School of Engineering
InIT Institut für Informatik

Bachelorarbeit Informatik

Optimierungsmethoden und Algorithmen für die PID- Regelung von allgemeinen technischen Systemen

Autoren	Lukas Gruber
Hauptbetreuung	Roland Büchi
Eingereicht am	7. Juni 2024

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:	**Name Studierende:**
Winterthur, 7. Juni 2024	Lukas Gruber

Zusammenfassung

Der PID-Regler ist eine weit verbreitete Regelungstechnik. PID steht für Proportional-Integral-Derivativ, was für die drei Strategien des Reglers steht. Der Regler kann für verschiedenste Systeme verwendet werden, von der Temperaturregelung in Heizgeräten bis zur Geschwindigkeitssteuerung von Motoren in der Robotik. Die Kunst besteht darin, die Regelparameter so zu wählen, dass das System das gewünschte Verhalten zeigt. Mithilfe von Simulink (Matlab) lässt sich die Sprungantwort eines PID-Regelkreises berechnen. Diese Arbeit erstellt ein Grundkonstrukt (Python-Skripts) um solche Regelkreise mit der Open-Source Plattform Python zu simulieren. Beliebige Systeme können simuliert werden. In dieser Arbeit wurden vier spezifische Systeme implementiert und untersucht. Die simulierten Sprungantworten wurden mit den Sprungantworten des äquivalenten Simulink-Modells validiert. Alle simulierten Systeme erzeugten dieselbe Sprungantwort wie das Simulink-Modell, und ihr ITAE-Kriterium (Integral of Time-weighted Absolute Error) war ebenfalls nur minimal verschieden.

Um optimale Regelparameter für die Systeme zu finden, wurde ebenfalls mit Python, die Partikelschwarmoptimierung, ein Optimierungsalgorithmus aus dem Bereich Künstliche Intelligenz, implementiert. Der PSO-Algorithmus sucht optimale Regelparameter für die gegebenen Systeme. Für jedes System wurden 100 PSO-Simulationen durchgeführt. Für zwei Systeme liegen bereits optimierte Parameter vor, die als Referenzwerte dienen. In dieser Arbeit werden die mittels PSO gefundenen Parameter mit den bekannten Referenzwerten verglichen, um die Zuverlässigkeit von PSO bei der Bestimmung guter Regelparameter zu bewerten. PSO liefert für die untersuchten Systeme sehr gute Regelparameter mit hoher Verlässlichkeit und in praktikabler Zeit, ohne dass spezielle Hardwareanforderungen nötig sind. PSO stellt eine echte Alternative zu herkömmlichen Methoden der Parameterfindung dar.

Abstract

The PID controller is a widely used control technique, standing for Proportional-Integral-Derivative, which represents the three strategies of the controller. The controller can be applied to various systems, from temperature control in heating devices to speed control of motors in robotics. The challenge lies in selecting the control parameters to achieve the desired system behavior. Using Simulink (Matlab), the step response of a PID control loop can be calculated. This work creates a basic framework (Python scripts) to simulate such control loops using the open-source platform Python. Any system can be simulated. In this work, four specific systems were implemented and examined. The simulated step responses were validated against the step responses of the equivalent Simulink model. All simulated systems produced the same step response as the Simulink model, and their ITAE criterion (Integral of Time-weighted Absolute Error) was also only minimally different.

To find optimal control parameters for the systems, particle swarm optimization (PSO), an optimization algorithm from the field of artificial intelligence, was also implemented using Python. The PSO algorithm searches for optimal control parameters for the given systems. For each system, 100 PSO simulations were conducted. For two systems, optimized parameters are already available, serving as reference values. In this work, the parameters found using PSO are compared with the known reference values to evaluate the reliability of PSO in determining good control parameters. PSO provides very good control parameters for the investigated systems with high reliability and in practical time, without requiring special hardware. PSO thus represents a viable alternative to conventional methods of parameter finding.

Inhaltsverzeichnis

1 Einleitung

Geschlossene Regelkreise sind in technischen Systemen von zentraler Bedeutung, wenn es darum geht, bestimmte Sollwerte zu erreichen und zu halten. Der PID-Regler ist dabei ein etabliertes Instrument, um den Eingang in die Regelstrecke zu steuern, um den gewünschten Sollwert zu erreichen. Der PID-Regler kommt in den unterschiedlichsten Systemen zum Einsatz und ist universal einsetzbar. Der Trick dabei ist, die drei Regelparameter (P-Anteil, I-Anteil, D-Anteil) so zu wählen, dass sie bestmöglich auf das konkrete System abgestimmt sind. Die Suche nach solchen optimierten Parametern beschäftigt die Elektrotechnik schon seit es PID-Regler gibt. Das erste etablierte Verfahren stammt von Ziegler-Nichols [1] aber etliche Weitere sind bekannt z.B. Chien, Hrones and Reswick [2]. Mit dem Aufkommen von neuen allgemeinen und Suchalgorithmen insbesondere aus dem Bereich der Künstlichen Intelligenz, stellt sich die Frage wie solche Verfahren, auch auf die Parametersuche für PID-Regler angewandt werden können. Die Arbeiten [3], [4] verwenden zum Beispiel Hill-Climbing für die Suche nach Parametern und sind dabei sehr erfolgreich. Das dabei verwendete Verfahren ist jedoch zu rechenintensiv, um in praktikabler Zeit Lösungen zu finden.

1.1 Problemstellung Portierung

Weit verbreitet ist die Nutzung von Simulink von Matlab zur Berechnung der Sprungantwort eines Regelkreises. Matlab-Lizenzen sind jedoch kostspielig und nicht Open-Source. Moderne Programmiersprachen wie Python bieten das Potenzial, die gleichen Berechnungen durchzuführen, die gleichen Systeme zu simulieren und auszuwerten, und stellen eine kostenlose Open-Source-Alternative dar. Der erste Teil dieser Arbeit beschäftigt sich damit, ausgewählte Systeme und Regelkreise mit einem PID-Regler, einer Stellgrössenbeschränkung und einer anschliessenden Übertragungsfunktion mithilfe von Python zu simulieren. Die Sprungantwort wird berechnet und daraus ein gegebenes Fehlerkriterium bestimmt. Um die Richtigkeit der Ergebnisse zu überprüfen, sollen die gleichen Systeme in Simulink modelliert werden. Anschliessend werden deren Sprungantwort sowie das resultierende Fehlerkriterium mit den in Python erzielten Ergebnissen verglichen, um zu zeigen, dass die Implementierung mit Python die gleichen Resultate liefert wie das Modell in Simulink. Ziel ist es, eine Alternative zu schaffen, um Regelkreise ausschliesslich mit Open-Source-Bibliotheken in Python zu simulieren.

1.2 Problemstellung Parameteroptimierung

Bei der Betrachtung eines beliebigen PID-Regelsystems als Blackbox erfolgt die Eingabe über die Regelparameter Kp, Ki und Kd, die Parameter entsprechen dem P, I, D Anteil des PID-Reglers. Alle anderen Eigenschaften des Systems, Stellgrössenbeschränkung, Sollwert, Übertragungsfunktion sind gegeben. Nur die Parameter Kp, Ki, Kd werden variiert. Aus dem gegebenen Regelkreis mit den PID-Parametern wird dann die Sprungantwort des Systems simuliert und daraus ein Fehlerkriterium, in diesem Fall ITAE, berechnet. Das Fehlerkriterium ist die Ausgabe dieser Blackbox und soll minimiert werden. Eine Methode, um ein Parameterset zu finden, wäre so lange zufällige Parameter zu wählen, bis ein Parameterset die Anforderung an das Fehlerkriterium besteht. Man wertet die Blackbox, also man berechnet das ITAE-Kriterium für zufällige Parameter so lange, bis das ITAE einen gewissen Wert unterschreitet. Wie bereits erwähnt wurden bessere Verfahren bereits gefunden [1], [2] als zufällig zu raten. Der Fortschritt in der Computertechnik, erlaubt es uns auf handelsüblicher Hardware, sehr viele Rechenschritte in kurzer Zeit auszuführen und so automatisiert das ITAE-Kriterium aus einem

Parameterset zu berechnen. Bereits frühere Arbeiten haben erfolgreich optimierte Regelparameter für allgemeine Systeme gefunden [3], [4]. Jedoch waren die Verfahren zu rechenintensiv, um in praktikabler Zeit Lösungen zu finden.

Das Ziel ist es, auf handelsüblicher Hardware, in vernünftiger Zeit, geeignete Regelparameter für ein System zu finden. Diese Arbeit konzentriert sich dabei auf die Partikelschwarmoptimierung, fortan PSO genannt, ein Algorithmus aus dem Bereich Machine-Learning. PSO verspricht schnelle Konvergenz, abschätzbaren Rechenaufwand sowie eine geringe Anfälligkeit zum Verfangen in lokalen Minima. Am Schluss wird evaluiert wie gut PSO optimierte Regelparameter findet und mit welcher Rechenintensität das verbunden ist.

2 Theoretische Grundlagen

In diesem Kapitel sollen die theoretischen Grundlagen dieser Arbeit erläutert werden.

2.1 Übertragungsfunktionen

In der Regelungstechnik wird häufig mit Übertragungsfunktionen gearbeitet, da diese eine effiziente Methode zur Analyse und Synthese von Regelkreisen bieten. Eine Übertragungsfunktion beschreibt das Verhalten eines linearen zeitinvarianten Systems im Frequenzbereich und stellt die Beziehung zwischen der Eingangs- und der Ausgangsgrösse in Form eines Bruchs rationaler Polynome dar. Die Übertragungsfunktion G(s) eines Systems wird durch die Anwendung der Laplace-Transformation auf die Differentialgleichung gebildet, wobei die Anfangsbedingungen auf null gesetzt werden. Dies führt zu einer algebraischen Darstellung im s-Bereich.

$$G(s) = \frac{Y(s)}{U(s)} \qquad (1)$$

Hierbei sind Y(s) und U(s) die Laplace-transformierten Ausdrücke der Ausgangs- bzw. Eingangsgrössen. [5]

2.1.1 Äquivalenz von Differentialgleichungen und Übertragungsfunktionen

Es ist wichtig zu betonen, dass Differentialgleichungen und Übertragungsfunktionen äquivalente Darstellungen eines Systems sind. Die Umwandlung von einer Darstellung zur Anderen ist in beide Richtungen möglich. Durch die Anwendung der Laplace-Transformation auf die Differentialgleichung wird die Übertragungsfunktion gebildet. Durch Rücktransformation und Umschreiben der Übertragungsfunktion können die entsprechenden Differentialgleichungen gebildet werden. [5]

2.2 PID-Regler

Der PID-Regler (Proportional-Integral-Derivative-Regler) ist ein fundamentaler Bestandteil in der Regelungstechnik, der in zahlreichen industriellen und technischen Anwendungen verwendet wird, wie für Temperaturregelung oder Motorsteuerung. Er bietet eine effiziente Methode zur Steuerung von Systemen, indem er kontinuierlich die Regelabweichung minimiert, also den Fehler zwischen einem gewünschten Sollwert und dem aktuellen Istwert. Die Stärke des PID-Reglers liegt darin, durch Anpassung der Regelparameter ein optimales Regelverhalten für beliebige Systeme zu erreichen. [6]

2.2.1 Funktionsweise

Ein PID-Regler kombiniert drei verschiedene Regelungsstrategien: proportional, integral und differential. Diese drei Komponenten arbeiten zusammen, um ein System auf einen Sollwert zu regeln.

2.2.1.1 Proportional-Anteil (P)

Der P-Anteil eines PID-Reglers ist direkt proportional zum aktuellen Fehler. Der Fehler e(t) wird mit einem konstanten Verstärkungsfaktor Kp multipliziert, Formel 2. [6]

$$u(t) = Kp * e(t) \qquad (2)$$

2.2.1.2 Integral-Anteil (I)

Der I-Anteil summiert den Fehler über die Zeit und berücksichtigt somit vergangene Fehler. Dies hilft, systematische Abweichungen (Offset) zu eliminieren, die ein reiner Proportional-Regler nicht ausgleichen kann. Der Integral-Anteil erhöht die Systemantwort kontinuierlich, bis der Fehler Null ist, was insbesondere bei stationären Zuständen wichtig ist. [6]

2.2.1.3 Derivativ-Anteil (D)

Der D-Anteil basiert auf der Änderungsrate des Fehlers. Er wirkt vorausschauend, indem er Beschleunigung in der Änderungsrate abbremst, insbesondere um überschwingende Systeme zu dämpfen. [6]

2.2.2 PID-Signal

Das Regelsignal veranschaulicht, wie es die Sprungantwort auf den Sollwert 1 regelt. Das Regelsignal ist auf die Stellgrösse [-2, 2] limitiert.

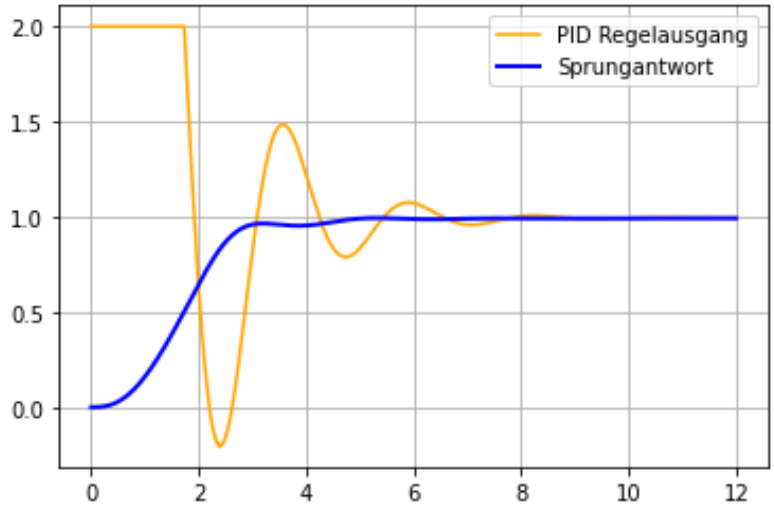


Abbildung 1 PID-Regelsignal und Sprungantwort

2.2.3 Struktur

Ein PID-Regler kann grundsätzlich auf zwei verschiedene Arten implementiert werden. Die Parallelstruktur und die Reihenstruktur. Der Hauptunterschied besteht darin, dass bei der Parallelstruktur alle Regel-Anteile unabhängig voneinander berechnet werden. Wohingegen bei der Reihenstruktur, der Ausgang des P-Anteils als Eingang für den I und D Anteil dient. Bei gleichen Regelparametern ist das Ausgangssignal beider Strukturen identisch.

2.2.3.1 Parallelstruktur

In der Parallelstruktur werden die Regelanteile (PID) unabhängig voneinander berechnet und dann addiert. Die drei Regelparameter Kp, Ki, Kd, sind die Verstärkungsfaktoren für die jeweiligen Anteile.

Der P-Anteil bildet sich aus dem Proportionalbeiwert Kp multipliziert mit dem Fehler e(t).

$$P = Kp * e(t) \qquad (3)$$

Der I-Anteil ist gleich dem Integralbeiwert Ki mal dem Integral des Fehlers seit Start.

$$I = Ki * \int_0^t e(t)\,dt \qquad (4)$$

Der D-Anteil bildet sich aus dem Differenzialbeiwert Kd multipliziert mit der Ableitung des Fehlers.

$$D = Kd * \frac{de(t)}{dt} \qquad (5)$$

Der vollständige PID-Regler summiert die drei Anteile. Der Reglerausgang u(t) setzt sich wie folgt zusammen, Formel 6.

$$u(t) = PID = Kp * e(t) + Ki \int_0^t e(\tau)\, d\tau + Kd \frac{de(t)}{dt} \qquad (6)$$

Alle Anteile erhalten denselben Input, nämlich den Fehler und sind somit parallelgeschaltet. Ihre Ausgänge werden summiert. Kp verstärkt den Fehler direkt (P-Anteil). Die Übertragungsfunktion $\frac{1}{s}$ integriert den Fehler und gewichtet entsprechend Ki (I-Anteil). Durch $\frac{s}{0.01s+1}$ wird der Fehler differenziert und bildet multipliziert mit Kd den D-Anteil. [7]

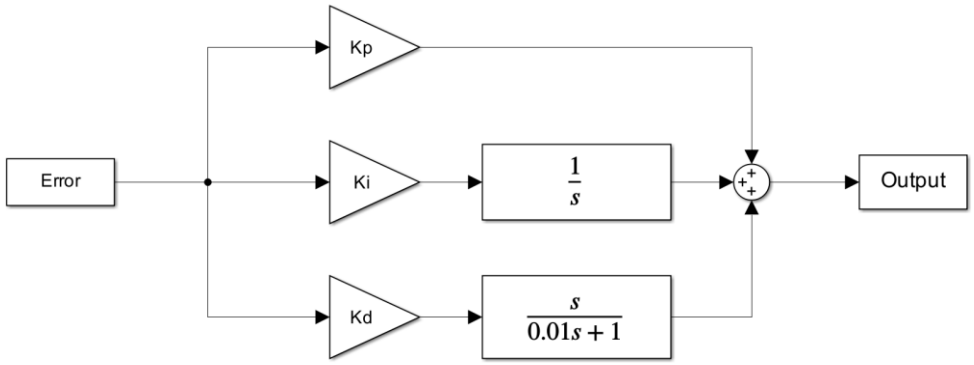


Abbildung 2 Schaltbild PID-Regler in Parallelstruktur

2.2.3.2 Reihenstruktur

Die zweite Art einen PID-Regler darzustellen ist die Reihenstruktur. Hierbei dient der Ausgang des P-Anteils als Eingang für den I und den D-Anteil. I und D werden in Reihe zu P geschaltet. Somit fallen die unabhängigen Beiwerte Ki und Kd weg. In der Reihenstruktur verwendet man stattdessen die Nachstellzeit Ti und die Vorhaltzeit Td. Der Proportionalbeiwert Kp bleibt wie gewohnt.

Der I-Anteil wird neu über die Nachstellzeit Ti geregelt und berechnet sich wie folgt.

$$I = \frac{Kp}{Ti} * \int_0^t e(\tau)\, d\tau \qquad (7)$$

Der D-Anteil ist proportional zu Kp und Td.

$$D = Kp * Td * \frac{de(t)}{dt} \qquad (8)$$

Der vollständige PID-Regler in Reihenstruktur berechnet sich nun gemäss Formel 9.

$$u(t) = PID = Kp * [e(t) + \frac{1}{Ti} \int_0^t e(\tau)\, d\tau + Td \frac{de(t)}{dt}] \qquad (9)$$

2.2.3.2.1 Nachstellzeit Ti

Die Nachstellzeit Ti ist die Zeitspanne, über die der Integralanteil eines PID-Reglers den Fehler aufsummiert, um systematische Abweichungen zu korrigieren. Sie bestimmt die Geschwindigkeit der Fehlerkorrektur. Ti kann zu Ki umgerechnet werden gemäss Formel 10. [7]

$$Ki = \frac{Kp}{Ti} \qquad (10)$$

Die Gewichtung des I-Anteils Ki wird umgekehrt proportional von Ti beeinflusst. Kleine Werte führen also zu einem hohen Wert für Kd und umgekehrt. Eine kurze Nachstellzeit führt zu schnellen, jedoch potenziell instabileren Reaktionen, wohingegen eine lange Nachstellzeit zu langsameren und stabileren Reaktionen führt. [7]

2.2.3.2.2 Vorhaltzeit Td

Die Vorhaltzeit Td gibt an wie weit der Differentialanteil die Änderung des Fehlers berücksichtigt. Eine lange Vorhaltzeit verstärkt die Reaktion auf schnelle Änderungen des Fehlers. Das System wird empfindlicher gegenüber schnellen Schwankungen, was zu einer stärkeren Dämpfung und einer Reduktion des Überschwingens führt. Allerdings kann dies auch zu einer verzögerten Reaktion des Systems führen. Eine kurze Vorhaltzeit reduziert den Einfluss auf schnelle Änderungen, wodurch das System schneller reagieren kann. Jedoch kann dies dazu führen, dass das System weniger stabil ist und zum Überschwingen neigt. Td lässt sich auch zu Kd umrechnen gemäss Formel 11. [7]

$$Kd = Kp * Td \qquad (11)$$

2.2.3.2.3 Schaltbild

Die Regelabweichung dient als Eingangssignal für den Regler, verstärkt durch Kp, bildet sie wie gewohnt den P-Anteil. Als Eingang für den I und D Anteil dient nun der P-Anteil, somit ist I und D in Reihe mit P geschaltet. Im Integral-Anteil wird das Signal durch die Nachstellzeit Ti geteilt und anschliessend auf integriert. Im D-Anteil wird das Signal nochmals durch den Faktor der Vorhaltzeit verstärkt und anschliessend differenziert.

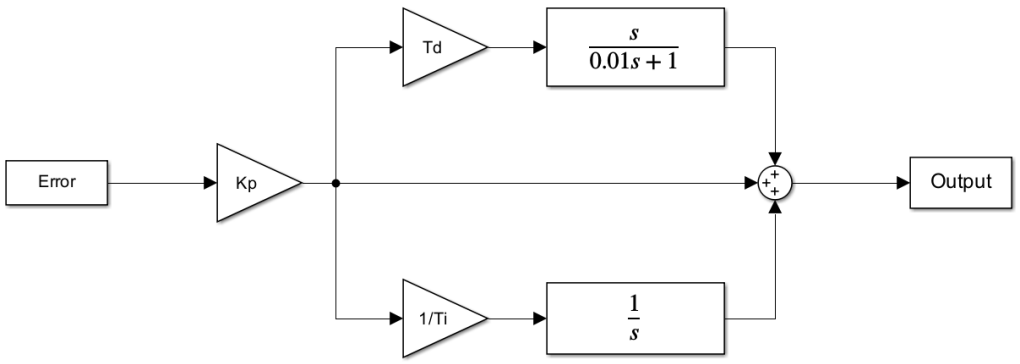


Abbildung 3 Schaltbild PID-Regler als Reihenstruktur

2.3 Stellgrössenbeschränkung

In der Realität kann die Steuergrösse nicht beliebige Werte annehmen. Betrachten wir einen Elektromotor ist die Stellgrösse auf eine bestimmte Spannung begrenzt. Die Ausgangsgrösse des Reglers muss auf einen bestimmten Wertebereich limitiert werden. Dadurch wird der Regelkreis jedoch nichtlinear. Wir gehen in jedem Beispiel von einer spezifischen Stellgrössenbeschränkung aus.

2.4 Identifikation eines überschwingenden Systems 2. Ordnung

Die Sprungantwort eines gedämpften, überschwingenden Systems 2. Ordnung, dargestellt in Abbildung 4, bildet die Basis, um die Parameter eines solchen Systems zu identifizieren.

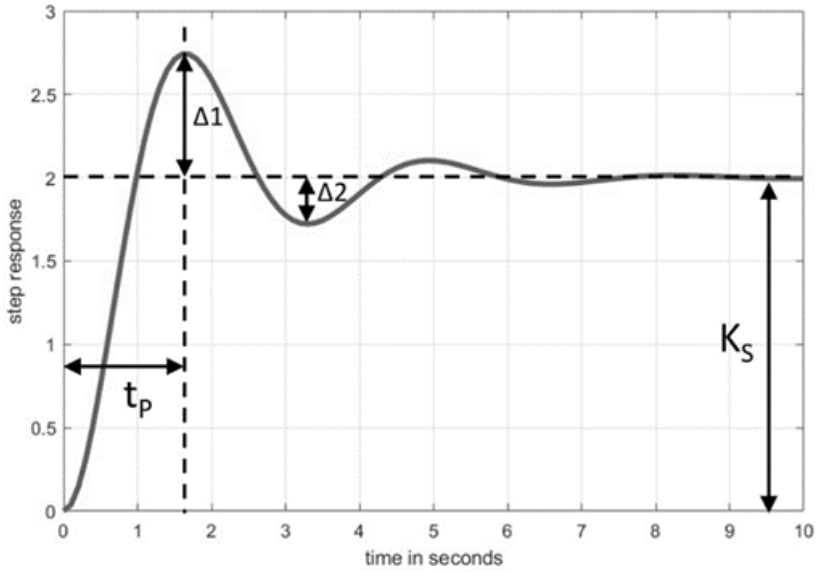


Abbildung 4 Sprungantwort eines gedämpften Systems zweiter Ordnung.

Diese überschwingenden Systeme zweiter Ordnung, die in praktischen Anwendungen häufig vorkommen, zeichnen sich durch ihr erstes Überschwingen, Δ1, relativ zu ihrem stationären Endwert aus. Für das betrachtete System wird dies als Δ1 = 0,37 dargestellt. Die Korrelation zwischen Δ1 und dem Dämpfungsfaktor des Systems, D, ist durch die Formel 12 definiert.

$$\Delta 1 = exp\left(-\frac{D \cdot \pi}{\sqrt{1-D^2}}\right) \qquad (12)$$

Formel (12) aufgelöst nach dem Dämpfungsfaktor D.

$$D = -\frac{ln\,(\Delta 1)}{\sqrt{\pi^2+(ln(\Delta 1))^2}} \qquad (13)$$

Die typische Übertragungsfunktion G(s), ist gegeben durch Formel 14.

$$G(s) = \frac{K_S}{T^2 \cdot s^2+2 \cdot D \cdot T \cdot s+1} \qquad (14)$$

Dabei ist Ks der Verstärkungsfakor, T die Schwingungsdauer und D der Dämpfungsfaktor. [8]

Als Beispiel dient Ks = 1, T = 1, D = 0.3. Daraus ergibt sich folgende Übertragungsfunktion.

$$G(s) = \frac{1}{s^2+0.6s+1} \qquad (15)$$

Mit der gegebenen Übertragungsfunktion (15) erhalten wir das folgende Blockschaltbild (Abbildung 5)

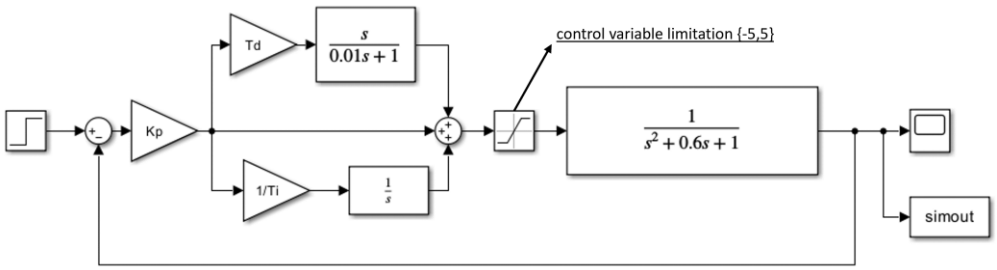


Abbildung 5 Blockschaltbild, überschwingendes System 2. Ordnung.

2.5 Identifikation eines PTn Systems

PTn-Systeme, hier als Beispiel ein PT3-System, werden verwendet, um zeitverzögerte Systeme zu modellieren. Die Übertragungsfunktion eines PT3-Systems wir durch Formel 16 beschrieben. [8]

$$G(s) = \frac{1}{T_1 * s+1} * \frac{1}{T_2 * s+1} * \frac{1}{T_3 * s+1} * K \qquad (16)$$

In Formel 16 repräsentieren T1, T2, T3 die Zeitkonstanten des Systems und K ist der statische Verstärkungsfaktor. Das in diesem Projekt untersuchte System verwendet T1 = T2 = T3 = K = 1, wodurch sich die Übertragungsfunktion zu Formel 17 vereinfacht.

$$G(s) = \frac{1}{s+1} * \frac{1}{s+1} * \frac{1}{s+1} \qquad (17)$$

Mit der gegebenen Übertragungsfunktion erhalten wir das folgende Blockschaltbild (Abbildung 6).

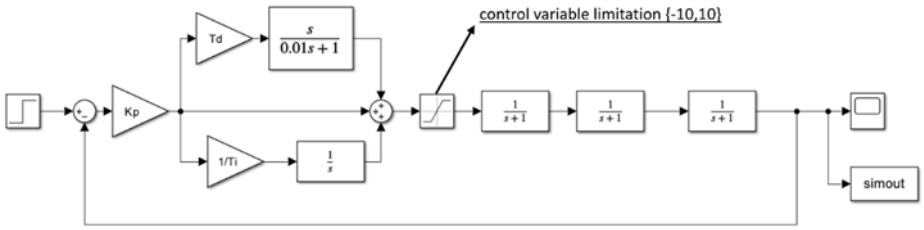


Abbildung 6 Blockschaltbild, PT3 System

2.6 Regelkreis

Der fertige Regelkreis, wie er in dieser Arbeit betrachtet wird, setzt sich nun zusammen aus einem Sprungsignal das abzüglich dem Ausgangssignal den Fehler bildet, der als Eingang für den PID-Regler in Reihenstruktur dient. Das Regelsignal wird auf eine Stellgrösse beschränkt, und das limitierte Signal dient als Eingang für die Übertragungsfunktion des Systems.

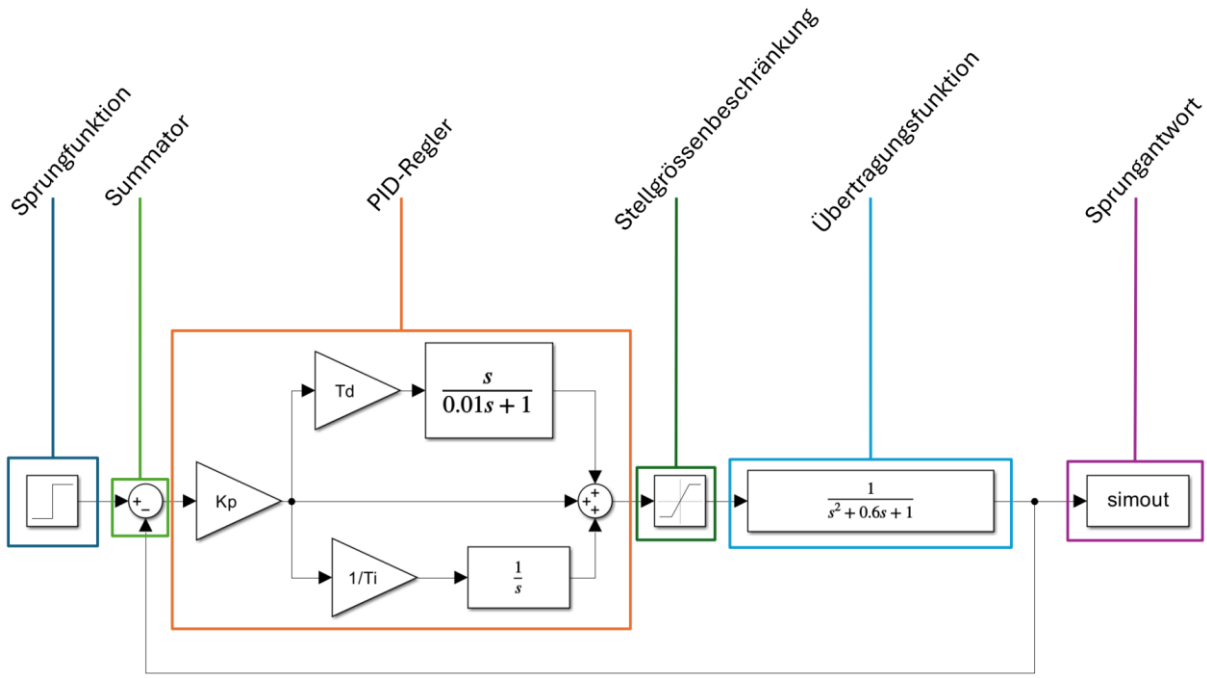


Abbildung 7 Aufbau Regelkreis

2.7 ITAE Kriterium

Um die Regelung eines Systems zu bewerten, gibt es verschiedene Kriterien, welche berechnet werden können. Die Sprungantwort ist die Grundlage, aus der ein Fehlerkriterium berechnet wird. Jedes Fehlerkriterium zielt darauf ab, ein Mass zu geben, wie gut, wie schnell der stationäre Sollwert erreicht wird. Das IAE-Kriterium ist beispielweise das Integral des Fehlers. Das ITAE Kriterium (Integral of Time-weighted Absolute Error) ist eine Erweiterung des IAE, da zusätzlich noch nach der Zeit gewichtet wird. Fehler zu einem späten Zeitpunkt werden stärker gewichtet, als Fehler zu einem frühen Zeitpunkt. Somit werden frühe Abweichung toleriert, um möglichst schnell den stationären Sollwert zu erreichen. Formel 18 zeigt die Berechnung des ITAE aus dem Fehler e(t). [9]

$$ITAE: \int_0^\infty |e(t)| \cdot t \cdot dt \qquad (18)$$

2.8 PSO (Particle Swarm Optimization)

PSO ist eine Optimierungsmethode aus der Klasse der genetischen Algorithmen. Sie wurde erstmals 1995 eingeführt. PSO ist an das Verhalten von Vogelschwärmen, oder Fischschwärmen, angelehnt. PSO verwendet ausgeklügelte Mechanismen, um lokalen Minima zu entkommen. Da jedoch die zu optimierende Funktion als Blackbox behandelt wird, kann, wie bei jedem Optimierungsalgorithmus, nie mit Sicherheit gesagt werden, ob das gefundene Minimum tatsächlich das globale Minimum ist. Im PSO-Algorithmus ist ein Partikel eine mögliche Lösung für das Optimierungsproblem. Die Eingabeparameter entsprechen den Koordinaten des Partikels, für jeden Parameter eine Dimension. Der zu minimierende oder

maximierende Ergebniswert wird Zielfunktion genannt. Eine bestimmte Anzahl von Partikeln wird im Suchraum verteilt. Jedes Partikel hat eine Initialgeschwindigkeit, mit der es sich durch den Suchraum bewegt. Eine Iteration erfolgt, wenn die Koordinaten aller Partikel gemäss ihrer Geschwindigkeit aktualisiert werden und die Zielfunktion an der neuen Position mit den neuen Parametern ausgewertet wird. Jedes Partikel merkt sich die beste Position, an der es sich je befunden hat, genannt PersonalBest pB. Jedes Partikel kennt auch den besten globalen Wert und Koordinaten, den ein Partikel im Schwarm jemals erreicht hat, genannt GlobalBest gB. Mit jeder Iteration werden auch die Geschwindigkeiten der Partikel so angepasst, dass das Partikel in Richtung pB und gB beschleunigt wird. Wie stark ein Partikel zu pB oder gB hingezogen wird, wird durch die Hyperparameter definiert [10].

2.8.1 Hyperparameter

PSO verfügt selbst über vier Parameter, um den Algorithmus zu kalibrieren.

Trägheit w besagt wie stark der aktuelle Schwung, die Geschwindigkeit, erhalten bleibt.
Kognitiver Gewichtungsfaktor c_1 bestimmt die Beschleunigung in Richtung PersonalBest.
Sozialer Gewichtungsfaktor c_2 bestimmt die Beschleunigung in Richtung GlobalBest.
Schwarmgrösse S ist die Anzahl Partikel im Schwarm.

2.8.2 Formel

Die neue Position ergibt sich durch Addieren des Geschwindigkeitsvektor.

$$x_{i+1} = x_i + v_i \qquad (19)$$

Der neue Geschwindigkeitsvektor wird mit Formel 20 berechnet, wobei w, c1 und c2 Hyperparameter und r1 und r2 Zufallszahlen zwischen 0 und 1 sind. pB und gB sind die Positionen des bestgefundenen persönlichen und globalen Funktionswertes. pi ist die Position des i-ten Partikels. [10]

$$v_{i+1} = w * v_i + c_1 * r_1 * (pB - p_i) + c_2 * r_2 * (gP - p_i) \qquad (20)$$

3 Vorgehen

Dieses Kapitel beschreibt wie ein Regelkreis in Python implementiert wird. Ebenso die genaue Umsetzung des PSO-Algorithmus in Python.

3.1 Portierung Regelkreis

Ein vollständiger Regelkreis wird in Python programmiert und die resultierende Sprungantwort mit der Sprungantwort des Äquivalenten Simulink Modells verglichen.

3.1.1 Implementierung PID

Die PID-Funktion nimmt zwei Parameter entgegen, der aktuelle Zeitpunkt t sowie der Systemausgang y, und liefert den Regler Ausgang PID.

```
def __pid_controller(self, t, y):
        error = self.mySetPoint - y[self.order_i]
        dt = t - self.myLastError[1]

        if dt > 0:
            error_diff = error - self.myLastError[0]
            derivative = error_diff / dt
            alpha = dt / (self.tau + dt)
            self.filtered_d = alpha * derivative + (1 - alpha) *
self.filtered_d
            self.myIntegralError += error * dt
        else:
            self.filtered_d = self.filtered_d

        P = self.myKp * error
        I = self.myKi * self.myIntegralError
        if self.myNegSat is not None and self.myPosSat is not None:
            I = np.clip(I, self.myNegSat, self.myPosSat)

        D = self.myKd * self.filtered_d
        PID = P + I + D
        if self.myNegSat is not None and self.myPosSat is not None:
            PID = np.clip(PID, self.myNegSat, self.myPosSat)
        self.myLastError = [error, t]
        return PID
```

Die Zeitdifferenz dt zwischen dem aktuellen Zeitpunkt und dem Zeitpunkt des letzten Fehlers wird berechnet. Falls dt grösser als null ist, was bedeutet, dass seit der letzten Fehlerberechnung Zeit vergangen ist, berechnet die Funktion die Änderungsrate des Fehlers (derivative), indem sie die Differenz zwischen dem aktuellen Fehler und dem zuletzt gespeicherten Fehler durch dt teilt. Diese Ableitung wird dann in self.prev_d gespeichert, um sie für den nächsten Aufruf der Funktion zu bewahren. Ausserdem wird das Integral des Fehlers (self.myIntegralError) durch Multiplikation des aktuellen Fehlers mit dt aktualisiert. Falls dt nicht grösser als null ist, was typischerweise nur beim ersten Funktionsaufruf oder bei sehr schnellen

Aufrufen passiert, wird der letzte gespeicherte Wert der Ableitung verwendet. Die Funktion berechnet dann die drei Komponenten des PID-Reglers.

Proportional-Anteil P wird als Produkt des Verstärkungsfaktor Kp (self.myKp) und des aktuellen Fehlers berechnet.

Integral-Anteil I errechnet sich aus dem Produkt des Integral-Koeffizienten (`self.myKi`) und des akkumulierten Fehlerintegrals. Um den Effekt des „Wind-ups“ zu vermeiden, bei dem sich der Integralanteil aufbauen kann, obwohl die Stellgrösse bereits ihre maximale oder minimale Grenze erreicht hat, wird I mithilfe der Funktion np.clip auf die Werte der Stellgrösse beschränkt. Diese Begrenzung verhindert, dass der Integral-Anteil zu extremen Werten anwächst, die nicht mehr zur Verbesserung der Regelung beitragen können. [11]

Differential-Anteil D wird als Produkt des Differential-Koeffizienten Kd (self.myKd) und der berechneten Ableitung (derivative) bestimmt.

PID-Wert ergibt sich aus der Summe von P, I und D. Falls Stellgrössenbeschränkungen für den Regelausgang vorgesehen sind, wird der PID-Wert auf die angegebenen Grenzen beschränkt.

Am Ende der Funktion wird der aktuelle Fehler zusammen mit dem aktuellen Zeitpunkt in self.myLastError gespeichert, für zukünftige Aufrufe, und der berechnete PID-Ausgang wird zurückgegeben.

3.1.2 Systeme

In dieser Arbeit wurden insgesamt vier verschiedene Systeme modelliert. Ein überschwingendes System 2. Ordnung und ein PT3-Glied wurden implementiert mit den gleichen Systemeigenschaften wie in den Arbeiten [3], [4], um eine Basis zu haben um die unterschiedlichen Implementierungen zu validieren. Zusätzlich wurden als praxisnahe Beispiele ein Motor der einen Werkzeugschlitten führt, sowie ein Wärmeübertragungssystem modelliert. Die Systeme wurden in Python programmiert. Mithilfe der Python Bibliothek Scipy ist es möglich effizient Differentialgleichungen zu lösen.

3.1.2.1 Scipy

Scipy ist eine offizielle Python Bibliothek, die Möglichkeiten bietet um ein Anfangswertproblem, ein System von gewöhnlichen Differentialgleichungen, zu lösen.

3.1.2.1.1 Funktion

Eine Python-Funktion muss bereitgestellt werden, welche den Integrationszeitpunkt t sowie den Zustand y zum Zeitpunkt t als Parameter entgegennimmt. Und die zeitliche Ableitung des Zustands y zur Zeit t als Rückgabe liefert. Die Methodensignatur ist wie folgt:
fun(t: float, y: List[float]): List[float]
Die Dimension von y, sowie die Dimension des Rückgabewerts, ist gleich wie die Dimension von y_0 der Initial-Zustand, Dimension der Anfangswerte. [12]

3.1.2.1.2 Solver

Scipy.integrate.solve_ivp ist die Integrationsmethode zum Lösen von Anfangswertproblemen. Es können verschiedene ODE-Solver konfiguriert werden. In dieser Arbeit wird RK45 verwendet basierend auf dem Runge-Kutta-Verfahren fünfter Ordnung. RK45 verwendet adaptive Schrittweiten und bietet eine hohe Genauigkeit. Zudem wird die maximale Schrittgrösse auf 0.01 beschränkt, grössere Schritte kann der Solver nicht wählen. Die Wahl des Verfahrens und der maximalen Schrittgrösse ist eine Abschätzung zwischen Genauigkeit und Rechenaufwand. Die gewählte Konfiguration liefert genau Ergebnisse mit überschaubarem Rechenaufwand.

3.1.2.2 Klassendiagramm

Jedes System wird durch eine Klasse repräsentiert. Jede «System» Klasse erbt und erweitert die abstrakte Basisklasse ClosedLoop. In der Basisklasse sind alle Variablen und Funktionen gespeichert, die jeder beschriebene Regelkreis hat. Der PID-Regler, die Stellgrössenbeschränkung, Berechnung der Systemantwort, etc. Eine konkrete «System» Klasse muss zusätzlich noch das Interface ClosedLoopSystem implementieren.

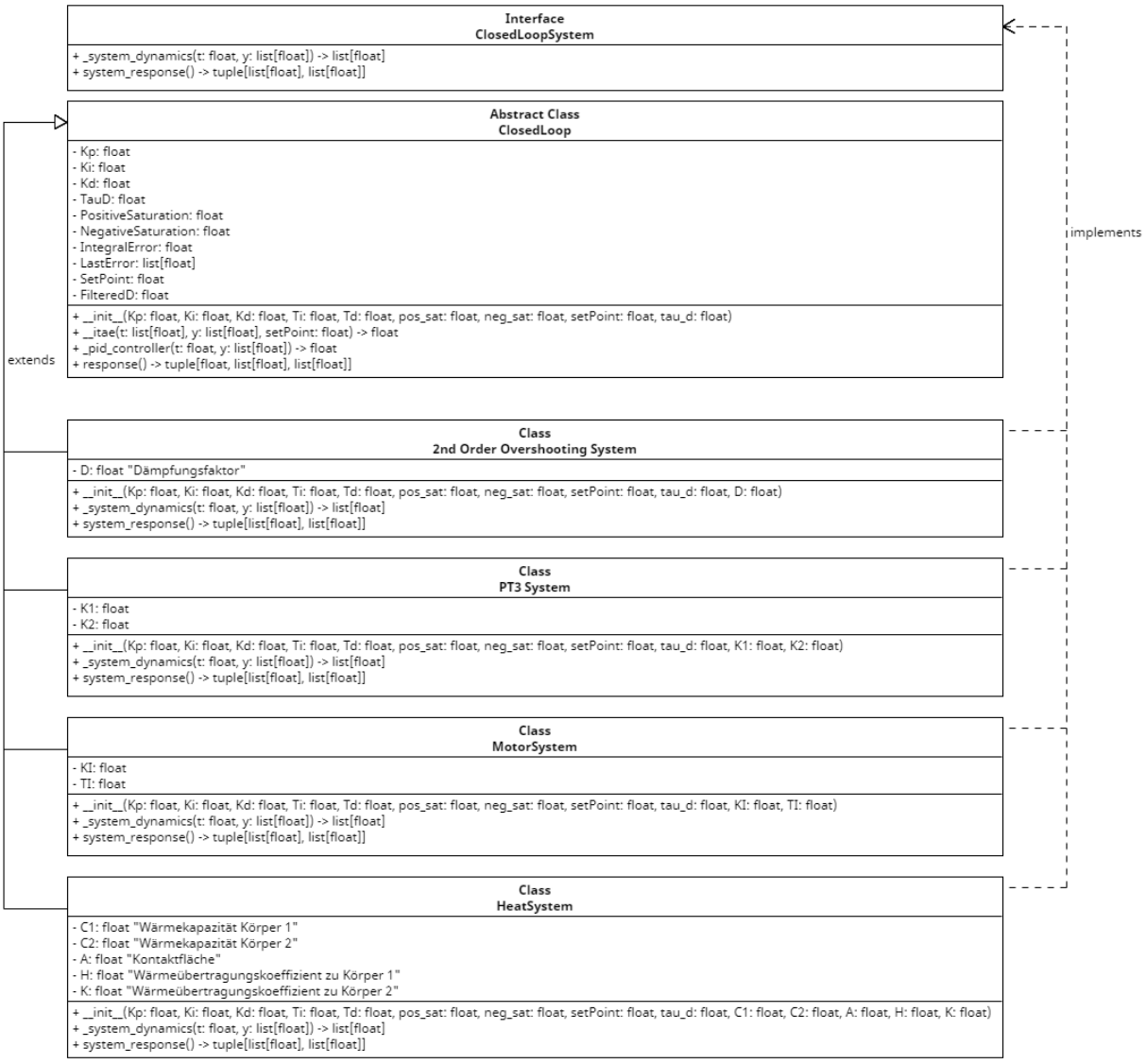


Abbildung 8 Klassendiagramm Systeme

3.1.2.2.1 Basisklasse ClosedLoop

Bei der Basisklasse ClosedLoop handelt es sich um eine abstrakte Klasse, da keine direkten Instanzen der Klasse erstellt werden. Es handelt sich um ein Grundgerüst um einen geschlossenen Regelkreis zu simulieren, Variablen und Funktionen, welche alle Systeme teilen, sind hier abgelegt. Zum Beispiel, die Regelparameter Kp, Ki, Kd, die Stellgrössenbeschränkung oder der Sollwert. Auch die Berechnung des ITAE-Kriteriums oder des PID-Ausgangs sind hier implementiert, da die Berechnung für jedes System identisch ist.

3.1.2.2.2 Interface ClosedLoopSystem

Das ClosedLoopSystem Interface ist die Ergänzung zur Basisklasse, um einen vollständigen Regelkreis zu bilden. Das Interface besagt, dass ein System neben der Basisklasse noch zwei weitere Funktionen implementieren muss.

3.1.2.2.2.1 _system_dynamics(t, [y1, y2, ...]) -> [y1, y2, ...]
Die Funktion beschreibt die eigentliche Differentialgleichung. Über diese Funktion integriert der Integrationsalgorithmus. Deshalb muss sie auch von jedem System individuell implementiert werden. Die Rückgabe der Funktion ist die rechte Seite der Differenzialgleichungen, die Änderungsraten der Zustandsvariablen als Array. Die Eingangsparameter sind die Zustandsvariablen y zum Zeitpunkt t. Die Zeitpunkte t, wann system dynamics ausgewertet wird, wird durch den Integrationsalgorithmus bestimmt.

3.1.2.2.2.2 system_response() -> [[t1, t2, ...], [y1, y2, ...], itae]
Diese Funktion simuliert die Sprungantwort des initialisierten Systems. Die Anfangszustände werden initialisiert sowie die Integrationsdauer gesetzt. Die Sprungantwort wird als zwei Arrays (Zeitpunkte t, Signal y) zurückgegeben. Der dritte Rückgabewert ist das, aus der Sprungantwort berechnete, ITAE-Kriterium.

3.1.2.2.3 System Klassen

Alle Klassen, welche das ClosedLoopSystem Interface implementieren gelten als Systemklassen, sie repräsentieren einen Regelkreis. Parameter die das System beschreiben werden als Klassenvariablen gespeichert, es wird immer davon ausgegangen, dass die Werte in SI-Einheiten vorliegen. Die Funktion system_dynamics liefert dann die Änderung der Zustandsvariablen gemäss den Klassenvariablen. Beim Erstellen eines neuen Systems können entweder Ki und Kd oder Ti und Td angegeben werden. Ti und Td werden entsprechend Formel 10 & 11 in Ki, Kd umgerechnet, da das Programm intern mit Ki, Kd rechnet.

In dieser Arbeit wurden insgesamt vier Systeme modelliert. Ein überschwingendes System 2. Ordnung, ein PT3-Glied, ein System der Wärmeübertragung sowie ein Motor der einen Werkzeugschlitten führt.

3.1.2.2.3.1 Überschwingendes System 2. Ordnung
Die Übertragungsfunktion ist bereits durch Formel 14 bekannt. Mit der inversen Laplace-Transformation erhalten wir die Differentialgleichungen, abhängig vom Verstärkungsfaktor Ks, der Schwingungsdauer T und dem Dämpfungsfaktor D.

$$\dot{y_1} = y_2 \qquad (21)$$

$$\dot{y_2} = \frac{Ks*u(t) - y_1 - 2*D*T*y_2}{T^2} \qquad (22)$$

Das System wird durch drei Parameter beeinflusst. Die Werte wurden wie folgt gewählt.

Tabelle 1 Systemparameter, Überschwingendes, gedämpftes System 2. Ordnung.

Symbol	Wert	Beschreibung
Ks	1	Verstärkungsfaktor
T	1	Schwingungsdauer
D	0.3	Dämpfungsfaktor

Die Stellgrössenbeschränkung wurde auf [-5, 5] gesetzt.

Daraus ergibt sich folgende Python-Funktion.

```
def __system_dynamics(self, t, y):
        Ks = self.Ks
        T = self.T
        D = self.D

        y1, y2 = y
        u = self.__pid_controller(t, y)
        dy1dt = y2
        dy2dt = (Ks * u - y1 - 2 * D * T * y2) / T**2
        return [dy1dt, dy2dt]
```

3.1.2.2.3.2 PT3-System

Aus der Übertragungsfunktion eines PT3-Glieds, Formel 15, erhalten wir folgendes System von Differentialgleichungen.

$$\dot{y_1} = \frac{K*u(t)-y_1}{T_1} \quad (23)$$

$$\dot{y_2} = \frac{y_1-y_2}{T_2} \quad (24)$$

$$\dot{y_3} = \frac{y_2-y_3}{T_3} \quad (25)$$

Wie bereits bekannt, wird ein PT3-Glied, durch vier Parameter beschrieben. Um die Gleichungen zu vereinfachen, wurden die Parameter auf 1 gesetzt.

Tabelle 2 Systemparameter, PT3-Glied

Symbol	Wert	Beschreibung
T_1	1	Zeitkonstante 1
T_2	1	Zeitkonstante 2
T_3	1	Zeitkonstante 3
K	1	Verstärkungsfaktor

Die Stellgrössenbeschränkung wurde auf [-10, 10] gesetzt.

```
def __system_dynamics(self, t, y):
        T1 = self.T1
        T2 = self.T2
        T3 = self.T3
        K = self.K

        y1, y2, y3 = y
        u = self.__pid_controller(t, y)
        dy1dt = (K * u - y1) / T1
        dy2dt = (y1 - y2) / T2
        dy3dt = (y2 - y3) / T3
        return [dy1dt, dy2dt, dy3dt]
```

3.1.2.2.3.3 Heat System

Dieses System regelt eine Wärmeübertragung einer externen Temperaturquelle auf Körper 1 und von Körper 1 auf Körper 2. Es wird angenommen, dass die Erhitzung der externen Temperaturquelle auf eine gewünschte Temperatur vernachlässigbar viel schneller passiert als

die Erwärmung von Körper 1 & 2. Das Regelsignal steuert eine externe Temperaturquelle, welche eine genaue Temperatur bereitstellt. Man geht von einem perfekt isolierten System aus, dass keine Wärme abgibt.

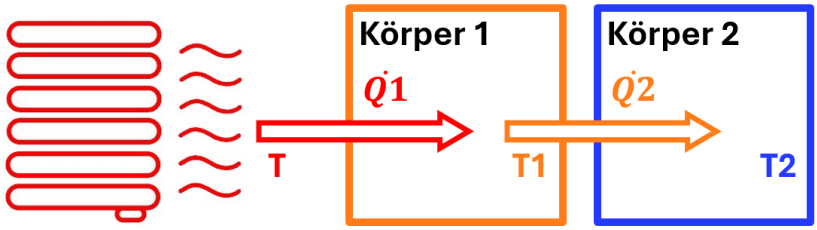


Abbildung 9 System der Wärmeübertragung

Das System wird durch sechs Parameter beschrieben. Die Parameter wurden so gewählt, dass ein vereinfachtes System beschrieben wird, bei dem eine Herdplatte einen Kochtopf gefüllt mit Wasser erhitzt. Die Stellgrössenbeschränkung wurde auf [20, 500] gesetzt. Also eine ausgeschaltete Herdplatte = 20° Celsius. Und eine maximale Temperatur von 500°. Folgende Parameter wurden verwendet.

Tabelle 3 Systemparameter, Wärmeübertragungs System.

Symbol	Wert	Beschreibung
C_1	$500\,\frac{J}{K}$	Wärmekapazität Körper 1, Pfanne
C_2	$4187\,\frac{J}{K}$	Wärmekapazität Körper 2, Wasser
A	$0.062\ m^2$	Kontaktfläche zur Wärmeübertragung
α_1	$400\ \frac{W}{m^2 * K}$	Wärmeübertragungskoeffizient 1, Herd zu Pfanne
α_2	$370\ \frac{W}{m^2 * K}$	Wärmeübertragungskoeffizient 2, Pfanne zu Wasser
m	$1\ kg$	Masse von Körper 1 & 2

Die Wärmeströme ergeben sich aus den folgenden Gleichungen.

$$\dot{Q}1 = \alpha_1 * A * (T - T_1) \qquad (26)$$

$$\dot{Q}2 = \alpha_2 * A * (T_1 - T_2) \qquad (27)$$

Zudem gelten die folgenden Beziehungen zwischen den Wärmeströmen und den Temperaturänderungen der Körper.

$$m * c_1 * \dot{T_1} = \dot{Q_1} - \dot{Q_2} \qquad (28)$$

$$m * c_2 * \dot{T_2} = \dot{Q_2} \qquad (29)$$

Somit ergeben sich folgende Differenzialgleichungen für die Temperaturänderung beider Körper.

$$\dot{T_1} = \frac{\alpha_1 * A * (T - T_1) - \alpha_2 * A * (T_1 - T_2)}{m * c_1} \qquad (30)$$

$$\dot{T_2} = \frac{\alpha_2 * A * (T_1 - T_2)}{m * c_2} \qquad (31)$$

In Python implementiert erhalten wir folgende Funktion system_dynamics.

```
def _system_dynamics(self, t, y):
        c1 = self.c1
        c2 = self.c2
        A = self.A
        alpha1 = self.alpha1
        alpha2 = self.alpha2
        m1 = self.m1
        m2 = self.m2

        T1, T2 = y
        T = self._pid_controller(t, y)
        dT1_dt = (alpha1 * A * (T - T1) - alpha2 * A * (T1 - T2)) / (m1 * c1)
        dT2_dt = (alpha2 * A * (T1 - T2)) / (m2 * c2)
        return [dT1_dt, dT2_dt]
```

3.1.2.2.3.4 Motor System

Dieses System modelliert einen Werkzeugschlitten der, angetrieben durch einen Gleichstrommotor, auf eine Ausgangsposition x geregelt werden soll. Die Eingangsgrösse ist die Klemmenspannung y. Wir führen eine Stellgrössenbeschränkung [-48, 48] ein und nehmen an, die maximal zulässige Betriebsspannung beträgt 48 Volt.

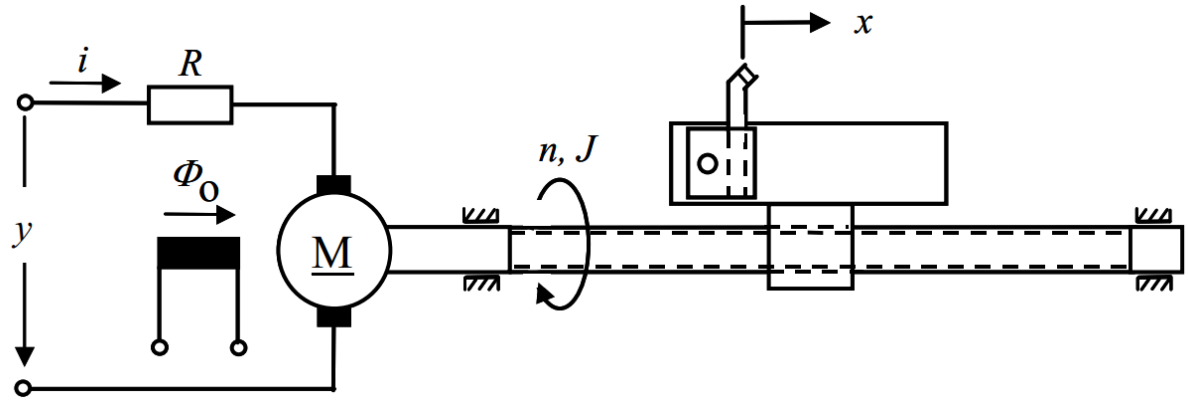


Abbildung 10 Modell Werkzeugschlitten

Das System wird durch fünf Parameter beschrieben. Es Für dieses konkrete System wurden folgende Werte angenommen.

Tabelle 4 Systemparameter, Motor System.

Symbol	Wert	Beschreibung
R	0.5 Ω	Ankerwiderstand
J	0.905 Nms^2	Trägheitsmoment der rotierenden Massen
$c\Phi_0$	0.6 Vs	Erregerfluss
n	-	Drehzahl der Motorwelle
a	4 mm/Umdrehung	Gewindesteigung

Führen wir die Abkürzungen

$$K_I = \frac{a}{2\pi c\Phi_0} = 1.061\frac{mm}{Vs} \quad (32)$$

und

$$T_I = \frac{JR}{(c\Phi_0)^2} = 1.257s \quad (33)$$

ein, so erhalten wir ein System bestimmt durch zwei Variablen, was zur Übertragungsfunktion Formel 34 führt. [13]

$$G(s) = \frac{K_I}{s(1+sT_I)} \quad (34)$$

Führen wir die Rückwärtstransformation durch, erhalten wir zwei Differentialgleichungen für die Änderungsraten der beiden Zustandsvariablen y1, y2. Mit u(t) als die geregelte Klemmenspannung zur Zeit t. [13]

$$\dot{y_1} = y_2 \quad (35)$$

$$\dot{y_2} = -\frac{1}{T_I} * y_2 + \frac{K_I}{T_I} * u(t) \quad (36)$$

Die Differentialgleichungen können nun in der system_dynamics Methode implementiert werden.

```
def _system_dynamics(self, t, y):
        KI = self.K_I
        TI = self.T_I

        y1, y2 = y
        u = self._pid_controller(t, y)
        dy1dt = y2
        dy2dt = -1/TI * y2 + KI/TI * u
```

Abbildung 11 Klassendiagramm Systeme

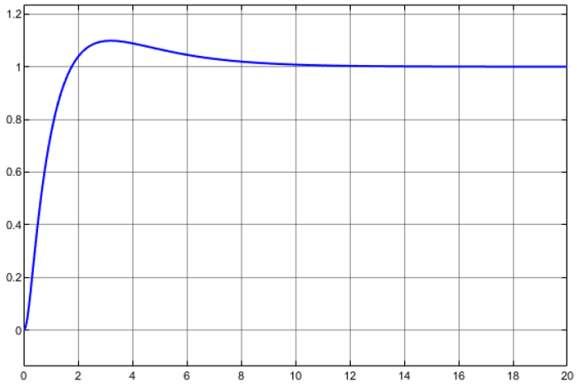
3.1.3 Validierung

In diesem Kapitel wird die Validierung der Simulationsergebnisse einer Regelkreissimulation in Python gegenüber einem bekannten Simulink-Modell betrachtet. Ziel ist es zu überprüfen, ob die Python-Simulation bei gleichen Parametern dieselbe Sprungantwort und dasselbe ITAE-Kriterium liefert wie das Simulink-Modell. Das ITAE-Kriterium dient hierbei nicht zur Bewertung der Parameter sondern als Indikator, wie ähnlich die Sprungantworten sind. Weicht das ITAE-Kriterium nur wenig ab, ist das ein Indiz, dass die Python Simulation die gleiche Sprungantwort produziert wie das Simulink Modell.

3.1.3.1 Überschwingendes System 2. Ordnung

Für das System 2.Ordnung wurden die Regelparameter: Ki = 10, Ti = 3, Td = 0.8, gewählt. Das Referenz-Simulink-Modell wurde bereits in Abbildung 5 eingeführt.
Abbildung 12 zeigt die Sprungantwort des Simulink Modells, verglichen mit der Sprungantwort der Python Simulation.

Simulink / Matlab, ITAE=2.2864



Python, ITAE=2.2705

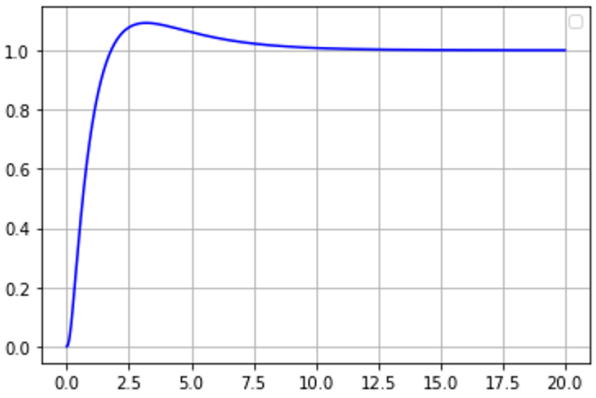


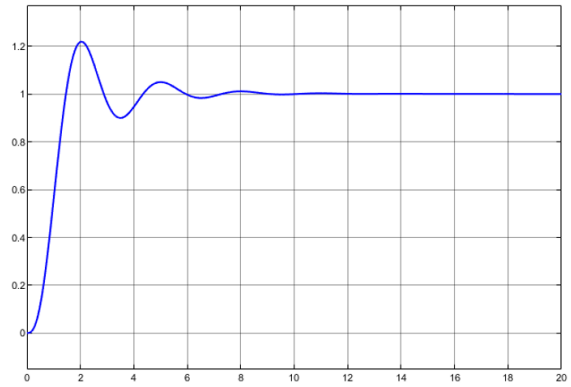
Abbildung 12 Vergleich der Sprungantworten des Simulink Modells zur Python-Simulation, Überschwingendes, gedämpftes System 2. Ordnung.

Die beiden Kurven sind von der Form nicht zu unterscheiden. Das ITAE-Kriterium unterscheidet sich um 1%.

3.1.3.2 PT3-Glied

Für das PT3-Glied wurden die Regelparameter: Kp = 8, Ti = 6, Td = 0.5, gewählt.
Das Referenz-Simulink-Modell wurde bereits in Abbildung 6 eingeführt.
Abbildung 13 zeigt die Sprungantwort des Simulink Modells, verglichen mit der Sprungantwort der Python Simulation.

Simulink / Matlab, ITAE=1.8547



Python, ITAE=1.8552

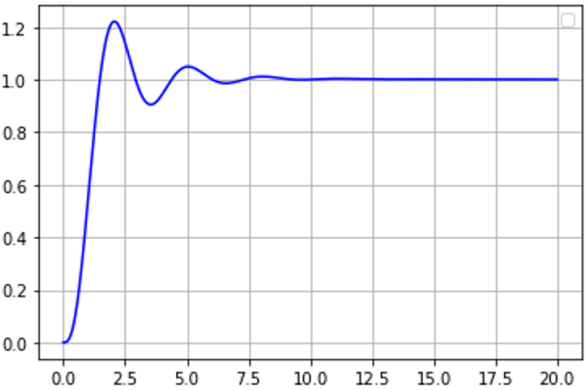


Abbildung 13 Vergleich der Sprungantworten des Simulink Modells zur Python-Simulation, PT3-Glied.

Wieder sind die beiden Kurven in der Form kaum zu unterscheiden. Die Abweichung des ITAE-Kriteriums beträgt wieder 1%.

3.1.3.3 Fazit

Die Validierung der Python-Simulation gegenüber dem Simulink-Modell zeigt, dass die Sprungantworten von Auge nicht zu unterscheiden sind. Eventuelle minimale Verzerrungen können höchstens auf verschiedene Plot-Einstellungen in MATLAB und Python zurückgeführt werden. Das ITAE-Kriterium unterscheidet sich zwischen den beiden Sprungantworten um lediglich 1%. Diese geringe Abweichung wird als akzeptabel angesehen, da ein komplett identisches ITAE-Kriterium, aufgrund unterschiedlicher Integrationsverfahren in MATLAB und Python, unwahrscheinlich ist.

Insgesamt bestätigt die Validierung, dass die Python-Simulation bei gleichen Parametern dieselben Ergebnisse wie das Simulink-Modell liefert und somit den Regelkreis korrekt modelliert hat.

3.2 PSO Portierung

Dieser Abschnitt behandelt die Implementierung des PSO-Algorithmus.

3.2.1 Anwendung auf PID-Regelkreise

Für jedes System resultiert ein Satz von Regelparametern (Kp, Ki, Kd) in einer eindeutigen Sprungantwort, aus der sich wiederum ein eindeutiges ITAE-Kriterium berechnen lässt. Daher kann die Simulation eines PID-Regelkreises als Funktion angesehen werden, welche die Regelparameter als Eingang nimmt und das ITAE-Kriterium als Rückgabewert liefert, Formel 37.

$$R^3 \rightarrow R^1, Kosten(Kp, Ti, Td) \rightarrow ITAE \qquad (37)$$

Es handelt sich um eine Funktion die aus dem R^3 auf einen Skalar R^1 abbildet. Ähnlich wie die Temperaturverteilung im Raum, wenn jedem Punkt im Raum eine Temperatur zugeordnet wird. Diese Berechnung des ITAE-Kriterium aus den Regelparameter dient als Kostenfunktion für den PSO-Algorithmus. Die Koordinaten eines Partikels sind die Regelparameter Kp,Ti,Td. Man kann sich einen Raum vorstellen mit Kp,Ti,Td als Achsen. Aus der Position eines Partikels also (Kp,Ti,Td) kann dann das ITAE-Kriterium berechnet werden. Der PSO-Algorithmus zielt darauf ab, Regelparameter (Positionen) zu finden, die zu einem möglichst minimalen ITAE-Kriterium führen.

3.2.2 Zeitkomplexität

Die Auswertung der Kostenfunktion eines Partikels ist der rechenintensivste Teil des Algorithmus. Aus den Regelparameter wird die Sprungantwort des Systems berechnet und daraus das ITAE-Kriterium berechnet. Der Rechenaufwand der übrigen Schritte des Algorithmus ist vernachlässigbar klein. Wir definieren die Auswertung der Kostenfunktion eines Partikels, also aus den Regelparametern das zugehörige ITAE-Kriterium berechnen, als fundamentale Einheit zur Messung des Zeitaufwands. Fortan wird das als Auswertung der Kostenfunktion bezeichnet. Da die Zeit, welche benötigt wird, um die Bewertungsfunktion einmal auszuwerten, von der verwendeten Hardware abhängt, können keine genauen Zeitangaben bezüglich der Rechenzeit gemacht werden. Es sei nur gesagt, dass der in dieser Arbeit verwendete handelsübliche 8-Kern Prozessor 10 Auswertungen pro Sekunde schafft. Im Kontext des PSO-Algorithmus bedeutet das, dass in jedem Iterationsschritt, die Kostenfunktion jedes Partikels im Schwarm einmal ausgewertet wird. Mit einer Schwarmgrösse S und einer Anzahl Iterationen I, wird die Kostenfunktion insgesamt F Mal ausgewertet.

$$F = (I + 1) * S \qquad (38)$$

Das +1 in der Klammer ergibt sich, da auch die Anfangsposition der Partikel ausgewertet wird.

3.2.3 Suchraum

Der Suchraum von PSO muss beschränkt werden. Für jede Dimension, jeden Parameter, werden Grenzen eingeführt. Die Position der Partikel wird also beschränkt. Grundsätzlich sollte der Suchraum so klein wie möglich gewählt werden, um die Konvergenz des Verfahrens zu verbessern. Werden die Grenzen des Suchraums jedoch zu klein oder schlecht gewählt, ist es möglich, dass ausserhalb des Suchraums bessere Parameter zu finden sind. Es ist nicht möglich

eine allgemeine Aussage über die Wahl geeigneter Grenzen zu machen, dies ist stark vom jeweiligen System abhängig. Durch technische Kenntnisse des Systems soll der Suchraum sinnvoll beschränkt werden. Für die in dieser Arbeit untersuchten, Systeme werden spezifische Suchräume definiert.

Für das System 2. Ordnung und das PT3-Glied wurde bereits in früheren Arbeiten nach optimalen Regelparametern gesucht, dabei wurden für alle Parameter Grenzen von 0 bis 10 verwendet. Mit Ausnahme für Ti wurde 0.1 als untere Grenze gesetzt, um Division durch Null zu vermeiden. Um die Vergleichbarkeit der Ergebnisse zu gewährleisten, definieren wir für unsere Referenzsysteme (2. Ordnung und PT3) ebenfalls die Parametergrenzen von 0-10.

Für das Wärmeübertragungs-System und das Motor System, werden die Parametergrenzen erweitert, um eine hohe Nachstellzeit (Ti) zu berücksichtigen. Die spezifischen Grenzen wurden wie folgt gewählt.

System	Kp	Ti	Td
2. Ordnung	[0 - 10]	[0.1 - 10]	[0 - 10]
PT3	[0 - 10]	[0.1 - 10]	[0 - 10]
HeatSystem	[0 – 100]	[0.1 – 10’000]	[0 - 100]
MotorSystem	[0 - 100]	[0.1 – 10’000]	[0 - 100]

Abbildung 14 Suchraumgrenzen der Systeme.

3.2.4 Algorithmus

Die grundlegende Funktionsweise des PSO-Algorithmus wurde bereits erläutert. Mit der Zeit wurde der Algorithmus um verschiedene Aspekte erweitert wurde, wie die dynamische Anpassung der Hyperparameter. In dieser Arbeit wurde eine PSO-Libary swarmlip erstellt, die den grundlegenden Algorithmus [10] implementiert und um die vorgeschlagenen Erweiterungen von Mezura [14] und Pedersen [15] erweitert. Zudem wurden eigene Abbruchkriterien definiert. Die Initialisierung und der genaue Ablauf des Algorithmus werden im Folgenden beschrieben.

3.2.4.1 Initialisierung

Die Parameter des Algorithmus werden wie folgt initialisiert.

SwarmSize S = 40, die Anzahl Partikel im Schwarm. Die Schwarmgrösse hat grossen Einfluss auf den Rechenaufwand. Desto mehr Partikel, desto hoher der Rechenaufwand, da die Kostenfunktion entsprechend mehr ausgewertet wird. Andererseits wird der Algorithmus mit mehr Partikel immuner gegen lokale Minima und findet bessere Werte. Abschliessend kann nicht gesagt werden, ob es besser ist, eine kleine Schwarmgrösse zu wählen und mehrere Simulationen durchzuführen und die insgesamt besten Parameter zu nehmen. Oder eine grosse Schwarmgrösse zu wählen und nur eine Simulation durchzuführen. Arbeit [15] verweist das 40 eine praktikable Schwarmgrösse ist. Alle PSO-Simulation, in dieser Arbeit, verwenden eine Schwarmgrösse von 40.

StallCounter = 0, zählt die Anzahl Iterationen, die vergangen sind, ohne eine Verbesserung der Kostenfunktion.

W = 1.1, Trägheitsfaktor der bestimmt wie stark die Geschwindigkeit eines Partikels erhalten bleibt.

C_1 = 1.49, Kognitiver Gewichtungsfaktor, regelt wie stark ein Partikel in Richtung seines PersonalBest gezogen wird.

C_2 = 1.49, Sozialer Gewichtungsfaktor, regelt wie stark ein Partikel in Richtung des GlobalBest seiner Nachbarschaft gezogen wird.

MinNeighborsFraction = 0.25, bestimmt die minimale Grösse der Nachbarschaft, die Nachbarschaft umfasst mindestens ein Viertel aller Partikel.

MinNeighborhoodSize = MinNeighborsFraction * SwarmSize, minimale Anzahl der Nachbar-Partikel.

InitialSwarmSpan = 100, unterteilt jede Achse in 100 Punkte. Die zufälligen Anfangswerte können nur Werte auf diesen Punkten annehmen. Mit Parametergrenzen von [0, 10] wäre also jeder 0.1 Schritt (5.1, 5.2) ein gültiger Anfangswert. Mit 3 Dimensionen ergeben sich eine Million mögliche Startpositionen, auf die sich die Partikel zufällig verteilen.

InertiaRange = [0.1, 1.1], Obere und Untere Grenze des Trägheitsfaktors W. Der Trägheitsfaktor wird zur Laufzeit dynamisch angepasst, die InertiaRange setzt die Grenzen des Trägheitsfaktors.

StallIterations = 15, dient als Abbruchkriterium, wenn über die letzten 15 Iterationen keine signifikante Verbesserung (ImprovementThreshold) der Kostenfunktion stattfindet.

ImprovementThreshold = 0.01, definiert eine signifikante Verbesserung der Kostenfunktion in Prozent.

SpaceCriteria = 0.001, dient als Abbruchkriterium. Sobald der Raum, den die Partikel spannen, nur noch den Bruchteil 1/1'000 des gesamten Suchraums beträgt terminiert der Algorithmus.

N, ist die Anzahl Nachbar-Partikel, die ein Partikel berücksichtigt. Wird auf einen zufälligen Wert zwischen der MinNeighborhoodSize und der Schwarmgrösse -1 gesetzt.

r, ein Vektor, der die anfänglichen Bereiche definiert, innerhalb derer die Anfangsgeschwindigkeiten der Partikel zufällig gewählt werden. Die Werte von r für jede Dimension werden aus der Differenz zwischen den oberen und unteren Schranken des Suchraums bestimmt. Dies gewährleistet, dass die Anfangsgeschwindigkeiten der Partikel in einem sinnvollen Bereich liegen.

Es werden entsprechend der Schwarmgrösse S Partikel zufällig innerhalb des Suchraums erstellt. Partikel i hat die Position x(i), welche ein Zeilenvektor mit d Elementen ist. Die Anfangsgeschwindigkeit der Partikel wird ebenfalls zufällig gewählt, innerhalb des Bereichs [-r,r]. Jedes Partikel evaluiert die Kostenfunktion an seiner Startposition und setzt seinen PersonalBest auf den aktuellen Wert und die aktuelle Position. GlobalBest wird entsprechend auf das beste Partikel im Schwarm gesetzt.

3.2.4.2 Iterationsablauf

Der Algorithmus aktualisiert den Schwarm wie folgt. Für Partikel i, welches sich an Position x_i befindet:

1. Wähle eine zufällige Teilmenge S von N Partikeln ausser i.
2. Bestimme fbest(S), die beste Kostenfunktion unter den Nachbarn, und g(S), die Position des Nachbarn mit der besten Zielfunktion.
3. Man bildet r_1 und r_2 als Zufallszahlen zwischen 0 und 1. Formel 39 wird angewandt, um die Geschwindigkeit eines Partikels zu aktualisieren.
   $$v_i = w * v_i + c_1 * r_1 * (pB - p_i) + c_2 * r_2 * (gP - p_i) \quad (39)$$
4. Die Position wird aktualisiert, indem der Geschwindigkeitsvektor dazu addiert wird.
   $$x = x + v$$
5. Die Grenzen des Suchraums werden angewandt. Wenn eine Komponente von x_i ausserhalb der Grenze des Suchraums liegt, wird die Komponente gleich der entsprechenden Grenze gesetzt. Jene Komponenten, die gerade auf eine Grenze gesetzt wurde, wenn die Geschwindigkeit v(i) dieser Komponente ausserhalb der Grenze zeigt, wird diese Geschwindigkeitskomponente auf null gesetzt.
6. Die Kostenfunktion wird an der neuen Position ausgewertet cost = f(x_i).
7. Wenn cost < f(p_i) wird $p_i = x_i$ gesetzt, damit p_i die beste Position ist, an der das Partikel i je war. Dieser Schritt aktualisiert den PersonalBest Wert sowie Position.
8. Wenn cost < f(g) wird $g = x_i$ gesetzt, damit g die beste Position ist, an der je ein Partikel war. Dieser Schritt aktualisiert den GlobalBest Wert sowie Position.
9. Falls im vorherigen Schritt ein neuer GlobalBest gefunden wurde, wird flag = true, ansonsten flag = false, gesetzt.
10. Falls flag = true:
    Setze c = max(0,c-1).
    Setze N = minNeighborhoodSize.
    Falls c < 2, dann setze W = 2*W.
    Falls c > 5, dann setze W = W/2.
    Falls flag = false:
    Setze c = c + 1.
    Setze N = min(N + minNeighborhoodSize,SwarmSize).

3.2.4.3 Parallelisierung

In der modernen Softwareentwicklung ist die Parallelisierung, insbesondere von rechenintensiven Prozessen fundamental. Handelsübliche Prozessoren verfügen über mehrere Kerne, also unabhängigen Recheneinheiten. Wird Programmiercode nicht explizit parallelisiert läuft der Code synchron auf nur einem Kern und der Grossteil der Prozessorleistung wird nicht genutzt. Voraussetzung für die Parallelisierung ist es, dass es sich um Prozesse handelt, die asynchron abgearbeitet werden können.

PSO lässt sich sehr gut parallelisieren. Jede Auswertung eines Partikels ist ein unabhängiger Prozess. Verschiedene Partikel können gleichzeitig auf verschiedenen Kernen ausgewertet werden. Ein modernen 8-Kern Prozessor kann «theoretisch» das ITAE-Kriterium von acht Parameter-Sets gleichzeitig berechnen. Aufgrund von Overhead durch Kommunikation und Synchronisation der Prozesse wird der Algorithmus nicht exakt acht Mal schneller, aber die Gesamtrechenzeit wird signifikant reduziert. Der Code wurde vollständig parallelisiert.

3.2.5 Abbruchkriterien

Einer der wichtigsten Aspekte eines Optimierungsalgorithmus ist die Abbruchbedingung. Terminiert der Algorithmus zu früh, wird keine optimierte Lösung gefunden. Läuft der Algorithmus zu lang, wird Rechenleistung verschwendet, ohne eine Verbesserung der Zielfunktion zu erreichen. Diese Arbeit kombiniert zwei Ansätze, um zu erkennen, ob der Algorithmus konvergiert.

3.2.5.1 StallIterations

Der Algorithmus terminiert, sobald sich das ITAE-Kriterium in den letzten S Iterationen um nicht mindestens X% verbessert hat. Verbessert sich die Zielfunktion über S Iterationen nur minimal, ist das ein Zeichen, dass der Algorithmus ein endgültiges Minima erreicht hat und konvergiert ist. Diese Arbeit wählt S = 15 und X = 1%.

3.2.5.2 SpaceCriteria

Die äussersten Partikel, in jeder Dimension, spanen einen Raum. Zu Beginn, da die Partikel zufällig verteilt sind, nehmen die Partikel meistens fast den gesamten Suchraum ein. Da alle Partikel in Richtung des GlobalBest gezogen werden, nähern sich alle Partikel aneinander an und der Raum, den sie spannen, wird kleiner. Sobald der Raum nur noch ein Bruchteil V des gesamten Suchraums beträgt, terminiert der Algorithmus. Die Partikel haben sich angenähert und können nicht mehr ausbrechen. Diese Arbeit wählt V = 1/1'000.

Sobald eines der Kriterien erreicht ist, terminiert der Algorithmus und der finale GlobalBest Parameter wird ausgegeben.

3.2.6 Visualisierung

Um den PSO-Algorithmus verständlich zu machen wird der Verlauf der Partikelschwarmoptimierung grafisch dargestellt. Anhand von sechs Bildern wird die Position der Partikel im Suchraum zu verschiedenen Iterationsschritten veranschaulicht. Die Partikel sind gemäss ihrem ITAE-Kriterium eingefärbt: Rot steht für einen hohen, schlechten ITAE-Wert, Gelb für mittlere Werte und Grün für einen tiefen, guten ITAE-Wert. Die Darstellung zeigt die schrittweise Annäherung der Partikel an eine optimale Lösung. Ein Schwarm von 20 Partikeln wurde simuliert, um eine überschaubare Darstellung zu geben.

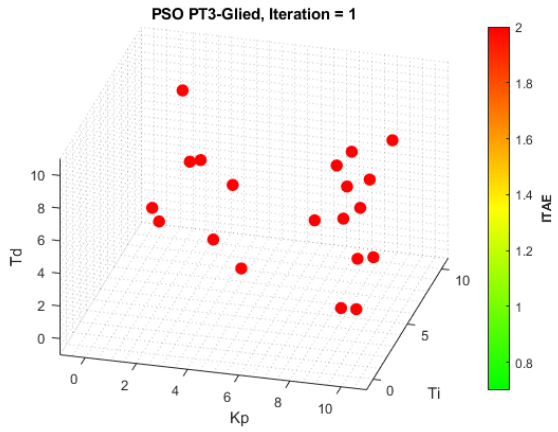


Abbildung 15 Schwarm, Iteration = 1

Abbildung 15 zeigt den initialen Schwarm in Iteration 1. Alle Partikel befinden sich in zufälligen Anfangspositionen und sind rot eingefärbt, kein Partikel befindet sich bereits an einer Position mit tiefem ITAE-Wert.

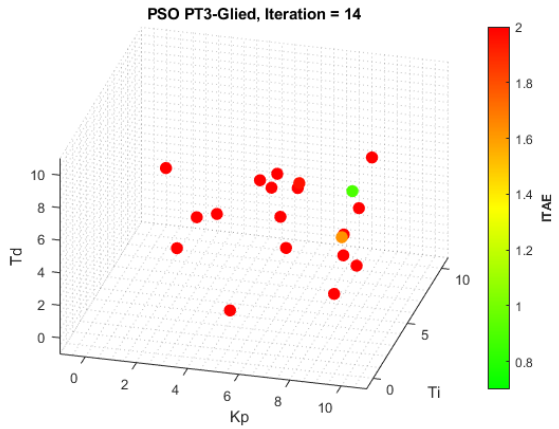


Abbildung 16 Schwarm, Iteration = 14

Nach vierzehn Iterationen beginnen einzelne Partikel, bessere Positionen im Suchraum zu entdecken, was durch das Auftreten von grün und gelb markierten Partikeln ersichtlich wird. Trotz der verbesserten Situation befinden sich die Partikel immer noch in grosser Distanz zueinander, der Algorithmus verfängt sich noch nicht in Minima, was die anfängliche Breite der Exploration zeigt.

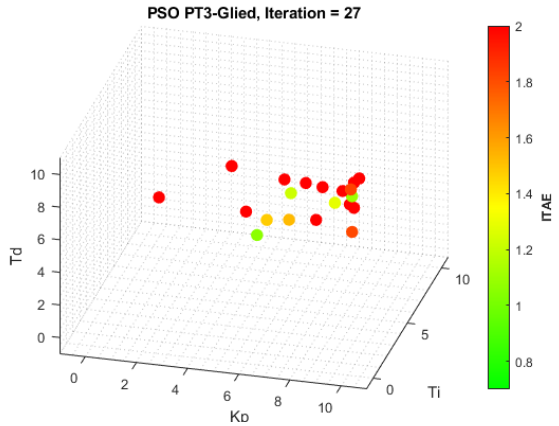


Abbildung 17 Schwarm, Iteration = 27

Weitere 13 Iterationen später zeigt sich eine erkennbare Annäherung des Schwarms. Mehr Partikel finden gute Parameter. Die Partikel passen sich basierend auf individueller und kollektiver Erfahrung an.

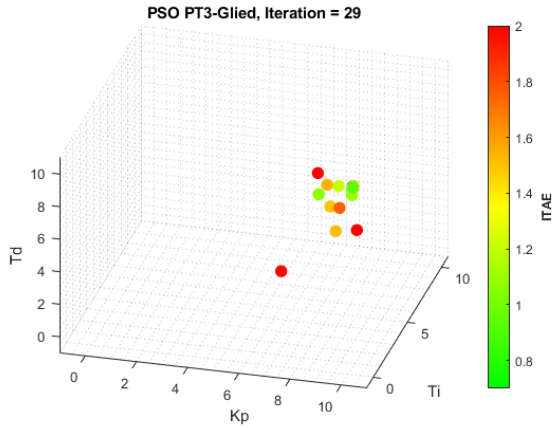


Abbildung 18 Schwarm, Iteration = 29

Nur zwei Iterationen später hat sich der Grossteil des Schwarms bereits deutlich in Richtung optimaler Positionen bewegt. Die Mehrheit der Partikel zeigt nun grüne Farben, die Partikel werden stark Richtung GlobalBest gezogen.

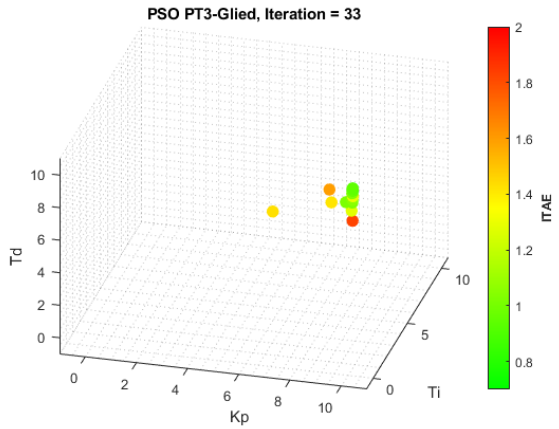


Abbildung 19 Schwarm, Iteration = 33

Fast alle Partikel haben einen guten Kostenwert. Der Raum zwischen den Partikel hat sich stark eingegrenzt, der Algorithmus konvergiert.

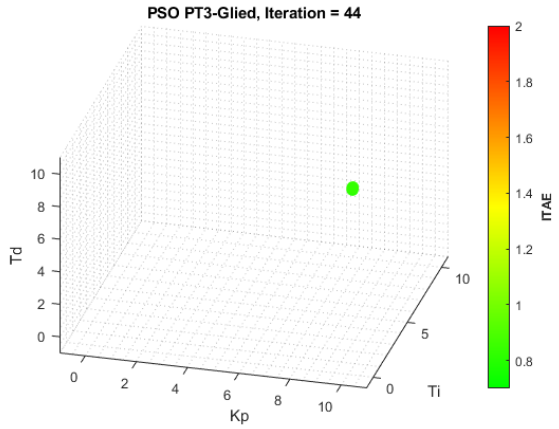


Abbildung 20 Schwarm, Iteration = 44

Elf Iterationen später hat der Algorithmus seinen Endpunkt erreicht. Es dauert noch relativ lange, da einige Partikel eine hohe Geschwindigkeit hatten, die abgebremst werden musste. Nun haben sich alle Partikel weiter angenähert und befinden sich dicht beieinander um das Optimum. Sie können dem Minima nicht entfliehen, was das Ende der Optimierung bedeutet. Der Algorithmus terminiert.

3.2.7 Datensammlung

Für jedes System wurde der PSO-Algorithmus 100-mal ausgeführt, um zu ermitteln, wie zuverlässig PSO gute Regelparameter findet. Dies ergibt 100 Parametersätze mit entsprechenden ITAE-Kriterien. Die Simulationsergebnisse werden in einer Datenbank gespeichert, welche die Grundlage für weitere Auswertung und Analyse der Resultate dient.

3.2.7.1 Datenbank

Eine Mysql Datenbank wurde erstellt, der Python Code stellt eine Verbindung zur Datenbank her und speichert nach jeder PSO-Simulation, die gefundenen Parameter, das ITAE-Kriterium, sowie andere bestimmende Eigenschaften.

Tabelle 5 Datenbankattribute.

Column	Type
Kp	Float
Ti	Float
Td	Float
Itae	Float unsigned
Iterations	Int unsigned
Function count	Int unsigned
Swarm size	Int unsigned
Kp min	Float

Kp max	Float
Ti min	Float
Ti max	Float
Td min	Float
Td max	Float

3.2.7.2 Abfragen

Es wurden mehrere SQl-Abfragen erstellt, um die Simulationsergebnisse auszuwerten. Die Daten werden in eine Form gebracht damit sie später, meistens in Python-Plots, dargestellt werden können.

4 Ergebnisse PSO

Dieses Kapitel untersucht die mit Partikelschwarm gefundenen Regelparameter und bewertet sie anhand bestehenden Referenzwerten. Ausserdem wird der Rechenaufwand des Algorithmus abgeschätzt.

Für das System 2. Ordnung sowie für das PT3-Glied, wie sie in dieser Arbeit implementiert wurden, wurden bereits, in vergangenen Arbeiten [3], [4] mit Hill-Climbing und sehr hohem Rechenaufwand, nach optimierten Parametern gesucht. Die dort gefundenen Parameter dienen als Referenzwerte, um neue mit PSO gefundenen Parameter zu bewerten. Die Referenzwerte sind in Tabelle 6 zu finden.

Tabelle 6 Referenzparameter des Systems 2. Ordnung sowie für das PT3-Glied.

System	Kp	Ti	Td	ITAE
System 2. Ordnung	9.9	5.4	0.45	0.221
PT3-Glied	10	9.7	0.7	0.810

Um die Performance des Algorithmus zu veranschaulichen, wird das ITAE-Kriterium jeder Simulation in aufsteigender Reihenfolge dargestellt. Jede Simulation repräsentiert einen gefundenen Parametersatz und das entsprechende ITAE-Kriterium. Die Referenzwerte aus den Publikationen [3][4] sind ebenfalls dargestellt. Das Diagramm zeigt, wie viele Simulationen Werte gefunden haben, die gleich oder besser als die Referenzwerte sind. Dies gibt ein Mass für den Erfolg des Algorithmus. Wenn weitere Simulationen durchgeführt würden, könnten niedrigere und höhere ITAE-Kriterien gefunden werden. Die Ergebnisse zeigen jedoch, wie die letzten 100 Simulationen im Verhältnis zu den genannten Referenzparametern abgeschnitten haben. Die Y-Achse zeigt das ITAE-Kriterium. Die X-Achse zeigt den Simulationsindex, sortiert nach aufsteigendem ITAE-Kriterium. Index 1 ist die Simulation, die die besten Parameter mit dem niedrigsten ITAE-Kriterium gefunden hat. Index 100 ist die Simulation mit den schlechtesten Parametern und dem höchsten ITAE-Kriterium. Jede Simulation wurde mit einer Schwarmgrösse von 40 berechnet.

4.1 Überschwingendes System 2. Ordnung

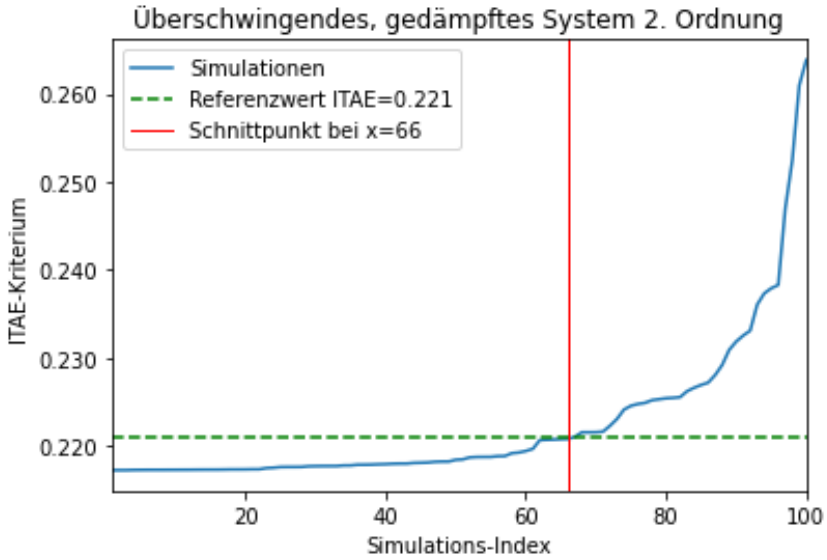


Abbildung 21 ITAE-Kriterium von 100 PSO-Simulationen, Überschwingendes, gedämpftes System 2. Ordnung.

Tabelle 7 Vergleich gefundene Regelparameter, Überschwingendes, gedämpftes System 2. Ordnung.

Wert	Kp	Ti	Td	ITAE
Referenz	9.9	5.4	0.45	0.221
Min (Index=1)	10	5.321	0.435106	0.217235
Median (Index=50)	10	5.12882	0.414282	0.218417
Max (Index=100)	7.7202	4.06828	0.452048	0.26387

Insgesamt haben 66 von 100 Simulationen gleich gute oder bessere Parameter gefunden als die Referenzwerte. Die restlichen Parameter sind jedoch nur minimal schlechter als die Referenz. Der in 100 Simulationen gefundene schlechteste Parameter-Satz weicht um 19.4% vom Referenzwert ab. Auffallend ist, dass die Referenz und Median Parameter sehr nahe beieinander liegen. Der schlechteste Parameter-Satz sich aber in einem anderen lokalen Minimum verfangen hat. Absolut betrachtet produziert auch der schlechteste Parameter-Satz ein akzeptables Regelverhalten. In Abbildung 22 wird die Sprungantwort der Referenzparameter mit der Sprungantwort der gefundenen Medianparameter verglichen.

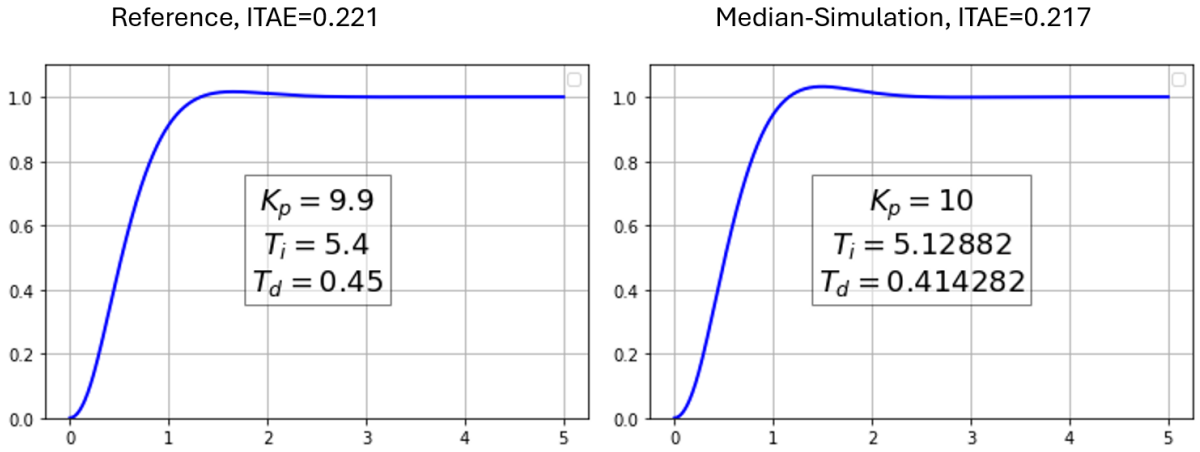


Abbildung 22 Vergleich der Sprungantwort der Referenzparameter zu der Sprungantwort der Parameter der Median-Simulation. Überschwingendes, gedämpftes System 2. Ordnung.

Die beiden Kurven sind sehr ähnlich. Die Medianparameter erzeugen einen leicht schnelleren Anstieg dadurch aber auch ein leicht stärkeres Überschwingen. Im Kontext des zeitgewichteten Fehlers wird so ein Verhalten bevorzugt, was in einem minimal geringeren ITAE-Kriterium gegenüber dem Referenzwert resultiert.

4.2 PT3-Glied

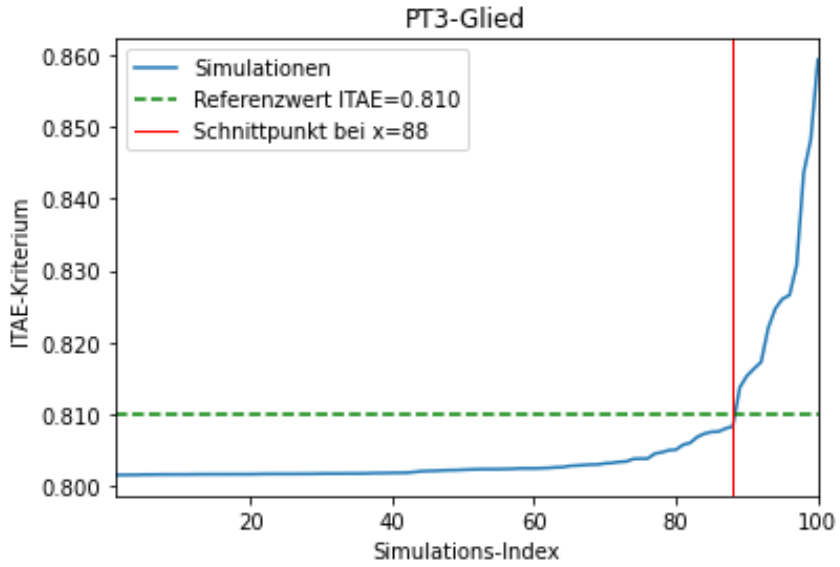


Abbildung 23 ITAE-Kriterium von 100 PSO-Simulationen, PT3-Glied.

Tabelle 8 Vergleich gefundene Regelparameter, PT3-Glied.

Wert	Kp	Ti	Td	ITAE
Referenz	10	9.7	0.7	0.810
Min (Index=1)	10	9.91944	0.720803	0.801581
Median (Index=50)	9.96387	9.87306	0.719043	0.802304
Max (Index=100)	7.77747	8.22063	0.731806	0.859277

Insgesamt haben 88 von 100 Simulationen gleich gute oder bessere Parameter als die Referenzwerte gefunden. Der schlechteste Parametersatz weicht um 6.1% vom Referenzkriterium ab, was immer noch zu einem akzeptablen Regelverhalten führt. Auch hier fällt auf das sich der schlechte Parametersatz in einem anderen Minimum als der Referenz oder Median Parametersatz befindet. In Abbildung 24 wird die Sprungantwort der Referenzparameter mit der Sprungantwort der gefundenen Medianparameter verglichen

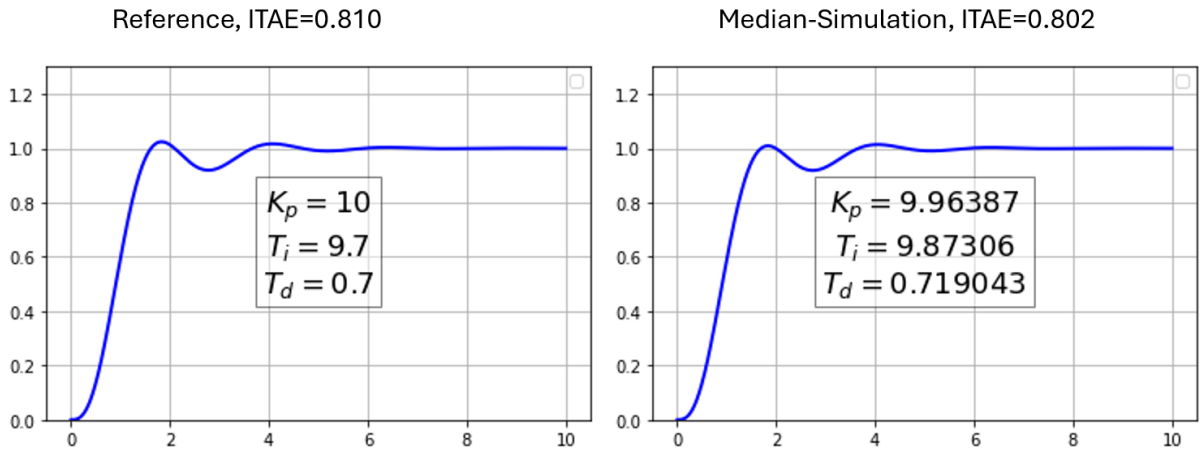


Abbildung 24 Vergleich der Sprungantwort der Referenzparameter zu der Sprungantwort der Parameter der Median-Simulation. PT3-Glied.

Die beiden Kurven sind fast identisch. Die Medianparamter produzieren einen minimal geringeren Überschwung, was zu einem marginal tieferen ITAE-Kriterium führt.

4.3 Wärmeübertragungs-System

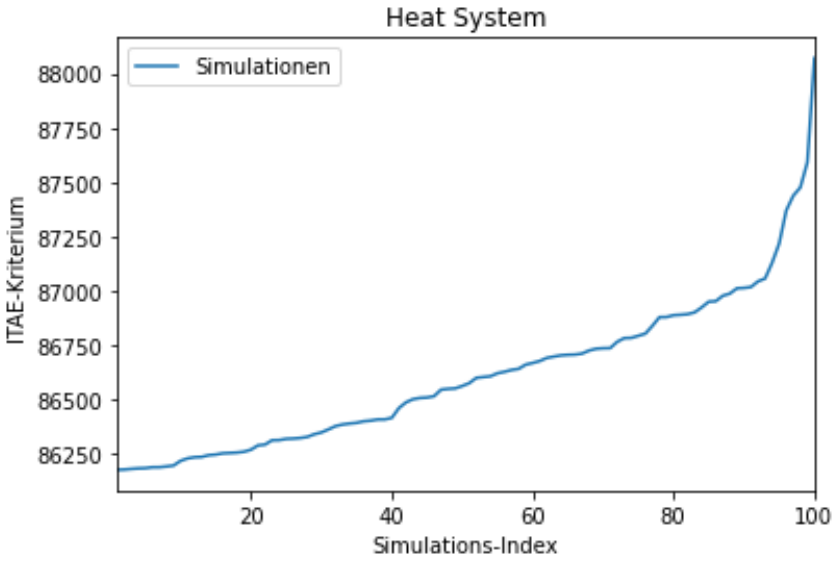


Abbildung 25 ITAE-Kriterium von 100 PSO-Simulationen, Wärmeübertragungs System.

Tabelle 9 Vergleich gefundene Regelparameter, Wärmeübertragungs System.

Wert	Kp	Ti	Td	ITAE
Min (Index=1)	99.9751	3318.88	5.92159	86177.4
Median (Index=50)	98.9129	3260.8	6.50453	86563
Max (Index=100)	59.8449	2043.5	5.72668	88070.9

Die Abweichung der Simulationsergebnisse ist sehr gering. Das ITAE-Kriterium der schlechtesten Simulation ist lediglich 2.2% höher als das Kriterium der besten Simulation. Jede Simulation liefert gute Regelparameter.

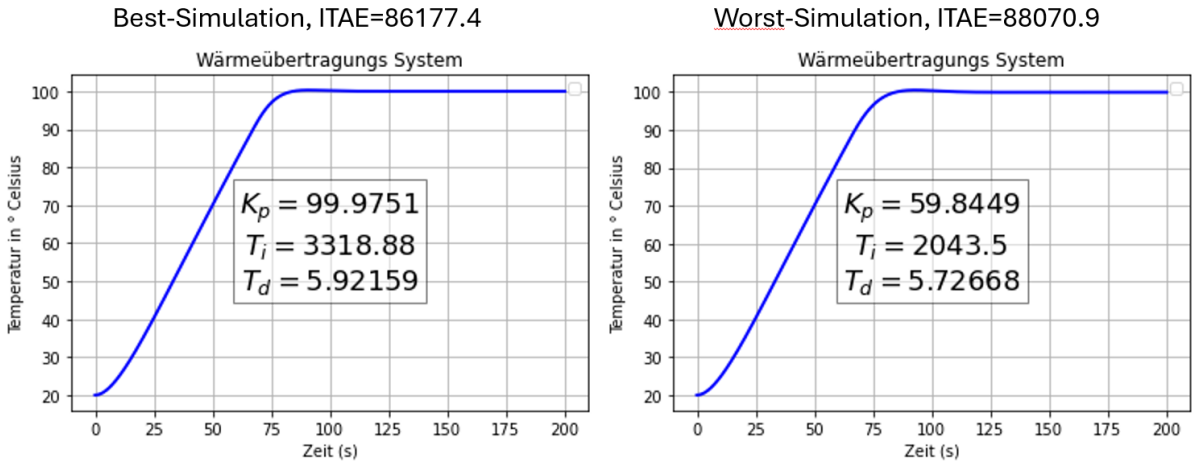


Abbildung 26 Vergleich der Sprungantwort der besten und schlechtesten gefundenen Parameter, Wärmeübertragungs System.

Bemerkenswert ist, obwohl die Parameter unterschiedlich sind, erzeugen sie fast eine identische Sprungantwort. Die Kostenfunktion hat mehrere fast identisch gute Minima. Ansonsten kann man keine Abweichung erkennen.

4.4 Motor System

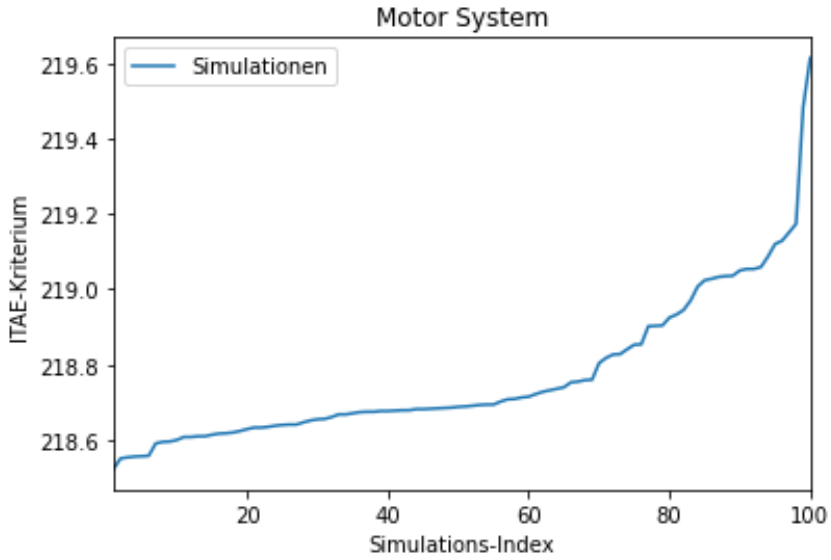


Abbildung 27 ITAE-Kriterium von 100 PSO-Simulationen, Motor System-

Tabelle 10 Vergleich gefundene Regelparameter, Motor System.

Wert	Kp	Ti	Td	ITAE
Min (Index=1)	99.9421	10’000	0.359373	218.523
Median (Index=50)	92.1261	10’000	0.354098	218.688
Max (Index=100)	49.8873	8179.66	0.367418	219.614

Auch hier liefert die schlechteste Simulation ein ITAE-Kriterium, welches nur 5% höher wie das tiefste ITAE-Kriterium, welches gefunden wurde. Auch hier ist zu sehen, dass der Grossteil der Simulationen das gleiche Minima erkennt. Die anderen Simulationen verfangen sich in einem Minima mit nur marginal schlechterem ITAE-Kriterium.

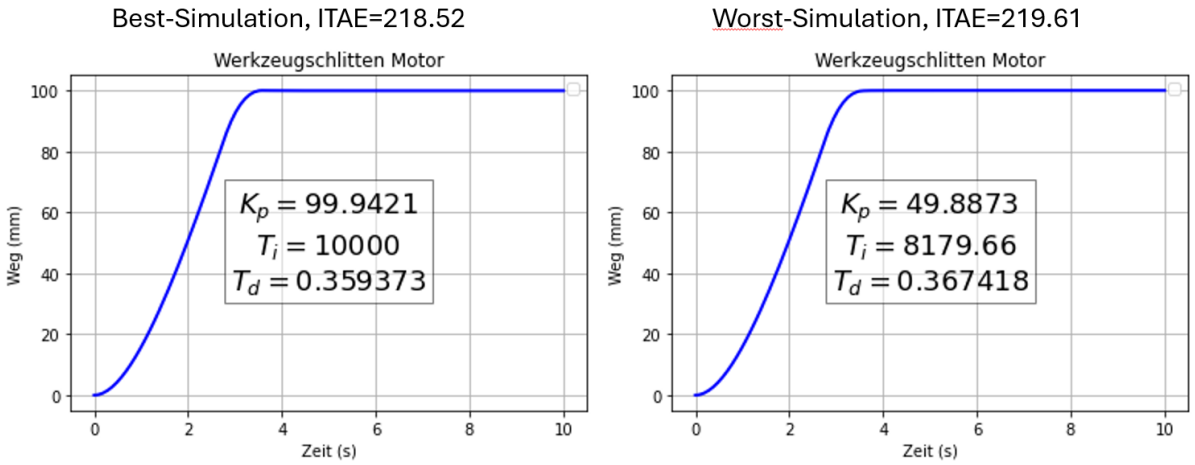


Abbildung 28 Vergleich der Sprungantwort der besten und schlechtesten gefundenen Parameter, Motor System.

Die Abweichung der Kurven ist nicht erkennbar.

4.5 Rechenaufwand

Die genaue Laufzeit, Anzahl Iterationen, von PSO kann nie im Voraus bestimmt werden, lediglich durch die Abbruchkriterien kann beeinflusst werden, wie schnell der Algorithmus terminiert, dabei ist das Gleichgewicht zwischen genügend Iterationen, um lokalen Minima zu entfliehen, und nicht zu viele Iterationen, um den Rechenaufwand zu minimieren einzuhalten. Die Laufzeit hängt direkt vom jeweiligen System ab. Je nach Beschaffenheit der Kostenfunktion kann der Algorithmus schnell konvergieren und ein Minimum finden. Hat die Kostenfunktion viele lokale Minima, in denen Partikel hängen bleiben, wird die Laufzeit verlängert.

Um eine Vorstellung zu geben wie viel Zeit eine PSO-Simulation benötigt, wird die durchschnittliche Anzahl von Iterationen pro System bestimmt, basierend auf den letzten 100 Simulationen. Die Anzahl der Aufrufe der Kostenfunktion wird dann als Multiplikation der Anzahl der Iterationen mit der Schwarmgrösse gebildet. Für diese Anzahl an Parameterkombinationen wurde das ITAE-Kriterium berechnet. Die Dauer der Berechnungen ist von der verwendeten Hardware abhängig. Für diese Arbeit wurde ein handelsüblicher 8-Kern Prozessor eingesetzt, der 0,2 Sekunden pro Evaluierung der Kostenfunktion benötigt, was einer Rate von zehn Auswertungen pro Sekunde entspricht. Unter der Annahme, dass zehn Auswertungen pro Sekunde durchgeführt werden können, lässt sich eine ungefähre Rechenzeit für die verschiedenen Systeme schätzen. Es ist zu betonen, dass diese Schätzung kein absolutes Mass darstellt, sondern vielmehr eine grobe Abschätzung der Rechenzeit unter Verwendung handelsüblicher Hardware bietet.

Tabelle 11 Funktionsaufrufe & Rechenzeit.

System	Ø Anzahl Iterationen	Schwarmgrösse	Ø Anzahl Funktionsaufrufe	Rechenzeit
2. Ordnung	20	40	800	1min 20sek
PT3-Glied	22	40	880	1min 28sek
Heat System	19	40	760	1min 16sek
Motor System	17	40	680	1min 8sek

5 Diskussion

Python bietet zwar keine direkte Alternative zu Simulink, ermöglicht jedoch die numerische Lösung von Differentialgleichungen, wodurch eine Sprungantwort eines Regelkreises simuliert werden kann. Ein Grundkonstrukt wurde entwickelt, das erweitert werden kann, um verschiedene Regelkreise zu simulieren und auszuwerten. Hierfür sind lediglich grundlegende Programmierkenntnisse erforderlich, um das Grundkonstrukt mit den gewünschten Übertragungsfunktionen zu ergänzen.

Die Partikelschwarmoptimierung (PSO) hat sich als hervorragend geeignet erwiesen, um optimierte PID-Regelparameter für allgemeine Systeme zu finden. Innerhalb weniger Minuten können auf handelsüblicher Hardware sehr genaue Regelparameter (Kp, Ki, Kd oder Ti, Td) bestimmt werden. PSO zeichnet sich durch seine Effizienz aus, da die Prozesse aufgrund ihrer Natur parallelisiert werden können, wodurch die Hardware optimal genutzt wird. Der Algorithmus verallgemeinert das Problem der Parameterfindung sehr gut, jedoch nicht vollständig.

Technisches Know-how über das System ist erforderlich, damit der Anwender die Suchgrenzen der jeweiligen Parameter festlegen kann. Je enger die Grenzen, desto kleiner der Suchraum und desto besser konvergiert der Algorithmus, wodurch geeignete Parameter innerhalb des Suchraums gefunden werden. Ist der Suchraum jedoch zu klein oder schlecht gewählt, können sich bessere Parameter ausserhalb des Suchraums befinden. Abschliessend lässt sich sagen, dass Partikelschwarmoptimierung eine praktikable Alternative mit sehr guten Resultaten zur Parameterfindung bietet.

6 Verzeichnisse

6.1 Literaturverzeichnis

1. J. B. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," ASME Transactions, v64 (1942), S. 759-768.
2. K. L. Chien, J. A. Hrones, and J. B. Reswick, "On the Automatic Control of Generalized Passive Systems," Transactions of the American Society of Mechanical Engineers., Bd. 74, Cambridge (Mass.), USA, Feb. 1952, S. 175–185.
3. R. Büchi, "PID Parameter Tables after ITAE to Control Overshooting Systems Found with AI Algorithm," in Proceedings of the International Conference on Mechanical, Automotive and Mechatronics Engineering (ICMAME 2023).
4. R. Büchi, "Optimal ITAE criterion PID parameters for PTn plants found with a machine learning approach," in 2021 9th International Conference on Control, Mechatronics and Automation (ICCMA), IEEE, 2021.
5. K. Ogata, “Modern Control Engineering,” (5. Edition), Prentice Hall, 2010, S. 200-230.
6. K. J. Åström and T. Hägglund, “PID Controllers: Theory, Design, and Tuning,” Instrument Society of America, 1995, S. 59-117.
7. A. Ahmed, E. Ahmed, A. A. Aziz, and M. Amin, "Comparison Study of Different Structures of PID Controllers," Research Journal of Applied Sciences, Engineering, and Technology. 11. 10.19026/rjaset.11.2026.
8. G. F. Franklin, J. D. Powell, and A. Emami-Naeini, “Feedback Control of Dynamic Systems,” (6. Edition), Pearson, 2015, S. 120-136.
9. D. E. Seborg, T. F. Edgar, and D. A. Mellichamp, “Process Dynamics and Control,” Hoboken, NJ: John Wiley & Sons, 2004, pp. 210–215.
10. J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of ICNN'95 - International Conference on Neural Networks, Perth, WA, Australia, 1995, S. 1942-1948.
11. A. Visioli, "Modified Anti-Windup Scheme for PID Controllers," IEE Proceedings - Control Theory and Applications, vol. 150, no. 1, pp. 1-7, Jan. 2003.
12. SciPy Developers, "scipy.integrate.solve_ivp," SciPy v1.10.0 Reference Guide, 2024. [Online]. Verfügbar: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html. [Zugriff: 11-Mai-2024].
13. S. Zacher and M. Reuter, "Regelungstechnik für Ingenieure: Analyse, Simulation und Entwurf von Regelkreisen, " (16. Edition) Wiesbaden, Germany: Springer Fachmedien Wiesbaden GmbH, 2022, S. 86 – 92
14. E. Mezura-Montes and C. A. Coello Coello, "Constraint-handling in nature-inspired numerical optimization: Past, present and future," Swarm and Evolutionary Computation, 2011, S. 173–194.
15. M. E. Pedersen, "Good Parameters for Particle Swarm Optimization," Luxembourg: Hvass Laboratories, 2010.

6.2 Abbildungsverzeichnis

6.3 Tabellenverzeichnis

7 Anhang

Der Python Code, der im Laufe dieser Bachelorarbeit erstellt wurde und mit dem die Ergebnisse berechnet wurden, findet man vollständig unter folgendem GitHub Repository.

https://github.zhaw.ch/grubelu1/PSO-Python.git

Ebenfalls findet man die Datenbank, auf der die gefundenen Parameter gespeichert sind. Mit dem Code und der Datenbank, kann man alle Ergebnisse dieser Arbeit selbst nachvollziehen.