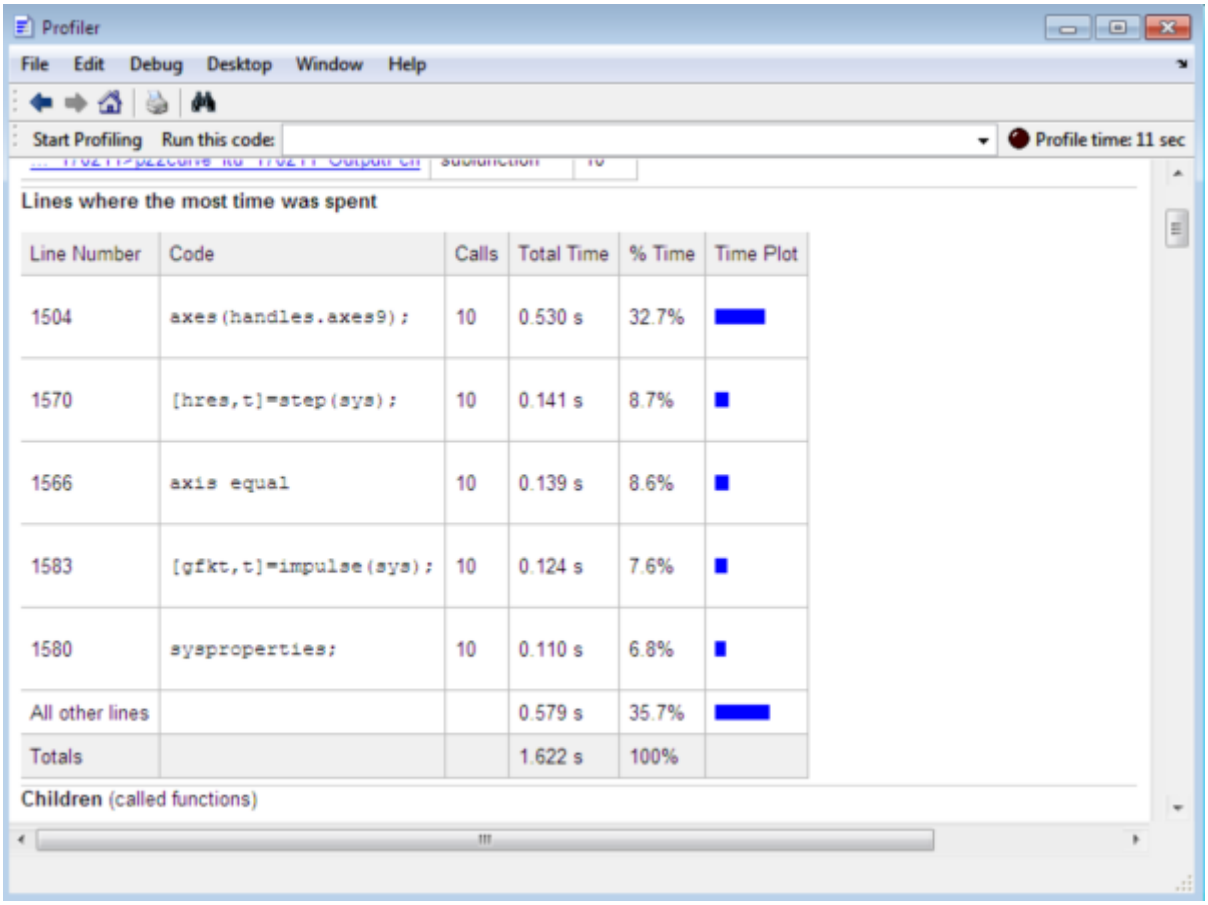

Abb. 4.25: profile report der Unterfunktion *plotting*

4.5 Überprüfen der Funktionen

Das Matlab m-file besteht aus vielen Funktionen, die bei einem Programmdurchlauf auch öfters ausgegeben werden. Um die Richtigkeit des Gesamtprogramms zu testen, bietet es sich an, die Einzelfunktionen extra zu überprüfen. Deshalb wird für komplexe Funktionen ein Test-file erzeugt. Dieses file wird im folgenden Text $run\_NamederFunktion$ genannt. Diese Vorgehensweise wird nun an einem Beispiel erläutert.

Eine Funktion, die sehr oft Verwendung findet ist diejenige, welche aus einer Übertragungsfunktion den jeweiligen String zur Darstellung im Edit Feld berechnet. Diese Funktion wurde $transferdisplay$ genannt. Die Beschreibung ist im Listing A.8 zu sehen. Die dazugehörige Testfunktion heißt $run\_transferdisplay$ und ist im Listing A.9 abgebildet. Für den Test wird ein Allpassglied 2ter Ordnung verwendet, mit den Polen

$-10+10j$, $-10-10j$, den Nullstellen $10+10j$, $10-10j$ und einem Verstärkungsfaktor $k = 1$. In den Zeilen 13 - 23 der $run\_transferdisplay$ Datei werden die korrekten Ergebnisse für die drei Darstellungsformen angezeigt. In Zeile 26 und 27 erfolgt die Bestimmung der Eingabeparameter für die Funktion $transferdisplay$. Anschließend werden jeweils die Strings des Zählers und Nenners für alle drei Möglichkeiten berechnet. Stimmen die ausgegebenen Zeichenketten mit den korrekten Ergebnissen überein, so wird zuletzt angezeigt, dass der Test erfolgreich durchgeführt wurde.

5 Anwendungsbeispiel Drehzahlregelung

Das Regelungstechnik Praktikum des Studiengangs Produktionstechnik der Hochschule Rosenheim besteht aus fünf Versuchen. Einer davon ist die Drehzahlregelung. In diesem Kapitel wird das System analysiert und anschließend die Auslegung und Simulation des Regelkreises mit Hilfe des entwickelten Programms erläutert.

5.1 Versuchsbeschreibung

Ein Gleichstrommotor mit einer Leistung von 100W und maximaler Drehzahl von 2000 $\frac{U}{min}$ bei 230V ist gegeben [13]. An der Ausgangswelle ist eine Last aufschaltbar, die eine Störgröße simulieren soll. Abbildung 5.1 zeigt das Blockschaltbild des Regelkreises mit den relevanten Messeinrichtungen. Es wird ein analoger PID-Universalregler verwendet. Wahlweise kann reines P-, PI-, PD- oder PID Verhalten eingestellt werden.

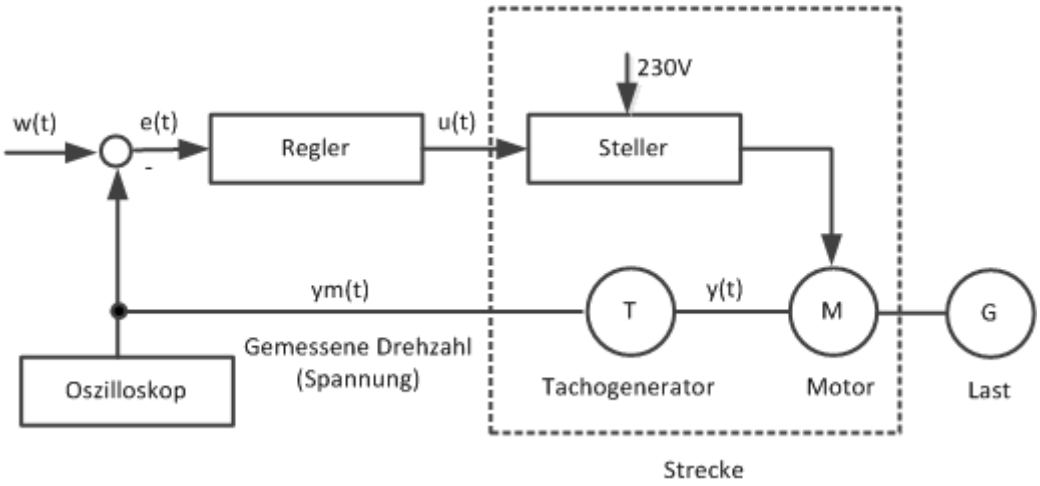

Abb. 5.1: Blockschaltbild des Regelkreises zur Drehzahlstabilisierung nach [13]

5.2 Versuchsdurchführung

Zunächst gilt es die Regelstrecke zu analysieren. Aufgrund der Charakteristik des Motors wird von einem PT2-Glied ausgegangen, mit folgender Übertragungsfunktion:

$$G_S(s) = \frac{K_S}{(1+T_1 s)(1+T_2 s)} = \frac{K_S}{1 + 2\frac{d}{\omega_0} + \frac{1}{\omega_0^2}s^2}$$

Die Parameter T_1, T_2 und K_S werden aus dem Verlauf der Sprungantwort ermittelt. Darauf wird nun nicht näher eingegangen.

Mit $\omega_0 = 10,823s$ und $d = 1$ konnte folgende Übertragungsfunktion in Zeitkonstantenform aufgestellt werden:

$$G_S(s) = \frac{1,28}{(0,097s+1)(0,088s+1)} = \frac{1,28}{0,00854s^2 + 0,185s + 1}$$

Zur Analyse dieses Systems im Programm kann die Übertragungsfunktion entweder direkt in den Edit-Feldern eingegeben werden, oder aus dem Matlab Workspace geladen werden. Letzteres wird erreicht indem

```
system=tf(1.28, [0.00854 0.185 1]);
```

im Command Window eingetippt wird. Im Programm muss nur noch *system/load system* ausgewählt werden. Das System mit zwei auf der Realachse liegenden Polen wird somit in die GUI eingeladen. Da die Pole dicht aneinander liegen kann die Achsenskalierung durch Drücken auf *resize* erweitert werden. Nun sieht man deutlich, dass es sich um zwei Pole handelt, siehte Abbildung 5.2.

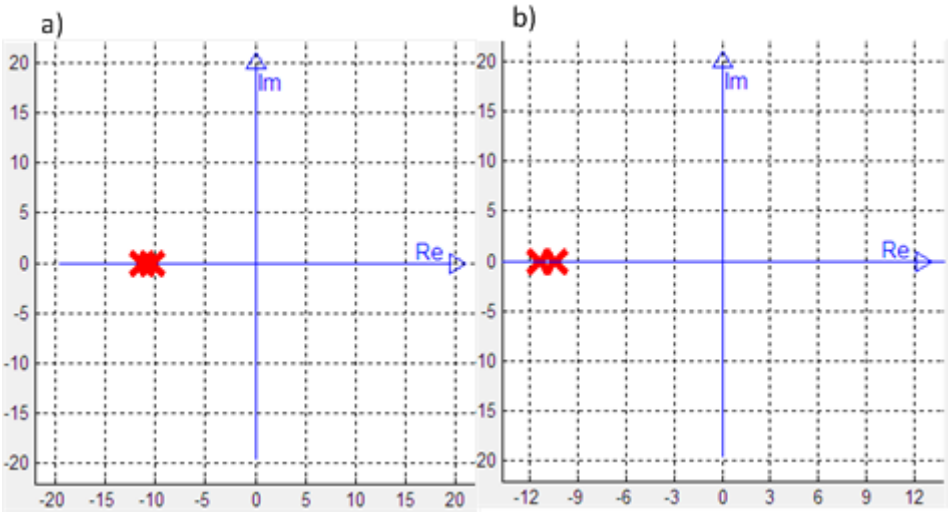

Abb. 5.2: komplexe Ebene, a) vor *resize*, b) nach *resize*

Das System ist stabil, da die Sprungantwort bei $t \to \infty$ gegen einen konstanten Wert und die Impulsantwort für $t \to \infty$ gegen Null konvergiert. Die vorhandene Stabilitätsreserve kann über Auswahl von *Nyquist* und anschließender Aktivierung der Checkbox *Plot Stabilitätsreserve* in Erfahrung gebracht werden (siehe Abbildung 5.3). Der Phasenrand beträgt 124°, der Amplitudenrand ist unendlich groß. Diese Werte können auch über das Bode Diagramm ermittelt werden, so in Abbildung 5.4 geschehen. Für den Phasenrand wird der Schnittpunkt des Amplitudengangs mit der 0 dB Linie in den Phasengang projeziert. Der Bereich zwischen Schnittpunkt dieser Linie mit dem Phasengang und der $\alpha = 180°$ Linie ist der Phasenrand.
Für den Amplitudengang wird der Schnittpunkt der $\alpha = 180°$ Linie mit dem Phasengang benötigt. Da der Phasenrand bei $\omega \to \infty$ gegen 180° konvergiert, gibt es diesen Schnittpunkt in dem speziellen Fall nicht. Deshalb ist der Amplitudenrand unendlich groß.

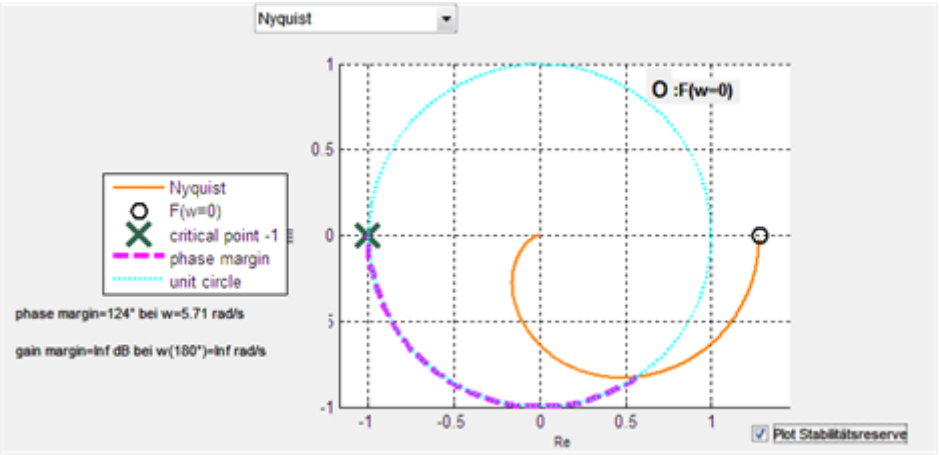

Abb. 5.3: Stabilitätsreserve über die Ortskurve

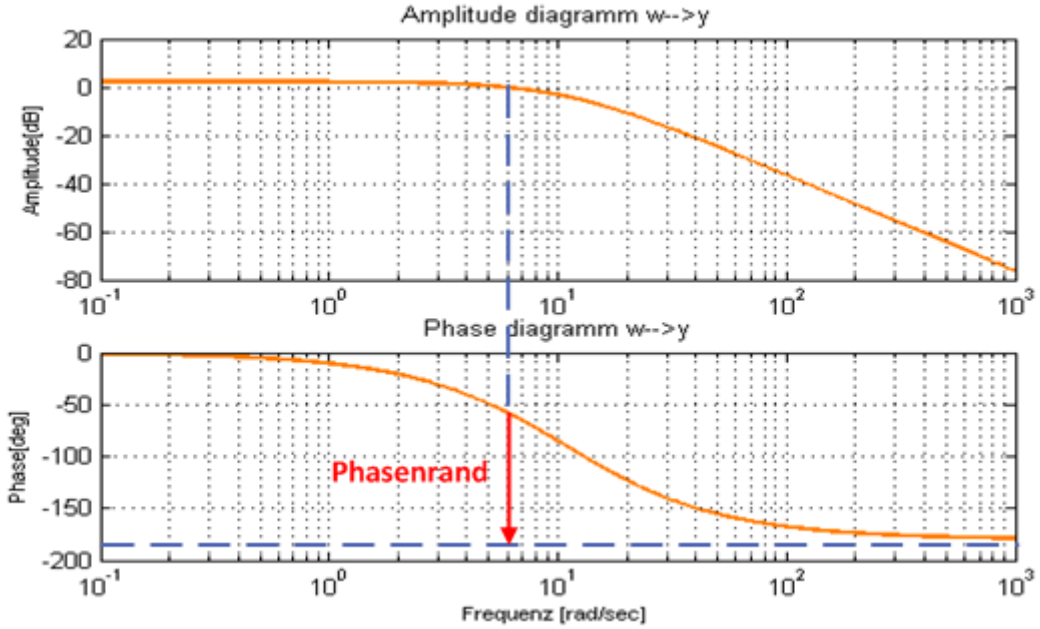

Abb. 5.4: Stabilitätsreserve über das Bode Diagramm

Der Motor soll im Praktikum mit 1000 $\frac{U}{min}$ betrieben werden. Dies entspricht einer Stellereingangsspannung von 3,8V. Wechselt man nun in *purpose/controller design* kann die Sprungantwort des offenen Systems mit einem Sprung von 3,8V simuliert werden, siehe Abbildung 5.5.

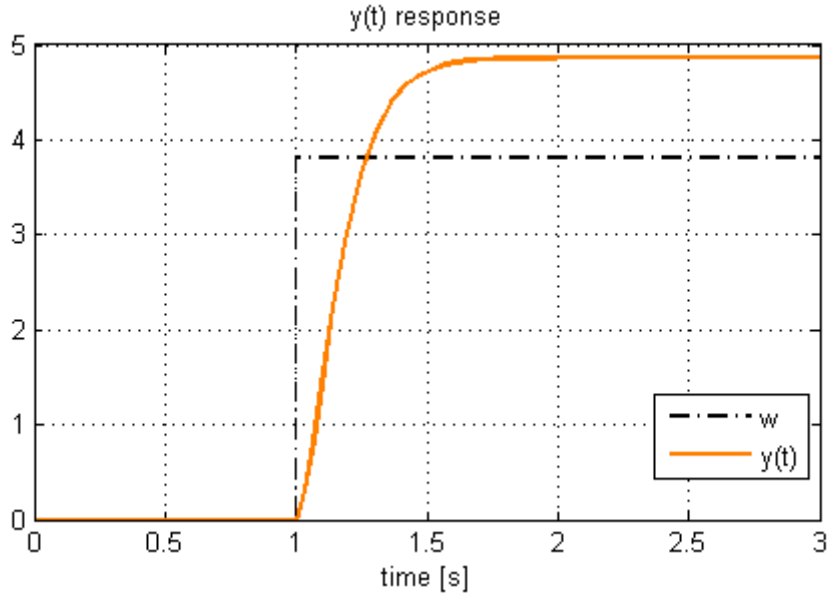

Abb. 5.5: Sprungantwort des offenen Systems bei einem Sollwertsprung von 3,8V

Das Ziel des Praktikums ist, die Regelstrecke so gut wie möglich zu regeln. Die Güte eines Systems kann durch die Überschwingweite, Anstiegszeit, Beruhigungszeit und die Überschwingzeit beschrieben werden. Im nächsten Schritt gilt es einen passenden Regler auszulegen.

5.2.1 P-Regler

Der Parameter K_R des P-Reglers wird durch die Vorgabe einen statischen Regelfaktor von $r = 0,2$ zu erreichen über die folgende Formel berechnet:

$$r = \frac{1}{1 + G_0(s)} mit\ s = 0$$

Für den P-Regler wurde somit die Übertragungsfunktion $G_R(s) = 3,125$ berechnet. Im Programm muss nun bei $G_S(s)$ die Streckenübertragungsfunktion eingegeben werden. Für einen frei wählbaren P-Regler wählt man beispielsweise *LS PID*. Nun kann der Wert von K_R im ersten Kästchen der Reglerübertragungsfunktion eingetippt werden. K_D und K_I müssen noch mit Null besetzt werden. Den Verlauf der geregelten Größe zeigt Abbildung 5.6.

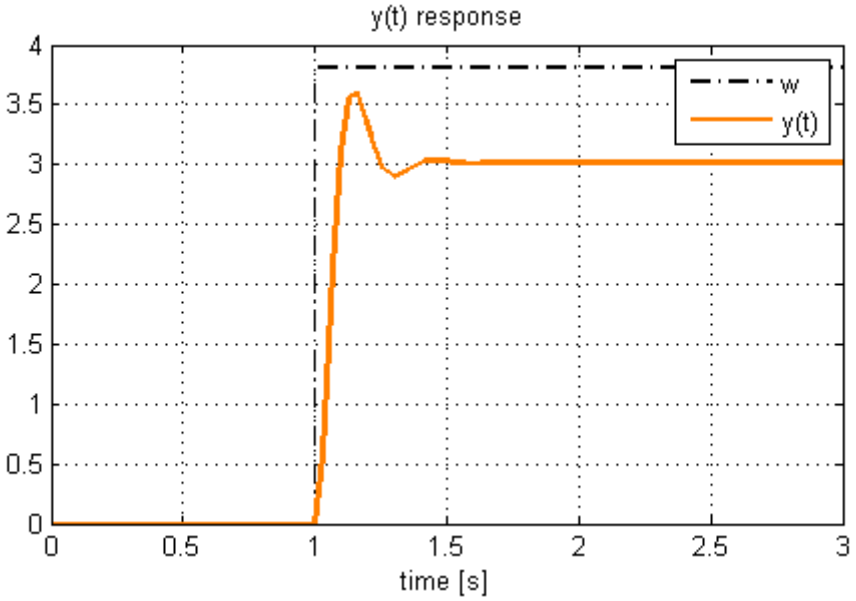

Abb. 5.6: Regelgröße mit P-Regler, $K_R = 3,125$

Das dargestellte Signal konvergiert bei $t \to \infty$ gegen 3V.
Bei sprungförmigem Eingangssignal entspricht der statische Regelfaktor der Regelabweichung. Deshalb gilt:

$$r = e(t) = \frac{3,8V - 3V}{3,8V} = 0,21$$

Dies belegt die Richtigkeit des simulierten Signals aus Abbildung 5.6

5.2.2 Reglerauslegung nach Chien, Hrones und Redwick

Eine weitere Möglichkeit den Regler auszulegen ist das Verfahren nach Chien, Hrones und Redwick anzuwenden. Dafür müssen die in Abbildung 5.7 gewählten Einstellungen eingegeben werden. Das Programm berechnet die Parameter für den PID-Regler. Zusätzlich hat man die Wahl nur P-, PI-, PD-, oder PID-Verhalten zu berücksichtigen (siehe Abbildung 5.7). Für den zu regelnden Motor wurden die folgenden die folgenden Übertragungsfunktionen berechnet:

$$G_{R_P}(s) = 2,29$$
$$G_{R_{PI}}(s) = 2,6 + \frac{8,3}{s}$$
$$G_{R_{PD}}(s) = 13,84 + \frac{0,18s}{1+0,1s}$$
$$G_{R_{PID}}(s) = 2,6 + \frac{10,35}{s} + \frac{0,03s}{1+0,1s}$$

Diese Gleichungen zeigen, dass jeweils eine neue Übertragungsfunktion berechnet wird. Beispielsweise wird bei Wechsel von einem PI-Regler zu einem P-Regler nicht lediglich der I-Anteil gelöscht, sondern auch einer neuer Wert für k_P berechnet.

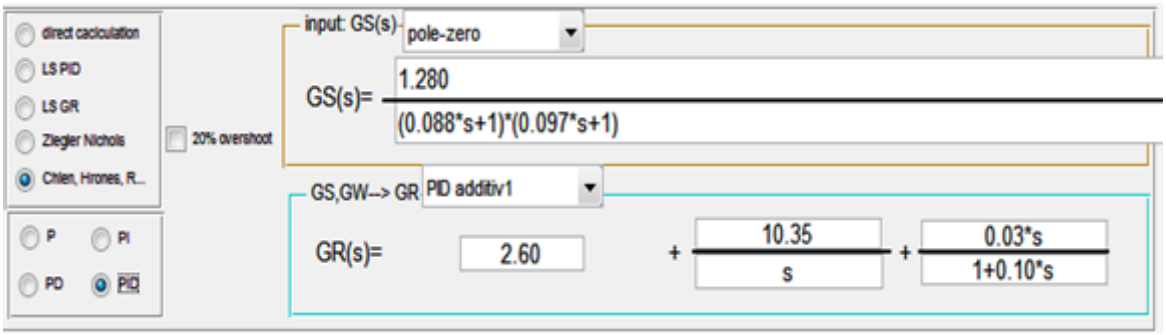

Abb. 5.7: Einstellungen für Reglerauslegung nach Chien, Hrones und Reswick

Abbildung 5.8 zeigt die Simulation des geschlossenen Regelkreises mit Reglerauslegung nach Chien, Hrones, Reswick. Als Strögröße wurde ein Sprung mit Sprunghöhe -2V gewählt. Der P- und PD-Regler haben, wie zu erwarten, eine bleibende Regelabweichung. Der PI- und PID- Regler weisen sehr ähnliches Verhalten auf. Das liegt daran, dass k_D des PID-Reglers mit 0,03 sehr gering ist.

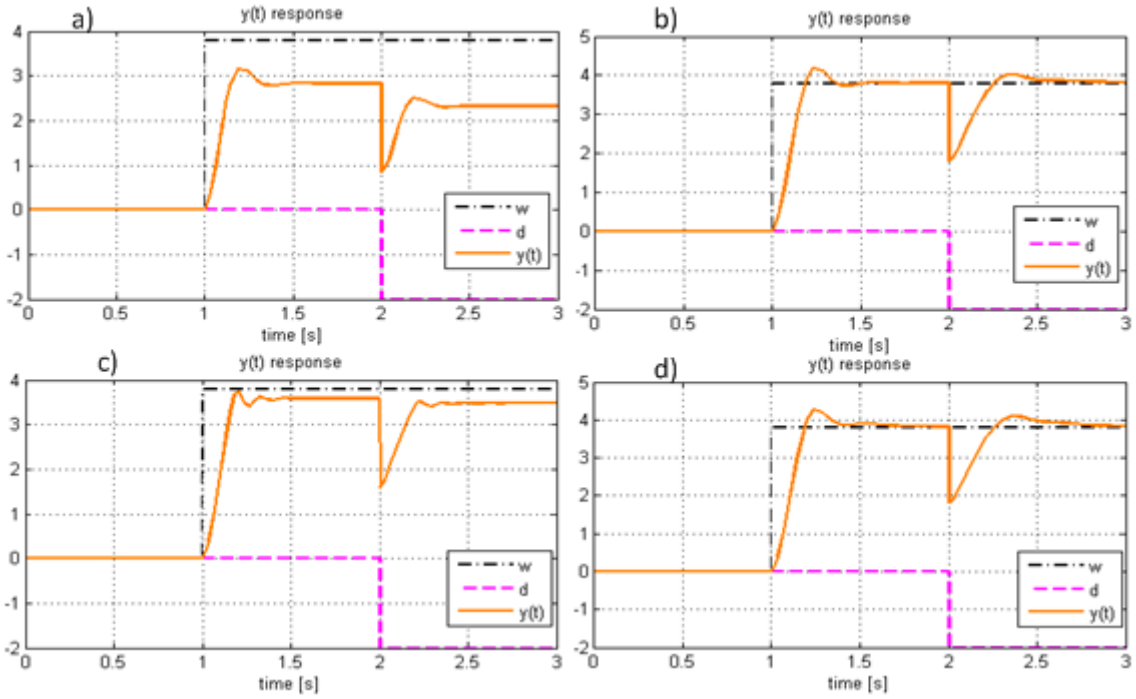

Abb. 5.8: Simulation der Regelgröße mit a)P-Regler, b) PI-Regler, c) PD-Regler, d)PID-Regler

6 Resümee und Ausblick

Im Rahmen dieser Arbeit wurde eine Matlab GUI entwickelt, die dem Regelungstechnik Neuling beim Verständnis der Anaylse eines dynamischen Systems Unterstützung bietet. Des Weiteren bietet das Programm verschiedene Möglichkeiten zum Reglerentwurf an.

Aufbauend auf dem Matlab m-File *pz2curve*, welches lediglich die Verschiebung zweier Pole und die Darstellung des Bode Diagramms, der Ortskurve sowie der Sprungantwort ermöglicht, wurde zunächst eine Bedienoberfläche mittels Matlab GUIDE eingerichtet. Diese wurde gemäß den Anforderungen aus Kapitel 1.2 ständig erweitert. Die zuvor festgelegten Forderungen konnten allesamt umgesetzt werden. Während der Bearbeitung wurden nach regelmäßiger Absprache mit Herrn Prof. Dr. Ing. Peter Zentgraf M.Sc. einige Erweiterungen eingearbeitet, welche nicht explizit Thema dieser Arbeit sind. Beispielsweise konnte eine Reglerabschätzung in PID Form mit Hilfe der Methode der kleinsten Fehlerquadrate umgesetzt werden. Ebenso gibt es die Möglichkeit eine bestimmte Reglerstruktur vorzugeben und daraus die resultierende Führungsübertragungsfunktion $G_W(s)$ berechnen zu lassen. Die Reglerentwicklung nach Ziegler und Nichols sowie nach Chien, Hrones und Reswick schließen dieses Angebot ab. Dazu lieferte Herr Prof. Dr. Ing. Peter Zentgraf M.Sc hilfreiche Beiträge.
Neben der Reglerauslegung besteht zeitgleich die Möglichkeit die Funktionalität des geschlossenen Regelkreises zu simulieren. Hierfür stehen die Signale Sprung, Impuls, eine beliebige zeitabhängige Funktion sowie ein Signal aus dem Workspace für die Führungsgröße zur Verfügung. Die Störgröße kann zusätzlich noch durch ein Rauschen realisiert werden. Aufgrund der Schnittstelle zum Matlab Workspace können Signalverläufe aus der alltäglichen Praxis verarbeitet werden.
In Kapitel 5 wurde das Programm an einem konkreten Beispiel angewandt.

Möglichkeiten der Optimierung des Programms gibt es vor allem hinsichtlich der Laufzeit. Im Bereich des Reglerentwurfs läuft das Verschieben der Pole mit einer geringen Zeitverzögerung ab.

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

A Ein Anhangskapitel

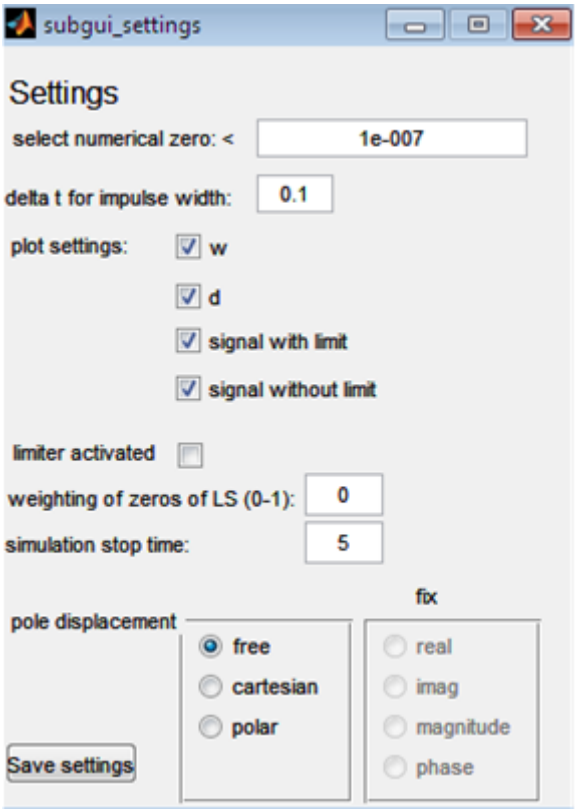

Abb. A.1: Nebenprogramm - Einstellungen

Listing A.1: Callback Funktion eines Push Buttons

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

Listing A.2: Code zum Verschieben eines Graphik Objekts

```
function plotdisplacement

if ~isempty(gcbo)
   while get(gcf,'userdata')
      p=get(0,'PointerLocation');
      pf=get(gcf,'pos');
      p(1:2)=p(1:2)-pf(1:2);
      set(gcf,'CurrentPoint',p(1:2));
      p=get(gca,'CurrentPoint');
      p=sign(p(1,1:2)).*min(abs(p(1,1:2)),[20 20]);
      set(gcbo,'xdata',p(1,1), 'ydata',p(1,2));
      drawnow;
   end
   return
else
   set(gcf,'userdata',0)
   set(gcf,'WindowButtonDownFcn','set(gcf,''userdata'',1)')
   set(gcf,'WindowButtonUpFcn' ,'set(gcf,''userdata'',0)')
   figure(1)
   h=plot(0,0,'rx','markersize',15,'Linewidth',5);
   set(h,'ButtonDownFcn',mfilename);
   axis equal;
   xlim([-20 20]);
   ylim([-20 20]);
   title('plotdisplacement');
end
return
```

Listing A.3: Code function rechtsklick

```
function rechtsklick(~, ~)
mouse=get(gcf,'SelectionType')
switch(mouse)
    case 'alt'
    cp=get(gcf,'CurrentPoint')
    cpa=get(gca,'CurrentPoint')
    setappdata(gcf,'posre',cpa)
    cp2=[cp(1,1) cp(1,2)-50]
    h1=getappdata(gcf,'h1')
    set(h1,'position',[cp2 190 80],'visible','on')
end
```

Listing A.4: Code zum Plotten der Pole

```
h(1)=plot(0,cpa(1,2),'x','color','r',...
        'Linewidth',4,'Markersize',14);
h(2)=plot(0,-cpa(1,2),'x','color','r',...
        'Linewidth',4,'Markersize',14);
set(h(1),'ButtonDownFcn',mfilename);
set(h(2),'ButtonDownFcn',mfilename);
```

Listing A.5: Code zum Löschen der Pole

```
po=findobj(gca,'color','r');
for i=1:length(po)
    xpo=get(po(i),'xdata');
    ypo=get(po(i),'ydata');
    abstand(i)=sqrt((abs(cpa(1,1)-xpo))^2 +...
    (abs(cpa(1,2)-ypo))^2);
end
    [y,i]=min(abstand);
    ypo=get(po(i),'ydata');
    posym=findobj(gca,'ydata',-ypo);
```

Listing A.6: Code für Bode Diagramm

```
function [mag, phase]=phasentest(sys)
w=logspace(-1,3,1000);
[mag01,phase01]=bode(sys,0.1);
fg01=freqresp(sys,0.1);
phideg=atan(imag(fg01)/real(fg01))*360/(2*pi);
[mag, phase]=bode(sys,w);
diffph=phase01-phideg;
phase=phase-diffph;
if nargout==1
   mag=diffph;
end
```

Listing A.7: Code für die Magnetschwebekörper Animation

```
function animationmagnet(hObject, handles);
sys=tf(1,[1 1 1]);
t1=linspace(0,30,500);
[bew, t]=step(sys, t1);
axes(handles.axes17);

%boden plotten
a=min(bew)-0.5;
xboden=[-1 1 1 -1 -1 -1 1; -1 1 1 -1 -1 -1 1]*2;
yboden=[-1 -1 1 1 -1 -1 -1; -1 -1 1 1 -1 1 1]*2;
minb=min(bew)
if minb<0
   if minb<=-1
      zboden=[0 0 0 0 0 0 0; -1 -1 -1 -1 -1 0 0]/3+minb*2;
   else
      zboden=[0 0 0 0 0 0 0; -1 -1 -1 -1 -1 0 0]/3+1/minb;
   end
else
   zboden=[0 0 0 0 0 0 0; -1 -1 -1 -1 -1 0 0]/3-1;
end
boden=surf(xboden, yboden, zboden,'facecolor',[0.6 0.6 0.6]);

%kugel plotten
[x,y,z]=sphere(30);
hold on
sp=surf(x,y,z);

%deckel plotten
xdeckel=[-1 1 1 -1 -1 -1 1; -1 1 1 -1 -1 -1 1]*2;
ydeckel=[-1 -1 1 1 -1 -1 -1; -1 -1 1 1 -1 1 1]*2;
```

```
zdeckel=[0 0 0 0 0 0 0; -1 -1 -1 -1 -1 0 0]/3+max(bew)+2;
deckel=surf(xdeckel, ydeckel, zdeckel,'facecolor',[0.6 0.6 0.6]);

axis equal;
axis off;
axis([-2 2 -2 2 min(bew)-1.5 2.5+max(bew)]);
view(20, 12);
lightDir=10*[1 0 1.2];
light('Position',lightDir,'Style','local');
pause(0.5);
for i=1:length(bew)
   stopanim=getappdata(handles.figure1,'stopanim');
  if stopanim==1
     break
  else
   zneu=z+bew(i);
   set(sp,'zdata',zneu);
   drawnow
   pause(.01);
  end
end
```

Listing A.8: Erläuterung von *transferdisplay*

```
%#####################################################################
%
% function [zaehler, nenner]= transferdisplay(G, display)
%
% PURPOSE:
% This function computes strings of the numerator and the
% denominator when the transfer function G is given.
%
% IN:
%     G         transfer function, either tf or zpk.
%     display   string, which decides whether the transfer function
%               should be displayed in Pole Zero, Polynom, or Time
%               Constant form.
%               possibilities:
%               - 'polezero'
%               - 'polynom'
%               - 'timeconstant'
%
% OUT: zaehler contains the string of the numerator
%      nenner   contains the string of the denominator
%
% Fabian Tutschka, 20.12.2011
%
%#####################################################################

%--------------------------------------------------------------------
% Additional functions needed:
%  [zn]=polynomdarstellung(koeff)
%  [ausgabestr]= pzdarstellung(porz,k)
%  [ausgabestr]= timeconstantdars(porz,ks)
%  [ks]=verstaerkungks(p1,z1,k)
%--------------------------------------------------------------------
```

Listing A.9: *run_transferdisplay*

```
%run transferdisplay
%
% test #1
% allpass 2nd order:
%     poles: -10+j*10
%            -10-j*10
%     zeros: 10+j*10
%             10-j*10
%
% correct results:
%
%        'polezero':
%     zaehler: (s-10+j*10)*(s-10-j*10)
%     nenner:  (s+10+j*10)*(s+10-j*10)
%
%        'polynom':
%     zaehler: s^2-20*s+200
%     nenner:  s^2+20*s+200
%
%        'timeconstant'
%     zaehler: 0.071^2*s^2-2*0.707*0.071*s+1
%     nenner:  0.071^2*s^2+2*0.707*0.71*s+1


G=tf([1 -20 200],[1 20 200]) ;
display={'polezero' 'polynom' 'timeconstant'};
%polezero
[zpz, npz]=transferdisplay(G, display(1));

%polynom
[zpoly, npoly]=transferdisplay(G, display(2));

%timeconstant
[ztime, ntime]=transferdisplay(G, display(3));

if strcmp(zpz,'1.00*(s-10.00-j*10.00)*(s-10.00+j*10.00)')==1...
&& strcmp(npz,'(s+10.00-j*10.00)*(s+10.00+j*10.00)')==1...
&& strcmp(zpoly,'1.00*s^2-20.00*s+200.00')==1...
&& strcmp(npoly,'1.00*s^2+20.00*s+200.00')==1...
&& strcmp(ztime,'0.071^2*s^2-2*0.707*0.071*s+1')==1...
&& strcmp(ntime,'0.071^2*s^2+2*0.707*0.071*s+1')==1
disp('test #1 successful');
end
```

Literaturverzeichnis

[1] Lunze J.: *Regelungstechnik 1-Systemtheoretische Grundlagen Analyse und Entwurf einschleifiger Regelungen*, Springer, 2001.

[2] Föllinger O.: *Regelungstechnik-Einführung in die Methoden und ihre Anwendung*, Hüthig, 2008.

[3] Orlowski P.F.: *Praktische Regeltechnik-Anwendungsorientierte Einführung für Maschinenbauer und Elektrotechniker*, Springer, 1999.

[4] Bode H.: *Matlab in der Regelungstechnik-Analyse linearer System*, Teubner, 1998.

[5] Zentgraf P.: *Vorlesungsmanuskript Regelungstechnik I*, 2011.

[6] Hering-Martin-Stohrer: *Physik für Ingenieure*, VDI Verlag, 1989.

[7] Meyberg-Vachenauer: *Höhere Mathematik 2- Differentialgleichungen, Funktionentheorie, Fourier-Analysis, Variationsrechnung*, Springer, 1991.

[8] Ludyk G.: *Theoretische Regelungstechnik 1-Grundlagen, Synthese linearer Regelungssystem*, Springer, 1995.

[9] Quaschning V.: *Regenerative Energiesysteme-Technologie, Berechnung, Simulation*, Hanser, 2009.

[10] Angermann A., Beuschel M., Rau M., Wohlfarth U.: *Matlab-Simulink-Stateflow*, Oldenbourg Verlag München, 2008.

[11] Marchand P.: *Graphics and GUIs with MATLAB-second edition*, Chapman/Hall/-CRC, 2000.

[12] *Matlab Product Documentation*

[13] Zentgraf P.: *Praktikumsmanuskript Drehzahlregelung*, 2011.