

Fachbericht Easy-PID

Windisch, 18. November 2015



| | |
|-----------------|--|
| AUFTAGGEBER: | PETER NIKLAUS |
| BETREUER: | PETER NIKLAUS (ELEKTROTECHNIK) RICHARD GUT (OBJEKTOIENTIERTE PROGRAMMIERUNG) PASCAL BUCHSCHACHER (PROJEKTMANAGEMENT) ANITA GERTISER (KOMMUNIKATION) |
| GRUPPE: | FS15 PRO2E TEAM 3 |
| TEAMMITGLIEDER: | YANICK FREI (PROJEKtleiter) SIMON WYSS (EXPERTE ELEKTROTECHNIK) JOSUA STIERLI (EXPERTE PROGRAMMIERUNG) ALEXANDER MURRAY (Stv. PROJEKtleiter) SIMON STURM |
| STUDIENGANG: | ELEKTRO- UND INFORMATIONSTECHNIK |

Abstract

Um in der Praxis einen geeigneten Regler zu dimensionieren, ist oftmals viel Zeit und Erfahrung notwendig. Zwar gibt es zur Hilfe Faustformeln, diese liefern jedoch oftmals nur ungenügende Ergebnisse, sodass man schliesslich auf grafische Methoden mit Papier und Bleistift zurückgreifen muss. Dies kann jedoch im 21. Jahrhundert nicht mehr als zeitgemäss betrachtet werden. „Easy-PID“ ist ein Softwaretool, welches im Rahmen des Projekt 2 der Fachhochschule Nordwestschweiz entwickelt wurde. Es ermöglicht die Reglerdimensionierung mit der grafischen Phasengangmethode von Zellweger sowie mehreren Faustformeln. „Easy-PID“ ermöglicht es, einen optimalen Regler zu dimensionieren. Mit Hilfe der Kenngrössen der Regelstrecke sowie weiteren Einstellungen wie beispielsweise dem maximalen Überschwingen und der Wahl des Reglertyps kann „Easy-PID“ alle benötigten Reglerparameter bestimmen.

Zur richtigen Reglerdimensionierung mussten zuerst die entsprechenden Formeln der Dimensionierung und der Sprungantwort hergeleitet werden. Die damit erstellten Berechnungsalgorithmen wurden in Matlab implementiert und die Ergebnisse mit gegebenen Werten verglichen, um die Korrektheit zu überprüfen. Um „Easy-PID“ flexibel in Java programmieren zu können, wurde auf das Model-View-Controller Entwurfsmuster zurückgegriffen. Die von „Easy-PID“ gelieferten Ergebnisse wurden ausserdem mit Matlab verifiziert.

Das Java-Programm wurde anschliessend weiter optimiert, um ein ideales Verhältnis zwischen Geschwindigkeit und Genauigkeit zu finden. So wurden beispielsweise zur Berechnung der Schrittantworten zwei Vorgehensweisen implementiert: Residuen und IFFT. Dabei hat sich gezeigt, dass mit Residuen sowohl genauere als auch schnellere Resultate erzielt werden können.

„Easy-PID“ kann durchaus als erfolgreiches Projekt bezeichnet werden. Das Programm bietet nebst dem Berechnen der Reglerparameter aufgrund der Eingabeparameter und der Darstellung der Sprungantwort diverse Zusatzfunktionen wie eine Mini-Version, den Export als PDF oder die nachträgliche Anpassung der Phasengang-Methode. Der User kann zusätzlich mit einem Hilfe-Menü hilfreiche Links aufrufen, die auftretende Fragen zum Thema Regelungstechnik beantworten.

Sämtliche Daten werden dabei in einem einfach zu bedienenden User-Interface dargestellt. So kann der Benutzer die Eingabeparameter in Textfeldern eingeben und mittels einem Dropdown-Menü den Reglertyp wählen. Die mit diesen Daten berechneten Regler werden tabellarisch aufgelistet. Die Sprungantwort der geschlossenen Regelstrecke wird ausserdem grafisch dargestellt, wobei auch einzelne Graphen ausgewählt werden können.

Projekt P2 - Aufgabenstellung vom Auftraggeber (FS_2015)

Reglerdimensionierung mit Hilfe der Schrittantwort

1. Einleitung

In der Praxis werden die klassischen Regler (PI, PID, PD, ...) oft mit sog. Faustformeln dimensioniert. Dazu benötigt man bestimmte Informationen der zu regelnden Strecke. Handelt es sich dabei um „langsame Strecken“ mit Zeitkonstanten im Bereich von Sekunden bis Minuten, so ist das Bestimmen und Ausmessen der Schrittantwort oft die einzige Möglichkeit zur Identifikation der Strecke. Typische Beispiele dafür sind Temperaturheizstrecken, welche meistens mit einem PTn-Verhalten modelliert werden können (Kaffeemaschine, Boiler, Raumheizungen, Lötkolben, Warmluftfön, usw.).

Die Schrittantwort wird mit Hilfe einer Wendetangente vermessen und die Kenngrößen Streckenbeiwert (K_s), Verzugszeit (T_u) und Anstiegszeit (T_g) werden bestimmt. Dies kann sowohl von Hand (grafisch) oder auch automatisiert durchgeführt werden, falls die Messdaten elektronisch vorliegen. Mit diesen drei Kenngrößen können mit Hilfe sog. Faustformeln PI- und PID-Regler dimensioniert werden (Ziegler/Nichols, Chien/Hrones/Reswick, Oppelt, Rosenberg). Die Faustformeln liefern zwar sehr schnell die Reglerdaten, aber die Schrittantworten der entspr. Regelungen sind teilweise weit vom "Optimum" entfernt und der Regelkreis kann sogar instabil werden. In der Praxis muss man diese "Startwerte" häufig nachoptimieren, damit die Schrittantwort der Regelung die Anforderungen erfüllt.

Die sog. "Phasengangmethode zur Reglerdimensionierung" wurde von Jakob Zellweger (FHNW) entwickelt und liefert Reglerdaten, welche näher am "Optimum" sind und für die Praxis direkt verwendet werden können. Dabei kann das Überschwingen der Schrittantwort vorgegeben werden (z.B. 20%, 10%, 2%, oder aperiodisch). Bei dieser Methode kann also das für viele Anwendungen wichtige Verhalten der Schrittantwort beeinflusst werden. Um die Phasengangmethode anwenden zu können, muss der Frequenzgang der Strecke bekannt sein (analytisch oder numerisch/gemessen). Mit Hilfe der Hudzovik-Approximation (oder anderer ähnlicher Verfahren) wird dieses Problem gelöst, in dem vorgängig aus den Kenngrößen der Schrittantwort (K_s , T_u, T_g) eine PTn-Approximation der Strecke erzeugt wird. Mit dem Frequenzgang der PTn-Approximation können dann die Regler dimensioniert werden (I, PI, PID). Die Phasengangmethode war ursprünglich eine grafische Methode, basierend auf dem Bodediagramm der Strecke. Aktuell soll die Methode direkt numerisch im Rechner durchgeführt werden.

In dieser Arbeit geht es um die Entwicklung und Realisierung eines Tools zur **Reglerdimensionierung mit der Phasengangmethode**. Ausgehend von der PTn-Schrittantwort der Strecke sollen "optimale Regler" (PI, PID-T1) dimensioniert werden, wobei das Überschwingen der Regelgröße vorgegeben werden kann. Zum Vergleich sollen die Regler auch mit den üblichen Faustformeln dimensioniert werden. Wünschenswert wäre auch eine Simulation der Schrittantwort des geschlossenen Regelkreises, so dass die Dimensionierung kontrolliert und evtl. noch "verbessert" werden könnte.

2. Aufgaben/Anforderungen an Tool

Entwerfen und realisieren Sie ein benutzerfreundliches Tool/Programm/GUI/usw. mit welchem PI- und PID-Regler mit der Phasengangmethode dimensioniert werden können. Dabei sind folgende Anforderungen und Randbedingungen vorgegeben:

- Die zu regelnden Strecken sind PTn-Strecken, wobei entweder die Schrittantwort grafisch vorliegt oder die Kenngrößen K_s , T_u und T_g schon bekannt sind
- Die Bestimmung einer PTn-Approximation wird vom Auftraggeber zur Verfügung gestellt und muss entsprechend angepasst und eingebunden werden (Matlab zu Java)
- Das Überschwingen der Regelgröße (Schrittantwort) soll gewählt werden können
- Zum Vergleich sind die Regler auch mit den üblichen Faustformeln zu dimensionieren.
- Das dynamische Verhalten des geschlossenen Regelkreises soll auch berechnet und visualisiert werden (Schrittantwort)

3. Bemerkungen

Die Software und das GUI sind in enger Absprache mit dem Auftraggeber zu entwickeln. Der Auftraggeber steht als Testbenutzer zu Verfügung und soll bei der Evaluation des GUI eingebunden werden. Alle verwendeten Formeln, Algorithmen und Berechnungen sind zu verifizieren, eine vorgängige oder parallele Programmierung in Matlab ist zu empfehlen. Zum Thema der Regelungstechnik und speziell zur Reglerdimensionierung mit der Phasengangmethode werden Fachinputs durchgeführt (Fachcoach).

Literatur

- [1] J. Zellweger, *Regelkreise und Regelungen*, Vorlesungsskript.
- [2] J. Zellweger, *Phasengang-Methode*, Kapitel aus Vorlesungsskript.
- [3] H. Unbehauen, *Regelungstechnik I*, Vieweg Teubner, 2008.
- [4] W. Schumacher, W. Leonhard, *Grundlagen der Regelungstechnik*, Vorlesungsskript, TU Braunschweig, 2003.
- [5] B. Bate, *PID-Einstellregeln*, Projektbericht, FH Dortmund, 2009.

Inhaltsverzeichnis

| | |
|---|-----------|
| 1 Einleitung | 3 |
| 2 Elektrotechnische Problemlösung | 4 |
| 2.1 Regelungstechnische Grundlagen | 4 |
| 2.2 Faustformeln | 6 |
| 2.2.1 Chien/Hrones und Reswick | 6 |
| 2.2.2 Oppelt | 7 |
| 2.2.3 Rosenberg | 8 |
| 2.3 Zellweger Methode | 9 |
| 2.3.1 Numerisches Beispiel PI-Regler | 12 |
| 2.3.2 Numerisches Beispiel PID-Regler | 15 |
| 2.4 Übertragungsfunktion des geschlossenen Regelkreises | 19 |
| 2.4.1 Berechnungsarten | 20 |
| 3 Software | 25 |
| 3.1 Model | 25 |
| 3.1.1 Regelstrecke | 27 |
| 3.1.2 Regler | 29 |
| 3.1.3 Reglerberechnung | 30 |
| 3.1.4 Regelkreis | 31 |
| 3.1.5 Resultate | 32 |
| 3.2 View | 33 |
| 3.2.1 Anforderungen an die Benutzeroberfläche | 33 |
| 3.2.2 Erstellung der Benutzeroberfläche | 33 |
| 3.2.3 Aufgabe der einzelnen Panels | 35 |
| 3.2.4 Die Klasse View im Klassendiagramm | 40 |
| 3.2.5 Funktionalität der Benutzeroberfläche | 41 |
| 3.3 Controller | 43 |
| 3.4 Ablauf einer Reglerberechnung (Use-Case) | 44 |
| 4 Validierung | 46 |
| 4.1 Matlab | 46 |
| 4.1.1 Programmierung | 46 |
| 4.1.2 Simulation | 46 |

| | |
|---|-----------|
| 4.2 Java Software | 46 |
| 4.2.1 Benutzeroberfläche | 46 |
| 4.2.2 Model | 46 |
| 4.2.3 Performancevergleich von Partialbruchzerlegung und IFFT | 46 |
| 4.2.4 JUnit-Tests | 50 |
| 5 Schlussfolgerung | 51 |
| Literatur | 52 |
| A Anhang Elektrotechnik | 53 |
| A.1 Herleitung der Berechnung von β für die Zellweger Methode | 53 |
| A.2 Herleitung Bodekonforme Darstellung | 55 |
| A.2.1 PI-Regler | 55 |
| A.2.2 PID-Regler | 55 |
| B Klassendiagramm | 58 |
| C CD-Rom | 60 |

1 Einleitung

Das Berechnen von Reglern ist oftmals eine schwierige Angelegenheit, die sowohl Erfahrung als auch Zeit benötigt. Zur richtigen Dimensionierung helfen entweder Faustformeln oder auch grafische Methoden, die jedoch von Hand ausgeführt werden müssen. Folglich ist die Berechnung entweder ungenau oder sehr aufwändig. Da es fragwürdig ist, ob solche manuelle Methoden in einer Zeit von Matlab und weiteren Berechnungstools noch zeitgemäß sind, wurde im Projekt 2 eine Software erstellt, welche Regler automatisch anhand der „Phasengangmethode zur Reglerdimensionierung“ von Jakob Zellweger dimensionieren kann.

Das Hauptziel des Projektes ist, ein funktionierendes Programm zu erstellen, welches die Reglerberechnung automatisch nach der Phasengangmethode von Zellweger sowie weiteren Faustformeln dimensioniert. Zur korrekten Funktion werden sowohl richtig implementierte Berechnungen benötigt als auch die richtige Darstellung der Ergebnisse, beispielsweise mittels sinnvoll skalierten Graphen. Zusätzlich wurden einige optionale Ziele definiert. Die wichtigsten davon sind die Simulation der geschlossenen Regelstrecke, eine Miniversion des Programms sowie eine Möglichkeit zum Nachjustieren der Eingabeparameter.

Easy-PID ist ein Programm, mit welchem Regler anhand der Parameter der Schrittantwort automatisch dimensioniert werden können. Es besitzt ein intuitiv zu bedienendes User-Interface, das neben den Ein- und Ausgabeparametern auch das dynamische Verhalten des Regelkreises grafisch darstellt. Die Regler werden durch verschiedene Methoden berechnet. Durch diese verschiedenen Dimensionierungen und der Möglichkeit, Parameter in Echtzeit zu manipulieren, kann ein optimaler Regler gefunden werden.

Dieser Bericht beschreibt die Problemlösung aufgeteilt in die drei Teilbereiche Elektrotechnik, Programmierung und Validierung. Der elektrotechnische Teil behandelt die hergeleiteten Formeln für die Reglerdimensionierung und die Programmierung dieser Formeln in Matlab, während der zweite Teil den Aufbau der Benutzeroberfläche und die Umsetzung der Matlab-Programmierung in Java beinhaltet. Die Validierung beschreibt den Aufbau und den Vorgang der Tests sowie deren Ergebnisse.

2 Elektrotechnische Problemlösung

In diesem Kapitel wird das Vorgehen zur elektrotechnischen Problemlösung genauer erläutert. Dabei wird näher auf die Grundlagen der Regelungstechnik, die Zellweger-Methode, einige Faustformeln und das Berechnen des geschlossenen Regelkreises eingegangen.

2.1 Regelungstechnische Grundlagen

Ein Regelkreis besteht aus einer Strecke und einem Regler.

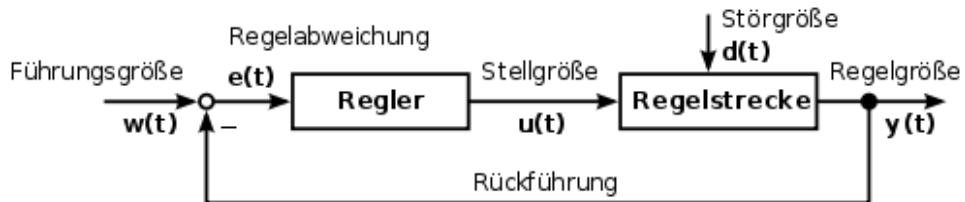


Abbildung 1: Regler und Regelstrecke [1]

Die Aufgabe des Regelkreises ist, die Regelgrösse konstant auf dem Wert der Führungsgrösse zu halten. Bei einer Heizung ist dies beispielsweise die momentane Zimmer-Temperatur, die auf die gewünschte Soll-Temperatur zu regeln ist. Die Strecke ist dabei der zu heizende Raum.

Um den Regelkreis zu beschreiben, braucht man zunächst die Schrittantwort der Strecke. Aus dieser kann man mithilfe der Wendetangentenmethode die Parameter K_s (Verstärkungsfaktor), T_u (Verzugszeit) und T_g (Ausgleichszeit) auslesen.

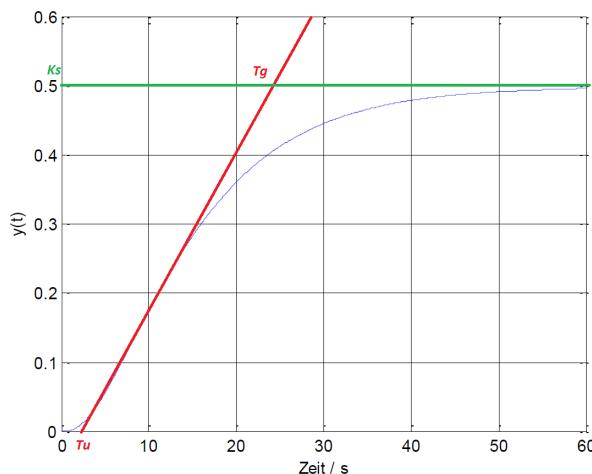


Abbildung 2: Regler und Regelstrecke [2]

Für das weitere Vorgehen wird die Sani Methode verwendet, die vom Auftraggeber in Form eines Matlab-Files zur Verfügung gestellt wurde. Die Sani Methode funktioniert ähnlich wie die früher verwendete Hudzovik Methode. Mit dieser können die Ordnung der Strecke sowie die Zeitkonstanten T_1 bis T_m berechnet werden, wobei N die Ordnung der Strecke ist und die Anzahl der Zeitkonstanten N entspricht. Die berechneten Zeitkonstanten werden dazu verwendet, die Übertragungsfunktion der Strecke $H(s)$ zu berechnen:

$$s = j \cdot w$$

$$H(s) = \prod_{m=1}^N \frac{1}{(1 + s \cdot T_m)} \quad (1)$$

Für die Übertragungsfunktion eines PI/PID Reglers gilt es zu beachten, dass es zwei Darstellungsarten gibt, die reglerkonforme sowie die bodekonforme.

Übertragungsfunktionen von I/PI/PID Reglern:

I (Reglerkonform/Bodekonform beide gleich):

$$G_R(s) = \frac{1}{s \cdot Ti} \quad (2)$$

PI Reglerkonform:

$$G_R(s) = K_R \left(1 + \frac{1}{s \cdot Tn} \right) \quad (3)$$

PI Bodekonform:

$$G_R(s) = K_R \left(\frac{1 + s \cdot Tn}{s \cdot Tn} \right) \quad (4)$$

PID Reglerkonform:

$$G_R(s) = K_R \left(1 + \frac{1}{s \cdot Tn} + \frac{s \cdot Tv}{1 + s \cdot Tp} \right) \quad (5)$$

PID Bodekonform:

$$G_R(s) = K_{RK} \left(\frac{(1 + s \cdot Tnk)(1 + s \cdot Tvk)}{s \cdot Tnk(1 + s \cdot Tp)} \right) \quad (6)$$

Auf die Verwendung dieser Funktionen und deren Herleitung wird im Kapitel 2.3, Zellweger Methode wie auch im Anhang A.2 genauer eingegangen.

2.2 Faustformeln

Zum Dimensionieren von verschiedenen Reglertypen gibt es eine Vielzahl von festen Einstellregeln. Diese sind einfach anzuwenden, da sie keine besonderen Vorkenntnisse benötigen, jedoch ist deren Genauigkeit oftmals ungenügend. Die für dieses Projekt relevanten Faustformeln sind:

- Chien/Hrones und Reswick
- Oppelt
- Rosenberg

Diese Faustformeln wurden gewählt, weil sie ebenfalls die Werte K_s , T_u und T_g verwenden, welche mittels Wendetangentialen aus der Schrittantwort der Regelstrecke herausgelesen werden können.

Zur Verifizierung der Faustformeln wurden verschiedene Quellen verglichen und die in der Mehrzahl vorkommenden Werte wurden als richtig angesehen.

2.2.1 Chien/Hrones und Reswick

PI

- Aperiodischer Verlauf
 - Gutes Störverhalten

$$Kr = 0.6 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = 4 \cdot Tu$$

- Gutes Führungsverhalten

$$Kr = 0.45 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = 1.2 \cdot Tg$$

- 20% Überschwingen
 - Gutes Störverhalten

$$Kr = 0.7 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = 2.3 \cdot Tu$$

- Gutes Führungsverhalten

$$Kr = 0.6 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = Tg$$

PID

- Aperiodischer Verlauf

– Gutes Störverhalten

$$Kr = 0.95 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = 2.4 \cdot Tu$$

$$Tv = 0.42 \cdot Tu$$

– Gutes Führungsverhalten

$$Kr = 0.6 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = Tg$$

$$Tv = 0.5 \cdot Tu$$

- 20% Überschwingen

– Gutes Störverhalten

$$Kr = 1.2 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = 2 \cdot Tu$$

$$Tv = 0.42 \cdot Tu$$

– Gutes Führungsverhalten

$$Kr = 0.95 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = 1.35 \cdot Tg$$

$$Tv = 0.47 \cdot Tu$$

2.2.2 Oppelt

PI

$$Kr = 0.8 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = 3 \cdot Tu$$

PID

$$Kr = 1.2 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = 2 \cdot Tu$$

$$Tv = 0.42 \cdot Tu$$

2.2.3 Rosenberg**PI**

$$Kr = 0.91 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = 3.3 \cdot Tu$$

PID

$$Kr = 1.2 \cdot \frac{Tg}{Ks \cdot Tu}$$

$$Tn = 2 \cdot Tu$$

$$Tv = 0.44 \cdot Tu$$

2.3 Zellweger Methode

Der PI- oder PID-Regler wird aus dem Amplitudengang- und Phasengang mit der Phasengang-Methode nach Zellweger berechnet. Der Amplitudengang- und Phasengang kann aus der Übertragungsfunktion $H(s)$ (Formel (1)) berechnet werden. Folgend werden die dazu verwendeten Formeln aufgeführt:

$$\text{Amplitudengang Strecke } A_S(\omega) = Ks \cdot \prod_{m=1}^N \frac{1}{\sqrt{1 + (\omega \cdot T_m)^2}}$$

$$\text{Phasengang Strecke } \phi_S(\omega) = -1 \cdot \sum_{m=1}^N \arctan(\omega \cdot T_m)$$

Gemäss Zellweger wird nun nach folgendem Rezept vorgegangen:

1. Aus dem Phasengang ϕ_S wird die Frequenz bei einem bestimmten Winkel ϕ_x herausgesucht. Der Winkel ϕ_x ist abhängig davon, welcher Regler-Typ dimensioniert werden soll. Beim I-Regler ist ϕ_I -45° , beim PI-Regler ist ϕ_{PI} -90° und beim PID-Regler ist ϕ_{PID} -135° .

Gemäss dem Skript „Regelkreise und Regelung“ [3] von Zellweger gibt es für die Regler zwei Darstellungsvarianten: Die reglerkonforme und die bodekonforme Darstellung.

Bei den weiteren Berechnungen des PID-Reglers muss für die Phasengang-Methode nach Zellweger in die bodekonforme Darstellung gewechselt werden (siehe Anhang Seite 55 Kapitel A.2).

Für PI- und PID-Regler unterscheidet sich das weitere Vorgehen. Es ist in die folgenden zwei Kapitel aufgeteilt.

Weiteres Vorgehen PI Regler:

Die gefundene Frequenz ω_{PI} bei ϕ_{PI} entspricht $1/Tn$.

$$Tn = \frac{1}{\omega_{PI}}$$

2. Mit dem gefundenen Tn und $Kr = 1$ wird der Amplitudengang- und Frequenzgang des offenen Regelkreises (G_O) berechnet. Das heisst Regler $G_R(s)$ und Strecke $G_S(s)$ in Serie, aber ohne Rückkoppelung.

Übertragungsfunktion PI-Regler (Reglerkonform) $G_R(s)$ siehe Formel (3)

Übertragungsfunktion PI-Regler (Bodekonform) $G_R(s)$ siehe Formel (4)

$$\text{Phasengang PI-Regler } \phi_R = \arctan(\omega \cdot Tn) - \frac{\pi}{2}$$

$$\text{Amplitudengang PI-Strecke } A_R = Kr \frac{\sqrt{1 + (\omega \cdot Tn)^2}}{\omega \cdot Tn}$$

$$\text{Phasengang Regelkreis offen } \phi_O = \phi_S + \phi_R$$

Amplitudengang Regelkreis offen $A_O = A_S \cdot A_R$

3. Jetzt wird abhängig vom gewünschten Überschwingen des Reglers ein Winkel auf dem Phasengang des offenen Regelkreises gesucht. Folgende Tabelle zeigt den Zusammenhang des für die Dimensionierung gewählten Überschwingens und des dazu gehörigen Winkels.

| Dämpfmaß d | ϕ_{Rand} | $\phi_{Strecke}$ | Überschwingen |
|------------|---------------|------------------|---------------|
| 0.42 | 45° | -135° | 23.3% |
| 0.5 | 51.5° | -128.5° | 16.3% |
| 0.7 | 65.5° | -114.6° | 4.6% |
| 1 | 76.3° | -103.7° | 0% |

Tabelle 1: Phasenwinkel abhängig vom gewünschten Überschwingen

Bei 0% Überschwingen muss beispielsweise der Winkel $\phi_{Strecke}$ gleich -103.7° auf dem Phasengang gesucht werden. Bei der gefundenen Frequenz wird der dazu gehörende Wert aus dem Amplitudengang des offenen Regelkreises herausgelesen. Um diesen Faktor muss die Verstärkung des Reglers vermindert werden. Der aus dem Amplitudengang herausgelesene Wert mit (-1) multipliziert ergibt also den Regler-Parameter Kr .

Variante I-Regler: Beim I-Regler entspricht ω_I bei ϕ_I direkt dem Kehrwert des Parameters des I-Reglers:

$$Ti = \frac{1}{\omega_I}$$

Damit ist die Berechnung des I-Reglers abgeschlossen. Die weiteren Schritte entfallen für den I-Regler.

Weiteres Vorgehen PID-Regler:

Beim PID-Regler wird analog zum beim PI-Regler beschriebenen Vorgehen gerechnet. Die gefundene Frequenz ω_{PID} für $\phi_{PID} = -135^\circ$ entspricht aber nicht direkt $1/Tn$ wie beim PI-Regler.

Die Zellweger'sche Phasengang-Methode berechnet beim PID-Regler Tnk , Tvk und Krk . Dies sind Parameter in der bodekonformen Darstellung. Diese Parameter können schliesslich wieder in die Parameter Tn , Tv und Kr der reglerkonformen Darstellung umgerechnet werden.

2. Für die Berechnung von Tnk und Tvk wird ein Parameter β benötigt. Folgend wird die Berechnung davon aufgeführt:

$$\frac{2 \cdot \beta}{1 + \beta^2} - w_{PID} \cdot \sum_{m=1}^N \frac{T_m}{1 + (w_{PID} \cdot T_m)^2} = -0.5$$

Folgendes wird mit Z substituiert:

$$Z = \frac{2 \cdot \beta}{1 + \beta^2}$$

$$Z = w_{PID} \cdot \sum_{m=1}^N \frac{T_m}{1 + (w_{PID} \cdot T_m)^2} - 0.5$$

β kann nun durch folgende Gleichung beschrieben werden:

$$\beta = \frac{1}{Z} - \sqrt{\frac{1}{Z^2} - 1}$$

β muss zwischen 0 und 1 liegen. Damit es keine komplexen Lösungen gibt, wird β für $Z > 1$ gleich 1. Für eine detaillierte Herleitung von β siehe Anhang A.1.

3. T_{nk} und T_{vk} können nun wie folgt mit β berechnet werden:

$$\frac{1}{T_{nk}} = \omega_{PID} \cdot \beta$$

$$\frac{1}{T_{vk}} = \frac{\omega_{PID}}{\beta}$$

4. Für die Berechnung des Dämpfungsfaktors K_{rk} muss beim PID-Regler folglich die Formel der Übertragungsfunktion (1) des PID-Reglers verwendet werden. Der Regler-Parameter T_p wird benötigt, weil der Regler nicht über eine unendlich kurze Zeit differenzieren kann. $1/T_p$ muss ca. 1 Dekade höher als $1/T_{vk}$ gewählt werden. Beim PID-Regler wird die bodekonforme Darstellung verwendet, weil T_{nk} und T_{vk} vorhanden sind, T_n und T_k jedoch noch nicht. Eine genauere Erläuterung dieser ist im Anhang Seite 55 Kapitel A.2 zu finden. Für die Berechnung des Amplitudenganges wird $K_{rk} = 1$ verwendet:

$$\frac{1}{T_p} = 10 \cdot \frac{1}{T_{nk}}$$

Übertragungsfunktion PID-Regler (Reglerkonform) $G_R(s)$ siehe Formel (5)

Übertragungsfunktion PID-Regler (Bodekonform) $G_R(s)$ siehe Formel (6)

Phasengang PID-Regler $\phi_R = \arctan(\omega \cdot T_{nk}) + \arctan(\omega \cdot T_{vk}) - \arctan(\omega \cdot T_p) - \frac{\pi}{2}$

Amplitudengang PID-Regler $A_R = K_{rk} \frac{\sqrt{1 + (\omega \cdot T_{nk})^2} \cdot \sqrt{1 + (\omega \cdot T_{vk})^2}}{\omega \cdot T_{nk}}$

Phasengang Regelkreis offen $\phi_O = \phi_S + \phi_R$

Amplitudengang Regelkreis offen $A_o = A_S \cdot A_R$

5. Abhängig vom gewählten Überschwingen wird wiederum ein Winkel auf dem Phasengang ϕ_o gesucht (Siehe Tabelle 1 Seite 10). Die Amplitude bei diesem Winkel mit (-1) multipliziert, ergibt K_{rk} .

2.3.1 Numerisches Beispiel PI-Regler

Im folgenden Abschnitt wird ein Berechnungsbeispiel zum PI-Regler durchgeführt. Folgende Werte werden verwendet:

$$K_s = 0.5$$

$$T_u = 2.5$$

$$T_g = 18.3$$

$$\phi_{Strecke} = -114.6^\circ \text{ (entspricht 4,6\% Überschwingen)}$$

Zunächst werden die Zeitkonstanten sowie die Ordnung der Strecke mithilfe der Sani Methode bestimmt. Bei unserer Strecke ergibt dies eine Strecke 3.Ordnung und die folgenden Zeitkonstanten:

$$T_1 = 1.0688s$$

$$T_2 = 3.3484s$$

$$T_3 = 10.4901s$$

Nun wird der Frequenzgang aus der Übertragungsfunktion (1) berechnet und in einem Plot aufgezeichnet (siehe Abbildung 3). Danach wird der -90° Punkt auf dem Phasengang gesucht (mit blau gestrichelter Linie in Abbildung 3 dargestellt). Dies ergibt eine Frequenz ω_{PI} von $0.1415s^{-1}$. Daraus berechnet sich die Zeitkonstante $T_n = 7.0647s$.

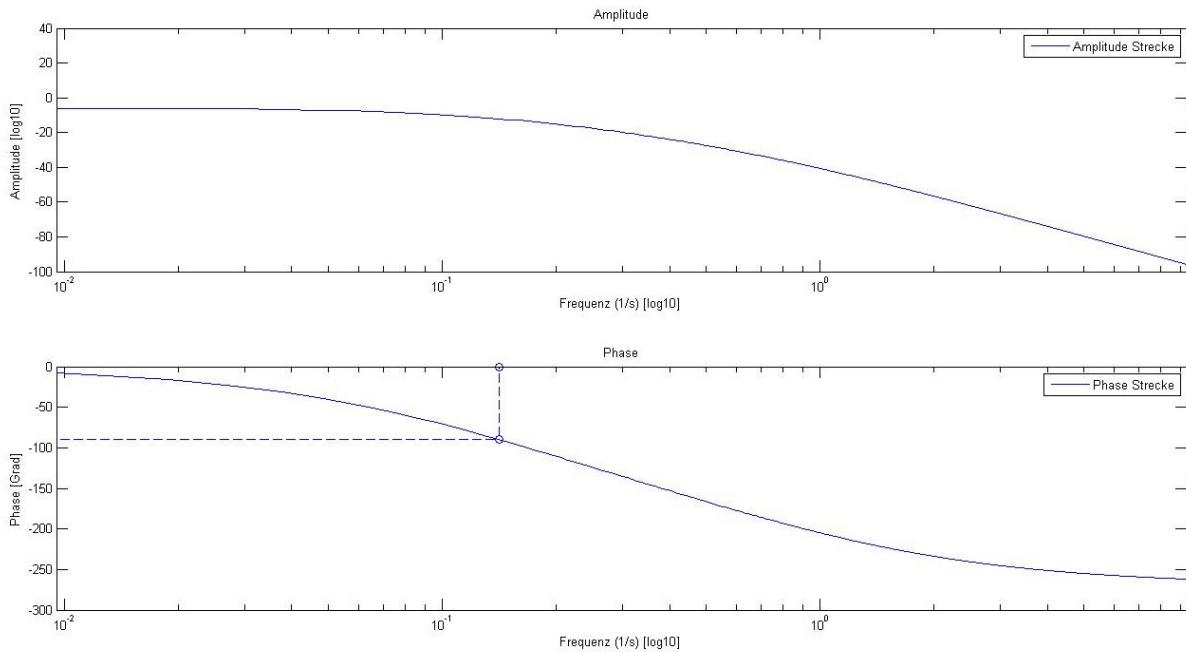


Abbildung 3: Amplituden- und Phasengang der Regelstrecke

Im nächsten Schritt berechnet man mithilfe von Tn und mit $Kr = 1$ den offenen Regelkreis (rot dargestellt in Abbildung 4).

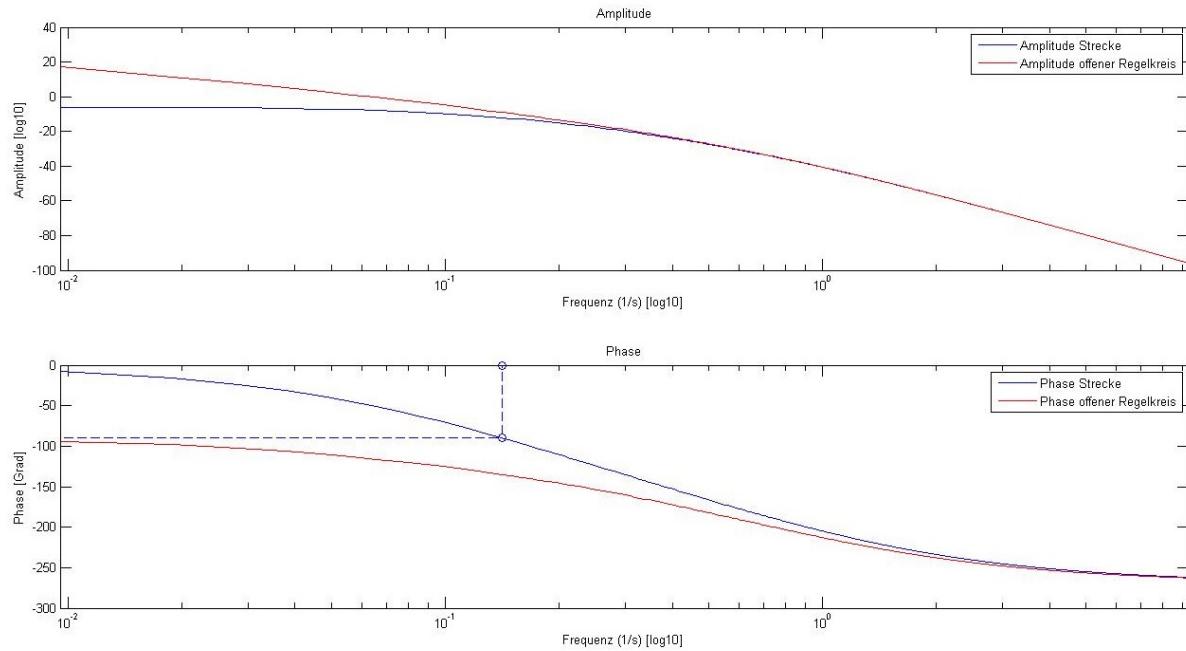


Abbildung 4: Frequenzgang der offenen Strecke

Auf dem Phasengang des offenen Regelkreises (Abbildung 5) wird nun die Frequenz bei $\phi_{Strecke}$ mithilfe der Tabelle 1 auf Seite 10 gesucht (rot gestrichelte Linie). Die gefundene Frequenz ist 0.0611s^{-1} .

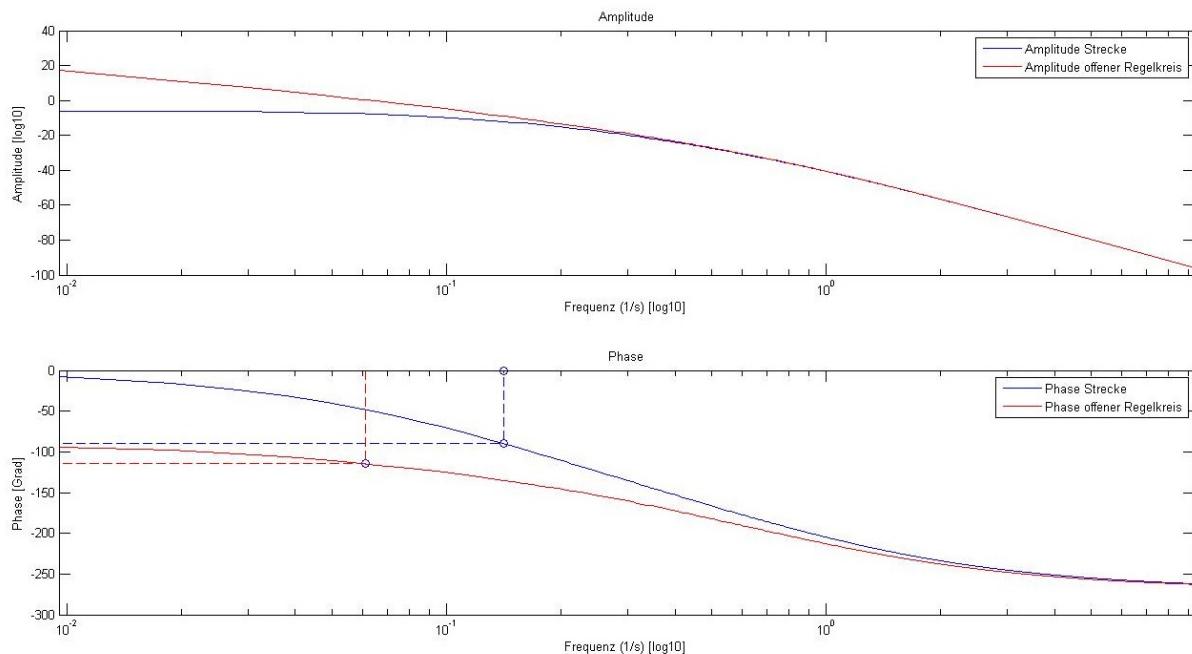


Abbildung 5: ω bei $\phi_{Strecke}$

Schliesslich wird die entsprechende Verstärkung zur gefundenen Frequenz aus dem Amplitudengang des offenen Regelkreises herausgelesen (Abbildung 6). In unserem Beispiel ergibt dies eine Verstärkung von 1.0390. Um diesen Faktor wird die Verstärkung des Reglers gemindert, dies ergibt den letzten benötigten Parameter Kr .

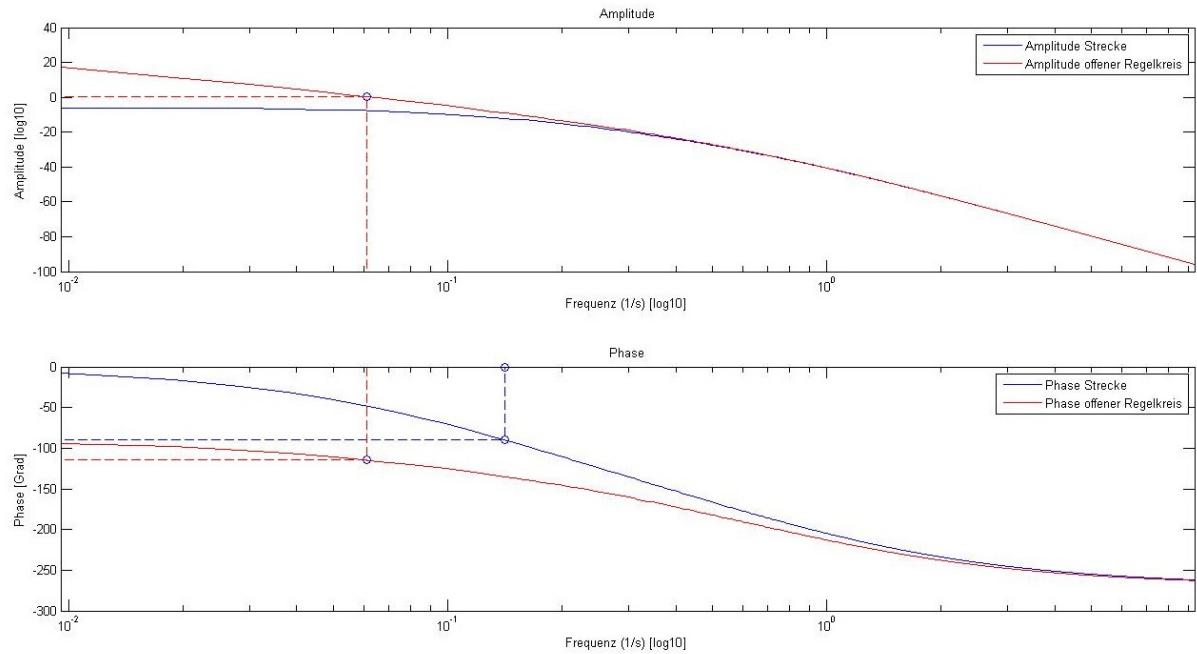


Abbildung 6: Bestimmung von Kr

Die Endresultate dieser Berechnung sind:

$$Tn = 7.0647s$$

$$Kr = 0.9625$$

2.3.2 Numerisches Beispiel PID-Regler

Die Berechnung eines PID-Reglers verläuft ähnlich wie die eines PI-Reglers. Für das Beispiel werden die gleichen Eingabeparameter verwendet wie beim PI-Regler:

$$K_s = 0.5$$

$$T_u = 2.5$$

$$T_g = 18.3$$

$$\phi_{Strecke} = -114.6^\circ$$

Im Gegensatz zum PI-Regler sind alle Ausgabeparameter in der bodekonformen Darstellung. Eine genauere Erklärung dazu ist im Anhang, Sektor A.2, Seite 55 zu finden.

Zunächst werden nun die Zeitkonstanten sowie die Ordnung der Strecke mithilfe der Sani Methode bestimmt. Bei unserer Strecke ergibt dies eine Strecke 3.Ordnung und die folgenden Zeitkonstanten:

$$T_1 = 1.0688s$$

$$T_2 = 3.3484s$$

$$T_3 = 10.4901s$$

Nun wird der Frequenzgang aus der Übertragungsfunktion der Strecke (1) berechnet und in einem Plot aufgezeichnet (siehe Abbildung 7). Danach wird nach dem -135° Punkt auf dem Phasengang gesucht. Dieser ist mit der horizontalen blau gestrichelten Linie eingezeichnet und ergibt folgende Frequenz: $0.2987s^{-1}$.

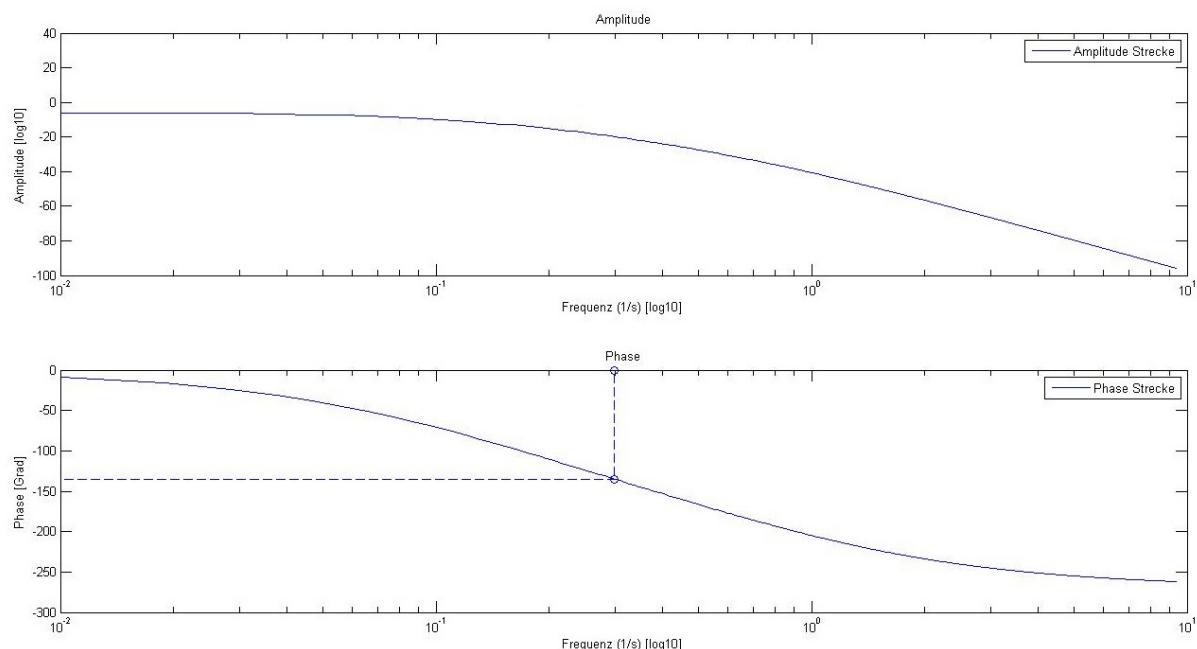


Abbildung 7: Amplituden- und Phasengang der Regelstrecke

Bevor der Frequenzgang des offenen Regelkreises berechnet wird, muss β bestimmt werden. Die Berechnung dazu ist im Anhang, Kapitel A.1 auf Seite 53 ersichtlich. Nach Ausführen dieser Berechnung erhält man $\beta = 0.3192$.

Mit β ist es möglich, $T_{nk} = 10.4902s$ und $T_{vk} = 1.0688s$ zu berechnen. Weiter wird noch T_p bestimmt (siehe Kapitel 2.3, Seite 9). In diesem Beispiel wurde T_p eine Dekade kleiner als T_{vk} gewählt, folglich ist der Wert von $T_p = 0.107s$. Mit diesen Werten und $K_{rk} = 1$ wird nun der offene Amplituden- und Phasengang berechnet (Abbildung 8).

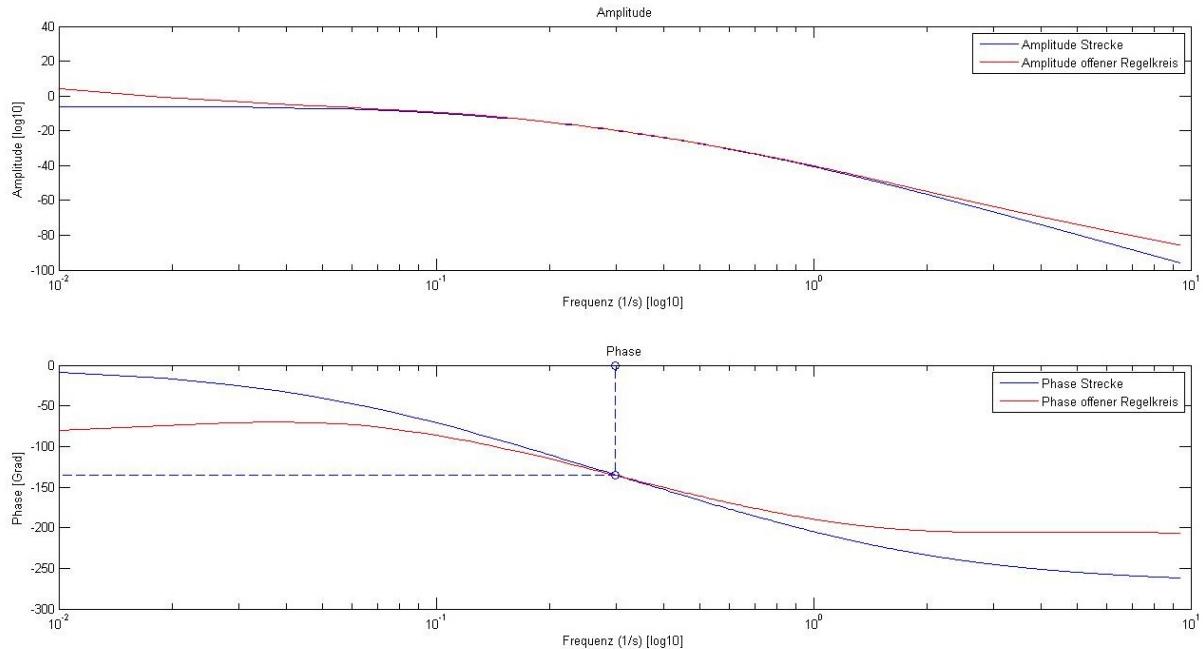
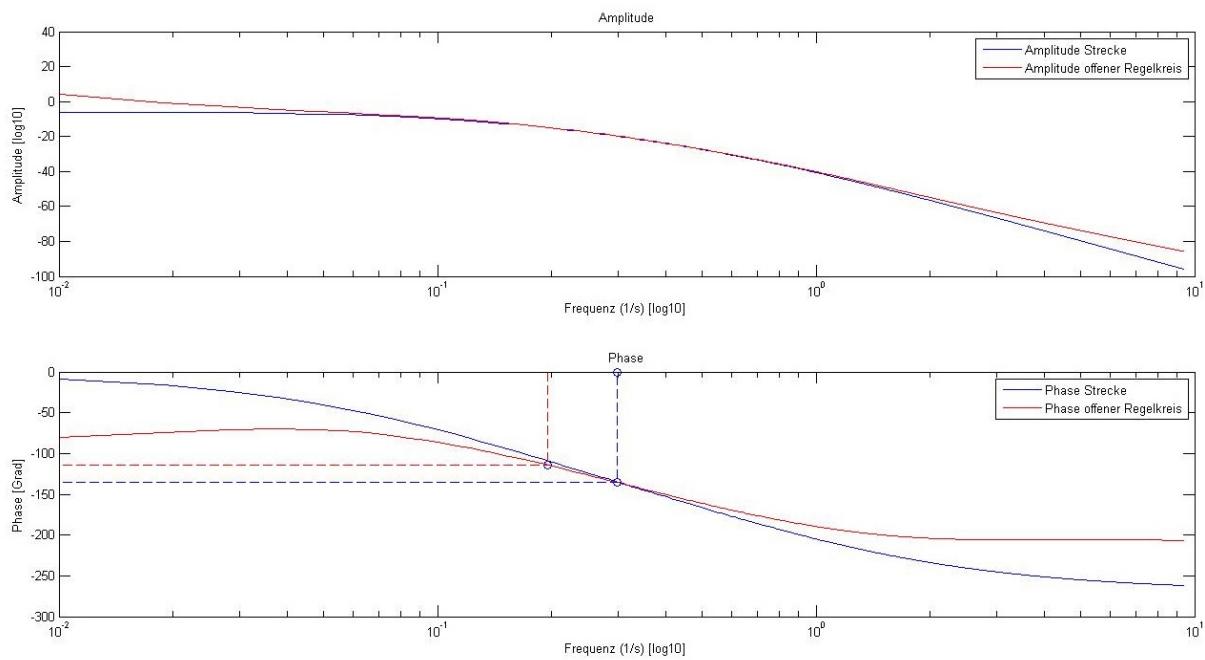
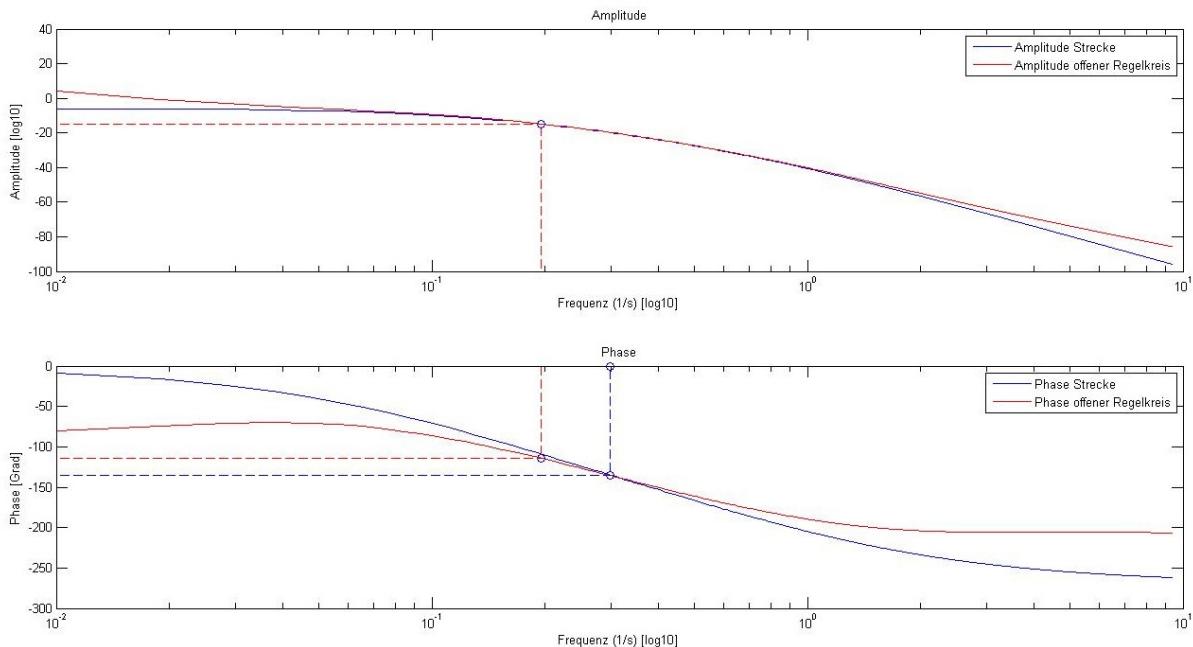


Abbildung 8: Frequenzgang der offenen Regelstrecke

Als nächster Schritt wird die Tabelle 1 auf Seite 10 zu Hilfe genommen. Daraus wird das gewünschte Überschwingen ausgewählt und die entsprechende Frequenz auf dem Phasengang des offenen Regelkreises gesucht (siehe Abbildung 9). Die gefundene Frequenz beträgt: $0.1317s^{-1}$.

**Abbildung 9:** ω bei $\phi_{Strecke}$

Zur gefundenen Frequenz liest man die Verstärkung von 0.3312 heraus. Der Kehrwert davon ergibt den letzten Faktor $Krk = 3.0194$. (Abbildung 10)

**Abbildung 10:** Bestimmung von Krk

Die Endresultate dieser Berechnung sind:

$$Tnk = 10.4902$$

$$Tvk = 1.0688$$

$$Krk = 3.0194$$

Wie schon erwähnt, sind diese Angaben alle in der bodekonformen Darstellung. Umgewandelt in die reglerkonforme Darstellung ergibt dies folgende Werte:

$$Tn = 11.4521$$

$$Tv = 0.8721$$

$$Kr = 3.2963$$

Den genauen Verlauf für die Umrechnung zur reglerkonformen Darstellung ist im Anhang, Sektor A.2, Seite 55 zu finden.

2.4 Übertragungsfunktion des geschlossenen Regelkreises

Die Berechnung des geschlossenen Regelkreises erfolgt mittels der Übertragungsfunktionen der Strecke (Formel (1)) und der des Reglers (Formeln (3) und (5)):

$$G(s) = \frac{G_R(s) \cdot G_S(s)}{1 + G_R(s) \cdot G_S(s)} \quad (7)$$

Die Funktionen $G_R(s)$ und $G_S(s)$ werden nun in $B_R(s)$, $A_R(s)$, $B_S(s)$, $A_S(s)$ aufgeteilt, wobei $B_R(s)$ und $A_R(s)$ das Zähler- bzw. das Nennerpolynom von $G_R(s)$ sind. Analog sind $B_S(s)$ und $A_S(s)$ die jeweiligen Polynome von $G_S(s)$.

Eingesetzt in die Formel 7 ergibt dies folgende Gleichung:

$$G(s) = \frac{\frac{B_R(s) \cdot B_S(s)}{A_R(s) \cdot A_S(s)}}{1 + \frac{B_R(s) \cdot B_S(s)}{A_R(s) \cdot A_S(s)}} \quad (8)$$

Ausmultipliziert erhalten wir:

$$G(s) = \frac{B_R(s) \cdot B_S(s)}{A_R(s) \cdot A_S(s) + B_R(s) \cdot B_S(s)} \quad (9)$$

Nun setzen wir $B_R(s) \cdot B_S(s)$ mit $B(s)$ und $A_R(s) \cdot A_S(s)$ mit $A(s)$ gleich, wodurch folgende Gleichung entsteht:

$$G(s) = \frac{B(s)}{A(s)} \quad (10)$$

Polynom-Bestimmung beim PI-Regler

Als Grundlage werden die Übertragungsfunktionen der Strecke und die des PI-Reglers genommen (Formeln 1 und 3). Dabei wird angenommen, dass die Strecke erster Ordnung sei.

Für die Strecke kann aus dem Zählerpolynom 0 herausgelesen werden. Das Nennerpolynom besteht aus den Koeffizienten T_i und 0. Eingesetzt ergibt das:

$$B_S(s) = [0]$$

$$A_S(s) = [T_i \ 0]$$

Die Funktion der Strecke muss umgeformt werden:

$$G_R(s) = \frac{s \cdot Tn \cdot Kr + Kr}{s \cdot Tn}$$

Daraus können wiederum $B_R(s)$ und $A_R(s)$ bestimmt werden:

$$B_R(s) = \begin{bmatrix} Tn \cdot Kr & Kr \end{bmatrix}$$

$$A_R(s) = \begin{bmatrix} Tn \end{bmatrix}$$

Polynom-Bestimmung beim PID-Regler

Als Regelstrecke wird wiederum eine Strecke 1. Ordnung verwendet, woraus sich direkt die Koeffizienten der Streckenpolynome bestimmen lassen:

$$B_S(s) = \begin{bmatrix} 0 \end{bmatrix}$$

$$A_S(s) = \begin{bmatrix} Ti & 0 \end{bmatrix}$$

Die Formel 5 muss zuerst noch umgeformt werden:

$$G_R(s) = \frac{s^2 \cdot (Kr \cdot Tp \cdot Tn + Kr \cdot Tn \cdot Tv) + s \cdot (Kr \cdot Tn + Kr \cdot Tp) + Kr}{s^2 \cdot (Tp \cdot Tn) + s \cdot Tn}$$

Folgende Koeffizienten können herausgelesen werden:

$$B_R(s) = \begin{bmatrix} Kr \cdot Tp \cdot Tn + Kr \cdot Tn \cdot Tv & Kr \cdot Tn + Kr \cdot Tp & Kr \end{bmatrix}$$

$$A_R(s) = \begin{bmatrix} Tp \cdot Tn & Tn \end{bmatrix}$$

2.4.1 Berechnungsarten

Zur weiteren Berechnung der Schrittantwort wurden folgende Methoden verwendet: IFFT (Inverse Fast Fourier Transformation) und Residuenberechnung (Partialbruchzerlegung).

IFFT

Zuerst wird bei der IFFT der Frequenzgang berechnet. Aus diesem kann anschliessend mit der IFFT selbst die Impulsantwort berechnet. Die aufsummierten Werte der Impulsantwort ergeben die Schrittantwort.

Residuen

Als erstes werden die führenden Nullstellen von $A(s)$ und $B(s)$ entfernt. Danach bestimmt man die Ordnung der Zähler- und Nennerpolynome. Falls diese die gleiche Ordnung haben, muss zuerst eine Polynomdivision durchgeführt werden, um die Konstante K zu bestimmen. Nun werden die Nullstellen von $A(s)$ und $B(s)$ berechnet, worauf die Residuen bestimmt werden können. Mittels dieser Residuen wird die Impulsantwort bestimmt. Aufsummiert ergibt dies die Schrittantwort.

Schrittantwort PI-Regler

Für die Simulation der Schrittantwort eines PI-Reglers werden wieder die gleichen Werte verwendet wie beim Beispiel in Kapitel 2.3.1:

$$K_s = 0.5$$

$$T_u = 2.5$$

$$T_g = 18.3$$

$$\phi_{Strecke} = -114.6^\circ \text{ (entspricht } 4,6\% \text{ Überschwingen)}$$

$$T_n = 7.0647s$$

$$K_r = 0.9625$$

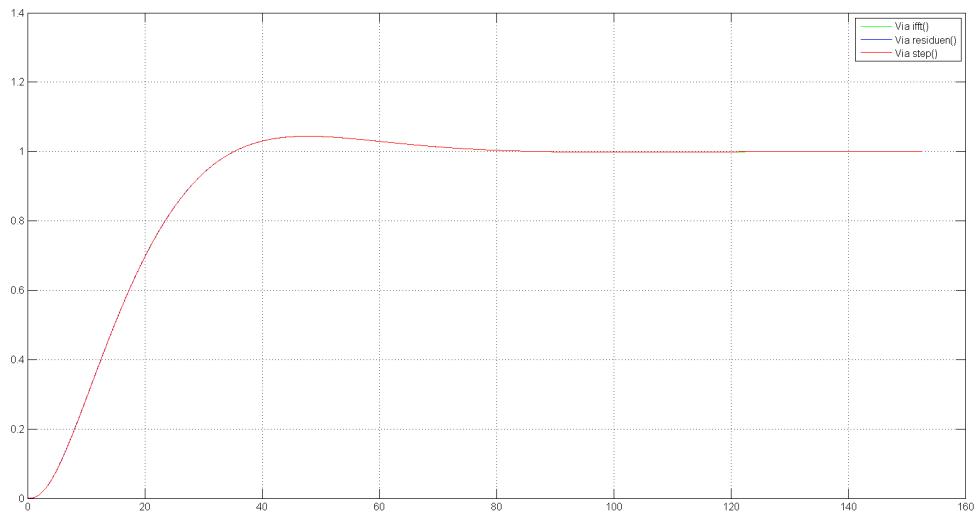


Abbildung 11: Simulation Schrittantwort

Als Vergleich wurde die step() Funktion von Matlab verwendet. Abweichungen sind fast nicht zu erkennen, erst wenn man die Abbildung vergrößert sind kleine Unterschiede festzustellen:

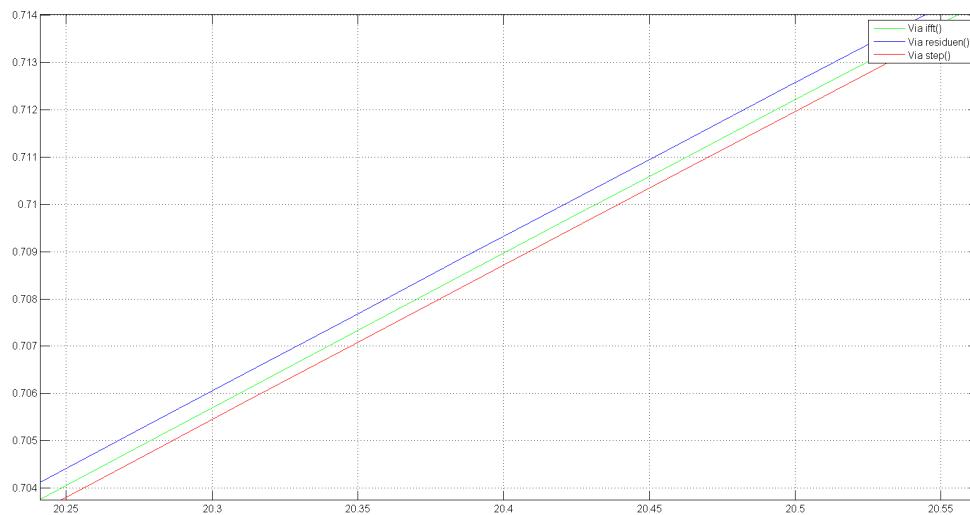


Abbildung 12: Abweichungen der Berechnungsarten

Schrittantwort PID-Regler

Für die Simulation der Schrittantwort eines PID-Reglers werden wiederum die gleichen Werte verwendet wie beim Beispiel in Kapitel 2.3.2:

$$\begin{aligned}K_s &= 0.5 \\T_u &= 2.5 \\T_g &= 18.3 \\\phi_{Strecke} &= -114.6^\circ \\T_n &= 11.4521 \\T_v &= 0.8721 \\K_r &= 3.2963\end{aligned}$$

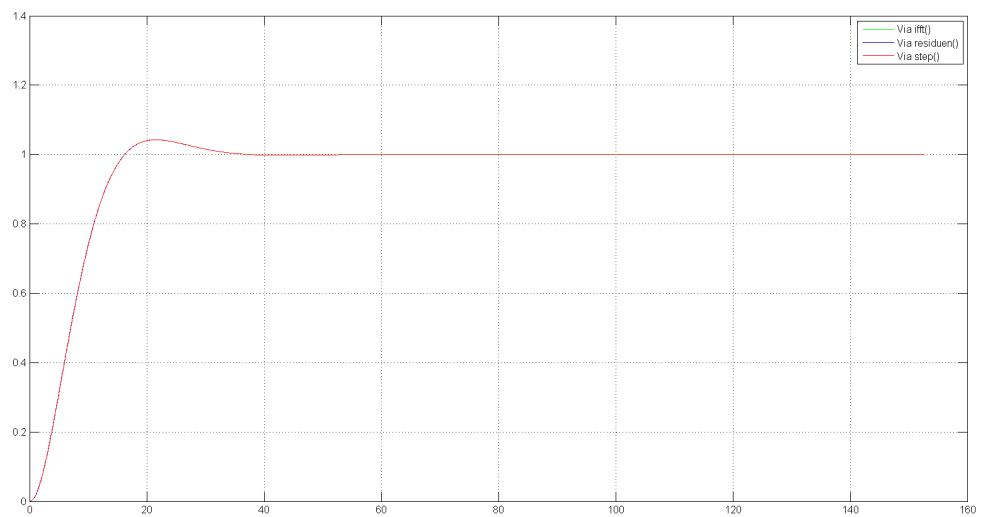


Abbildung 13: Simulation Schrittantwort

Als Vergleichswert wurde auch wieder die step() Funktion von Matlab verwendet. Um Abweichungen zu verdeutlichen, muss die Abbildung vergrößert werden:

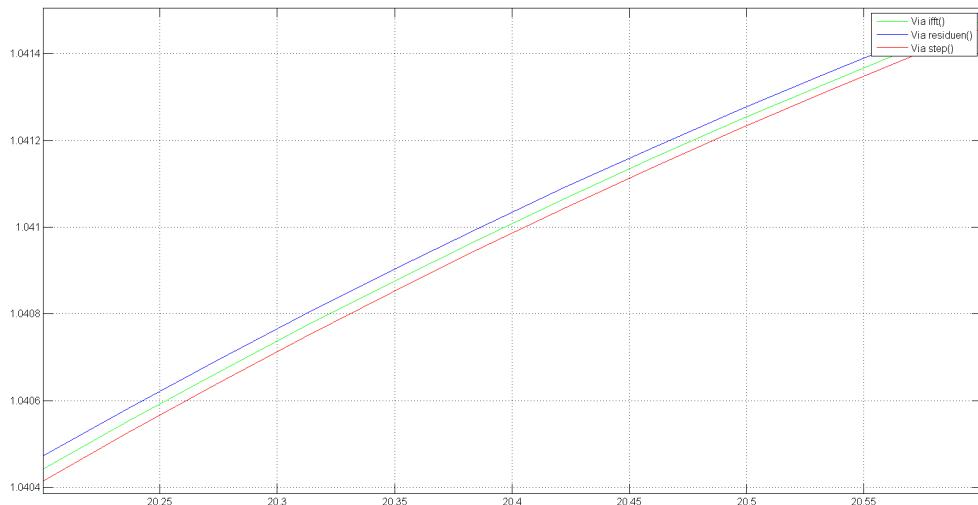


Abbildung 14: Abweichungen der Berechnungsarten

3 Software

Im folgenden Abschnitt wird der Aufbau der Software beschrieben. Die Software basiert auf dem Model-View-Controller Entwurfsmuster, welches für Applikationen mit graphischer Benutzeroberfläche aufgrund seiner hohen Flexibilität und Wiederverwendbarkeit als Standard gilt. Das Model-View-Controller Prinzip basiert auf den drei Hauptklassen *Model*, *View* und *Controller*, welche kurz erläutert werden:

Model: Das *Model* enthält die Daten und Berechnungsmethoden, welche für die Software benötigt werden. Diese Daten sind unabhängig von der grafischen Darstellung der Software. Falls Daten im *Model* geändert werden, informiert das *Model* die Klasse *View*, welche die Daten für den Benutzer darstellt.

View: Die *View* beinhaltet und visualisiert die Benutzeroberfläche. Sie erhält dazu die Daten der Klasse *Model*, welche grafisch dargestellt werden und leitet Eingaben an die Klasse *Controller* weiter. In der Klasse *View* werden keine Berechnungen durchgeführt. Das *Model* ist mit der *View* mittels Observable-Observer Entwurfsmuster verknüpft, wobei das *Model* als Observable und die *View* als Observer agiert.

Controller: Der *Controller* überprüft die Benutzereingaben und leitet diese, falls sie korrekt sind, zur weiteren Berechnung an das *Model* weiter.

Im folgenden werden wiederholt Ausschnitte des Klassendiagramms gezeigt, das komplette Klassendiagramm findet sich im Anhang B.

3.1 Model

Das *Model* ist das Herzstück sämtlicher Berechnungen und führt folgende Funktionen aus:

- Die Koordination der unabhängigen und parallel verlaufenden Berechnungen der Reglertypen und deren Schrittantworten.
- Das Verwalten der dazu notwendigen Objekte, beispielsweise der Sani Kurven.
- Die Implementation des iterativen Approximationsverfahrens zur genauen Bestimmung des Überschwingens.

| Model |
|--|
| <ul style="list-style-type: none"> - overswing : double - parasiticTimeConstantFactor : double - curvesVisible : boolean[] = {true,true,true,true,true,true,true} - zellwegerPhaseInflectionAdjustingCalculationOngoing : boolean = false + setPlant(tu : double, tg : double, ks : double) : void + setRegulatorType(regulatorTypeName : String) : void + setParasiticTimeConstantFactor(parasiticTimeConstantFactor : double) : void + setOverswing(overswing : double) : void + simulateAll() : void + updateZellweger(phaseInflectionOffset : int) : void + selectCalculation(name : String) : void + hideSelectedCalculation() : void + showSelectedCalculation() : void + registerListener(listener : IModelListener) : void + unregisterListener(listener : IModelListener) : void - clearSimulation() : void - validatePlantIsPIDCompliant() : void - getCalculators() : ArrayList<CalculationCycle> - notifyAddCalculation(loop : ClosedLoop) : void - notifyRemoveCalculation(loop : ClosedLoop) : void - notifySimulationBegin(numberOfCalculators : int) : void - notifySimulationComplete() : void - notifyHideCalculation(closedLoop : ClosedLoop) : void - notifyShowCalculation(closedLoop : ClosedLoop) : void - notifySetPlant(plant : Plant) : void + onStepResponseCalculationComplete(closedLoop : ClosedLoop) : void |

Abbildung 15: Klassendiagramm des Models

Als erstes wird die zu Grunde liegende Terminologie erläutert:

Unter einer *Calculation* versteht man den Prozess, einen Regler zu berechnen, einen geschlossenen Regelkreis zu erstellen und davon die Schrittantwort zu berechnen. Es gibt meistens mehrere *Calculations* für jede *Simulation*. Eine *Calculation* wird von der inneren Klasse *CalculationCycle* implementiert.

Eine *Simulation* beinhaltet mehrere *Calculations* (entspricht der Anzahl Reglertypen). Unter *Simulation* versteht man den Prozess, alle zu den entsprechenden Parametern gehörenden *Calculations* durchzuführen. Da die *Calculations* rechenintensiv, aber von einander unabhängig sind, implementiert die Klasse *CalculationCycle* das Interface *Runnable*, damit die Berechnungen mittels einem Thread-Pool parallel verlaufen können. Eine *Simulation* wird direkt von der *Model*-Klasse übernommen.

Der Prozess, die Schrittantwort eines geschlossenen Regelkreises zu berechnen, fängt mit der Erstellung eines *Plant*-Objektes an, welches die zu regelnde Strecke beschreibt. Es gibt sehr viele Möglichkeiten und Verfahren, einen passenden Regler für die Strecke zu berechnen, was auch von der komplexen Vererbungshierarchie der verschiedenen *AbstractControllerCalculator*-Klassen reflektiert wird. Was zwischen den Rechner-Klassen gemeinsam bleibt ist, dass sie als Eingabe ein *Plant*-Objekt entgegennehmen und als Ausgabe ein *Controller*-Objekt herausgeben. Es kann zum Beispiel mit Hilfe der *ZellwegerPID*-Klasse ein *ControllerPID*-Objekt mittels der Zellweger Methode erstellt werden. Mit dem *Controller*-Objekt kann zusammen mit dem *Plant*-

Objekt ein geschlossener Regelkreis mit der Klasse *ClosedLoop* erstellt werden. Diese Klasse erlaubt das Berechnen einer Schrittantwort.

Die *Model*-Klasse hat viele Methoden, die das Einstellen einer *Calculation* beziehungsweise einer *Simulation* steuern kann. Die wichtigsten Methoden sind folgend aufgezählt:

- *Model::setPlant()*: Erlaubt es dem Anwender, die Strecke mit den Parametern T_u , T_g und K_r zu definieren.
- *Model::setRegulatorType()*: Mit dieser Methode kann der Reglertyp ausgewählt werden. Dabei kann zwischen *I*, *PI* und *PID* ausgewählt werden. Insgesamt unterstützt „Easy-PID“ 15 Varianten, einen Regler zu berechnen. Diese Methode filtert dabei die Varianten, so dass nur die Regler berechnet werden, die mit dem ausgewählten Reglertyp übereinstimmen.
- *Model::setOverswing*: Erlaubt es dem Benutzer, das gewünschte Überschwingen in Prozent einzugeben. Dabei wird bei der Zellweger-Methode der Parameter K_r iterativ angepasst, bis das Überschwingen mit dem angegebenen Wert übereinstimmt. Die Faustformeln werden von dieser Methode nicht beeinflusst.

Die Methode *Model::setPlant()* erlaubt es dem Anwender, die Strecke mit den Parametern T_u , T_g , und K_r zu definieren.

Außerdem kann die parasitäre Zeitkonstante angegeben werden. Die Zellweger-Regler berechnen T_p , indem sie diesen Faktor mit T_{vk} multiplizieren. Die Faustformeln berechnen T_p , indem sie diesen Faktor mit T_v multiplizieren. Gewöhnlich wird hier etwa 10% benutzt.

3.1.1 Regelstrecke

Eine Regelstrecke, im Programm vom Englischen als *Plant* bezeichnet, beschreibt die zu regelnde Strecke anhand einer *TransferFunction* (Übertragungsfunktion), einer Liste von *timeConstants* (Zeitkonstanten) und natürlich anhand der Parameter T_u , T_g und K_r . Bei der Instanzierung eines *Plant*-Objektes werden die drei Parameter zusammen mit einer Instanz der Klasse *SaniCurves* übergeben.

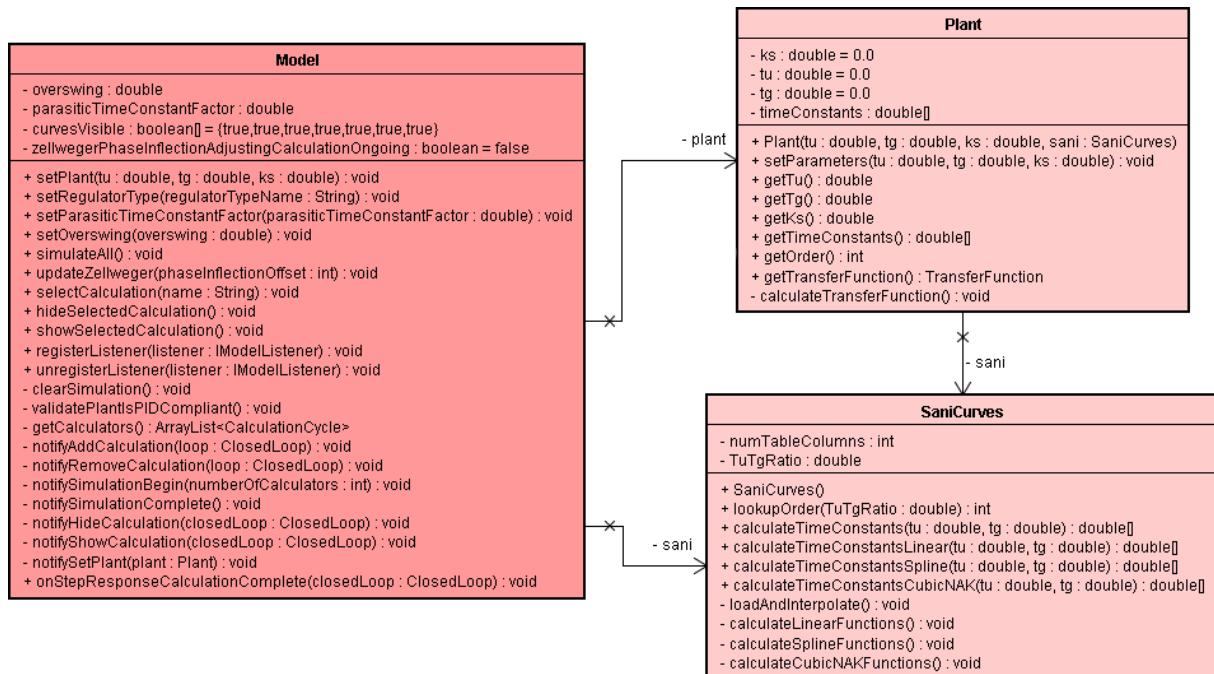


Abbildung 16: Klassendiagramm der Regelstrecke

Die Sani-Kurven sind die Grundlage zur Berechnung der *TransferFunction*, diese bildet wiederum die Basis zur späteren Berechnung der Schrittantwort.

Die Sani-Kurven werden nicht vom Programm selbst berechnet, sondern werden zu Programmbeginn aus einer Text-Datei importiert, welche wiederum von Matlab exportiert wurde. Da die Kurven als diskrete Punkte gespeichert sind, werden sie beim Importieren in Java kubisch interpoliert, um eine höhere Auflösung zu erreichen. Das Interpolieren erfolgt mittels der *SplineNAK*-Klasse, welche von C in Java übersetzt wurde. [4]

Da diese Interpolationsmethode mit der in Matlab verwendeten übereinstimmt, wird ein stabiler Benchmark für Tests ermöglicht.

Das *SaniCurves*-Objekt wird nur einmal im *Model* erstellt, um die Festplattenzugriffszeit zu minimieren. Das *Model*-Objekt hat zu jedem Zeitpunkt eine Referenz auf den aktuellen *Plant*.

Jedes Mal, wenn sich mindestens einer der drei Parameter T_u , T_g oder K_r im *Plant*-Objekt ändert, sei es durch eine Instanzierung oder durch Aufrufen der Methode *Plant::setParameters()*, werden mittels *SaniCurves* die Zeitkonstanten der Strecke und die Übertragungsfunktion neu berechnet.

3.1.2 Regler

Der Regler, im Programm vom Englischen als *Controller* benannt, beschreibt den Regler der Strecke anhand einer *TransferFunction* (Übertragungsfunktion).

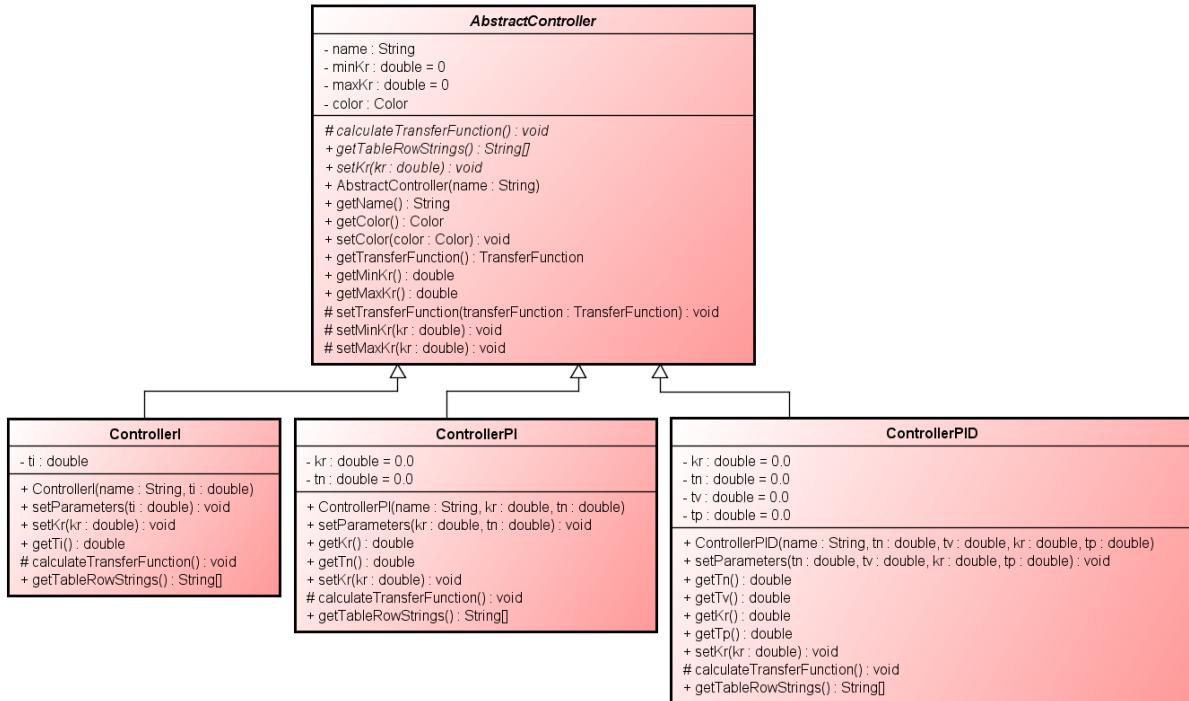


Abbildung 17: Klassendiagramm des Reglers

Nicht alle Regler haben gemeinsame Parameter. Bei den verschiedenen Reglertypen sind die Parameter anders benannt oder gar nicht vorhanden. Beispielsweise hat der PI-Regler die Parameter T_u und T_g , aber es fehlen die PID-Parameter T_v und T_p . Außerdem wird abhängig vom Reglertyp die Übertragungsfunktion anders berechnet. Es gibt aber auch Gemeinsamkeiten, wie der Name oder die Farbe des Reglers und die Tatsache, dass die Parameter in einer Tabelle eingefügt werden müssen. Für den Rest des Programms ist außerdem nicht ersichtlich, um was für einen Reglertypen es sich handelt. Für die Schrittantwort wird lediglich die Übertragungsfunktion verwendet.

Aus diesen Gründen wurde mit Vererbung gearbeitet. Die Klassen *ControllerI*, *ControllerPI* und *ControllerPID* implementieren jeweils die Details der entsprechenden Reglertypen. Die Superklasse *AbstractController* stellt eine gemeinsame Schnittstelle zur Verfügung.

Immer, wenn sich die Reglerparameter ändern, also dann wenn der Regler instanziert wird, wird seine Übertragungsfunktion berechnet. Diese ist mittels der Methode *AbstractController::getTransferFunction()* abrufbar.

Jeder Regler definiert eine Farbe und einen Namen für den Plot. Auf diese Informationen kann mittels der Methoden *AbstractController::getName()* und *AbstractController::getColor()* zugegriffen werden.

Die erbenden Klassen implementieren die Methode *getTableRowStrings()*, welche erlaubt, die Reglerparameter über die Superklasse in die Tabelle einzufügen ohne zu wissen, um was für einen Reglertypen es sich handelt.

Wurde der Regler mittels einer Zellwegermethode berechnet, kann ein Fenster für den Minimal- und den Maximalwert des Parameters K_r über die Methoden `AbstractController::getMinKr()` und `AbstractController::getMaxKr()` geholt werden. Dieses Fenster wird für das iterative Approximieren des Überschwingens verwendet. Alle anderen Berechnungsmethoden, also die Faustformeln, berechnen diese Fenster nicht.

3.1.3 Reglerberechnung

Die Reglerberechnung ist hierarchisch und technisch das komplizierteste Teilstück des Programms. Es gibt insgesamt 15 verschiedene Klassen, die auf 15 verschiedene Arten einen passenden Regler für eine Strecke berechnen können. Alle erben von der Klasse `AbstractControllerCalculator`.

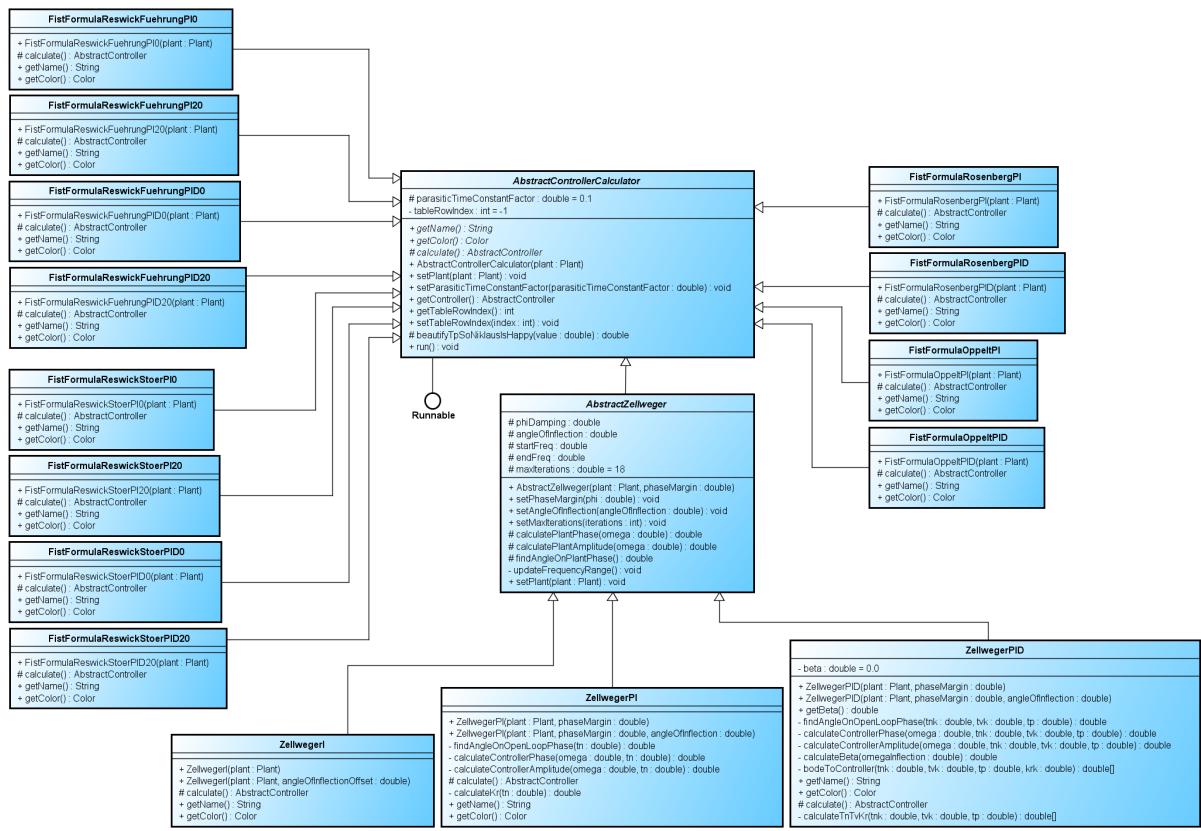


Abbildung 18: Klassendiagramm zur Reglerberechnung

Grundlegend kann ein neues `ControllerCalculator`-Objekt mit einem `Plant`-Objekt (die Strecke) instanziert werden, es kann `AbstractControllerCalculator::run()` aufgerufen werden und es kann mittels der Methode `AbstractControllerCalculator::getController()` der resultierende Regler als `AbstractController` geholt werden.

Wie der Regler berechnet wird und welchem Typ der Regler angehört, ist dabei durch die Vererbung definiert.

3.1.4 Regelkreis

Mit der Klasse *ClosedLoop* kann mit einem *Plant*-Objekt und einem *AbstractController*-Objekt ein geschlossener Regelkreis erstellt werden. Auch der *ClosedLoop* hat eine Übertragungsfunktion (ein *TransferFunction*-Objekt), welche aus den Übertragungsfunktionen der Strecke und des Reglers berechnet wird.

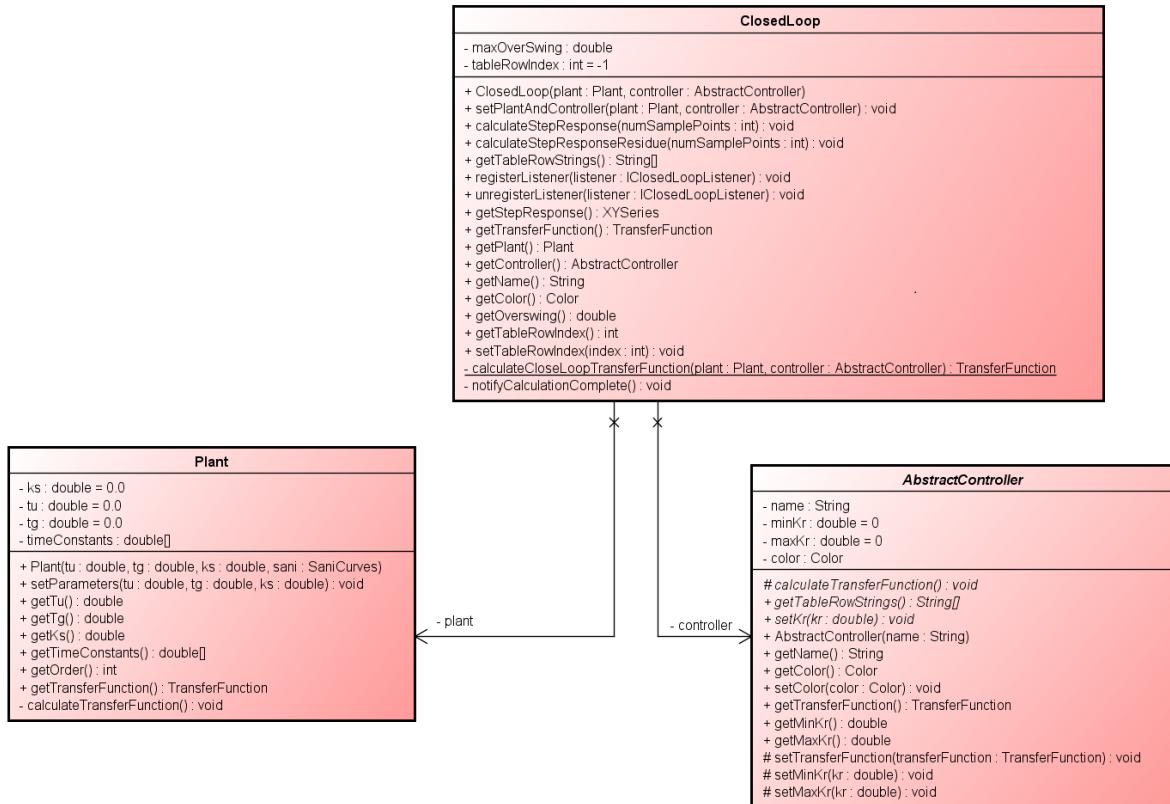


Abbildung 19: Klassendiagramm zum Regelkreis

Immer wenn sich das *Plant*-Objekt oder das *AbstractController*-Objekt ändert, sei es durch eine Instanzierung oder durch Aufrufen der Methode *ClosedLoop::setPlantAndController()*, wird aus den beiden Übertragungsfunktionen des *Plant*-Objektes und des *AbstractController*-Objektes die Übertragungsfunktion des geschlossenen Regelkreises berechnet.

Durch Aufrufen der Methode *ClosedLoop::calculateStepResponse()* wird die Sprungantwort mittels der Residuenmethode berechnet. Die resultierende Schrittantwort kann mit der Methode *ClosedLoop::getStepResponse()* als *XYSeries* für JFreeChart geholt werden. Die Residuenmethode wurde gewählt, da sie robuster und schneller ist als die Variante mit IFFT. Ein Vergleich dazu ist im Abschnitt Validierung unter 4.2.3 aufgeführt.

Weiter wird bei der Berechnung der Sprungantwort das maximale Überschwingen gemessen. Dieser Wert kann mit der Methode *ClosedLoop::getOverswing()* geholt werden.

3.1.5 Resultate

Die Resultate der Berechnungen sind die Reglerparameter T_n , T_v und K_r , die parasitäre Zeitkonstante T_p , das Überschwingen, der Name der verwendeten Methode und die Schrittantwort.

Die Schrittantworten werden mit JFreeChart geplottet und die restlichen Parameter können mit der Methode `AbstractController::getTableRowStrings()` als String-Array geholt werden und werden im GUI in eine Tabelle eingefügt.

3.2 View

Die Klasse *View* und ihre Unterklassen sind zuständig für die Darstellung der Benutzeroberfläche. Am oberen Fensterrand befindet sich die *MenuBar*, welche sich über die gesamte Breite erstreckt. Sie enthält Einstellungsmöglichkeiten und weiterführende Informationen. Auf der linken Seite befinden sich zwei Panels für die Ein- und Ausgabe. Im *InputPanel* können die Eingabeparameter eingetragen, der gewünschte Regler und das Überschwingen gewählt und die Simulation mit einem Button gestartet werden. Das *OutputPanel* stellt alle berechneten Graphen mit der Angabe ihrer Kennwerte in einer Tabelle dar und besitzt einen Trimmer für das manuelle justieren des Phasenwinkels. Auf der rechten Seite des Fensters werden die Graphen mit der Klasse *GraphPanel* angezeigt. Unter diesem Plot befindet sich das *GraphDisplayPanel* und das *GraphSettingPanel*, welche beide für die Auswahl der darzustellenden Kurven verantwortlich sind. Im folgenden Abschnitt wird die Darstellung des GUIs (Graphical User Interface) genauer erläutert.

3.2.1 Anforderungen an die Benutzeroberfläche

Die Benutzeroberfläche wurde so umgesetzt, dass der Anwender das Tool intuitiv bedienen kann und es einen grafisch ansprechenden Eindruck hinterlässt. Des Weiteren wird nach Wunsch des Auftraggebers das Programm in einem einzelnen Fenster dargestellt. Dies ermöglicht es dem Anwender, alles zu überblicken und mit einem einzigen Printscreen das gesamte Programm (Einstellungen, Graph) in einer Dokumentation festzuhalten oder mit anderen Methoden zu vergleichen. Dies führt zu einer einfacheren Handhabung der erstellten Simulationen.

Vom Auftraggeber wurden folgende Punkte für die Benutzeroberfläche gewünscht:

- Es soll lediglich der Graph der Schrittantwort dargestellt werden.
- Eine automatische Skalierung über einen sinnvollen Bereich für den Graphen soll implementiert sein. Falls dies nicht gelingt, sollten die Achsen manuell einstellbar sein.
- Das im Graphen ausgemessene Überschwingen soll in einer Tabelle dargestellt werden.
- Die Fenstergröße soll veränderbar sein, zusätzlich sollte eine Miniversion enthalten sein.
- Alle Informationen sollten in einem Fenster angezeigt werden, sodass mit einem Blick alles erkannt werden kann.

3.2.2 Erstellung der Benutzeroberfläche

Für die Erstellung der Benutzeroberfläche wurden hauptsächlich die Swing-Klassen der Java Foundation Classes (JFC) verwendet. Sie enthalten die notwendigen Komponenten wie Textfelder, Buttons usw., mit denen grafische Oberflächen erstellt werden können. Erkennbar sind sie auch durch das „J“, welches dem Namen vorangestellt ist (beispielsweise JButton). Auf die ausführliche Beschreibung dieser Standardkomponenten wird bewusst verzichtet und es werden nur die zusätzlichen Komponenten erläutert, welche verwendet wurden.

Für die Platzierung der Komponenten gibt es in Java verschiedene Layoutmanager, um die Komponenten anzurichten. Um einen kurzen Überblick über diese Manager zu geben, werden die verwendeten in der folgenden Aufzählung kurz beschrieben:

1. *GridBagLayout*: Er ist der mächtigste LayoutManager aus dem Java AWT-Package und ermöglicht es, die Komponenten in ein anpassbares Raster (Grid) zu setzen. Die einzelnen Spalten können dabei unterschiedlich breit und die einzelnen Zeilen unterschiedlich hoch sein. Die Komponenten werden in eine Zelle gesetzt und können von dort in X- und Y-Richtung weitere Zellen überlappen. Weiter gibt es viele zusätzliche Anpassungsmöglichkeiten.

2. *GridLayout*: Setzt die Komponenten in ein gleichmässiges Raster (Grid), bei dem alle Zellen die selbe Grösse haben.
3. *BorderLayout*: Besitzt fünf Bereiche (Norden, Süden, Osten, Westen, Zentrum), in welche Komponenten platziert werden können.
4. *WrapLayout*: Alle Komponenten werden nacheinander von links nach rechts eingefüllt. Beim Verändern der Fenstergrösse gibt es Zeilenumbrüche, falls die Breite des Panels nicht mehr ausreicht. Dieses Layout gehört nicht zu den Standard-Typen und wird bei der Erläuterung des *GraphDisplayPanels* genauer beschrieben.

Für weitere Informationen zu Layoutmanagern wird auf die Dokumentation von Oracle verwiesen [5].

Aufbau der Benutzeroberfläche

Der Aufbau des GUIs wurde so erstellt, dass die gesamte grafische Benutzeroberfläche in einem JFrame platziert ist. Dieses Frame enthält jeweils eine Instanz der Klasse *MenuBar* und *View*. Die *View* besteht wiederum aus den vier JPanels *InputOutputPanel*, *GraphPanel*, *GraphDisplayPanel* und *GraphSettingPanel*. Das *InputOutputPanel* fasst das *InputPanel* und das *OutputPanel* zu einem gemeinsamen JPanel zusammen. Auf die Layoutmanager wird bei der Beschreibung der einzelnen Panels genauer eingegangen.

In der folgenden Grafik wurde der gesamte Zusammenhang dargestellt:

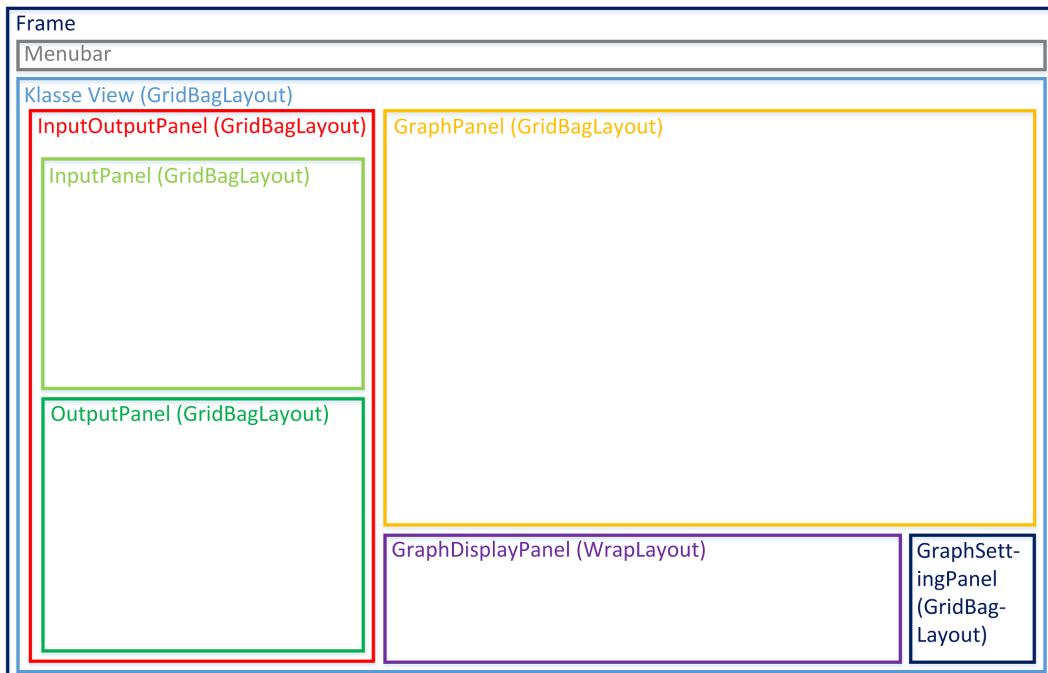


Abbildung 20: Aufbau des GUIs

3.2.3 Aufgabe der einzelnen Panels

Die Benutzeroberfläche beinhaltet, wie vorhin beschrieben, mehrere Panels. Das *InputPanel* dient zur Eingabe der Kenngrößen der Regelstrecke und das *OutputPanel* zur Anzeige der berechneten Reglerparameter und zur nachträglichen Manipulation des Phasenwinkels. Das *GraphPanel* visualisiert die Schrittantwort, das *GraphDisplayPanel* ermöglicht die Auswahl der anzuzeigenden Kurven und das *GraphSettingPanel* besitzt drei Buttons für die Einstellungen zum Graphen. Ein Grossteil der verwendeten Komponenten stammt aus der Swing Klasse von Java, alle zusätzlichen Elemente werden speziell erläutert. Folgend werden die Panels genauer beschrieben:

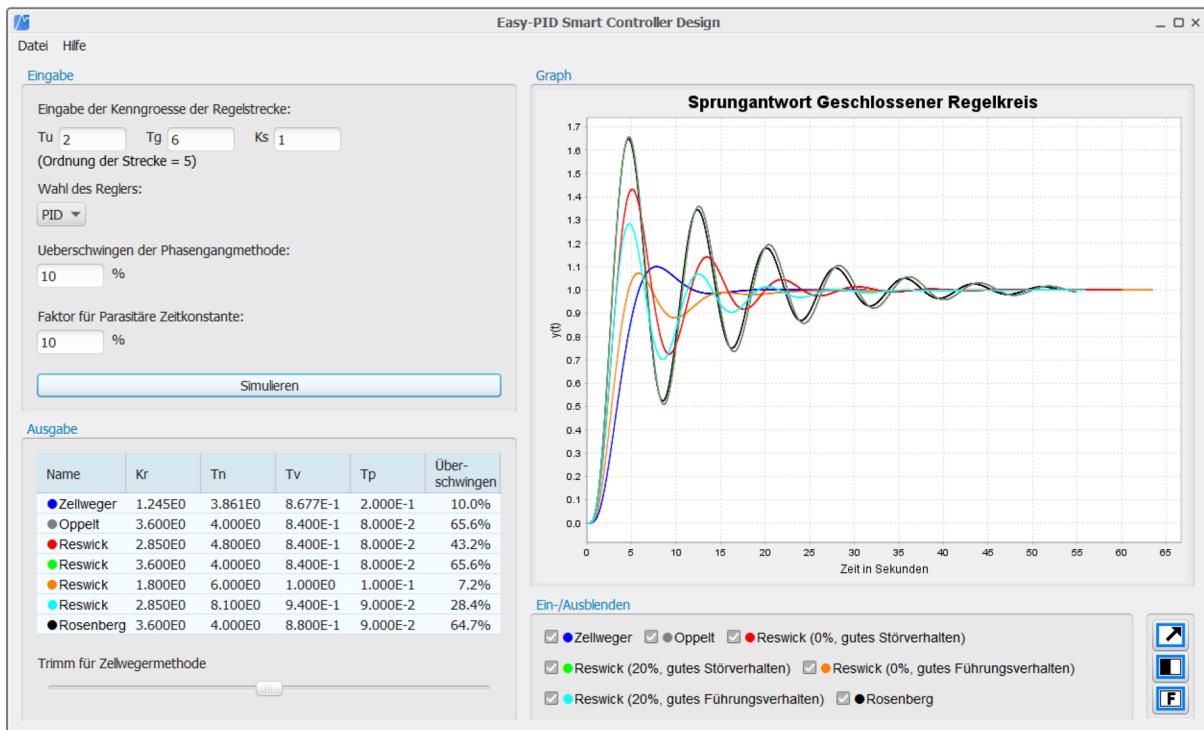


Abbildung 21: Übersicht über das GUI

InputOutputPanel: Die Aufgabe dieses im GridBagLayout erstellten Panels besteht einzig darin, das *InputPanel* und das *OutputPanel* in einem gemeinsamen Panel zu vereinen. Dieses Panel wurde erstellt, um die Handhabung im GridBagLayout der Klasse *View* zu verbessern.

InputPanel: Dieses Panel wurde im GridBagLayout erstellt und besitzt einen Rahmen mit dem Titel „Eingabe“. Es enthält zuoberst die drei Eingabefelder für die Parameter T_u (1), T_g (2) und K_s (3) der Regelstrecke. Anschliessend folgt eine leere Zeile, die zu Beginn keinen Inhalt hat. Sie zeigt nach einer Simulation den Grad der Berechnung an und bei fehlerhaften Eingaben rote Hinweise für den Anwender. Als nächstes folgt das Dropdown-Menü für die Auswahl des Reglers (4) (*I*, *PI* oder *PID*). In den nächsten beiden Textfeldern (4 und 5) kann das Überschwingen und die parasitäre Zeitkonstante eingetragen werden. Diese beiden Eingaben sind abhängig von der Auswahl des Reglers und deshalb nicht für alle Regler aktiviert. Wenn die Felder deaktiviert sind, werden sie ausgegraut und es können keine Werte eingetragen werden. Beim I-Regler werden beide und beim PI-Regler nur das Feld für die parasitäre Zeitkonstante deaktiviert. Beim PID-Regler sind beide aktiviert. Am Ende dieses Panels befindet sich über die gesamte Breite die Schaltfläche zum Starten der Simulation (7).

Alle Beschriftungen sind *JLabels*, alle Textfelder sind *JFormattedDoubleTextfields*, die Schaltfläche ist ein *JButton* und für die Wahl des Reglers wurde eine *JComboBox* verwendet. Weiter verfügen alle Textfelder über ToolTips (Zusatzinformationen bei Objekten), welche dem Nutzer zusätzliche Informationen bieten.

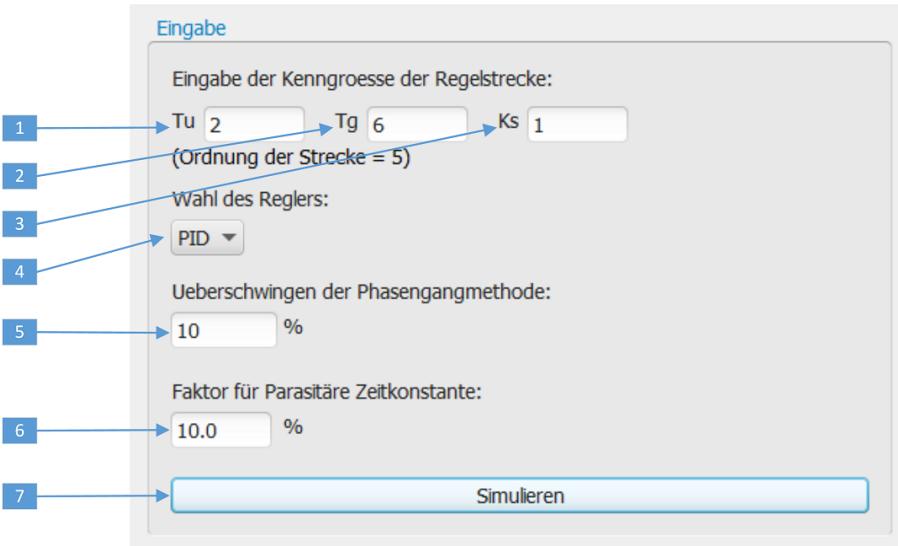


Abbildung 22: InputPanel

OutputPanel: Dieses Panel wurde im GridBagLayout erstellt und besitzt einen Rahmen mit dem Titel „Ausgabe“. In einer übersichtlichen Tabelle werden die Kennwerte jeder simulierten Kurve aufgelistet. Diese Tabelle besteht aus den sechs Spalten mit den folgenden Inhalten:

1. *Name*: Als erstes steht ein grosser farbiger Kreis in der Farbe der dazugehörigen Kurve. Dadurch kann der Anwender den Zusammenhang zwischen Tabelle, Graph und Auswahl der Kurven leicht erkennen. Nach dem Kreis folgt im selben Feld der Name des Reglers.
2. *Kr*: Enthält den berechneten Wert für die Reglerverstärkung
3. *Tn*: Enthält den berechneten Tn-Reglerparameter
4. *Tv*: Enthält den berechneten Tv-Reglerparameter
5. *Tp*: Enthält den berechneten Tp-Reglerparameter
6. *Überschwingen*: Enthält das gemessene Überschwingen in Prozent

Für die nachträgliche Optimierung folgt unter der Tabelle ein Slider (*Trimm für Zellwegermethode*), mit dem der Phasenwinkel manuell verändert werden kann (7). Die Tabelle des *OutputPanels* ist eine JTable mit einem TableHeader und der Slider ein JSlider.

The diagram illustrates the *OutputPanel* interface. At the top, there is a title bar labeled "Ausgabe". Below it is a *JTable* with the following structure:

| | Name | Kr | Tn | Tv | Tp | Über-schwingen |
|---|-----------|---------|---------|----------|----------|----------------|
| ● | Zellweger | 1.245E0 | 3.861E0 | 8.677E-1 | 2.000E-1 | 10.0% |
| ● | Oppelt | 3.600E0 | 4.000E0 | 8.400E-1 | 8.000E-2 | 65.6% |
| ● | Reswick | 2.850E0 | 4.800E0 | 8.400E-1 | 8.000E-2 | 43.2% |
| ● | Reswick | 3.600E0 | 4.000E0 | 8.400E-1 | 8.000E-2 | 65.6% |
| ● | Reswick | 1.800E0 | 6.000E0 | 1.000E0 | 1.000E-1 | 7.2% |
| ● | Reswick | 2.850E0 | 8.100E0 | 9.400E-1 | 9.000E-2 | 28.4% |
| ● | Rosenberg | 3.600E0 | 4.000E0 | 8.800E-1 | 9.000E-2 | 64.7% |

Below the table is a slider labeled "Trimm für Zellwegermethode". A number "7" is highlighted with a blue box and an arrow points from it to the slider.

Abbildung 23: OutputPanel

GraphPanel: Das *GraphPanel* stellt die Schrittantworten der im *GraphDisplayPanel* ausgewählten Regler-Typen dar. Die Achsen werden dabei automatisch skaliert und falls im *GraphDisplayPanel* Anpassungen vorgenommen wurden, werden die einzelnen Kurven ein- oder ausgebendet. Für das Anzeigen des Graphen wird ein XYPlot der Klasse *JFreeChart* verwendet, der im BorderLayout platziert wurde.

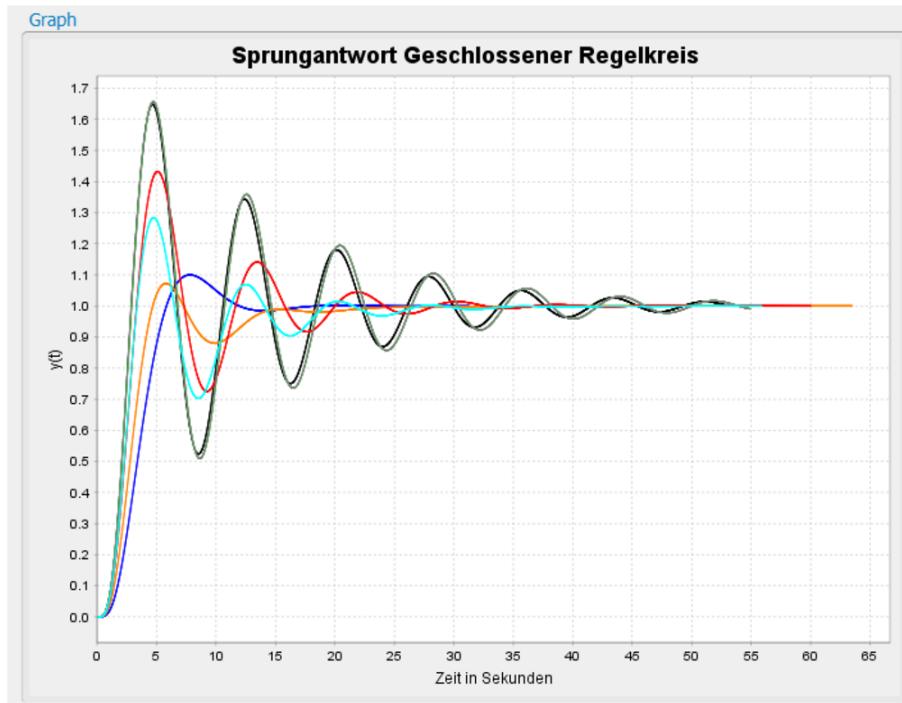


Abbildung 24: GraphPanel

GraphDisplayPanel und GraphSettingPanel: Im *GraphDisplayPanel* können mit Hilfe der Checkboxen die Regler ausgewählt werden, welche im *GraphPanel* angezeigt werden sollen (1). Rechts daneben wurde das *GraphSettingPanel* platziert. Dieses zusätzliche Panel verfügt über drei Buttons mit den folgenden Funktionen:

- (2) Zur Original-Ansicht wechseln (Zoom auf 100%).
- (3) Alle Kurven ein-/ausblenden.
- (4) Alle Faustformeln ein-/ausblenden.

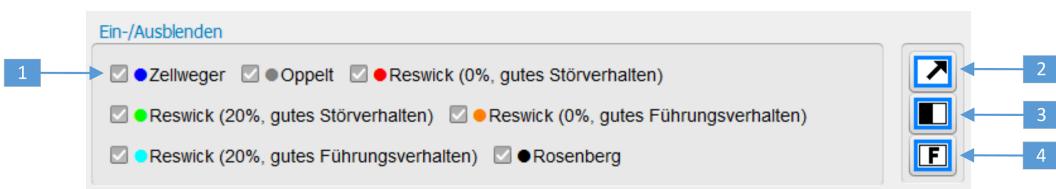


Abbildung 25: GraphDisplayPanel

Alle Checkboxen sind JCheckBoxen der Swing-Klassen und die drei Button sind JButtons mit selbst erstellten Icons.

Beim *GraphDisplayPanel* wurde der *WrapLayoutManager* verwendet. Dies ist kein Standard Layoutmanager und wurde auf einem Web-Blog gefunden [6]. Dieser Manager ist von der Funktion her vergleichbar mit dem FlowLayout und füllt alle Komponenten von links nach rechts in ein JPanel. Speziell an ihm ist jedoch, dass er beim Verändern der Fenstergrösse den Umbruch der Zeilen richtig handhabt, auch wenn das Panel in einem GridBagLayout platziert ist.

Ausserdem befindet sich am oberen Rand des Fensters die *MenuBar* mit den beiden Schaltflächen *Datei* und *Hilfe*. Sie wurde direkt ins Frame gesetzt und wurde nicht in der Klasse *View* hinzugefügt.

Das Menü *Datei* besteht aus den folgenden Menüeinträgen:

- *Zur Mini-Version wechseln*: Wechselt von der Gesamtansicht zur Mini-Version. Elemente wie das *GraphPanel*, das *GraphDisplayPanel* und der Trimmer werden dabei ausgeblendet. Der Name dieses Menüeintrages ändert sich von Mini-Version zu Normal-Version.
- *Export als PDF*: Öffnet ein Fenster, mit dem der Pfad und der Dokumentname ausgewählt werden kann. Nach der Bestätigung mit dem Button „Speichern“ wird ein PDF generiert, welches die gesamte Ansicht des Tools exportiert.
- *Schliessen*: Beendet die gesamte Applikation.

Das Menü *Hilfe* besteht aus den folgenden Menüeinträgen:

- *Info*: Es öffnet sich ein Fenster mit Informationen zum Projekt und den Autoren.
- *Hilfreiche Links*: Mit ihnen können Links und PDF-Dokumente geöffnet werden, welche weiterführende Information zur Reglerdimensionierung und der Zellweger-Methode bieten.

Das gesamte Menü wurde mit Komponenten der Swing-Klassen erstellt.

3.2.4 Die Klasse View im Klassendiagramm

Im folgenden Ausschnitt aus dem Klassendiagramm ist der Zusammenhang der Klasse *View* zu den Panels und den ganzen Swing-Komponenten dargestellt. Für das gesamte Klassendiagramm wird auf den Anhang B verwiesen.

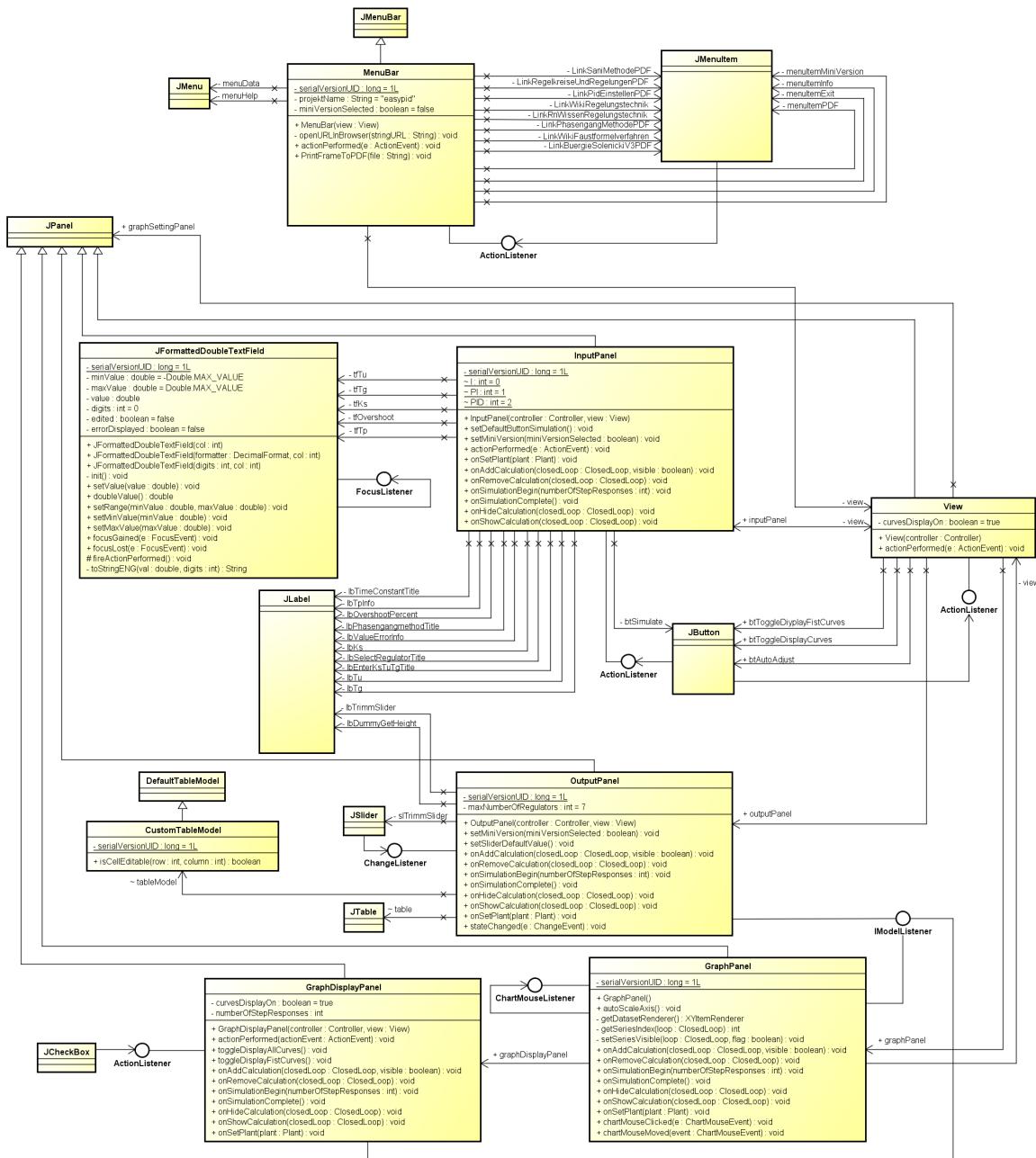


Abbildung 26: Klasse View und ihre Unterklassen

3.2.5 Funktionalität der Benutzeroberfläche

Eingabewerte

Für alle Textfelder wurden JDoubleTextFields verwendet. Diese Klasse bekamen wir im Rahmen des Fachinputs durch Herrn Prof. Dr. Richard Gut empfohlen. Sie handhabt die unerlaubten Eingaben der Textfelder. Dabei wurden folgende Einstellungen vorgenommen:

1. Die Felder werden nicht gerundet, verfügen also über keinen Formater.
2. Es können nur Zahlen, keine Buchstaben und keine anderen Zeichen eingegeben werden (Ausnahme dabei ist das „e“ für Engineering-Eingaben).
3. Die Fehlermeldung direkt im Textfeld wurde deaktiviert. Fehler werden in diesem Tool auf einer extra Zeile dargestellt.

Fenstermanagement

Wenn das Tool gestartet wird, öffnet sich das Programm in der Normalansicht (alle Panels werden angezeigt) mit der kleinstmöglichen Fenstergrösse. Dazu wird das Frame so gepackt, dass alle Komponenten mit ihrer Minimalgrösse platziert werden. Um das Tool auch bei unterschiedlichen Bildschirmauflösungen verwenden zu können, wurden dafür keine absoluten Pixelwerte eingesetzt.

Wenn die Fenstergrösse durch den Anwender verändert wird, hat dies einen unterschiedlichen Einfluss auf die einzelnen Panels. Das *OutputPanel* wird nur in der Vertikalen und das *GraphDisplayPanel* und die *MenuBar* nur in der Horizontalen verändert. Weiter wird das *GraphPanel* in beide Richtungen gestreckt oder gestaucht. Das *InputPanel* und das *GraphSettingPanel* bleiben dabei unverändert. Zur Veranschaulichung wurde dieses Verhalten in der folgenden Grafik dargestellt. Die roten Pfeile zeigen dabei, in welche Richtungen sich die Panels verändern können.

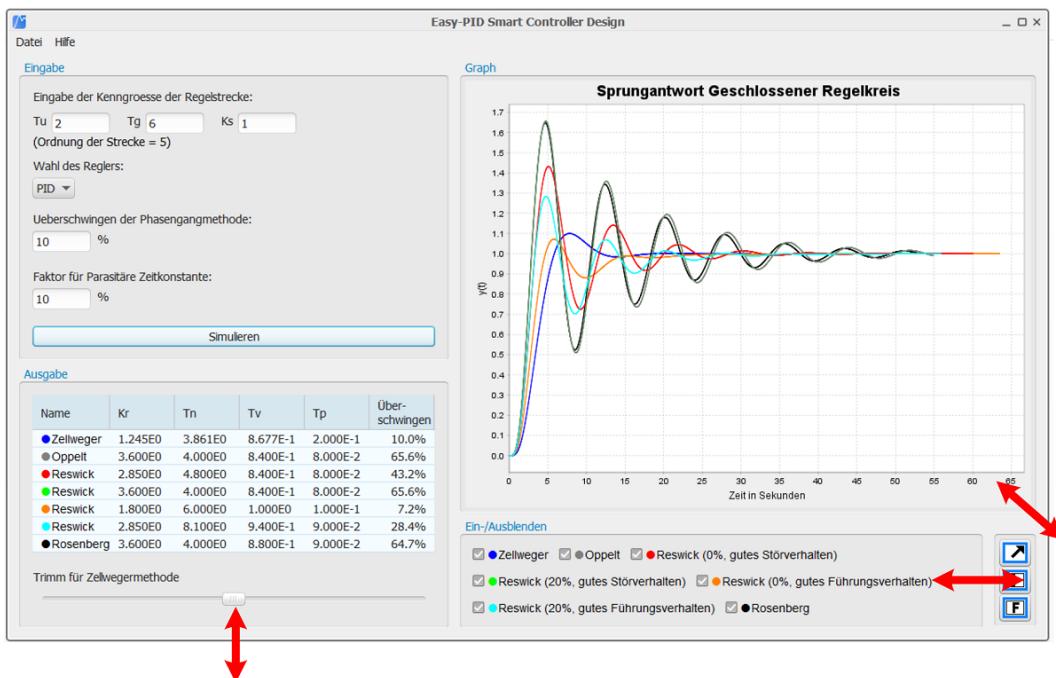


Abbildung 27: Verändern der Fenstergrösse

Mini-Version

Durch eine Schaltfläche in der *MenuBar* kann zur Mini-Version gewechselt werden. Diese reduzierte Ansicht verzichtet auf den Graphen und die dazugehörigen Panels und wurde speziell für fortgeschrittene Anwender implementiert. Die folgende Grafik zeigt die Mini-Version des Tools.

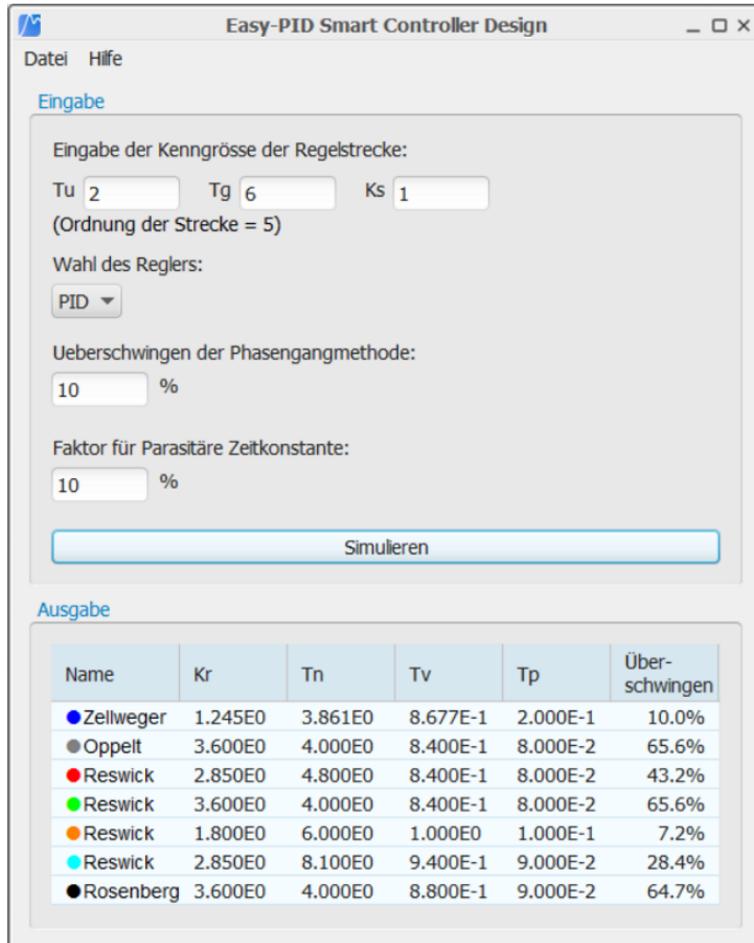


Abbildung 28: Ansicht der Mini-Version

3.3 Controller

Der *Controller* ist sehr schlank gehalten und ist ein Teil des Model-View-Controller-Pattern. Der Controller hat dabei mehrere Aufgaben:

- Beim Drücken der Simulationsschaltfläche im *InputPanel* werden die entsprechenden Methoden des Models aufgerufen, um die neuen Regler mit den neuen Eingabeparametern zu berechnen, die alten Simulationen zu löschen und neue Simulationen zu starten.
- Wenn der Trimm-Slider im *OutputPanel* verändert wird, so wird die entsprechende Methode im Model aufgerufen.
- Wenn der Anwender eine Checkbox im *GraphDisplayPanel* anklickt, wird die entsprechende Methode im Model aufgerufen, um eine Kurve ein- oder auszublenden.

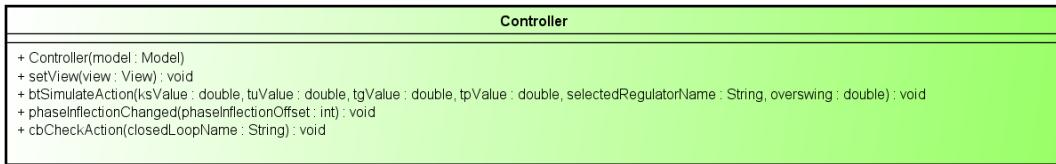


Abbildung 29: Klasse Controller

3.4 Ablauf einer Reglerberechnung (Use-Case)

Im folgenden Abschnitt wird die Dimensionierung eines Reglers mit Easy-PID an einem Beispiel erklärt. Als Beispiel dient eine Regelstrecke mit den Kenngrößen $Tu = 2$, $Tg = 6$ und $Ks = 1$, die parasitäre Zeitkonstante Tp wird bei 10% belassen, was der Standardeinstellung entspricht. Gewünscht ist ein PID-Regler mit einem maximalen Überschwingen von 8%.

1.

Die Werte für Tu , Tg und Ks werden in den entsprechenden *JFormattedDoubleTextField* eingegeben, für Tp muss nichts eingegeben werden, da 10% bereits die Standardeinstellung ist. Mit dem Dropdown-Menü, welches mittels *JComboBox* implementiert wurde, wird der Reglertyp PID ausgewählt und im folgenden Textfeld das Überschwingen auf 8% festgelegt.

2.

Durch Drücken der Schaltfläche *Simulieren* wird die Methode *InputPanel::actionPerformed()* aufgerufen, welche die Eingabeparameter in die entsprechenden Attribute speichert. Die Eingabeparameter werden im *View* saniert, damit ungültige Einstellung möglichst früh erkannt werden können. Ist alles in Ordnung, wird die Methode *Controller::btSimulateAction* aufgerufen. Bei nicht akzeptierten Eingabeparametern erscheint unter den Eingabefeldern ein *JLabel* mit der entsprechenden Fehlermeldung. Bei akzeptierten Eingabeparametern wird die Simulation gestartet.

3.

Der *Controller* konfiguriert das *Model*-Objekt mit den Eingabewerten mittels der Methoden *Model::setRegulatorType()* - welche den Typ zu *I*, *PI* oder *PID* setzt, *Model::setPlant()* - welche ein neues *Plant*-Objekt erstellt und als Attribut der Klasse *Model* speichert, *Model::setParasiticTimeConstantFactor()* und *Model::setOvershoot()*. Danach ruft sie die Methode *Model::simulateAll()* auf, um die Simulation zu beginnen.

4.

Das *Model* baut sich mit Hilfe seines *Plant*-Objekts, welches die vom Benutzer eingegebene Strecke beschreibt, eine Liste von *CalculationCycle*-Objekten. Diese Objekte sind dazu fähig, eine gesamte Berechnung von der Reglerberechnung bis zur Schrittantwort durchzuführen. Welche Regler ausgerechnet werden ist abhängig von der Auswahl des Reglertyps, also *I*, *PI* oder *PID*.

Es wird mittels *Model::notifySimulationBegin()* allen Listener mitgeteilt, dass eine Simulation beginnt. Dies bewirkt unter anderem, dass sich die verschiedene Panels auf die Resultate vorbereiten können, dies umfasst beispielsweise die Tabelle oder den Plot zu löschen.

Die *CalculationCycle*-Klasse erbt von *Runnable*. Es wird ein *ThreadPool* erstellt und alle *CalculationCycle*-Objekte werden parallel ausgeführt. Das Programm wartet, bis alle Berechnungen vollendet sind.

5.

Zu diesem Zeitpunkt teilt sich der Programmfluss in einer Unmenge kleiner Teilchen auf. Wir verfolgen nur einen Pfad, nämlich die Berechnung eines PID-Reglers mittels Zellweger Methode.

Beim Instanziieren der Klasse *CalculationCycle* wird ein *AbstractControllerCalculator* übergeben. In diesem Fall ist das ein Objekt, dass als konkrete Klasse den *ZellwegerPID* hat. Dieses Objekt wurde schon vom *Model* konfiguriert und es muss nur *AbstractControllerCalculator::run()* aufgerufen werden, um einen passenden Regler für die vom Benutzer definierte Strecke zu berechnen.

6.

Die *run()*-Methode führt dazu, dass die Methode *ZellwegerPID::calculate()* ausgeführt wird.

Der implementierte Algorithmus von Herrn Zellweger wird in dieser Methode gebraucht, um ein neues *ControllerPID*-Objekt zu erstellen. Dieses Objekt beschreibt den Regler Anhand der Reglerparametern T_n , T_v , K_r und T_p sowie einer Übertragungsfunktion mit der Klasse *TransferFunction*. Weiter berechnet das *ZellwegerPID*-Objekt einen Minimal- und Maximalwert für K_r aus, was später für das Approximieren des Überschwingens gebraucht wird.

7.

Der Regler ist berechnet und kann in der *CalculationCycle* mit *AbstractControllerCalculator::getRegulator()* als *AbstractController* geholt werden. Der Regler wird zusammen mit dem *Plant*-Objekt zu einem *ClosedLoop*-Objekt zusammengefügt. Das *ClosedLoop*-Objekt berechnet aus der Strecke und dem Regler seine eigene *TransferFunction* (Übertragungsfunktion).

8.

In der *CalculationCycle* wird nun die Schrittantwort mit der Methode *ClosedLoop::calculateStepResponse()* ausgerechnet. Diese Methode Berechnet mittels Partialbruchzerlegung die Schrittantwort und misst auch gleich das maximale Überschwingen. Wenn der *AbstractController* ein Minimal- und Maximalwert für K_r ausgerechnet hat, was bei allen Zellweger Reglern der Fall ist, dann wird die Schrittantwort mehrmals ausgerechnet, bis das Überschwingen, welches mit der Methode *ClosedLoop::getOvershoot()* geholt werden kann, mit dem vom Benutzer definierten Wert übereinstimmt.

9.

Ist die *CalculationCycle* abgeschlossen, wird mittels der Methode *CalculationCycle::notifyStepResponseCalculationComplete()* allen Listener mitgeteilt, dass die Berechnung einer Schrittantwort fertig ist. Dabei werden die Reglerparameter K_r , T_n , T_v , T_p und das Überschwingen in einer Tabelle im *OutputPanel* eingetragen und die Schrittantwort wird in einem 2D-Plot auf der rechten Seite des GUIs in der Klasse *GraphPanel* gezeichnet.

10.

Wenn der Threadpool keine Jobs mehr hat, dann wird allen Listener mittels der Methode *Model::notifySimulationComplete()* mitgeteilt, dass alle Berechnungen beendet sind.

11.

Im *GraphDisplayPanel* kann mittels der Checkboxen ausgewählt werden, welche Schrittantworten visualisiert werden können, um so einen möglichst passenden provisorischen Regler zu finden. In unserem Beispiel ist dies der Graph des Reglers, welcher mit der Phasengang-Methode dimensioniert wurde.

12.

Mittels eines *JSliders* kann die Phasengangmethode nachträglich optimiert werden. Dazu wird die Methode *OutputPanel::stateChanged* aufgerufen, welche *Controller::angleOfInflectionChanged* aufruft. Anschliessend wird die Methode *Model::updateZellweger* aufgerufen. Diese Methode verändert den Offset des Phasenrandes und simuliert den Regler anschliessend neu.

4 Validierung

Im folgenden Abschnitt wird die Validierung beschrieben. Die Validierung dient zum Auffinden und Beheben von Fehlern, um so eine korrekte Funktion des Programms garantieren zu können. Zum einen wurden die Berechnungen und Simulationen von Matlab überprüft, zum anderen die Software, aufgeteilt in die Teilbereiche Benutzeroberfläche und Berechnungen.

4.1 Matlab

4.1.1 Programmierung

Die Berechnung der Regler wurde zuerst mittels m-Files in Matlab implementiert. Zur Überprüfung der von Matlab gelieferten Werte wurden diese mit den Werten vom Fachcoach Herr Prof. Peter Niklaus verglichen.

4.1.2 Simulation

Mittels Simulink und Matlab wurde der geschlossene Regelkreis simuliert.

4.2 Java Software

4.2.1 Benutzeroberfläche

Das Java-Tool wurde jeweils nach der Implementation eines Elementes auf richtige Funktionsweise geprüft. Weiter wurde das Programm am Ende durch die Teammitglieder getestet. Neben gewöhnlichen Eingaben wurde versucht, die Eingabefelder mittels Sonderzeichen und speziellen Werten zu überlisten. Die Resultierenden Erkenntnisse flossen anschliessend in die Endversion des Java-Tools mit ein. Für eine zusätzliche Überprüfung wurde eine Testversion an den Auftraggeber gesendet.

4.2.2 Model

Die Plausibilität der Schrittantworten konnte relativ einfach optisch mit den Graphen überprüft werden, da das Verhalten der einzelnen Regler bekannt ist. Zur genauen Überprüfung der einzelnen Klassen wurden sämtliche Klassen des Models mittels JUnit mit den Ergebnissen von Matlab verglichen.

4.2.3 Performancevergleich von Partialbruchzerlegung und IFFT

Es wurde festgestellt, dass das Berechnen der Schrittantwort mittels IFFT-Methode langsam ist, vor allem dann, als das iterative Approximieren des Überschwingens implementiert wurde, was dazu führte, dass die Schrittantwortberechnung mehrmals ausgeführt werden musste.

Durch Reduzieren der Anzahl Punkte kann die Berechnung beschleunigt werden, jedoch geht dies auf Kosten von Genauigkeit, was anhand des immer zunehmenden Unterschwingens in der Figur 30 zu sehen ist. Weiter sind die Anzahl Punkte bei der IFFT-Methode auf Zahlen der Basis 2 limitiert, was eine genaue Optimierung der Anzahl Punkte unmöglich macht. Durch Experimentieren stellten wir fest, dass ein Minimum von 4096 Punkten für ein visuell genaues Resultat gebraucht wird. Dabei konnten wir auf 2048 Punkte für das iterative Approximieren des Überschwingens zurückgehen. Die Berechnungszeit war aber immer noch merkbar langsam.

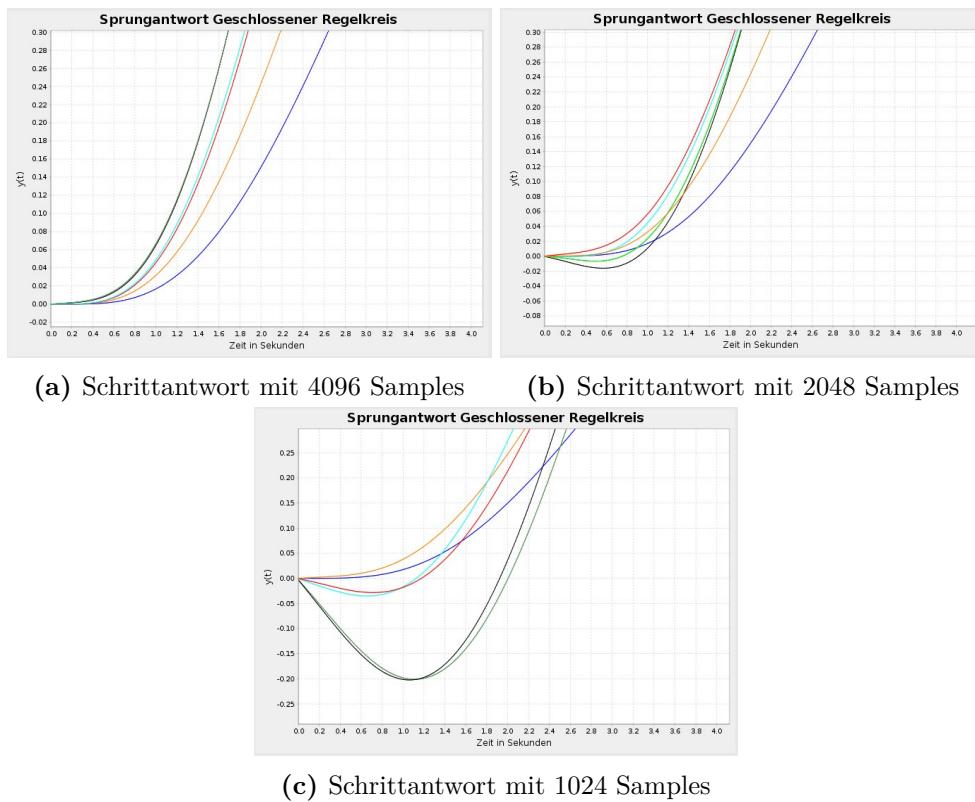


Abbildung 30: Vergleich einer IFFT Schrittantwort mit verschiedener Anzahl Punkte

Es wurde entschieden, eine alternative Berechnungsmethode zu implementieren: Die Schrittantwortberechnung mittels Partialbruchzerlegung.

Um die Berechnungsgeschwindigkeit beider Methoden messen und vergleichen zu können, wurden mehrere Tests durchgeführt, welche hier im Detail dargelegt werden.

Die Berechnungszeit wurde mit der Methode `System.currentTimeMillis()` gemessen. Weil die Messresultate stark variieren, wurde die Messung mehrmals mittels einer For-Schleife ausgeführt, wie im Java-Code in der Figur 31 zu sehen ist.

```
for(int iteration = 0; iteration < 20; iteration++) {
    long time = System.currentTimeMillis();
    model.simulateAll();
    System.out.println("[" + (System.currentTimeMillis() - time));
}
```

Abbildung 31: Zeitmessung der Methode `Model::simulateAll()`

Wichtig zu bemerken ist, dass das Programm für jede Messung neu gestartet wurde, damit die *Java Virtual Machine* (JVM) "frisch" bleibt.

Aus Gründen der Reproduzierbarkeit und/oder Vergleichsmöglichkeit wurden die Messungen auf einem System mit folgenden Eigenschaften ausgeführt:

- **OS:** Linux twilight 3.18.12-gentoo
- **Arch:** x86_64
- **CPU:** AMD Phenom(tm) II X6 1090T Processor, 4.2GHz
- **RAM:** 4 · 2GB Dual channel, 1.6GHz
- **Java:** Oracle JDK 1.8.0.45, Java HotSpot(TM) 64-Bit Server VM

Als erstes wurden beide Methoden mit fix 2048 Punkten verglichen. Das sind bei der IFFT-Methode das Minimum, das gebraucht wird, um ein relativ genaues Überschwingen der Zellweger Methode approximieren zu können; deswegen die Entscheidung, mit 2048 Punkten zu testen. Wie in der Figur 32 zu sehen ist, leistet die Methode mit Partialbruchzerlegung das gleiche Resultat innert halber Zeit.

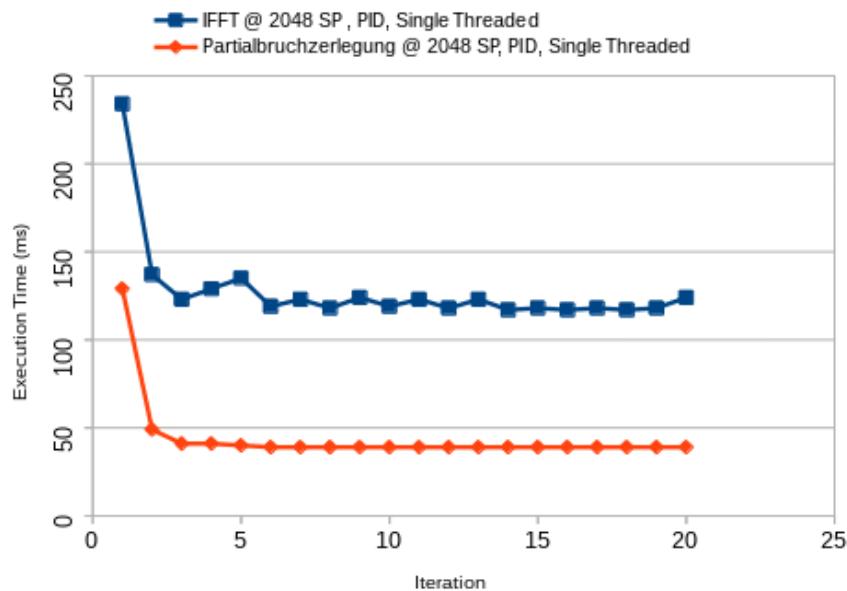


Abbildung 32: IFFT vs Partialbruchzerlegung mit fix 2048 Samples und keinem Threadpool. Partialbruchzerlegung ist etwa zweimal schneller.

Ein Nachteil der IFFT Methode ist, dass ihre Komplexität $\mathcal{O}(n \log n)$ skaliert. Die Partialbruchzerlegungsmethode hingegen skaliert nur linear mit $\mathcal{O}(n)$. Dies ist gut ersichtlich, wenn man Figur 32 und 33 vergleicht.

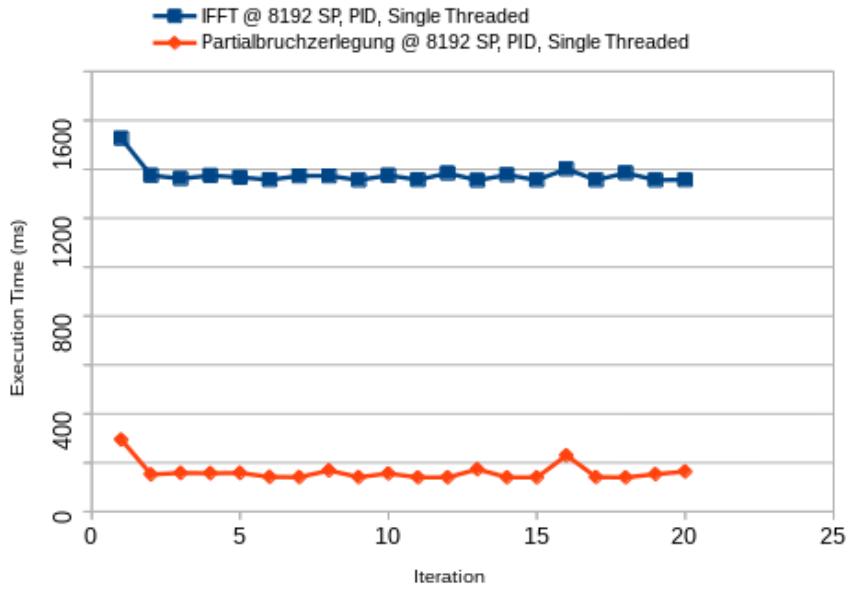


Abbildung 33: IFFT vs Partialbruchzerlegung mit 8192 Samples und keinem Threadpool. Berechnungszeit von IFFT skaliert quadratisch, Partialbruchzerlegung nur linear.

In der Realität braucht die Partialbruchzerlegungsmethode aber viel weniger als 2048 Punkte. Je nachdem was für eine Strecke vorliegt, ändern sich die Anzahl Punkte. Mit einer Strecke von $T_u = 2$ und $T_g = 6$ wurden in der Figur 34 nur 145 Punkte gebraucht, was zu einer Berechnungszeit von etwa 8ms führte. Die Berechnung kann mittels Multithreading weiter beschleunigt werden. In diesem Beispiel konnte sie auf weniger als 1ms reduziert werden, wie in Figur 34 auch zu sehen ist. Man beachte, dass nur 7 Berechnungen parallel liefen. Das heisst, es ist nicht auszuschliessen, dass Multithreading noch schneller sein könnte, wenn mehr Berechnungen vorhanden wären.

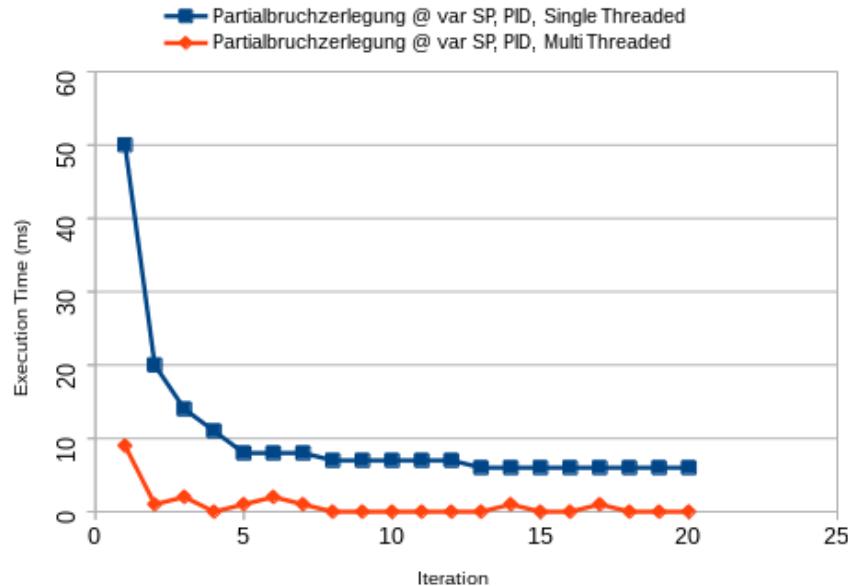


Abbildung 34: Single Threaded vs Multi Threaded: Partialbruchzerlegung mit variabler Anzahl Samples

4.2.4 JUnit-Tests

Unit Testing ist in der Softwareentwicklung ein extrem hilfreiches Verfahren, das dem Programmierer erlaubt, individuelle Methoden oder Funktionen isoliert vom Rest des Programms zu testen. Dabei kann für jede Methode ein *Test Case* erstellt werden, indem ein Eingabewert und eine Erwartung deklariert werden kann. Das *Unit Testing Framework* - zum Beispiel *JUnit* - führt alle *Test Cases* aus und überprüft, dass die Ausgaben den erwarteten Werten entsprechen. Ist dies nicht der Fall, wird ein Fehlermeldung generiert.

Der Programmierer kann somit sehr einfach und vor allem schnell schwierige Methoden auf ihre Richtigkeit überprüfen. Der Vorteil von *Unit Testing* liegt darin, dass Methoden nicht nur auf "richtige" Eingaben geprüft werden können, sondern auch auf unmögliche, vielleicht sogar maliziöse Eingaben getestet werden können. Sind die Tests robust, so kann davon ausgegangen werden, dass der Rest des Programms auch robust sein wird.

In den folgenden Bereichen wurde *Unit Testing* anhand des Frameworks *JUnit4* verwendet:

- **ClosedLoop:** Die Schrittantwort wurde direkt mit dem Resultat von Matlab verglichen. Dabei wurde die Kurve von Matlab exportiert und als Liste von XY-Punkte in Java importiert und mit dem Resultat von *ClosedLoop::calculateStepRespose()* verglichen.
- **In der Klasse MathStuff:** Alle Mathematikfunktionen, die von Matlab portiert wurden, sowie zusätzliche Mathematikfunktionen, die im Laufe des Projekts hinzugefügt wurden.
- **Faustformeln:** Die von Hand ausgerechnete Resultate wurden mit allen Faustformel-Klassen verglichen.
- **Regler:** Alle Reglerklassen - *ControllerI*, *ControllerPI* und *ControllerPID* - sind mit Matlab verglichen.
- **Strecke:** Die Strecke und deren Übertragungsfunktion wurde mit Matlab verglichen.
- **Sani:** Die Sani-Kurven wurden mit Matlab verglichen.

5 Schlussfolgerung

Easy-PID konnte durchaus erfolgreich abgeschlossen werden. Das Tool erfüllt dabei sämtliche Anforderungen, welche an ein solches Tool gestellt werden. Einige weitere Funktionen, namentlich die Mini-Version, die Simulation der geschlossenen Regelstrecke und der Export der Berechnung als PDF wurden als Zusatzfunktionen im Programm implementiert. Eine Nachjustierungsmöglichkeit für die Reglerparameter wurde dabei bewusst nicht umgesetzt, jedoch kann mittels eines Schiebereglers die Phasengangmethode optimiert werden.

Der Benutzer kann die Parameter der Regelstrecke in der grafischen Benutzeroberfläche eingeben, woraufhin Easy-PID die Reglerparameter mittels verschiedener Methoden berechnet und die entsprechenden Sprungantworten grafisch darstellt. Die einzelnen Methoden können dabei auch ausgeblendet werden. Für die Phasengangmethode existiert außerdem ein Trimm-Regler, mit welchem die Phasengangmethode angepasst werden kann. Mit diesen Funktionen ermöglicht es Easy-PID dem Benutzer, den idealen Regler zu dimensionieren.

Das Projekt ist nicht ohne Probleme verlaufen: Das zu Beginn des Projektes aufgestellte Klassendiagramm konnte beispielsweise nicht implementiert werden, da gewisse Aufgaben noch nicht erkannt wurden. Deswegen musste im späteren Projektverlauf ein neues Klassendiagramm erstellt werden, welches sämtliche Aufgaben und Funktionen beachtet und somit gut umgesetzt werden konnte.

Ein weiteres Problem war die Umsetzung von Matlab-Code in Java. Viele Funktionen, die in Matlab bereits vorimplementiert sind, mussten für das Tool neu programmiert werden. Auch haben die vielen kleinen Unterschiede zwischen den beiden Programmiersprachen immer wieder zu Problemen geführt.

Am Ende konnte das Projekt jedoch termingerecht und korrekt funktionierend abgeliefert werden. Trotzdem gibt es viele Features, welche in zukünftigen Projekten noch realisiert werden könnten. Eine Möglichkeit wäre das Importieren und Exportieren von Projekten. Dank dem Model-View-Controller Entwurfsmuster kann Easy-PID auch problemlos für eine Mobile-App adaptiert werden.

Literatur

- [1] Bild Regelkreis. Zugriff am 20.05.2015. [Online]. Available: http://de.wikipedia.org/wiki/Regelkreis#/media/File:Einfacher_Regelkreis_n.svg
- [2] Wendetangente Auftraggeber. Zugriff am 20.05.2015. [Online]. Available: [http://fsemu18.edu.ds.fhnw.ch/e_18_data11\\$/E1861_Unterrichte_EIT/E1861_2Ea/pro2E/Aufgabenstellung_vom_Auftraggeber](http://fsemu18.edu.ds.fhnw.ch/e_18_data11$/E1861_Unterrichte_EIT/E1861_2Ea/pro2E/Aufgabenstellung_vom_Auftraggeber)
- [3] J. Zellweger, "Phasengang Methode," unveröffentl. Vorlesungsskript, ohne Jahr, S. x-y.
- [4] C quell code. Zugriff am 07.05.2015. [Online]. Available: <http://www.pcs.cnu.edu/~bbradie/cpp/interp.C>
- [5] Layout manager guide. Oracle. Zugriff am 01.06.2015. [Online]. Available: <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>
- [6] Rob Camick. (2008, 11) Wraplayout. Zugriff am 10.05.2015. [Online]. Available: <https://tips4java.wordpress.com/2008/11/06/wrap-layout/>

A Anhang Elektrotechnik

A.1 Herleitung der Berechnung von β für die Zellweger Methode

Für den offenen Regelkreis Go gilt:

$$\begin{aligned}\phi_{RE} &\text{ Phase Regler} \\ \phi_S &\text{ Phase Strecke} \\ \phi_O &\text{ Phase offener Regelkreis} \\ \phi_{RE}(w_{PID}) + \phi_S(w_{PID}) &= \phi_O(w_{PID})\end{aligned}$$

Für die Steigung (Ableitung) gilt:

$$\frac{d\phi_{RE}(w_{PID})}{dw} + \frac{d\phi_S(w_{PID})}{dw} = \frac{d\phi_O(w_{PID})}{dw} \quad (11)$$

Der offene Regelkreis berechnet sich wie folgt:

$$Go = \frac{1}{s \cdot T0 \cdot (1 + s \cdot T1)}$$

Die Phase von O berechnet sich wie folgt:

$$\begin{aligned}Tk &= \frac{1}{w_{PID}} \\ \phi_O(w_{PID}) &= -\frac{\pi}{2} - \arctan\left(w \cdot \frac{1}{w_{PID}}\right) = -\frac{pi}{2} - \arctan(w \cdot Tk)\end{aligned}$$

ϕ_O wird nach w abgeleitet und für Tk wird $\frac{1}{w_{PID}}$ eingesetzt. Da in w_{PID} abgeleitet wurde wird w gleich w_{PID} gesetzt:

$$\frac{d\phi_O(w_{PID})}{dw} = -\frac{Tk}{1 + w^2 \cdot Tk^2} = -\frac{Tk}{1 + w_{PID}^2 \cdot \left(\frac{1}{w_{PID}}\right)^2} = -\frac{1}{2}Tk = -\frac{1}{2 \cdot w_{PID}} \quad (12)$$

Aus der Ableitung der Phase des offenen Regelkreises (12) und der Phasengleichung (11) des offenen Regelkreises folgt:

$$\phi_{RE}(w_{PID}) + \phi_S(w_{PID}) = -\frac{1}{2 \cdot w_{PID}}$$

Mit w_{PID} multipliziert

$$w_{PID} \cdot \frac{d\phi_{RE}(w_{PID})}{dw} + w_{PID} \cdot \frac{d\phi_S(w_{PID})}{dw} = -0.5$$

Wie man einfach erkennt folgt daraus:

$$w_{PID} \cdot \frac{d\phi_{RE}(w_{PID})}{dw} = \frac{2 \cdot \beta}{1 + \beta^2}$$

Eingesetzt in die Phasengleichung:

$$\frac{2 \cdot \beta}{1 + \beta^2} + w_{PID} \cdot \frac{d\phi_S(w_{PID})}{dw} = -0.5$$

Für $\phi_S(w_{PID})$ wird eingesetzt:

$$\frac{2 \cdot \beta}{1 + \beta^2} + w_{PID} \cdot (-\arctan(w_{PID} \cdot T1) - \arctan(w_{PID} \cdot T2) - \arctan(w_{PID} \cdot Tz)) = -0.5$$

Als Summe geschrieben:

$$\frac{2 \cdot \beta}{1 + \beta^2} - w_{PID} \cdot \sum_{m=1}^N \arctan(w_{PID} \cdot T_m) = -0.5$$

$\phi_S(w_{PID})$ abgeleitet:

$$\frac{2 \cdot \beta}{1 + \beta^2} - w_{PID} \cdot \left(\frac{T_1}{1 + (w_{PID} \cdot T_2)^2} + \frac{T_2}{1 + (w_{PID} \cdot T_2)^2} + \frac{T_m}{1 + (w_{PID} \cdot T_m)^2} \right) = -0.5$$

Als Summe geschrieben:

$$\frac{2 \cdot \beta}{1 + \beta^2} - w_{PID} \cdot \sum_{m=1}^N \frac{T_m}{1 + (w_{PID} \cdot T_m)^2} = -0.5$$

Nun wird so umgeformt, dass alle β auf einer Seite stehen:

$$\frac{2 \cdot \beta}{1 + \beta^2} = w_{PID} \cdot \sum_{m=1}^N \frac{T_m}{1 + (w_{PID} \cdot T_m)^2} - 0.5$$

Die rechte Seite wird mit Z substituiert:

$$Z = w_{PID} \cdot \sum_{m=1}^N \frac{T_m}{1 + (w_{PID} \cdot T_m)^2} - 0.5$$

$$Z = \frac{2 \cdot \beta}{1 + \beta^2}$$

Nach β aufgelöst:

$$\beta = \frac{1}{Z} - \sqrt{\frac{1}{Z^2} - 1}$$

A.2 Herleitung der Bodekonformen und Reglerkonformen Darstellung

Regler können in verschiedenen Darstellungen abgebildet werden. Unterschieden werden hier die reglerkonforme und die bodekonforme Darstellung, wobei oftmals standartmäßig die reglerkonforme Darstellung gewählt wird. Folgend sind beide Darstellungen von PI- und PID-Regler aufgeführt. Beim PID-Regler wird zudem die Beziehung zwischen den beiden Formen hergeleitet.

A.2.1 PI-Regler

Der PI-Regler hat die Parameter Kr und Tn .

Reglerkonform:

$$G_R(s) = Kr \left(1 + \frac{1}{s \cdot Tn} \right) \quad (13)$$

Bodekonform:

$$G_R(s) = Kr \left(\frac{1 + s \cdot Tn}{s \cdot Tn} \right) \quad (14)$$

A.2.2 PID-Regler

Reglerkonform:

Der PID-Regler hat in der reglerkonformen Darstellung die Parameter Kr , Tn , Tv und Tp .

$$G_R(s) = Kr \left(1 + \frac{1}{s \cdot Tn} + \frac{s \cdot Tv}{1 + s \cdot Tp} \right) \quad (15)$$

Umformung:

$$\begin{aligned} G_R(s) &= Kr \left(\frac{s \cdot Tn(1 + s \cdot Tp) + (1 + s \cdot Tp)(s \cdot Tn \cdot s \cdot Tv)}{s \cdot Tn(1 + s \cdot Tp)} \right) \\ &= Kr \left(\frac{s^2 \cdot Tn \cdot Tp + s^2 \cdot Tn \cdot Tv + s \cdot Tp + s \cdot Tn + 1}{s \cdot Tn(1 + s \cdot Tp)} \right) \\ &= Kr \left(\frac{s^2(Tn \cdot Tp + Tn \cdot Tv) + s(Tp + Tn) + 1}{s \cdot Tn(1 + s \cdot Tp)} \right) \\ &= Kr \left(\frac{s^2 \cdot Tn(Tp + Tv) + s(Tp + Tn) + 1}{s \cdot Tn(1 + s \cdot Tp)} \right) \end{aligned} \quad (16)$$

Bodekonform:

Der PID-Regler hat in der bodekonformen Darstellung die Parameter Krk , Tn , Tv und Tp .

$$G_R(s) = Krk \left(\frac{(1 + s \cdot Tnk)(1 + s \cdot Tvk)}{s \cdot Tnk(1 + s \cdot Tp)} \right) \quad (17)$$

Umformung:

$$G_R(s) = Krk \left(\frac{s^2(Tnk \cdot Tvk) + s(Tnk + Tnk) + 1}{s \cdot Tnk(1 + s \cdot Tp)} \right) \quad (18)$$

Koeffizienten-Vergleich:

Damit ein Koeffizienten-Vergleich zwischen der reglerkonformen und der bodekonformen Darstellung des PID-Reglers durchgeführt werden kann, müssen die beiden Formen gleichgesetzt werden (links reglerkonform, rechts bodekonform):

$$K_R \left(\frac{s^2 \cdot Tn(Tp + Tv) + s(Tp + Tn) + 1}{s \cdot Tn(1 + s \cdot Tp)} \right) = K_{RK} \left(\frac{s^2(Tnk \cdot Tvk) + s(Tnk + Tnk) + 1}{s \cdot Tnk(1 + s \cdot Tp)} \right)$$

s wird auf beiden Seiten im Nenner gekürzt:

$$K_R \left(\frac{s^2 \cdot Tn(Tp + Tv) + s(Tp + Tn) + 1}{Tn(1 + s \cdot Tp)} \right) = K_{RK} \left(\frac{s^2(Tnk \cdot Tvk) + s(Tnk + Tnk) + 1}{Tnk(1 + s \cdot Tp)} \right)$$

Lässt man die Frequenz w gegen Null gehen, strebt auch s gegen Null und die Terme lassen sich wie folgt vereinfachen:

$$K_R \left(\frac{1}{Tn} \right) = K_{RK} \left(\frac{1}{Tnk} \right)$$

An der oben stehenden Umformung erkennt man, dass man einen Koeffizientenvergleich machen kann. Folgend werden zuerst die Koeffizienten mit s , danach die s^2 und am Ende die Koeffizienten ohne s verglichen und nach den Parameter der reglerkonformen Darstellung umgeformt.

2. Koeffizient

$$\begin{aligned} s(Tn + Tp) &= s(Tnk + Tvk) \\ Tn + Tp &= Tnk + Tvk \\ Tn &= Tnk + Tvk - Tp \end{aligned} \tag{19}$$

Damit ist Tn bestimmt.

1. Koeffizient:

$$\begin{aligned} s^2(Tn \cdot Tp + Tn \cdot Tv) &= s^2(Tnk \cdot Tvk) \\ (Tn \cdot Tp + Tn \cdot Tv) &= (Tnk \cdot Tvk) \\ Tn(Tv + Tp) &= Tnk \cdot Tvk \end{aligned} \tag{20}$$

Obenstehende Gleichung wird nun nach Tv umgeformt:

$$\begin{aligned} Tv + Tp &= \frac{Tnk \cdot Tvk}{Tn} \\ Tv &= \frac{Tnk \cdot Tvk}{Tn} - Tp \end{aligned}$$

Die Variable Tn wird in der folgenden Gleichung durch die Beziehung von Formel 19 ersetzt:

$$Tv = \frac{Tnk \cdot Tvk}{Tnk + Tvk - Tp} - Tp \tag{21}$$

Damit ist Tv bestimmt. Kr wird wie folgt bestimmt:

3. Koeffizient:

$$\frac{Kr}{s \cdot Tn(1 + s \cdot Tp)} = \frac{Krk}{s \cdot Tnk(1 + s \cdot Tp)}$$

s und $(1 + s \cdot Tp)$ werden gekürzt und Tn ersetzt:

$$\frac{Kr}{Tnk + Tvk - Tp} = \frac{Krk}{Tnk}$$

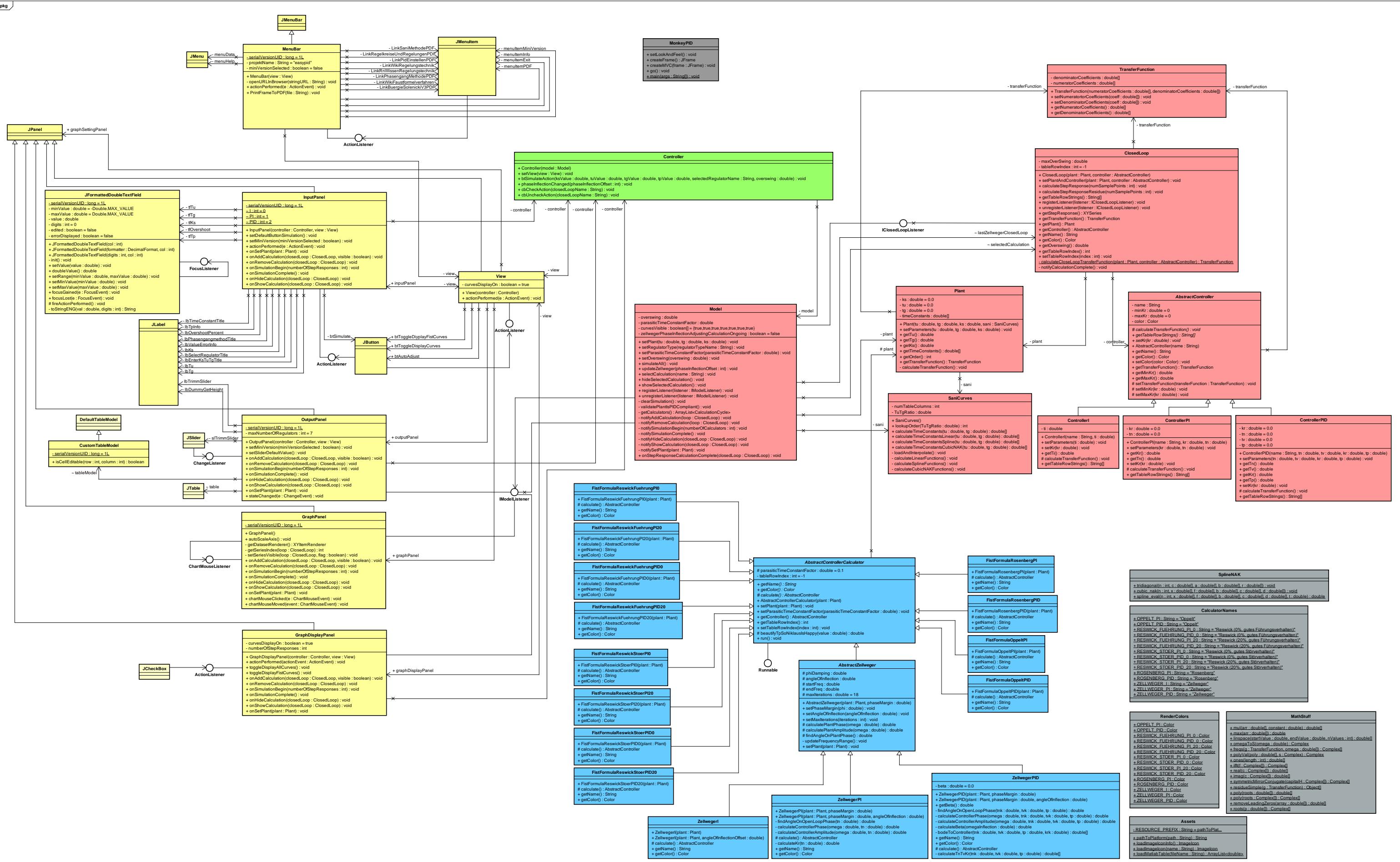
$$Kr = \frac{Krk(Tnk + Tvk - Tp)}{Tnk}$$

Das im Verhältnis zu Tvk ca. zehn mal kleinere Tp kann vernachlässigt werden:

$$Kr = \frac{Krk(Tnk + Tvk)}{Tnk}$$

$$Kr = \frac{Krk(1 + Tvk)}{Tnk} \quad (22)$$

B Klassendiagramm



C CD-Rom