**Regelung eines Hebelarms**

**Freiwilliges praktisches Projekt  
Systemtheorie II**

**Lösung WS24/25**

Die vorliegende Arbeit wurde am 16.12.2024 eingereicht bei:

RWTH Aachen University

Faculty of Electrical Engineering and Information Technology

Institute for Automation of Complex Power Systems

Univ.-Prof. Antonello Monti, Ph. D.

|  |  |  |
| --- | --- | --- |
| Gruppenmitglied | Matrikelnummer | (Digitale) Unterschrift |
| Jan Rothaug | 123456 | Jan Rothaug |
| Valentin Rentschler | 443229 | Valentin Rentschler |
| Antoneta Gjeka | 444168 | Antoneta Gjeka |
| Ruyuan Wang | 443897 | Ruyuan Wang |
| Haitian Liu | 444603 | Haitian Liu |

Durch Unterschrift versichern sich die Teilnehmerinnen und Teilnehmer an Eides Statt, dass das vorliegende Dokument selbstständig und ohne unzulässige fremde Hilfe erarbeitet wurde. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde in gleicher oder ähnlicher Form noch nicht zur Bewertung am Lehrstuhl für Automation of Complex Power Systems eingereicht.

Inhaltsverzeichnis

[Abbildungsverzeichnis ii](#_Toc117699631)

[Tabellenverzeichnis iii](#_Toc117699632)

[1 Aufgabe 1: Aufbau und grundlegende Motorsteuerung 1](#_Toc117699633)

[1.1 Aufgabe 1.1 1](#_Toc117699634)

[1.2 Aufgabe 1.2 1](#_Toc117699635)

[1.3 Aufgabe 1.3 2](#_Toc117699636)

[2 Aufgabe 2: Positionsregelung des Hebelarms 3](#_Toc117699637)

[2.1 Aufgabe 2.1 3](#_Toc117699638)

[2.2 Aufgabe 2.2 3](#_Toc117699639)

[2.3 Aufgabe 2.3 4](#_Toc117699640)

[2.4 Aufgabe 2.4 4](#_Toc117699641)

[3 Aufgabe 3: Positionsregelung 5](#_Toc117699642)

[3.1 Aufgabe 3.1 5](#_Toc117699643)

[3.2 Aufgabe 3.2 7](#_Toc117699644)

[4 Aufgabe 4: Kompensation einer Störung 8](#_Toc117699645)

# Abbildungsverzeichnis

[Abbildung 1‑1: Screenshot Positionssignal 1](#_Toc118096376)

[Abbildung 1‑2: Screenshot Positionssignal **Error! Bookmark not defined.**](#_Toc118096377)

[Abbildung 1‑3: Screenshot Positionssignal 2](#_Toc118096378)

[Abbildung 2‑1: Screenshot Positionssignal und Ableitung des Positionssignals 3](#_Toc118096379)

[Abbildung 3‑1: Screenshot Verstärkungsfaktor 1 6](#_Toc118096380)

[Abbildung 3‑2: Screenshot Verstärkungsfaktor 2 6](#_Toc118096381)

[Abbildung 3‑3: Screenshot Verstärkungsfaktor 3 6](#_Toc118096382)

[Abbildung 3‑4: Screenshot Verstärkungsfaktor 4 6](#_Toc118096383)

[Abbildung 3‑5: Veranschaulichung 7](#_Toc118096384)

[Abbildung 4‑1: Screenshot Blockschaltbild **Error! Bookmark not defined.**](#_Toc118096385)

[Abbildung 4‑2: Screenshot TYPE1 8](#_Toc118096386)

[Abbildung 4‑3: Screenshot TYPE2 8](#_Toc118096387)

[Abbildung 4‑4: Screenshot TYPE3 8](#_Toc118096388)

[Abbildung 4‑5: Screenshot TYPE4 9](#_Toc118096389)

[Abbildung 4‑6: Screenshot TYPE5 9](#_Toc118096390)

[Abbildung 4‑7: Screenshot TYPE6 9](#_Toc118096391)

[Abbildung 4‑8: Screenshot TYPE7 9](#_Toc118096392)

[Abbildung 4‑9: Screenshot TYPE8 10](#_Toc118096393)

# Tabellenverzeichnis

[Tabelle 1: Messergebnisse. 3](#_Toc118096394)

# Aufgabe 1: Aufbau und grundlegende Motorsteuerung

## Aufgabe 1.1

A graph with purple lines

Description automatically generated

Abbildung ‑: Screenshot Positionssignal

**Erläuterung:**

Der Graph beginnt bei einem beliebigen Wert, je nach initialer Ausrichtung. Dann steigt der Graph linear in zeitdiskreten Schritten bis zu einem Maximalwert von ca. 1010 und wird anschließend auf den Wert 0 zurückgesetzt.

## Aufgabe 1.2

**Erläuterung:**

Das Mittel über fünf Abtastwerte ergibt eine Abtastperiode von 15,42 ms bzw. 64,85 Hz. Daraus folgt mithilfe der Nyquist-Frequenz, dass das System maximal mit (64,85/2)Hz bzw. 32,425 Hz geregelt werden darf. Das entspricht einer Regelzeitkonstante von 30,84 ms.

Die Regelung auf einem Arduino kann ohne Overhead eines Betriebssystems wie Windows oder anderer Applikationen erfolgen. Außerdem ist ein Arduino für Echtzeitanforderungen optimiert und kann unter anderem Sensoren schneller auslesen.

## Aufgabe 1.3

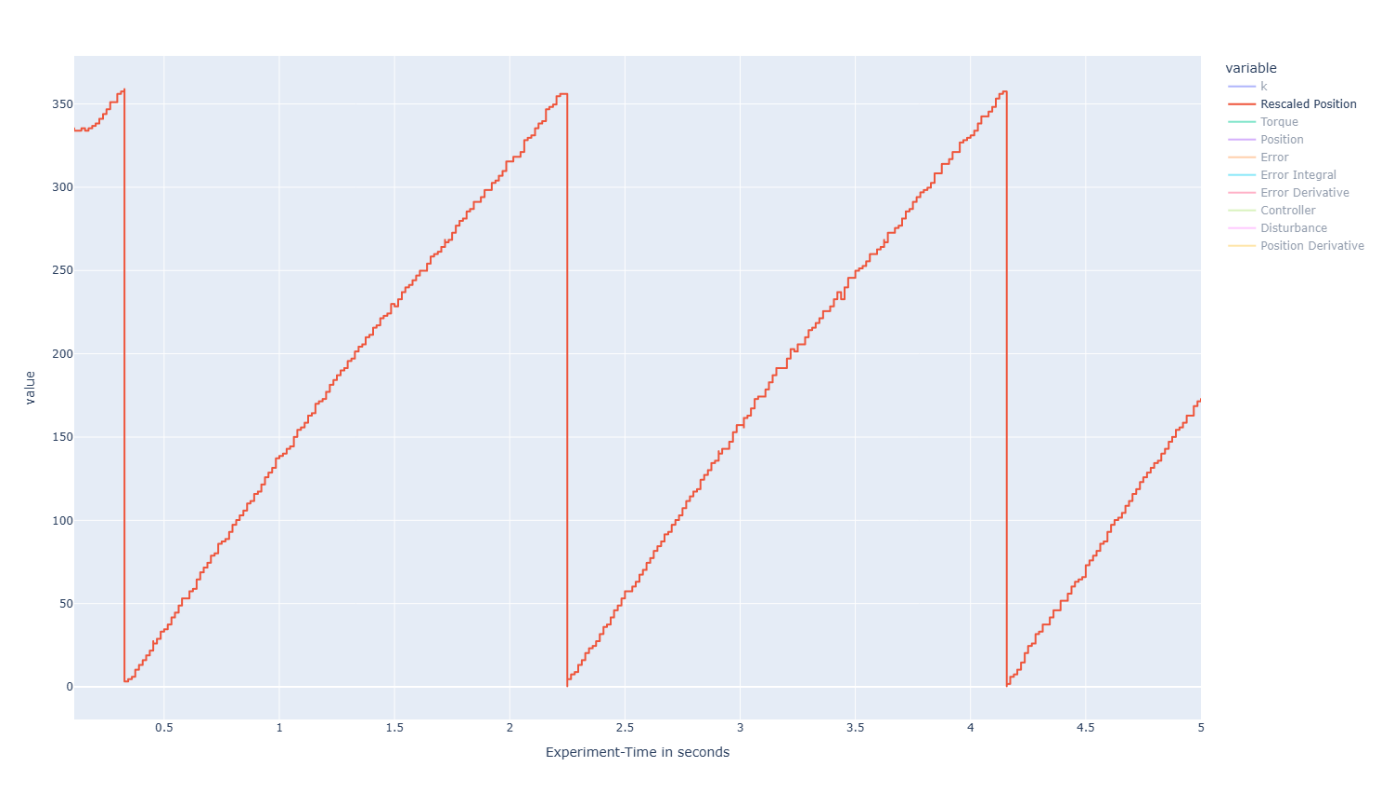


Abbildung ‑: Screenshot Positionssignal

**Erläuterung:**

Das Positionssignal wurde mit dem Faktor 360/1010 multipliziert und um 8,91 Grad gegen den Uhrzeigersinn verschoben damit die untere Gleichgewichtslage 180 Grad entspricht. Entsteht durch die Nullpunktverschiebung ein Winkel kleiner Null werden 360 Grad addiert.

# Aufgabe 2: Störsignale des Hebelarms

|  |  |  |  |
| --- | --- | --- | --- |
| Torque | Amplitude | Proportionalitätsfaktor | Proportionalitätsfaktor nach Offset von -2 |
| -20 | -8 | 2.5 | 2 |
| -10 | -4 | 2.5 | 1.66 |
| -8 | -2 | 4 | 2 |
| -5 | 0 | Inf | 2.5 |
| 5 | 4 | 1.25 | 2.5 |
| 8 | 6 | 1.3 | 2 |
| 10 | 8 | 1.25 | 1.66 |
| 20 | 12 | 1.66 | 2 |

Tabelle 1: Messergebnisse.

## Aufgabe 2.1

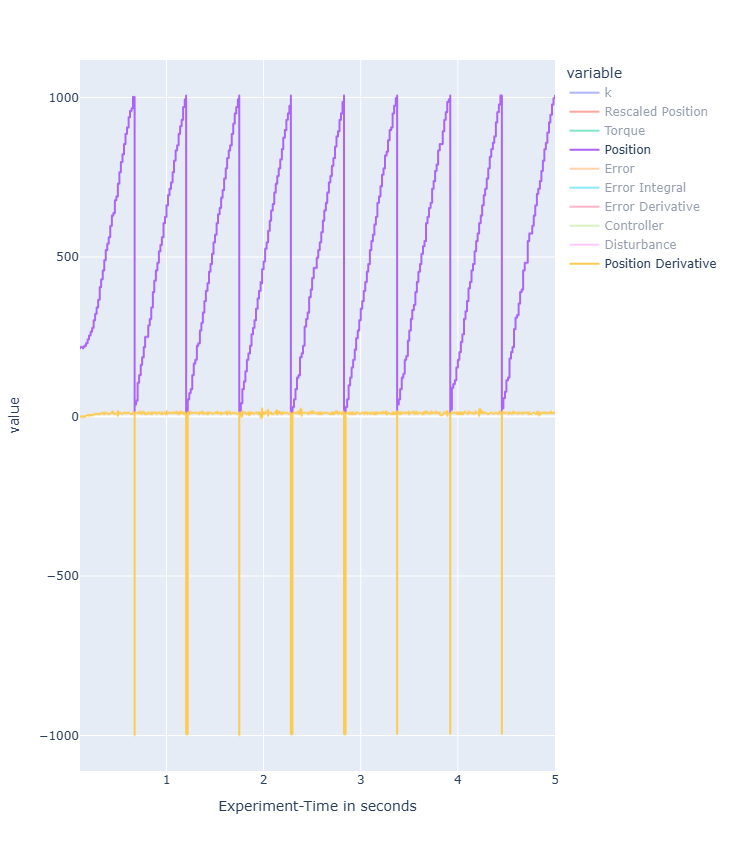


Abbildung ‑: Screenshot Positionssignal und Ableitung des Positionssignals

**Erläuterung:**

Erkläre, wie möglicherweise auftretende Sprünge zustande kommen?

Wenn das Pendel mit einer Umdrehung fertig ist, dann springt die Positionsmessung (Winkel) von 360o auf 0o. Diese Spitzen bedeuten nichts anders als eine Ableitung, die unendlich groß ist.

Erkläre, wieso die Ableitung nicht immer konstant ist, sondern z.B zwischen Null und einem Wert x springt?

Die Position ist ein diskretes Signal, d.h ein Halteglied 0ter Ordnung wurde benutzt (Treppensturuktur erkennbar). An diese Stellen, wo das Signal konstant bleibt, ist die Ableitung logischerweise null. An den Stellen, wo sich das Positionssignal ändert, ist die Ableitung ungleich 0.

## Aufgabe 2.2

**Erläuterung:**

Positive Werte für Torque ergaben einen Proportionalitätsfaktor von ca. 1,25 wohingegen der Faktor bei negativen Werten ca. doppelt so groß war (2,5). Sowohl dies als auch die Tatsache, dass sich bei einem Torque-Wert von -5 der Heble gar nicht mehr bewegt weißt auf einen Offset hin. Dieser Offset kann durch die Addition von -2 zu großen Teilen ausgeglichen werden. Unterschiedliche Proportionalitätsfaktoren weißen auf Nichtlinearitäten hin, welche auf die Bauweise des Servomotors zurückführbar sein könnten.

## Aufgabe 2.3

**Erläuterung**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Störsignaltyp** | **Signalbeschreibung** | **Amplitude** | **Frequenz/Hz** |
| ERROR1 | Sinusförmige Störung | 13 | 3.18 |
| ERROR2 | Sinusförmige Störung | 7 | 0.79 |
| ERROR3 | Rechteck Signal | 15 | 0.83 |
| ERROR4 | Rechteck Signal | 5 | 3.05 |
| ERROR5 | Rechteck Signal | 10 | 32.25 |
| ERROR6 | Sinusförmige Störung | 9 | 0.8 |
| ERROR7 | Sinusförmige Störung | 9 | 0.16 |
| ERROR8 | Rechteck Signal | 10 | 1 |
| ERROR9 | Sinusförmige Störung | 9 | 8 |

## Aufgabe 2.4

**Erläuterung:**

Die maximale Störsignalfrequenz, die mit den Simulationseinstellungen gemessen werden kann, wurde anhand der Abtastzeit und dem Shannon Theorem bestimmt.

Aus den Graphen können wir ablesen, dass die Abtastzeit T=15ms ist.

Deshalb erhalten wir eine Abtastfrequenz fsampling = 1/T = 66.67Hz

Laut Shannon Theorem muss die Abtastfrequenz mindestens **doppelt so hoch** sein wie die höchste Störfrequenz, um die Signale korrekt zu erfassen. Das heißt: fsampling≥ 2\*f = 66.7Hz. Das maximale Störsignalfrequenz f =33.33Hz

Wenn die Störsignalfrequenz höher liegt, könnten **Aliasing-Effekte** auftreten, bei denen Signale falsch interpretiert werden.

# Aufgabe 3: Positionsregelung

## Aufgabe 3.1

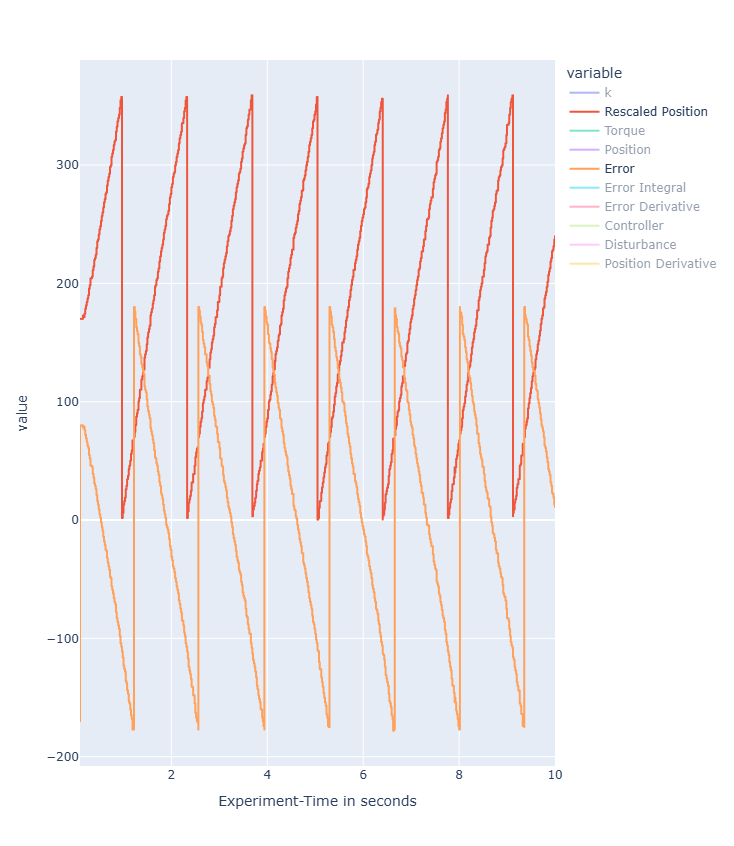


Abbildung 3‑: Screenshot Fehlersignal und Positionssignal

**Erläuterung:**

In der Abbildung ist zu erkennen, wie das Fehlersignal jeweils seinen Nulldurchgang bei 250 Grad des Positionssignal besitzt, was der Erwartung entspricht. Außerdem ist erkennbar, dass das Fehlersignal zwischen 180 und -180 oszilliert, da das Signal nur mit einem Fehler von +-360/2 Grad behaftet sein kann.

## Aufgabe 3.2

Um die PID Funktion zu diskretisieren muss sowohl die Ableitung als auch das Integral approximiert werden. Dies geschieht in den error\_derivative und error\_integral Funktionen. Hier wurde die Ableitung durch die Rückwärtsdifferenz und das Integral durch eine Stufenintegration approximiert. Der Ausgang des Reglers ist dann eine mit den Reglerkoeffizienten gewichtete und dem Offset addierte Linearkombination.

## Aufgabe 3.3

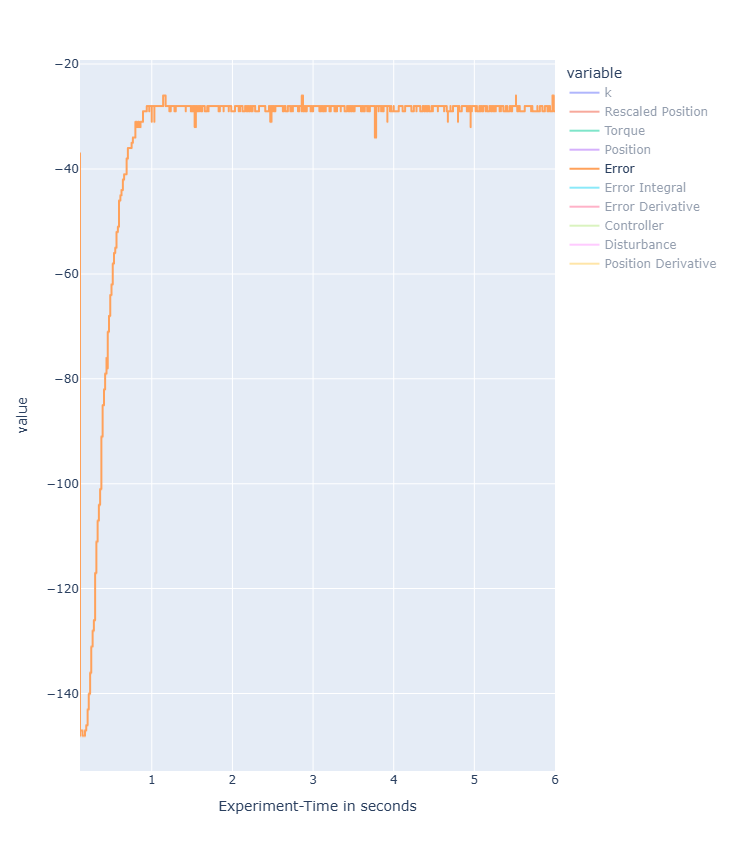


Abbildung ‑3: Screenshot Verstärkungsfaktor 0.1

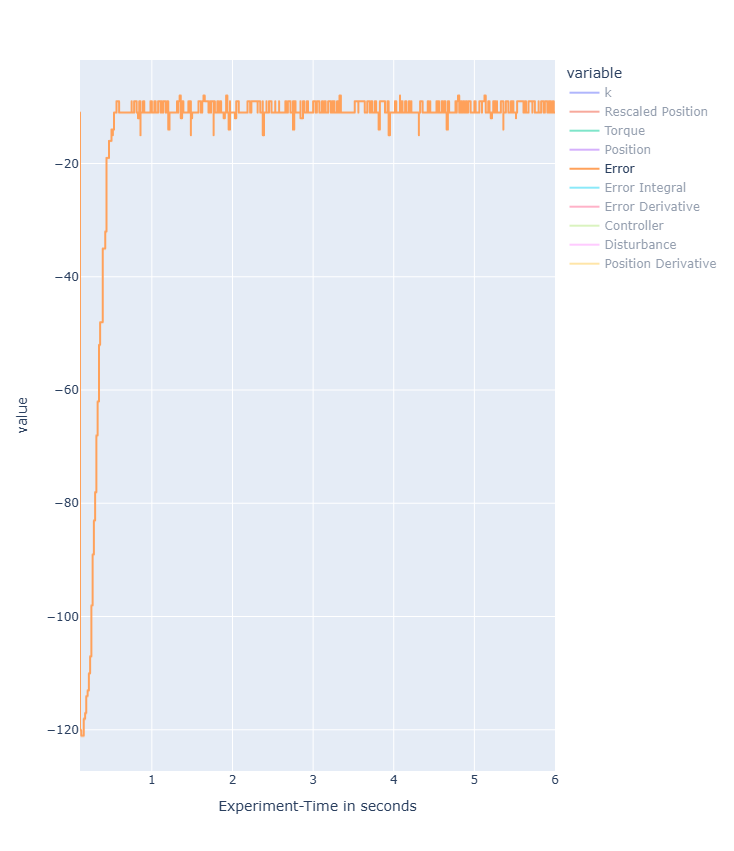


Abbildung ‑4: Screenshot Verstärkungsfaktor 0.2

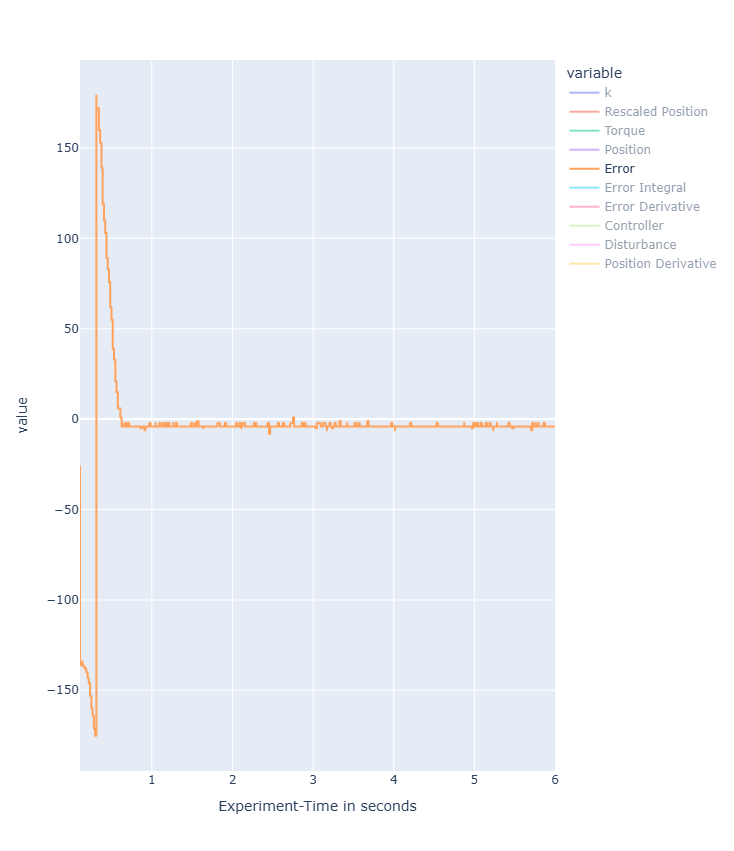


Abbildung ‑5: Screenshot Verstärkungsfaktor 0.3

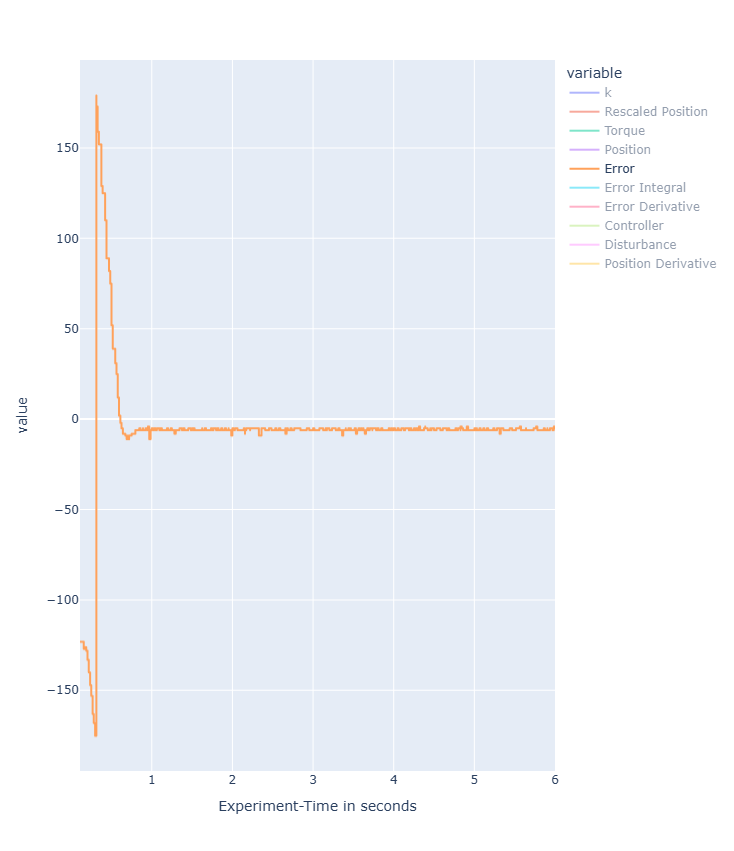


Abbildung ‑6: Screenshot Verstärkungsfaktor 0.4

**Beschreibung:**

In dieser Aufgabe wurde der P-Regler mit vier verschiedenen Koeffizienten benutzt (0,1; 0,2; 0,3; 0,4), um das System auf einen bestimmten Winkel zu regeln. Dabei entstand bei einem Faktor von 0,1 eine bleibende Regelabweichung von 29 Grad. Bei 0,2 eine Abweichung von 9 Grad. Der Faktor 0,3 erzielte eine Abweichung von 2 Grad und der Faktor 0,4 eine von 4 Grad. Erwartungsgemäß sinkt der stationäre Fehler bei höheren Vorfaktoren. Der stationäre Regelfehler lässt sich mit einem P-Regler jedoch nie auf 0 reduzieren, was bereits aus Systemtheorie 1 bekannt ist. Auch beginnt das System bei höheren Vorfaktoren zu schwingen an, was bereits in Abbildung 3-6 erkennbar ist. Auch ist zu erkennen, dass die Regelabweichung bei dem Wert von 0,4 wieder ansteigt was aber unserer Meinung der Genauigkeit und der Ausganslage des Arduinos zu verschulden ist, da größere Werte sich im Durchschnitt wieder einer kleineren Fehlerabweichung annähern.

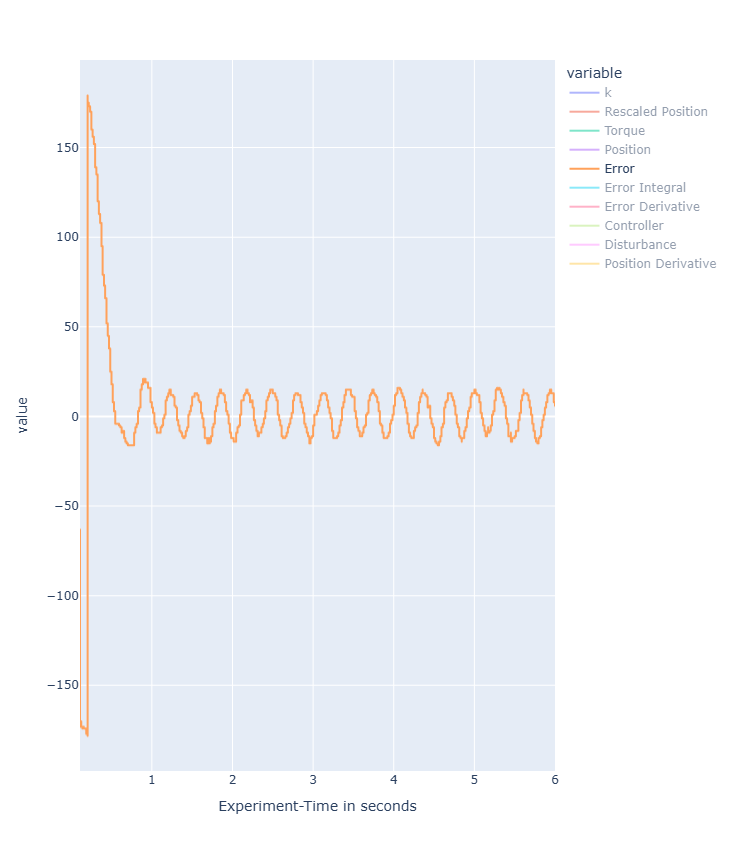
## Aufgabe 3.4

**Erläuterung:**

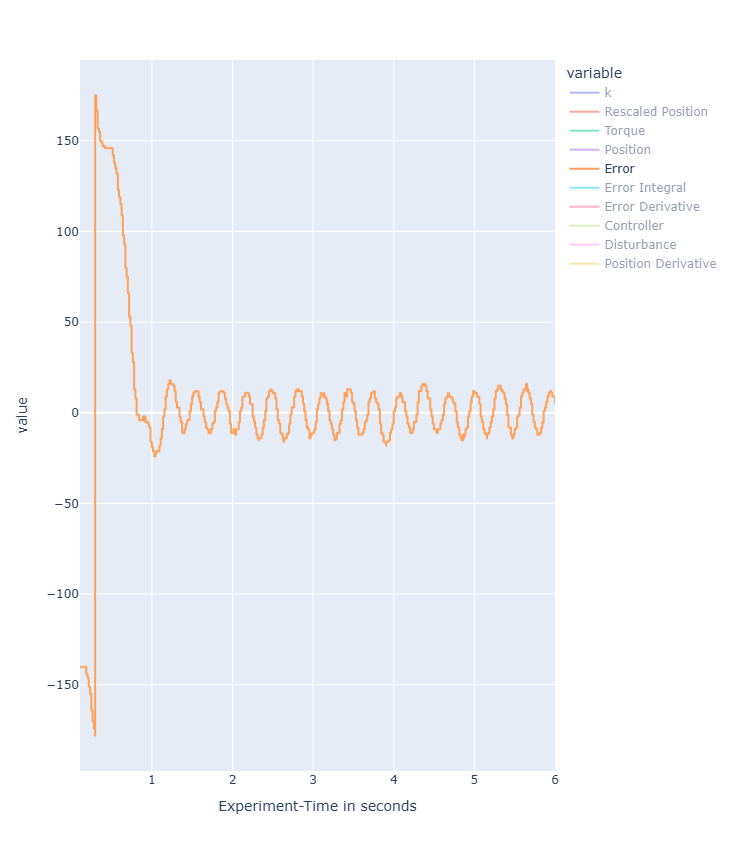
In dieser Aufgabe wurde versucht mithilfe der bereitgestellten Tabelle und einem Schwingversuch die optimalen Parameter für einen PID-Regler zu entwerfen.

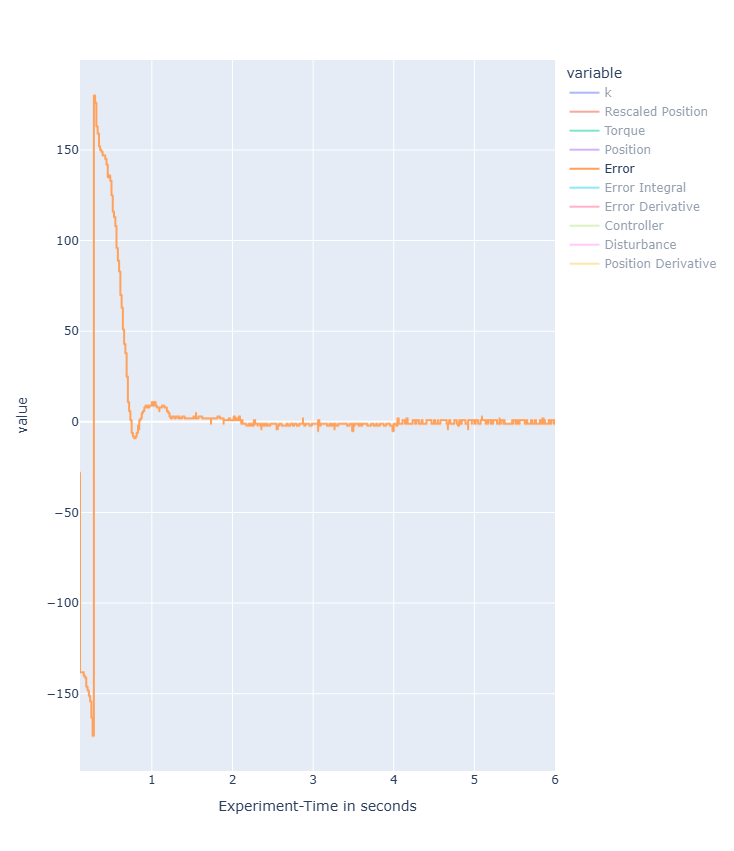
Kr\_krit wurde der Wert 1,4 in dem Schwingversuch gefunden und T\_krit konnte auf 0,312s bestimmt werden. Aus diesen Werten ergeben sich mithilfe der Tabelle die Parameter für die jeweilige Regler Art (P, PI, PID).

**P-Regler: (KR = 0,7)**

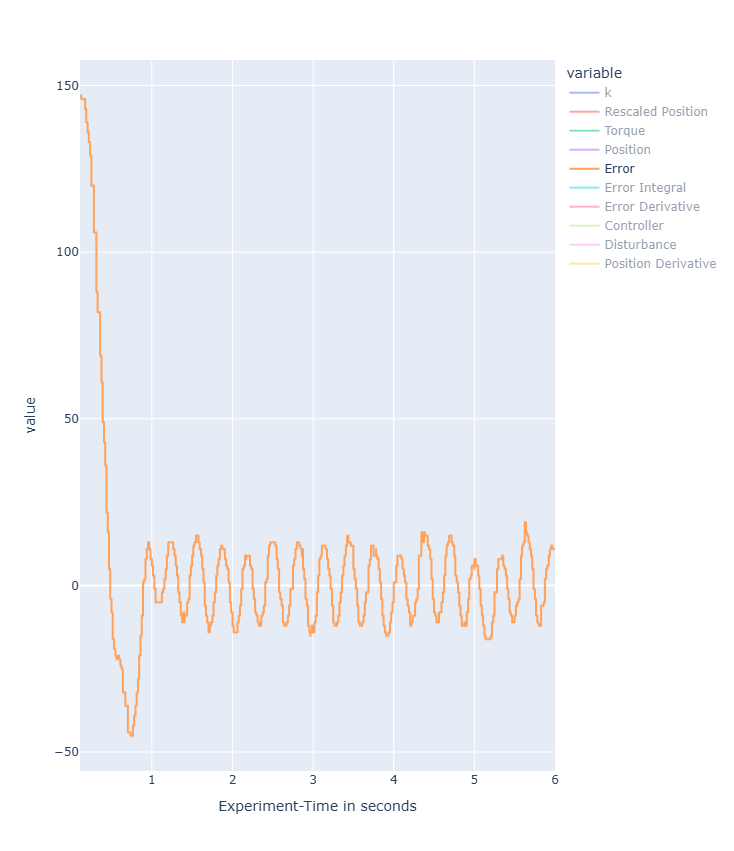
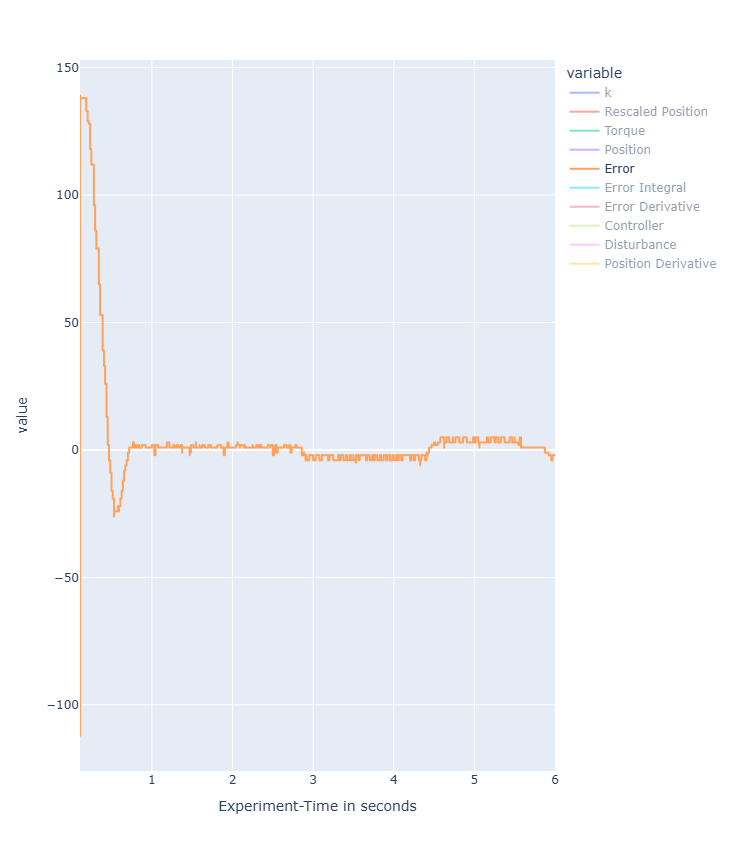
**Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte BeschreibungOhne Fehler ERROR1**

**PI-Regler: (KR = 0,63, T\_i = 0,259)**

**Ohne Fehler ERROR1**

**PID-Regler(KR = 0,49, T\_i = 0,25, T\_D = 0,0312)(eigene Werte, da Tabellenwerte nicht funktionierten)**

**Ohne Fehler ERROR1**

Für eine Regelung ohne Fehler erwies sich der P-Regler am zuverlässigsten, auch wenn theoretisch eine kleine Regelabweichung übrigbleibt. Tatsächlich war die Regelabweichung des PI-Reglers gleichgroß und brachte so keinen größeren Vorteil. Bei Signalen mit Fehlersignalen konnte der PID-Regler zum Teil bessere Ergebnisse liefern, welche im obigen Beispiel jedoch nicht zu sehen sind. Auch stellte das D-Glied Herausforderungen und machte das System schnell instabil.

# Aufgabe 4: Kompensation einer Störung

**Beschreibung:**

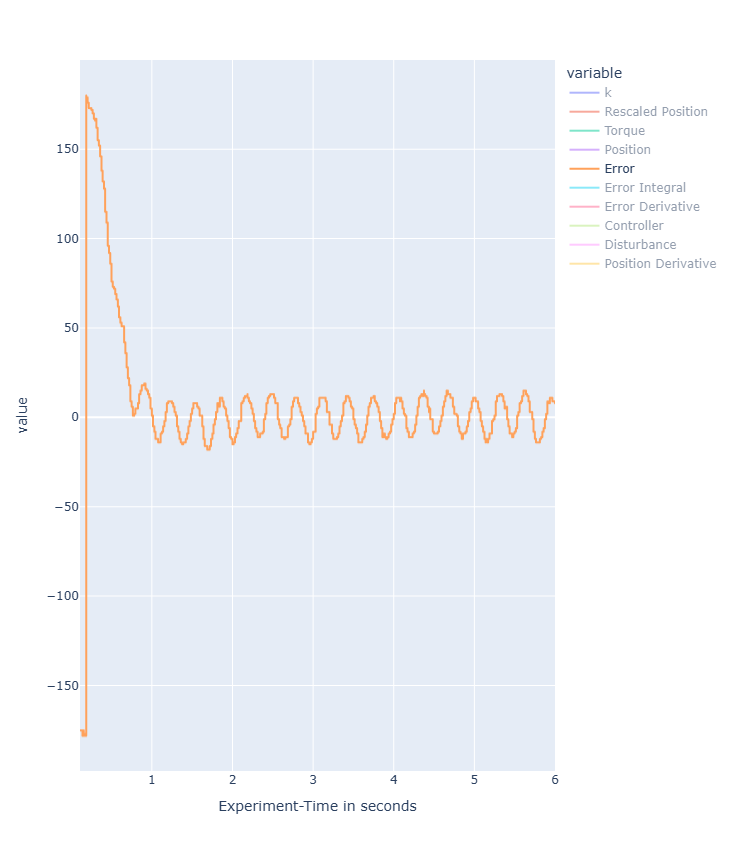
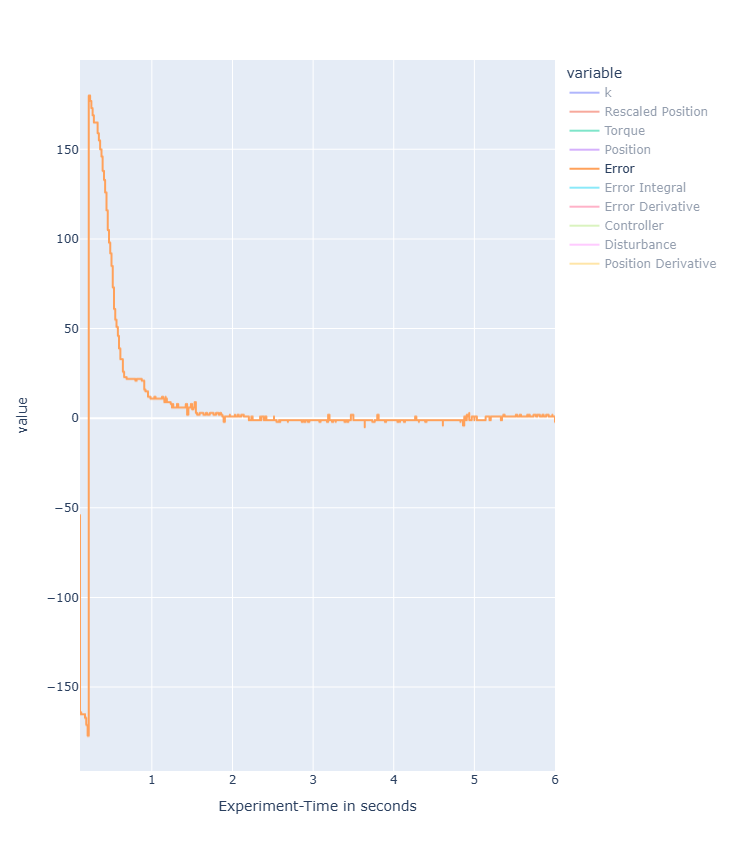
Dies ist ein Typoblindtext. An ihm kann man sehen, ob alle Buchstaben da sind und wie sie aussehen. Manchmal benutzt man Worte wie Hamburgefonts, Rafgenduks oder Handgloves, um Schriften zu testen.

Abbildung ‑: Screenshot ERROR1

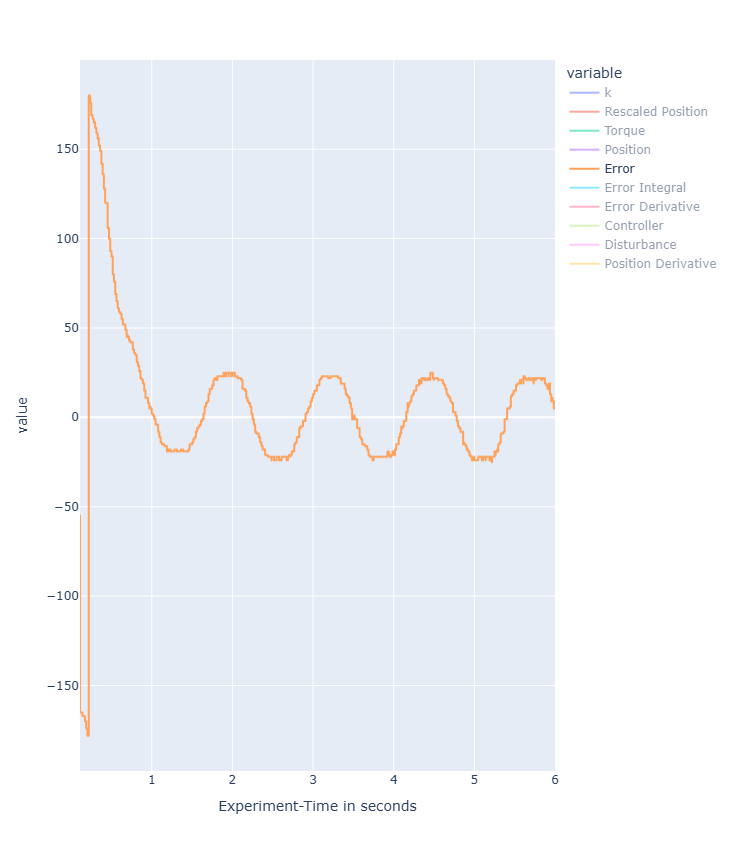
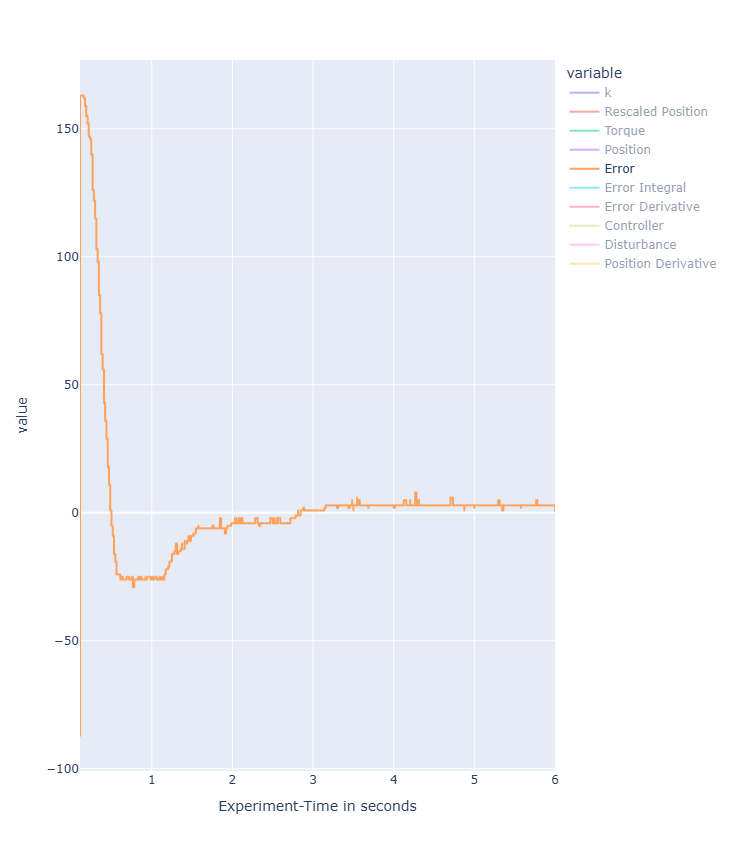


Abbildung ‑: Screenshot ERROR2

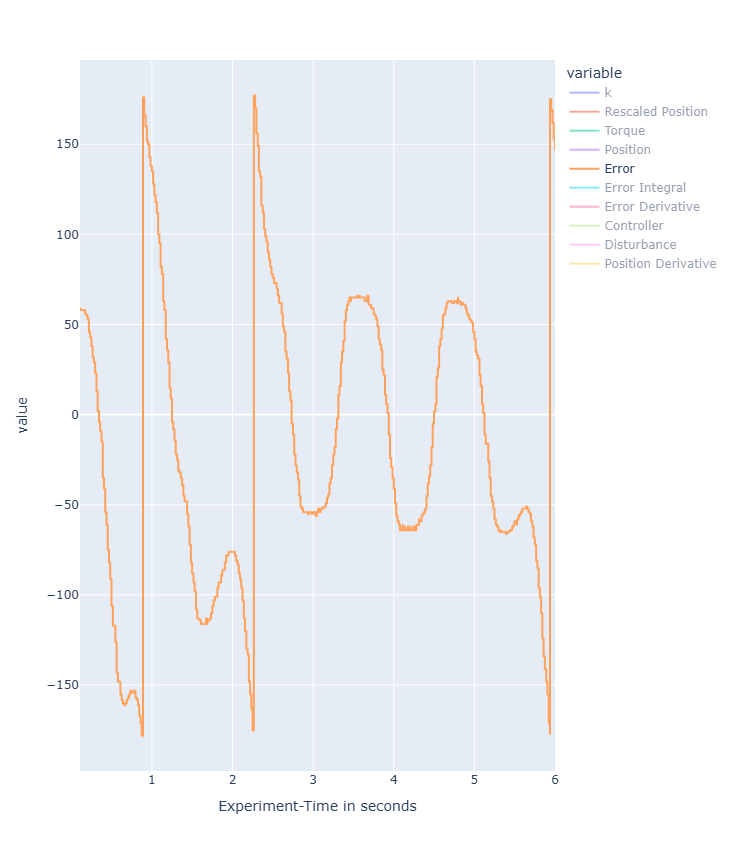
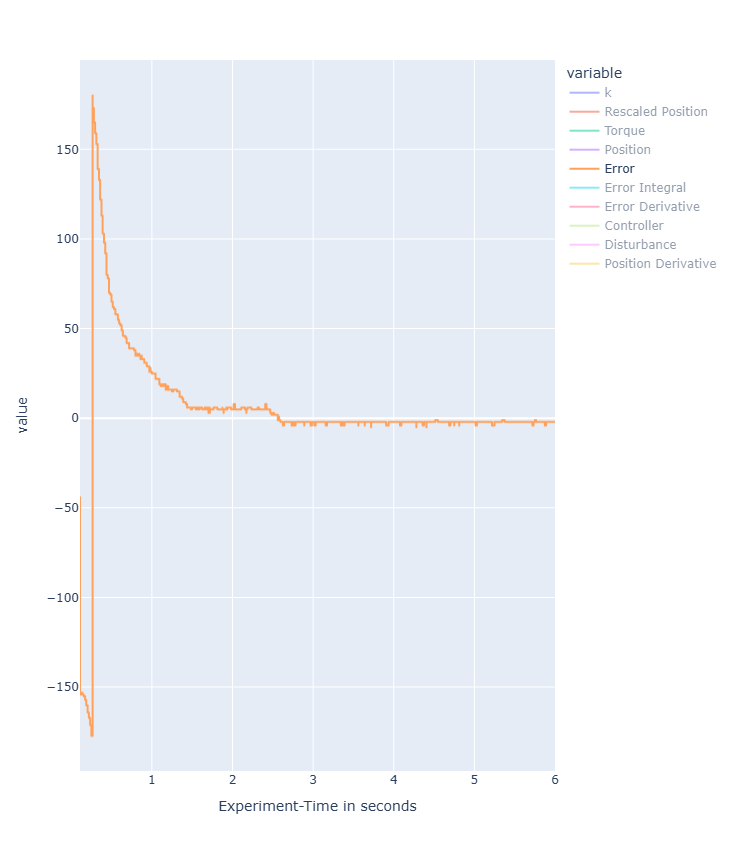


Abbildung ‑: Screenshot ERROR3

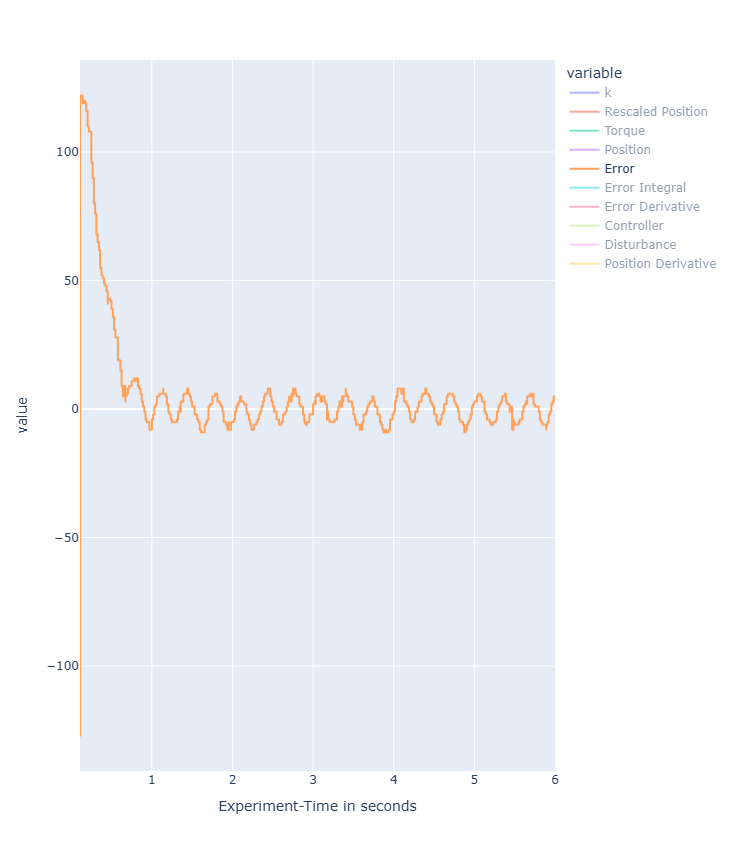
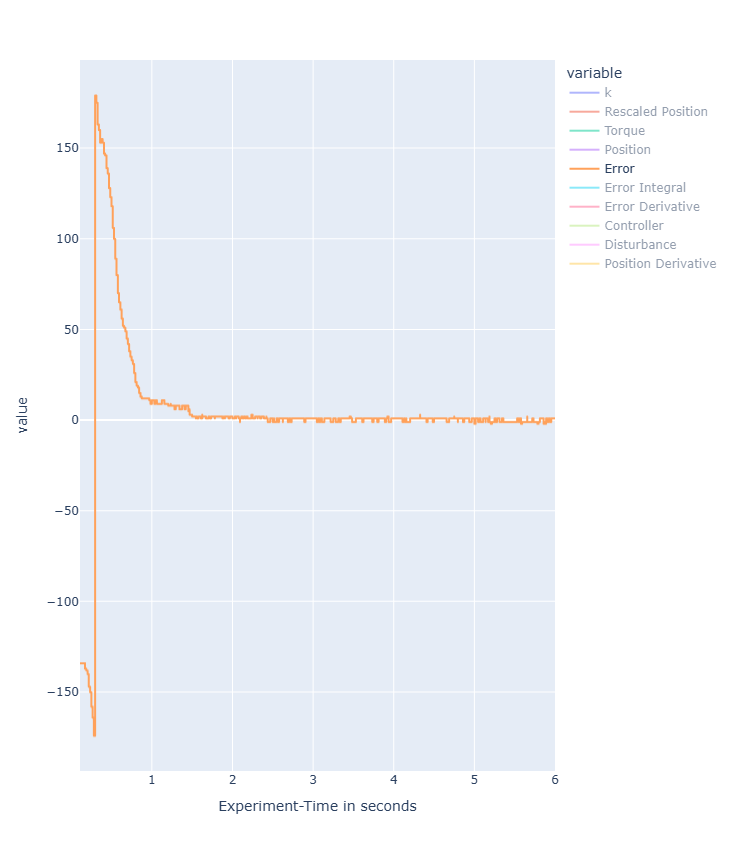


Abbildung ‑: Screenshot ERROR4

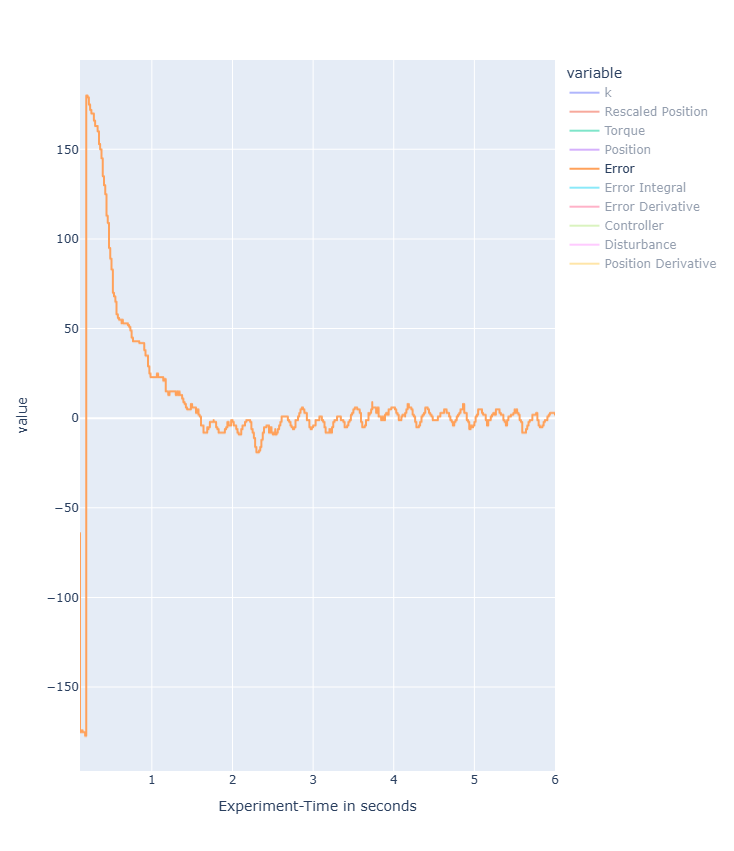
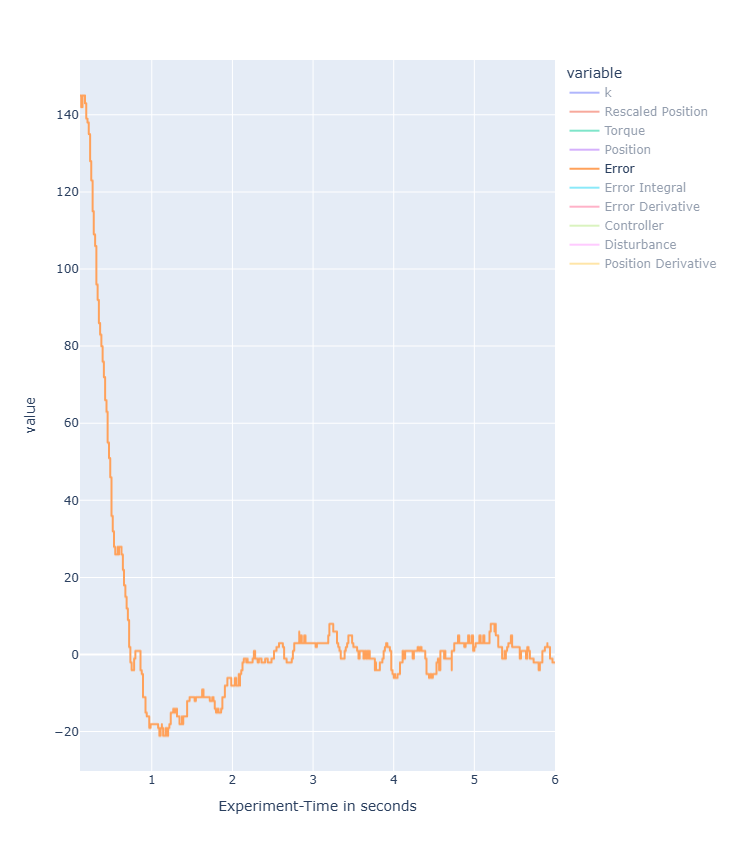


Abbildung ‑: Screenshot ERROR5

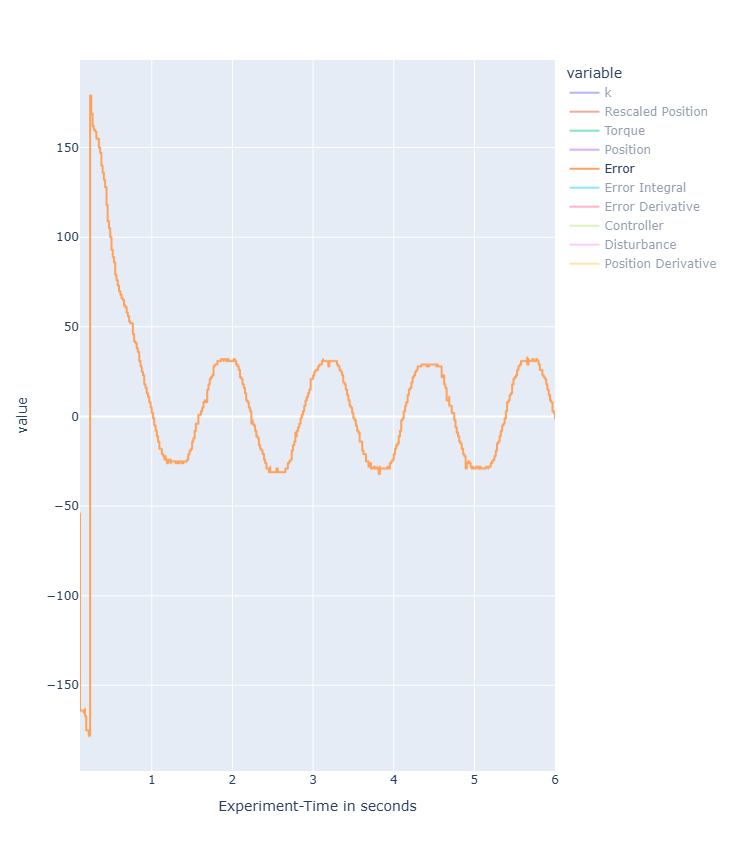
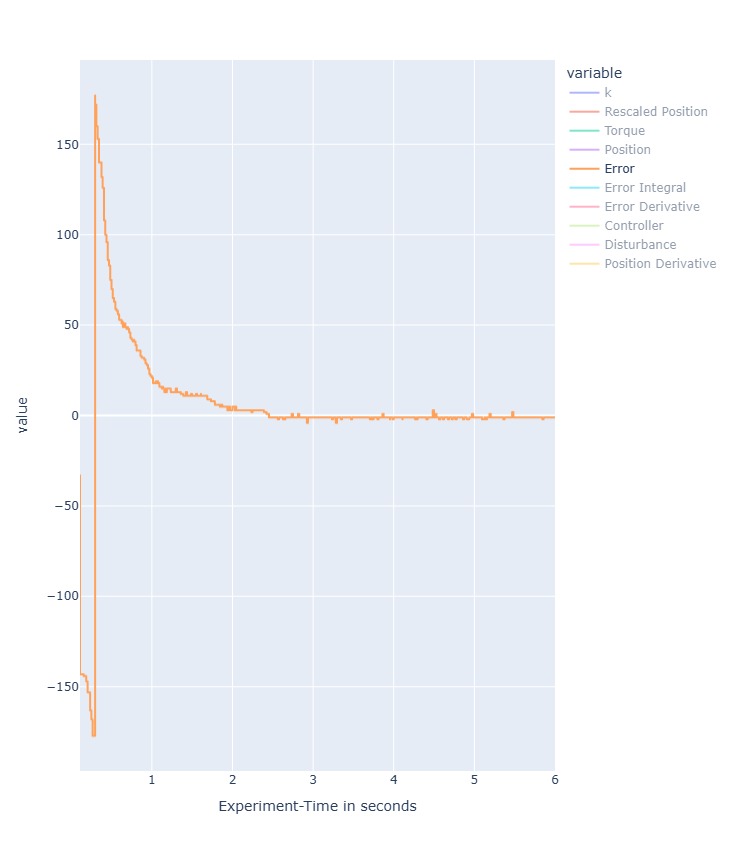


Abbildung ‑: Screenshot ERROR6

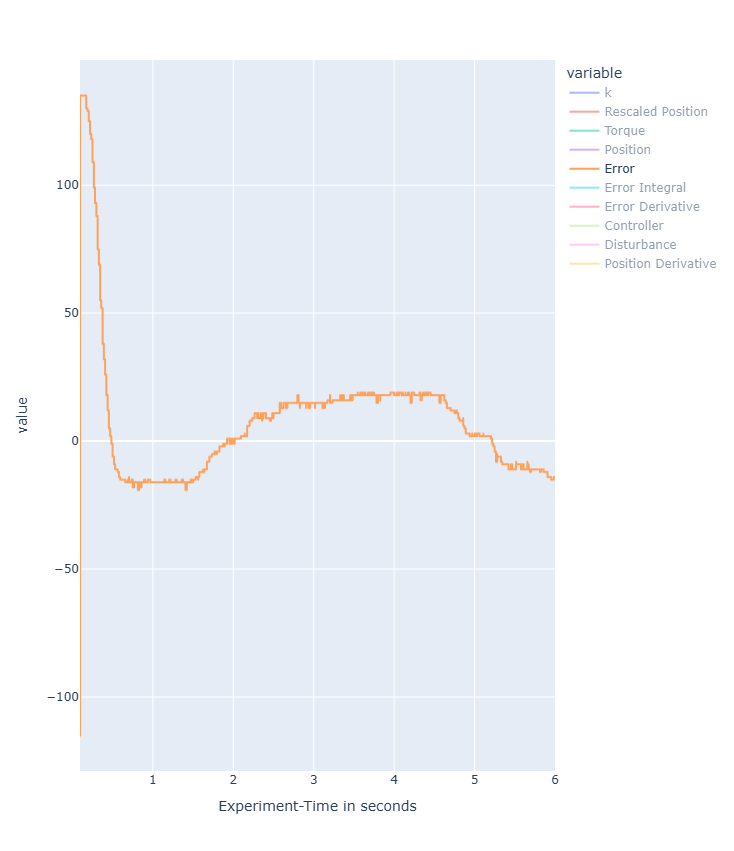
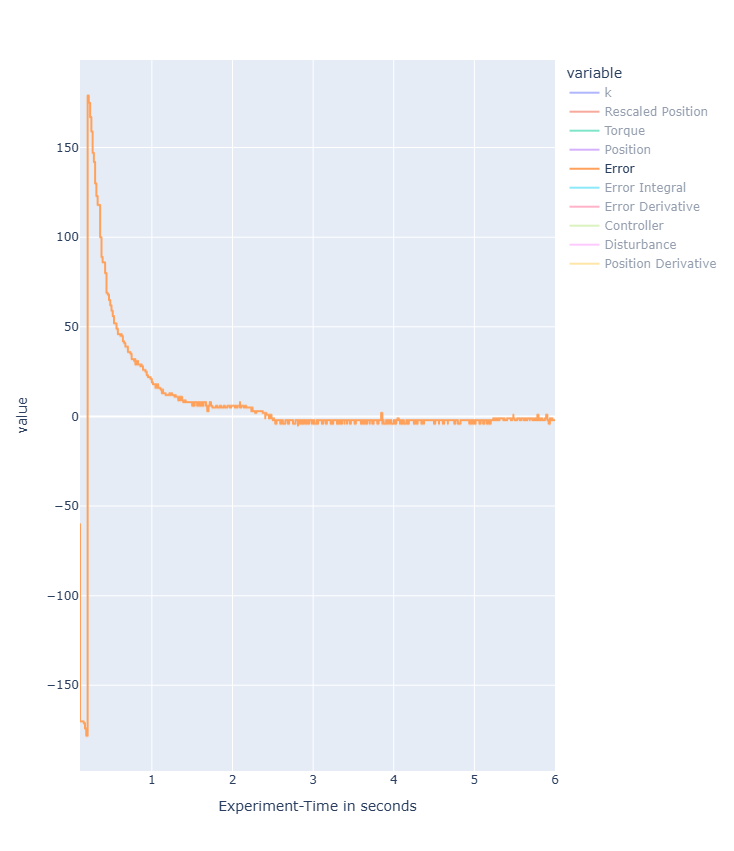


Abbildung ‑: Screenshot ERROR7

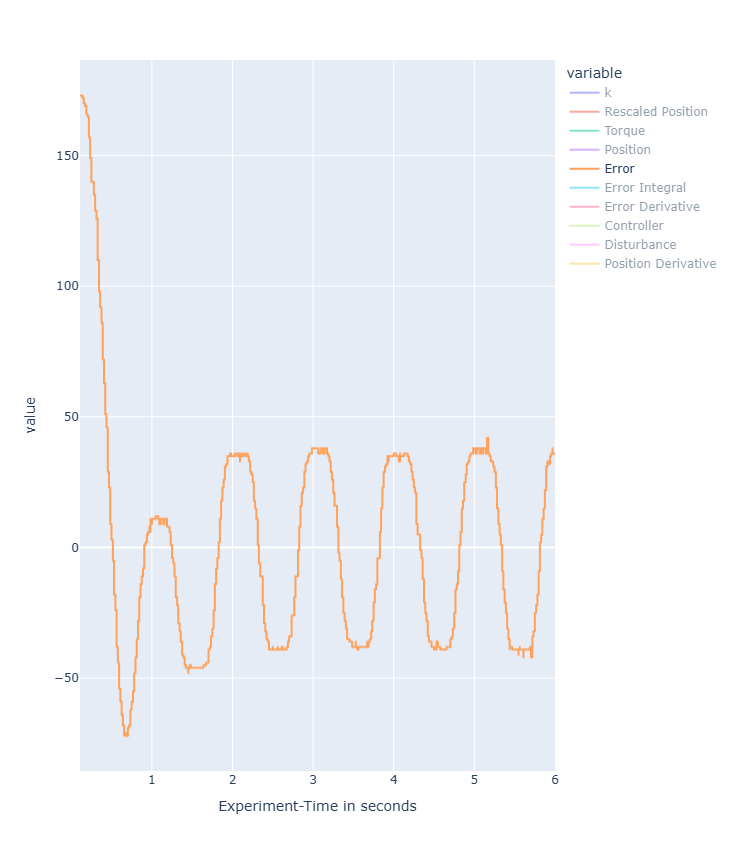
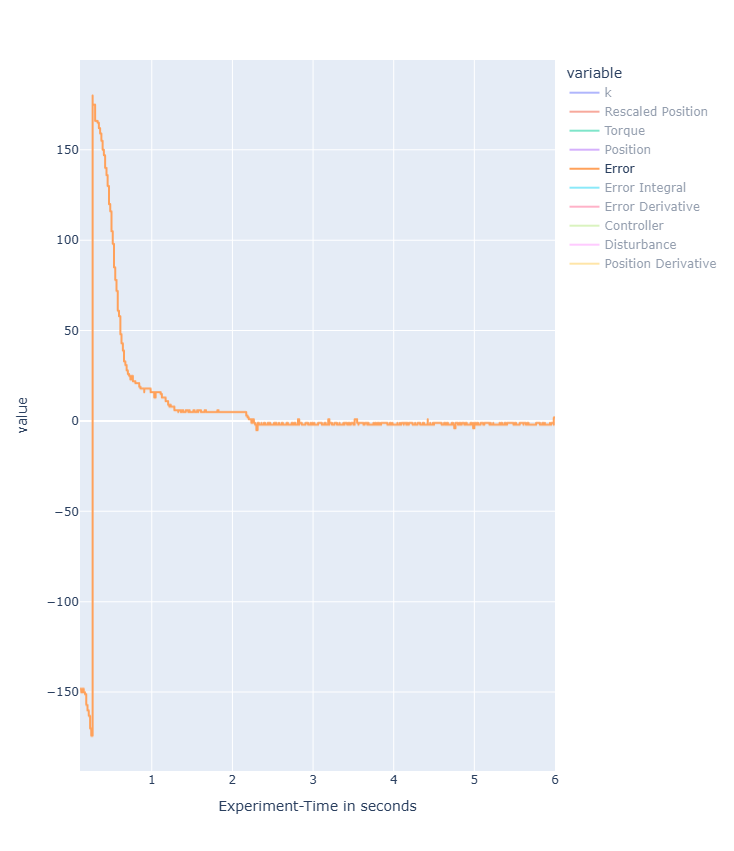


Abbildung ‑: Screenshot ERROR8

**Beschreibung:**

In den obigen Abbildungen ist jede Störung mit PID-Regler jeweils einmal ohne und einmal mit Störkompensation zu erkennen. Aus den Abbildungen ist klar zu erkennen, dass mit Störkompensation der Regler wie im störfreien Fall seine Position mit kleinem Regelfehler auf die gewünschte Position Regeln kann. Im Störfall hingegen sind zum Teil große Abweichungen erkennbar die nur schlecht von Regler unterdrückt werden.

Totzeit kann zu einer Phasenverschiebung im erhaltenen zum tatsächlichen Signal führen. Dies kann im Extremfall zu instabilem Verhalten führen.

**Fügen Sie ihrer Abgabe die *student.ino*, die von Ihnen bearbeitet wurde,hinzu!**

#include <SignalSource.h>

#include <math.h>

// Global Variables

//TODO

double dervlasterrorval;

unsigned long dervlasttimestamp;

unsigned long integrallasttimestamp;

double errintegral;

double T\_krit = 0.312;

double K\_krit = 1.4;

double controltorque;

double T\_i = 0.8\*T\_krit;

double T\_d = 0.1\*T\_krit;

double K\_p = 0.35\*K\_krit;

double K\_i = K\_p/T\_i;

double K\_d = K\_p \* T\_d;

//Resets the controller state when an experiment is started.

void reset(){

  //TODO

}

/\*\*

 \* Rescales the position value from the motor.

 \*

 \* @param position The original position value to be rescaled.

 \* @return The rescaled position value, adjusted to remain within a range of 0 to 360.

 \*/

float rescalepos(int16\_t position){  //rescaling function

  double angle = (360\*(double)position)/1010.00;

  //set to down to 180 degrees

  angle -= 8.91;

  if(angle<0){angle += 360;}

  return angle;

}

/\*\*

 \* Calculates the error between the desired setpoint and the current position.

 \*

 \* @param setpoint The desired target value.

 \* @param currentpos The current position value.

 \* @return The error value, normalized to account for smallest angle to target value from current value.

 \*/

int16\_t calculate\_error(int16\_t setpoint, int16\_t currentpos){

  //TODO

  int16\_t angle = 0;

  currentpos = rescalepos(currentpos);

  angle = setpoint - currentpos;

  if(angle > 180){

    angle -= 360;

  }

  if(angle < -180){

      angle += 360;

    }

  return angle;

}

/\*\*

 \* Calculates the derivative of the error for use in a PID controller.

 \*

 \* @param error The current error value.

 \* @return The change in error (delta) since the last call.

 \*/

int16\_t error\_derivative(int16\_t error){

  //TODO

  int16\_t deriv = (error-dervlasterrorval);

  dervlasterrorval = error;

  return deriv;

}

/\*\*

 \* Calculates the integral of the error for use in a PID controller.

 \*

 \* @param error The current error value.

 \* @return The accumulated error over time (integral).

 \*/

int16\_t error\_integral(int16\_t error){

  //TODO

  errintegral += error\*(double)(millis()-integrallasttimestamp)/1000.0;

  integrallasttimestamp = millis();

  return errintegral;

}

/\*\*

 \* Implements a basic controller using proportional, integral, and derivative (PID) control.

 \*

 \* @param error The current error value.

 \* @param error\_i The integral of the error.

 \* @param error\_d The derivative of the error.

 \* @param measured\_disturbance The measured disturbance value.

 \* @return The calculated control torque.

 \*/

int16\_t controller(int16\_t error, int16\_t error\_i, int16\_t error\_d, int16\_t measured\_disturbance){

  //TODO

  controltorque = (K\_p\*error) + (K\_i \* error\_i \* T\_s) + (K\_d \* error\_d / T\_s) - 2 - 1\*measured\_disturbance;

  //controltorque = (K\_p\*error) + (K\_i \* error\_i) + (K\_d \* error\_d) - 2 ;

  return controltorque;

}

/\*\*

 \* Placeholder for the main program loop. Do not change this.

 \*/

void loop(){

  start\_loop();

}

/\*\*

 \* Sets up the controller by registering student-defined functions and initializing signals.

 \* Do not change this.

 \*/

void setup(){

  register\_student\_fcns((studentFcns){

    .calculate\_position=rescalepos,

    .calculate\_error=calculate\_error,

    .calculate\_error\_integral=error\_integral,

    .calculate\_error\_derivative=error\_derivative,

    .calculate\_control=controller,

    .reset=reset});

  setup\_signal();

}