Lehrstuhl für Umformtechnik

Masterarbeit

Digitalisierung eines Walzwerks mit der Retrofitting Methode

Marcel Sorger, BSc

September 2020

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 21.09.2020

Unterschrift Verfasser/in
Marcel, Sorger

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die mich im Laufe meines Studiums und beim Verfassen meiner Masterarbeit unterstützt haben.

Mein Dank gebührt meinen Betreuern DDI Benjamin Ralph und Univ.-Prof. Dipl.-Ing. Dr.techn. Martin Stockinger für die hilfreichen Anregungen, tatkräftige Unterstützung und konstruktive Kritik, die diese Masterarbeit erst möglich gemacht hat.

Des Weiteren möchte ich bei meinen Eltern, meiner Familie und meinen Freunden für die Unterstützung und den Rückhalt bedanken, ohne die mein akademischer und persönlicher Werdegang nur schwer vorstellbar gewesen wäre.

Kurzfassung

Durch die Einführung des Begriffes der Industrie 4.0 in 2011 werden in der Technik vermehrt Anstrengungen unternommen, um IT-Technologien stärker zu integrieren, um Prozesse zu automatisieren, digitalisieren und optimieren. Das Walzen zählt zu einem der bedeutendsten Umformerfahren in der metallverarbeitenden Industrie. Bei älteren Walzwerken, die heute noch oft zum Einsatz kommen, besteht die Möglichkeit eines Upgrades auf den Stand der Industrie 4.0 durch ein „Retrofitting“. Ein „Retrofitting“ eines Walzwerkes macht dann wirtschaftlich Sinn, wenn das zu modernisierende Walzwerk seine Anforderungen noch ausreichend erfüllt und die Kosten der Anschaffung eines neuen Walzwerkes die Kosten des „Retrofittings“ überschreitet. Dazu wird das zu modernisierende Walzwerk mit einer auf die zu messenden Zustandsgrößen abgestimmten Sensorik ausgestattet, deren Daten durch eine geeignete Software gesammelt, ausgewertet und analysiert werden können. Dies kann durch die digitale Abbildung des Prozesses in Form eines „Digital Shadows“ (DS) oder „Digital Twins“ (DT) und die Integration in ein „Cyber Physical Production System“ (CPPS) erreicht werden.

Diese Masterarbeit befasst sich mit der Konzipierung und Implementierung eines „Digital Twins“ durch ein „Retrofitting“ des Walzwerkes am Lehrstuhl für Umformtechnik an der Montanuniversität Leoben. Im Zuge dieser Masterarbeit werden zwei Konzipierungen zur Sensorik und Programmierung vorgestellt, die mittels „Proof of Concept“ (POC) beweisen sollen, dass auch „Open-Source“-Programmiersprachen mittlerweile ausgereift genug sind, um ein CPPS abbilden zu können. Des Weiteren wird die Implementierung eines erarbeitenden Konzeptes am Walzwerk behandelt, um dieses auf einen Industrie 4.0 Stand zu bringen.

Abstract

With the introduction of the term Industry 4.0 in 2011, more efforts are being made to integrate IT technologies to automate, digitize and optimize processes. Rolling is one of the most important forming processes in the metal forming industry. In the case of older rolling mills, which are still frequently used today, there is the possibility of upgrading to the Industry 4.0 standard by means of a "retrofitting". A "retrofitting" of a rolling mill makes economic sense if the rolling mill to be modernized still sufficiently meets its requirements and the costs of purchasing a new rolling mill exceed the costs of the "retrofitting". For this purpose, the rolling mill to be modernized is equipped with a sensor system adapted to the condition variables to be measured and suitable software so the data can be collected, evaluated and analyzed. This can be done by digitally depict the process in the form of a “Digital Shadow” (DS) or “Digital Twin” (DT) and the integration into a “Cyber Physical Production System” (CPPS).

This master thesis deals with the designing and realizing of a digital twin through a “retrofitting” of the rolling mill at the Chair of Metal Forming at the Montanuniversitaet Leoben. During this master thesis, two concepts for sensor technology and programming are carried out, which will by means of “Proof of Concept” (POC) show that today’s Open-Source-Programming-Languages are mature enough to depict a CPPS. Furthermore, the implementation of one concept is realized in order to bring the existing rolling mill system up to the state of Industry 4.0.

Inhaltverzeichnis

Abkürzungsverzeichnis

Zeichen	Einheit	Beschreibung
μ	[-]	Reibungskoeffizient
A	[mm²]	Fläche
a	[mm]	Standardabstand zwischen Wirbelstromsensor und Walzenoberfläche
b	[mm]	Walzgutbreite
b_0	[mm]	Anfangsbreite
b_1	[mm]	Endbreite
C	[MN/mm]	Gerüstmodul
c_p	[J/(kg*K)]	Spezifische Wärmekapazität
d	[mm]	Durchmesser
d_w	[mm]	Walzendurchmesser
E	[N/mm²]	Elastizitätsmodul
F	[N]	Kraft
F_{id}	[N]	Ideelle Umformkraft
F_{max}	[N]	Maximale Walzkraft des Walzwerkes
F_N	[N]	Normalkraft
h	[mm]	Walzguthöhe
h_0	[mm]	Anfangshöhe
h_0'	[mm]	Einlaufdicke des Walzgutes
h_0''	[mm]	Einlaufdicke des Walzgutes
h_1	[mm]	Endhöhe
h_1'	[mm]	Auslaufdicke des Walzgutes
h_1''	[mm]	Auslaufdicke des Walzgutes
I	[mm⁴]	Flächenträgheitsmoment
k_f	[N/mm²]	Fließspannung
k_{fm}	[N/mm²]	Mittlere Fließspannung
k_w	[N/mm²]	Formänderungswiderstand
l	[m]	Länge
l_0	[mm]	Anfangslänge
l_1	[mm]	Endlänge
l_B	[mm]	Ballenlänge der Walze

l_d	[mm]	Gedrückte Länge
m	[kg]	Masse
R	[mm]	Walzenradius
R_B	[Ω]	Bürdenwiderstand
R_W	[Ω]	Widerstandsbahnwiderstand
R_X	[Ω]	Schleifkontaktwiderstand
s_0	[mm]	Walzspaltöffnung
T	[°C]	Temperatur
T_m	[°C]	Schmelztemperatur
U	[V]	Elektrische Spannung
u_1	[mm]	Messwert des Wirbelstromsensors 1
u_2	[mm]	Messwert des Wirbelstromsensors 2
u_2	[mm]	Messwert des Wirbelstromsensors 3
U_a	[V]	Schleifkontaktspannung
U_S	[V]	Angelegte Spannung
V	[mm³]	Volumen
v	[m/s]	Geschwindigkeit
v_0	[m/s]	Eintrittgeschwindigkeit des Walzgutes
v_1	[m/s]	Austrittsgeschwindigkeit des Walzgutes
v_{st}	[1/s]	Umformgeschwindigkeit
w_{max}	[mm]	Maximale Durchbiegung der Walze
W_Q	[J]	Wärmeenergie
W_U	[J]	Umformenergie
x	[mm]	Position des Schleifkontaktes auf der Widerstandsbahn
$x_{1,i}$	[mm]	x-Koordinate des Sensormesspunktes 1 in der Ebene x_i
$x_{2,i}$	[mm]	x-Koordinate des Sensormesspunktes 2 in der Ebene x_i
$x_{3,i}$	[mm]	x-Koordinate des Sensormesspunktes 3 in der Ebene x_i
x_i	[mm]	x-Ebene der Wirbelstromsensoren
x_{max}	[mm]	Widerstandsbahnlänge
$y_{1,i}$	[mm]	y-Koordinate des Sensormesspunktes 1 in der Ebene x_i
$y_{2,i}$	[mm]	y-Koordinate des Sensormesspunktes 2 in der Ebene x_i
$y_{3,i}$	[mm]	y-Koordinate des Sensormesspunktes 3 in der Ebene x_i
$z_{1,i}$	[mm]	z-Koordinate des Sensormesspunktes 1 in der Ebene x_i
$z_{2,i}$	[mm]	z-Koordinate des Sensormesspunktes 2 in der Ebene x_i

$z_{3,i}$	[mm]	z-Koordinate des Sensormesspunktes 3 in der Ebene x_i
α	[°]	Greifwinkel
α_s	[°]	Anstellwinkel des Wirbelstromsensors
ΔF	[N]	Änderung der Kraft
Δh	[mm]	Höhenänderung
Δh_{max}	[mm]	Maximale Höhenänderung
Δl	[mm]	Längenänderung
Δs_i	[mm]	Anstellungsänderung
Δt	[s]	Änderung der Zeit
$\Delta\vartheta$	[K]	Temperaturerhöhung
$\Delta\varphi$	[-]	Änderung des Umformgrades
ε	[-]	Dehnung
ε_I, ε_{II}, ε_{III}	[-]	Hauptnormaldehnungen
ρ	[kg/m³]	Dichte
σ	[N/mm²]	Spannung
σ_h	[N/mm²]	Horizontalspannung
σ_I, σ_{II}, σ_{II}	[N/mm²]	Hauptnormalspannungen
σ_v	[N/mm²]	Vertikalspannung
σ_x	[N/mm²]	Spannung in x-Richtung
σ_y	[N/mm²]	Spannung in y-Richtung
φ	[-]	Umformgrad
$\dot{\varphi}$	[1/s]	Umformgeschwindigkeit
φ_{max}	[-]	Maximaler Umformgrad

1. Einleitung

Im Zuge der vierten industriellen Revolution, der Industrie 4.0, kommt es zu einer fortschreitenden Digitalisierung von Prozessen, bei denen vor allem eine systemübergreifende Kommunikation zwischen verschiedenen Prozessen, aber auch jene zwischen Menschen und Prozess im Vordergrund stehen [44].
Aus diesem Grund wurde der Entschluss gefasst, das Walzwerk am Lehrstuhl für Umformtechnik an der Montanuniversität Leoben auf den Stand der Industrie 4.0 zu bringen, um dieses für Forschung und Lehre künftig besser nutzen zu können. Die Ziele dieser Masterarbeit bestehen daher in der Erarbeitung eines geeigneten Konzeptes der Sensorik, in der Programmierung der Datenerfassung, -auswertung und -visualisierung in einer „Open-Source"-Programmiersprache, sowie einer geeigneten Implementierung, um ein „Cyber Physical Production System" (CPPS) zu erschaffen [29].
Hierfür wurden Überlegungen zu den zu messenden Größen angestellt und wie es möglich gemacht werden kann, diese mittels geeigneter Sensorik zu erfassen. In der Folge wurden Prototypen zur Programmierung entwickelt, welche diese Daten analysieren, auswerten und visualisieren können, um diese auch dem Bediener leicht zugänglich zu machen.
Im zweiten Kapitel wird ein kurzer Überblick über den Stand der Technik gegeben, gefolgt von einer Übersicht über die technischen Grundlagen, die für diese Masterarbeit von Relevanz sind. Darauf folgt eine Übersicht über den derzeitigen Stand des Walzwerkes (Kapitel 4) und eine Erläuterung der Motivation eine „Open-Source"-Programmiersprache (Kapitel 5) zu verwenden. Das Kernstück der Arbeit wird in Kapitel 6 behandelt, die Konzipierung der Sensorik, die Programmierung sowie deren Ergebnisse mit einem theoretischen „Proof of Concept", gefolgt von der Auswertung und den Ergebnissen (Kapitel 7). In Kapitel 8 wird die Implementierung der Sensorik und Software des aus Kapitel 6 erarbeiten Konzeptes behandelt, um den „Proof of Concept" zu erbringen. Abschließend wird die Masterarbeit in Kapitel 9 mit einer Zusammenfassung der Arbeit und einem Ausblick abgeschlossen.

2. Stand der Technik

Der Begriff Industrie 4.0, steht für eine vermehrte Durchdringung von Informationstechnologie (IT) in der Industrie [33].

Durch den wachsenden Einsatz von IT soll im Zuge der vierten industriellen Revolution mittels neuer Technologien die Digitalisierung und Vernetzung der Industrie vorangetrieben werden [7, 34].

Hierzu werden bereits etablierte Technologien der dritten industriellen Revolution, wie etwa Sensoren und Aktoren, mit neuen digitalen Technologien kombiniert, um wirtschaftliche, betriebliche und produktionsspezifische Daten zu gewinnen, verarbeiten und analysieren [34].

Neben dem Sammeln, Verarbeiten und Analysieren von Daten stehen viele weitere vielseitige und facettenreiche Ziele und Schwerpunkte im Fokus, welche mit Industrie 4.0 Ansätzen realisiert werden sollen – Kundenvernetzung, Produkt- und Produktionsablaufveränderungen, Produktinnovationen, Abstimmung und Vereinheitlichung von Prozessen und Systemen, Automatisierung von Transportsystemen, Selbststeuerung und Automatisierung des Produktionsprozesses [33] – um nur einige davon zu nennen.

Eine digitale Vernetzung verändert die Grundfunktionen einer Maschine nicht, sondern steuert neue unterstützende Zusatzfunktionen bei [36].

Als „Enabler“ für solch eine intelligente Vernetzung agieren sogenannte „Embedded Systems“ [36]. „Embedded Systems“ sind Steuerzentralen, welche im Hintergrund ohne das externe Zutun Steuer- und Überwachungsaufgaben übernehmen. Sie stellen eine Kombination aus Hardware, wie etwa Sensorik, Aktorik, und Mikrocontrollern und Software dar [36, 37, 38].

Durch den anhaltenden Trend von immer kompakterer und leistungsstärkerer Elektronik, der steigenden Bandbreite von Netzwerken und der einfachen Verfügbarkeit beider werden sich nach aktuellem Forschungsstand immer mehr solcher Embedded Systems in den Maschinen der Zukunft finden [39].

Das „Cyber Physical System“ (CPS), und dessen Erweiterung, das „Cyber Physical Production System“ (CPPS), dienen als Grundstein der Industrie 4.0 [33].

Ein CPS stellt ein digital-physisches System dar und dient als Brücke zwischen der digitalen und realen Welt, welches mithilfe von Internettechnologien („Internet of Things“ (IoT)) Daten sammelt, speichert, analysiert und verarbeitet und diese Daten in der virtuellen und realen Welt integriert, um in weiterer Folge eine menschliche Interaktion zu ermöglichen [7, 29].

CPS bzw. CPPS sind in der Automobilindustrie am stärksten verbreitet, finden aber auch immer häufiger in der Medizintechnik Verwendung [39].

Ein „Digital Shadow“ (DS) oder „Digital Twin“ (DT), bildet den digitalen Teil eines CPS bzw. CPPS und ist eine digitale Repräsentanz des physischen, in der realen Welt vorhandenen Produktes oder ganzheitlichen Prozesses. DS oder DT werden in der Literatur oftmals als integraler Bestandteil eines CPS bzw. CPPS verstanden [29].

Durch das Vorhandensein eines CPS bzw. CPPS und die dadurch ermöglichte Interaktion zwischen Menschen und Maschine durch ein „Graphical User Interface“ (GUI), können vom Maschinenbediener durch die Visualisierung der Prozessdaten schnell und einfach flexible Änderungen am Prozess vorgenommen werden, wodurch man sich ökonomische als auch soziologische Vorteile verspricht [33, 35].

Durch eine weitreichende Vernetzung und Datenintegration ergeben sich zahlreiche Innovationspotentiale in Hinsicht auf die Produktionssteuerung und -planung. Ein in Echtzeit ablaufender Datenaustausch ermöglicht eine immense Prozessverbesserung in vielerlei Hinsicht – flexible Regelung des Produkt- und Warenstroms, flexible Bearbeitung von Kundenwünschen, dynamische Einsatzplanung von Beschäftigten und Belegungsplanung von Maschinen auf Basis der Auftragslage, Tracking der Produkte im Produktionsprozess – wobei diese Aufgaben auch mit ihrem Umfang an Komplexität hinzugewinnen [7, 33, 37].

Durch eine „Manufacturing Intelligence“, eine Echtzeit-Datenintegrationsmethode um Produktionsprozesse in Echtzeit zu analysieren, modellieren und simulieren zu können, soll das Zusammenführen verschiedener Datenquellen und das Auswerten und Analysieren der darin enthaltenen Daten unter dem Begriff „Operational Buisness Intelligence“ (OpBI), realisiert werden. Ihr Ziel besteht in der Entscheidungsunterstützung und Flexibilisierung von Geschäftsprozessen, um die hierarchische „Top-Down-Steuerung“ zu dezentralisieren und diese durch eine selbstorganisierende Steuerung zu substituieren. [33, 36].

Die Verwirklichung dieser Ziele führte zur Etablierung und Konkretisierung neuer und auch bereits bekannter fachspezifischen Termini – „Ubiquitous Computing“, „Pervasive Computing“, „Ambient Intelligence“, „Smart Objects“ bzw. „Smart Devices“, IoT, „Systems of Systems“, „Smart Factories“ und „Learning Factories“ sind gängige Begriffe im Bereich der Industrie 4.0 [38, 39].

Immer mehr „Smart Factories“, gekennzeichnet durch das Vorhandensein eines CPS, und „Learning Factories“, deren Ziele die Weiterbildung und Forschung im Bereich der Industrie 4.0 sind, halten im Industrie- und Forschungssektor Einzug [7, 35, 36].

Ganzheitlich gesehen bieten CPS bzw. CPPS nicht nur großes Potential in der Produktion, wie etwa als integrale Bestandteile einer „Smart Factory“, sondern auch in anderen Manifestierungen der Industrie 4.0, wie z.B. „Smart Mobility“, „Smart Health“ und „Smart Grids“ [39].

Bereits im Zuge der dritten industriellen Revolution wurden große Anstrengungen in die Automatisierung und Regelung von Walzwerken investiert. Moderne Walzwerke entsprechen in vielen Belangen bereits dem Standard der Industrie 4.0, jedoch bestehen diese oft aus digitalen Insellösungen, die eine umfassende System- und Datenintegration erschweren und deshalb aus technologischer Sicht keine „Smart Factory“ vorliegt. Da sich der allgemeine Trend aber eindeutig in diese Richtung bewegt, werden zum jetzigen Zeitpunkt immer mehr solcher Systeme in CPPS transformiert, um eine Prozessverbesserung und Flexibilisierung entlang der gesamten Prozesskette zu erreichen [7].

3. Grundlagen

Die folgenden Abschnitte bieten einen Überblick über die wichtigsten Begriffe und mathematischen Beziehungen, die für diese Masterarbeit von Relevanz sind. Zuerst werden die umformtechnischen Grundbegriffe behandelt, gefolgt von den Grundlagen des Walzens, welche auch im Laufe der Masterarbeit angewendet werden. Anschließend wird auf die Grundbegriffe der Sensorik und für diese Arbeit relevanten Sensortypen eingegangen. Als Abschluss folgt eine kurze Erläuterung der Funktionsweise von A/D-Wandlern sowie den wichtigsten Terminologien der Industrie 4.0.

3.1. Umformtechnische Grundbegriffe

DIN8550 definiert das Umformen als eine plastische Formänderung eines festen Körpers unter Beibehalt der Masse und des Stoffzusammenhalts [1].

Die Dehnung ε gibt Auskunft über die Längenänderung eines Körpers, wobei eine Streckung eine positive Dehnung und eine Stauchung eine negative Dehnung beschreibt [1].

$$\varepsilon = \frac{\Delta l}{l_0} = \frac{l_1 - l_0}{l_0} \tag{3.1}$$

In der Umformtechnik hat sich als Formänderungskenngröße der Umformgrad φ etabliert, welcher auch als logarithmische oder wahre Dehnung bezeichnet wird. Der Umformgrad bietet einige Vorteile:

- das Vorzeichen des Umformgrades gibt Auskunft über die Richtung der Umformung, so handelt es sich bei einem positiven Vorzeichen um eine Streckung und bei einem negativen Vorzeichen um eine Stauchung des Werkstückes
- die Umkehrbarkeit des Umformgrades, welche bei Rückrechnung auf die vorherige Umformung den gleichen Absolutwert ergibt
- die Summe aller Umformgrade der jeweiligen Umformungsschritte ergeben den Gesamtumformgrad [1].

Der Umformgrad φ kann als Maß für die Größe der plastischen Formänderungen angesehen werden [1].

$$\varphi = ln\left(\frac{h_1}{h_0}\right) \tag{3.2}$$

Aus der Volumenskonstanz, hier für einen quaderförmigen Körper, ergibt sich somit für die Summe aller Umformgrade gleich null [1].

$$V = l_0 \cdot b_0 \cdot h_0 = l_1 \cdot b_1 \cdot h_1 = konst. \tag{3.3}$$

$$\frac{l_1 \cdot b_1 \cdot h_1}{l_0 \cdot b_0 \cdot h_0} = 1 \tag{3.4}$$

$$ln\left(\frac{l_1}{l_0}\right) + ln\left(\frac{b_1}{b_0}\right) + ln\left(\frac{h_1}{h_0}\right) = \ln(1) = 0 \tag{3.5}$$

Die Umformgeschwindigkeit $\dot{\varphi}$, auch als v_{st} bezeichnet, kann durch die Ableitung des Umformgrades nach der Zeit beschrieben werden [1].

$$\dot{\varphi} = \frac{d\varphi}{dt} \cong \frac{\Delta\varphi}{\Delta t} \; oder \; \lim_{\Delta t \to 0} \frac{\Delta\varphi}{\Delta t} \tag{3.6}$$

Die Fließspannung k_f (oder Formänderungsfestigkeit) ist die im einachsigen Spannungszustand ermittelte Normalspannung, die im Werkstoff die kritische zum Fließen erforderlichen Schubspannung hervorruft. Die Fließspannung hängt im Wesentlichen von vier Parametern ab:

- vom Werkstoff (Legierungsbestandteile, Gefüge)
- von der Temperatur T
- vom Umformgrad φ
- von der Umformgeschwindigkeit $\dot{\varphi}$ [1]

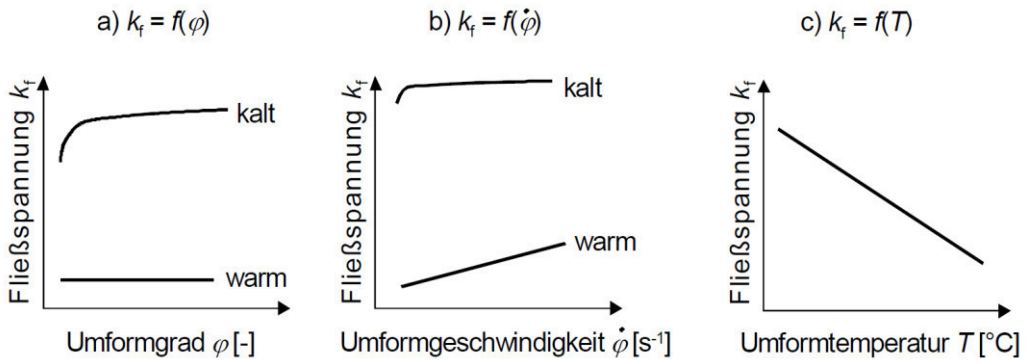


Abb. 1: Abhängigkeit der Fließspannung k_f vom Umformgrad φ (a), von der Umformgradgeschwindigkeit $\dot{\varphi}$ (b) und Temperatur T (c) [1]

Die Unterscheidung zwischen Kalt- und Warmumformung erfolgt durch die Temperatur, wobei als Grenzwert $0{,}42*T_m$ (Schmelztemperatur des Werkstoffes), die Rekristallisationstemperatur, herangezogen wird. Ist die bei der Umformung vorherrschende Temperatur $T > 0{,}7*T_m$ so spricht man von einer Warmumformung. In einem Temperaturbereich von $0{,}7*T_m \leq T \geq 0{,}42*T_m$ spricht man von einer Halbwarmumformung. Geschieht die Umformung unterhalb der Rekristallisationstemperatur $T < 0{,}42*T_m$, spricht man von einer Kaltumformung [1, 32].

Im Zwischenbereich von Kalt- und Warmumformung liegt die sogenannte Halbwarmumformung. Von einer Halbwarmumformung spricht man in einem Temperaturbereich die unter jener der Warmumformung und über jener der Kaltumformung liegt [2].

Durch die geringe Temperatur gegenüber der Warmumformung bietet dies Vorteile in Bezug auf die verringerte Zunderbildung und höhere Maßgenauigkeit, jedoch auch Nachteile wie höhere Umformkräfte [1].

Wie in Abb. 1 zu sehen ist, sinkt mit der Temperatur die zur plastischen Umformung benötige Fließspannung eines Werkstoffes (siehe Abb. 1, c). Des Weiteren flacht auch die Fließkurve mit zunehmender Temperatur bei zunehmenden Umformgrad ab (siehe Abb. 1, a.).

3.2. Walzen

Das Walzen zählt zu den Fertigungsverfahren der Gruppe des Druckumformens. Der Werkstoff wird zwischen mindestens zwei rotierenden Werkzeugen, den sogenannten Walzen, in der Umformzone umgeformt, was zu einer Querschnittverringerung des Walzgutes führt [3].

Abhängig von der Walzguttemperatur kann, wie in Abschnitt 3.1. beschrieben, zwischen Kalt-, Halbwarm- und Warmwalzen unterschieden werden.

Die zweite Hauptgruppe aus DIN 8580 wird nach DIN 8582 weiter nach dem herrschenden Spannungszustand unterteilt (siehe Abb. 2) [1].

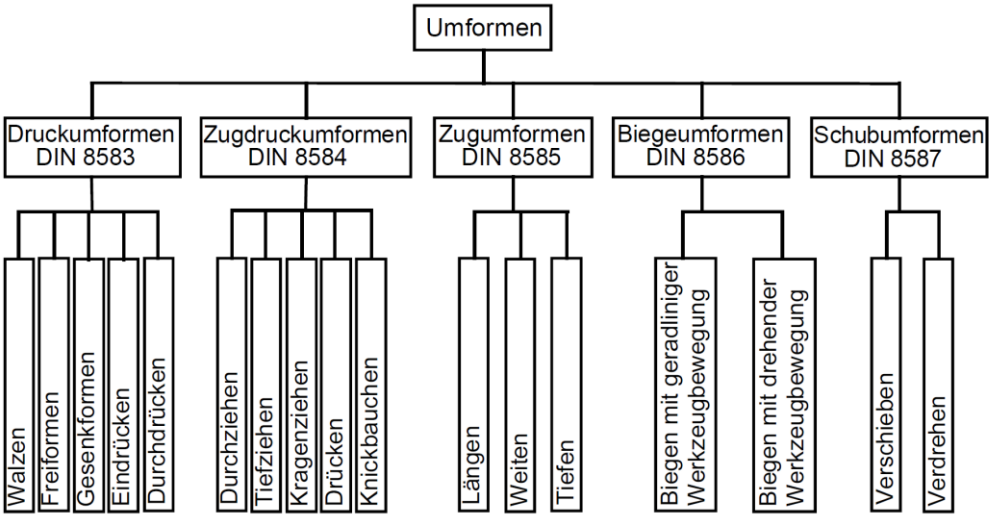


Abb. 2: Gliederung der sechs Hauptgruppen der Fertigungsverfahren (nach DIN 8582) [1]

Der Mohrsche Spannungskreis gibt Aufschluss über den Spannungszustand eines Punktes oder infinitesimalen Volumens. Beim breitungslosen Walzen wird von einem ebenen Spannungszustand ausgegangen, wie in Abbildung 3 dargestellt.

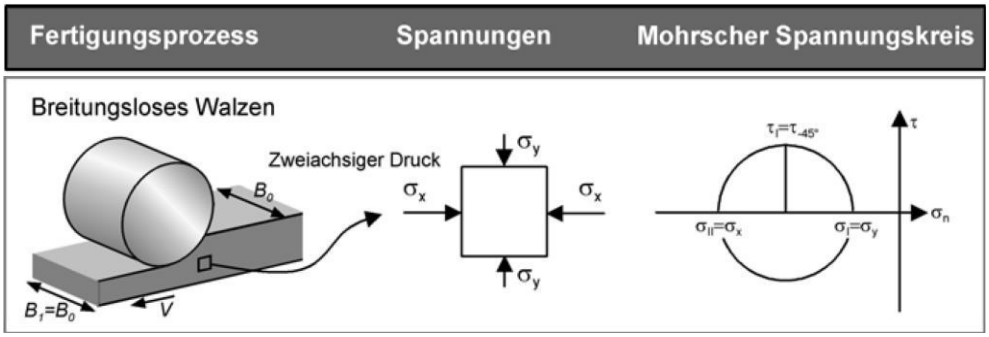


Abb. 3: Mohrscher Spannungskreis des breitungslosen Walzens [1]

Betrachtet man das breitungslose Walzen, so liegt ein ebener Formänderungszustand vor. Durch das Vorliegen eines ebenen Formänderungszustandes ist eine der Hauptformänderungsgeschwindigkeiten, sowie der Umformgrad und die Formänderung in dieser Richtung während der gesamten Dauer des Walzens gleich null (Abb. 4) [1].

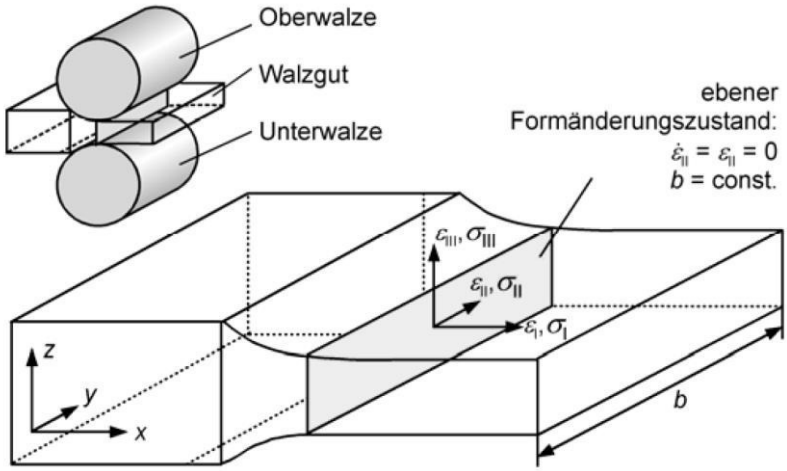


Abb. 4: Spannungszustand beim breitungslosen Walzen [1]

Bei der plastischen Deformation des Walzgutes im Walzspalt kommt es zu einer Längung und in manchen Fällen auch zu einer Breitung des Walzgutes [1].

Die Längung des Walzgutes beschreibt dessen plastische Verformung in die Längsrichtung, wohingegen die Breitung die platische Verformung des Walzgutes in die Querrichtung beschreibt. Bei einem Breiten/Dicke-Verhältnis > 10 wird die Breitung vernachlässigbar klein und es wird vom breitungslosen Walzen gesprochen [5].

Diese durch Längung und/oder Breitung bedingte Deformation des Walzgutes führt zu einer Dickenreduzierung, welche als Walzgrad bezeichnet wird [5].

Zur weiteren Klassifizierung von Fertigungsverfahren werden diese durch die Art der Krafteinleitung in Verfahren mit mittelbarer und unmittelbarer Krafteinleitung eingeteilt. Bei Umformprozessen mit unmittelbarer Krafteinleitung wird die Umformkraft der Umformmaschine direkt ohne weitere Umwege in die Umformzone eingeleitet. Die benötigten Umformkräfte bei unmittelbaren Prozessen sind, wie auch bei mittelbaren Umformprozessen, von den Reibverhältnissen und der Geometrie der Umformzone und der Fließspannung des umzuformenden Werkstoffes abhängig [1].

Aus dieser Definition geht hervor, dass das Walzen den Verfahren der unmittelbaren Krafteinleitung angehört, da die Krafteinleitung der Walzkraft durch die Ober- und Unterwalze unmittelbar auf das Walzgut erfolgt.

Die plastische Verformung des Walzgutes erfolgt im Walzspalt, wenn der Formänderungswiderstand des Werkstoffes überschritten wird. Der Formänderungswiderstand ist dann erreicht, wenn die Vergleichsspannung bestehend aus allen einwirkenden Spannungen gleich dem

Formänderungswiderstand ist. Aufgrund der komplexen Zusammenhänge im Walzspalt werden für eine analytische Betrachtung des Warmwalzens folgende Annahmen getroffen:

1. Die Formänderung des Walzgutes wird als rein plastisch betrachtet, elastische Anteile der Formänderung werden vernachlässigt.
2. Der Walzvorgang wird als stationär betrachtet, die Ein- und Austrittsgeschwindigkeit des Walzgutes in den Walzspalt ist konstant.
3. Die Beschleunigungskräfte im Walzspalt, die zur Geschwindigkeitsänderung im Walzspalt führen, werden vernachlässigt.
4. Der Reibungskoeffizient zwischen den Oberflächen der Walze und des Walzgutes wird als konstant betrachtet. Abhängigkeiten von Kraft, Druck oder Geschwindigkeit werden vernachlässigt.
5. Bei dünnem, breitem Walzgut tritt keine Breitung auf, die Fließrichtung entspricht der Walzrichtung [6].

Die ideelle Umformkraft F_{id} bei einer unmittelbaren Krafteinleitung kann wie folgt berechnet werden [1]:

$$F_{id} = A \cdot k_{fm} \tag{3.7}$$

Die mittlere Fließspannung k_{fm} berechnet sich dabei aus dem Umformgrad φ (Gl. 3.2) und der Fließspannung k_f [1]:

$$k_{fm} = \frac{1}{\varphi} \int_0^{\varphi} k_f \cdot d\varphi \tag{3.8}$$

Die Fläche in der das Walzgut mit den Arbeitswalzen in Kontakt tritt, wird als Walzspalt bezeichnet. Im Walzspalt tritt das Walzgut mit der Eintrittshöhe h_0 ein und verlässt den Walzspalt mit der Austrittshöhe h_1. Projiziert man diesen Kontaktbogen auf die Walzgutmittenebene spricht man von der gedrückten Länge l_d (siehe Abb. 5). Diese Kontaktfläche A, auch als gedrückte Fläche bezeichnet, ist das Produkt aus Walzgutbreite b und der gedrückten Länge l_d [6].

$$A = b \cdot l_d = b \cdot \sqrt{R \cdot \Delta h - \frac{\Delta h^2}{4}} \tag{3.9}$$

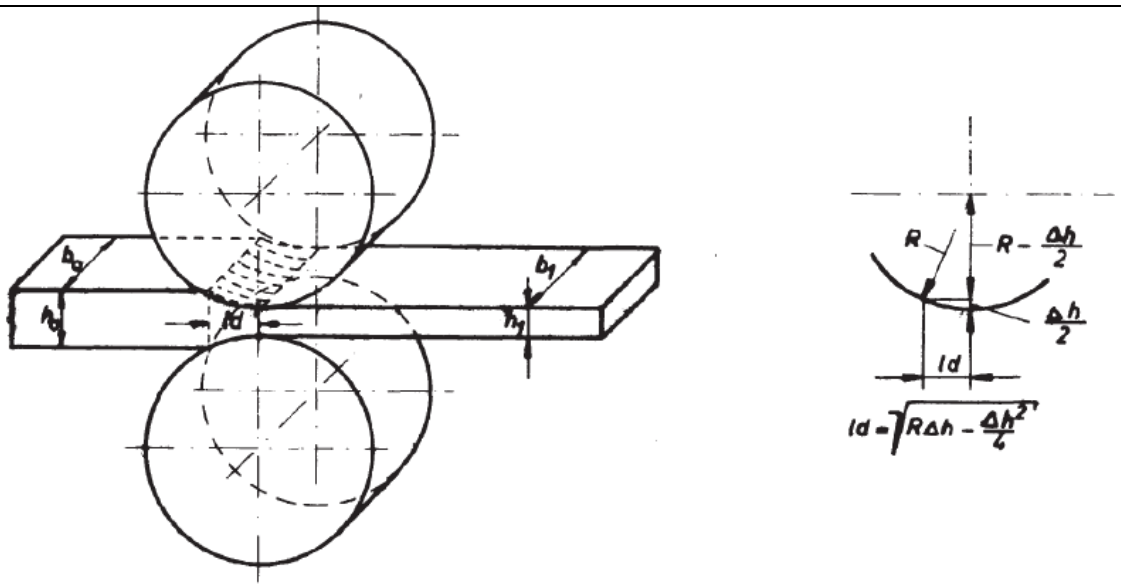


Abb. 5: Geometrie des Walzspaltes [6]

In der Umformzone kommt es zu einer Deformation der Körner des metallischen Gefüges des Walzgutes. Während der Umformung kommt es aufgrund der inneren Reibung und der äußeren Reibung zwischen Werkzeug und Walzgut zur Temperaturerhöhung des Walzgutes. Beim Kaltwalzen geht man davon aus, dass mehr als 90% der ideellen Umformarbeit W_{id} in Wärmeenergie umgewandelt werden. Die nicht in Wärmeenergie umgewandelte Umformarbeit verbleibt als potentielle Energie in Form von Eigenspannungen und Verfestigungserscheinungen im Walzgut [1].

Unter der Annahme, dass die gesamte Umformarbeit unter adiabatischen Verhältnissen in Wärme umgewandelt wird, erhält man Gl. 3.10. Da wie oben erwähnt fast die gesamte Umformarbeit W_U in Wärmeenergie W_Q umgesetzt wird kann $W_U = W_Q$ angenommen werden, woraus die Temperaturerhöhung $\Delta\vartheta$ des Walzgutes in Abhängigkeit von der mittleren Fließspannung k_{fm}, dem maximalen Umformgrad φ_{max}, der Dichte ρ und der spezifischen Wärmekapazität c_p berechnet werden kann (Gl. 3.11) [5]:

$$W_U = V \cdot k_{fm} \cdot |\varphi|_{max} = W_Q = m \cdot c_p \cdot \Delta\vartheta \tag{3.10}$$

$$\Delta\vartheta = \frac{k_{fm} \cdot |\varphi|_{max}}{\rho \cdot c_p} \tag{3.11}$$

Aufgrund der Volumenskonstanz (Gl. 3.2) ergibt sich ein zeitlich konstanter Volumenstrom an Walzgut im Walzspalt (Gl. 3.12) [6].

$$v_0 \cdot b_0 \cdot h_0 = v \cdot b \cdot h = konst. \tag{3.12}$$

Setzt man b = b_0, ergibt sich der Geschwindigkeitsverlauf des Walzgutes im Walzspalt (Abb. 6) [6].

$$v = v_0 \cdot \frac{h_0}{h} = v_1 \cdot \frac{h_1}{h} \tag{3.13}$$

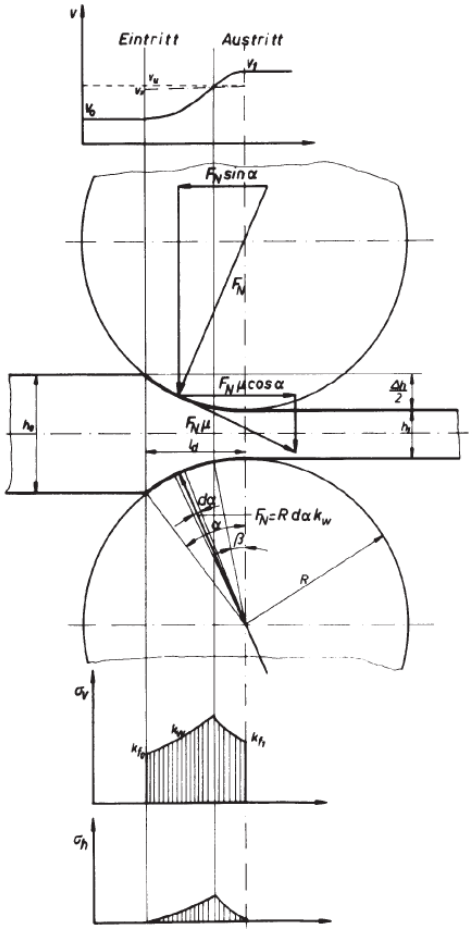


Abb. 6: Vorgänge im Walzspalt [6]

Zwischen dem Ein- und Austrittpunkt des Walzgutes wird zwischen der Nacheilzone, Fließscheide und Voreilzone unterschieden, was auf die Relativgeschwindigkeiten zwischen Walzenoberfläche und Walzgut zurückführen ist (Abb. 6, erstes Diagramm). Im Einlaufbereich ist die Walzenumfangsgeschwindigkeit höher als jene des Walzgutes, was aufgrund der Relativgeschwindigkeit zu einer Aufstauung an Walzgut führt, weshalb dieser Bereich als Nacheilzone bezeichnet wird. Als Fließscheide wird jener Bereich bezeichnet, in dem die Walzgutgeschwindigkeit der Walzenumfangsgeschwindigkeit entspricht, es liegt daher keine Relativgeschwindigkeit zwischen Walzgut und Walzenoberfläche vor. Zusätzlich wirken in der

Fließscheide die höchsten Druckspannungen auf das Walzgut. Im Austrittsbereich ist die Walzgutgeschwindigkeit höher als die Walzenumfangsgeschwindigkeit, was aufgrund der Relativgeschwindigkeit zu einer Ausstoßung an Walzgut führt, weshalb dieser Bereich als Voreilzone bezeichnet wird [5, 6].

Damit das Walzgut von den Arbeitswalzen eingezogen wird, muss die sogenannte Greifbedingung erfüllt sein [9].

Um die Greifbedingung zu erfüllen, müssen die horizontalen, in Richtung der Walzrichtung wirkenden Reibungskräfte größer sein als die horizontalen, in Gegenrichtung der Walzrichtung wirkenden Reibungskräfte. In anderen Worten, die einziehenden Reibungskräfte müssen größer als die ausstoßenden Reibungskräfte sein, um das Walzgut in den Walzspalt einzuziehen (Abb. 7) [5].

Hierbei wird, wie bereits erwähnt, nach [6] von einem konstanten Reibungskoeffizienten μ zwischen Walzgut und Walzenoberfläche ausgegangen, welcher weder druck- noch geschwindigkeitsabhängig ist. Aus Abb. 7 geht der in Gl. 3.13 beschriebene Zusammenhang zwischen dem Reibungskoeffizienten μ, der Normalkraft F_N und dem Greifwinkel α hervor:

$$\mu \cdot F_N \cdot \cos\alpha \geq F_N \cdot \sin\alpha \tag{3.13}$$

Somit folgt aus Gl. 3.13 die Greifbedingung:

$$\mu \geq \tan\alpha \tag{3.14}$$

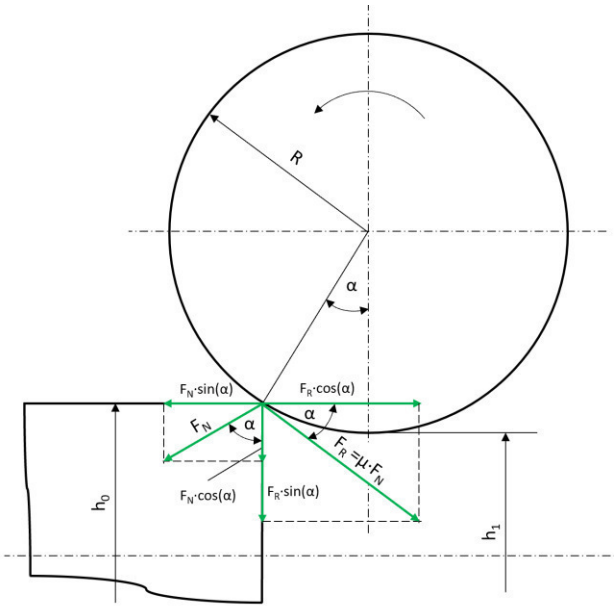


Abb. 7: Auftretende Kräfte für die Einziehbedingung [5]

Während des Walzprozesses kommt es bedingt durch die Krafteinleitung auf das Walzgut zu einem Kraftfluss durch das Walzgerüst. Alle Teile des Walzgerüstes, die sich im Kraftfluss befinden, erfahren dabei eine elastische Deformation. Betroffen Teile sind dabei unter anderem die Walzen, Walzenlager, Kraftmessdosen, Ständerholme, Anstellelemente und das Ständerquerhaupt. Diese elastische Verformung führt zum Auffedern des Walzspaltes. Diese elastische Verformung bzw. Auffederung des Walzgerüstes wird im sogenannten Anstelldiagramm zusammen mit der Walzgutdicke in Abhängigkeit voneinander dargestellt (siehe Abb. 8) [6].

Die maximale Höhenabnahme Δh_{max} des Walzgutes kann näherungsweise als Produkt des Walzenradius R und des Reibungskoeffizienten μ zum Quadrat beschrieben werden [5]:

$$\Delta h_{max} = \mu^2 \cdot R \tag{3.15}$$

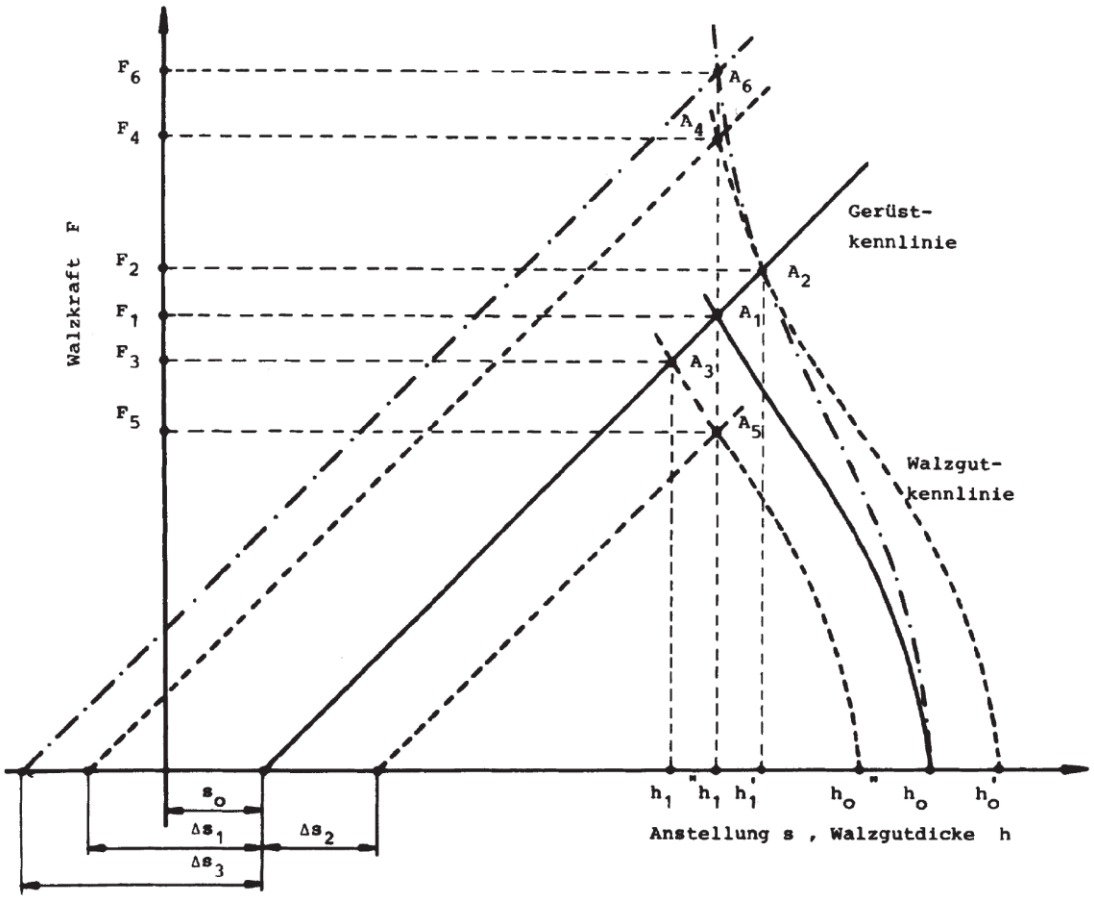


Abb. 8: Anstelldiagramm [6]

Der Schnittpunkt zwischen der Gerüstkennline und der Walzgutkennlinie im Anstelldiagramm wird Arbeitspunkt genannt. Der Arbeitspunkt gibt Aufschluss über die Austrittshöhe des Walzgutes in Abhängigkeit von der Walzkraft (Abb. 8) [6].

Die Linearität der Federkennlinie ist dabei auf die Annahme eines ideal elastischen Verformungsverhaltens des Walzgerüstes und deren Bauteile zurückzuführen.

In Abb. 8 bezeichnet F die Walzkraft, s_0 die Walzspaltöffnung des unbelasteten Gerüsts, h_0 die Einlaufhöhe des Walzgutes und A die Arbeitspunkte. Die Steigung der Gerüstkennline wird Gerüstmodul C [MN/mm] genannt (siehe Gl 3.16). Bildet man den Kehrwert des Gerüstmodul erhält man die Auffederung [mm/MN] [6].

$$C = \frac{\Delta F}{\Delta s} = \frac{\Delta F}{h_1 - s_0} \tag{3.16}$$

Die Austrittshöhe des Walzgutes lässt sich nach der gage-meter-Gleichung berechnen [6]:

$$h_1 = s_0 + \frac{F}{C} \tag{3.17}$$

Somit ist es möglich die Austrittshöhe des Walzgutes in Abhängigkeit von Temperatur, Werkstofffestigkeit, Walzspalthöhe und weiteren Parametern zu ermitteln [8].

Zusätzlich nehmen die Wärmedehnung des Werkstoffes und der Walzen und der Verschleiß der Walzen Einfluss auf die Auslaufdicke des Walzgutes h_1.

Walzgerüste werden bezüglich der Anordnung und Anzahl ihrer Walzen unterschieden. Das Präfix bezieht sich auf die Anzahl der Walzen des Walzgerüstes, so besitzt ein Duo-Walzwerk zwei Walzen, ein Quattro-Walzwerk vier Walzen, etc. (Abb. 9). Die Art des Walzgerüstes ist stark abhängig vom Einsatzgebiet. Unterscheidet man zwischen Kalt- und Warmwalzen ist die Auswahl abhängig von mehreren Gesichtspunkten, wie z.B. Fließkurven des Werkstoffes, Temperatur, Maß- und Formhaltigkeit etc…, weshalb sich die Walzgerüste stark in Ausführung hinsichtlich Bauart (Abb. 6), Dimensionierung, Kühl- und Schmiersystemen unterscheiden [9].

Wie in Abschnitt 3.1. bereits angeführt kann beim Umformen zwischen Kalt-, Halbwarm- und Warmumformen unterschieden werden. Beim Walzen wird aufgrund der Temperatur des Walzgutes zwischen Kalt- und Warmwalzen unterschieden. Beim Kaltwalzen müssen gegenüber dem Warmwalzen aufgrund der geringen Temperatur des Walzgutes wesentlich höhere Umformkräfte aufgebracht werden,

um den gleichen Umformgrad zu erzielen (Abb. 1, a), was einen signifikanten Einfluss auf die Auslegung bzw. Auswahl des Walzgerüstes hat.

Der Vorteil den beispielsweise Quattro-Walzwerke gegenüber Duo-Walzwerken bieten ist jener, dass das Quattro Walzwerk über zwei Stützwalzen verfügt, welche die beiden Arbeitswalzen, die mit dem Walzgut in Kontakt treten, stützen. Die Stützwalzen stützen die Arbeitswalze während des Walzvorganges und reduzieren somit die Durchbiegung der Arbeitswalzen. Die Reduktion der Durchbiegung der Arbeitswalzen führt in weiterer Folge zu einer geringeren Profilabweichung bzw. Planheitsabweichung.

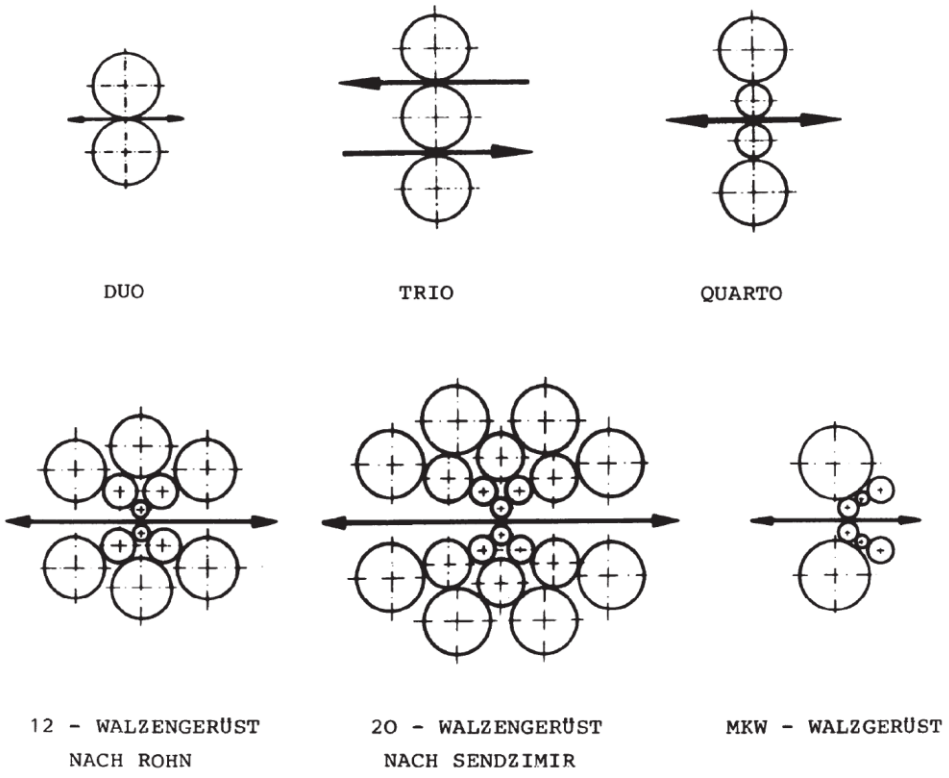


Abb. 9: Verschiedene Arten von Walzgerüsten [6]

Zusätzlich zur Auffederung des Walzgerüstes und der Durchbiegung der Arbeitswalzen nimmt die Walzenplättung Einfluss auf den Prozess. Durch die Flächenpressung zwischen Walzgut und Walzenoberfläche kommt es zu einer reversiblen bzw. elastischen Deformation der Walze im Bereich des Walzspaltes, was auch als Walzenplättung bezeichnet wird. Diese Walzenplättung führt zu einer Vergrößerung des Walzenradius und dadurch in weiterer Folge auch zu einer Vergrößerung der gedrückten Länge [8].

Zudem führt dieses Phänomen in Abhängigkeit von lokalen Kräften zu einer Vergrößerung des Walzspaltes und zu einer Profil- bzw. Planheitsabweichung des gewalzten Walzgutes. Profil- bzw. Planheitsabweichungen treten vor allem bei steigender Walzgutdicke auf. Zur Vermeidung von Planheitsfehlern des Walzgutes können Arbeitswalzen zum Ausgleich der Walzendurchbiegung vorgebogen werden, oder ein CVC-System eingesetzt werden. Das CVC-System verwendet flaschenförmige Walzen, welche durch eine gezielt axiale Verschiebung der Walzen die Steuerung der Dickenabnahme über die Breite ermöglicht. Durch axial verstellbare Zwischenwalzen kann die Walzendurchbiegung zusätzlich reduziert werden – man spricht dann von einem Sechswalzensystem (siehe Abb. 10) [9].

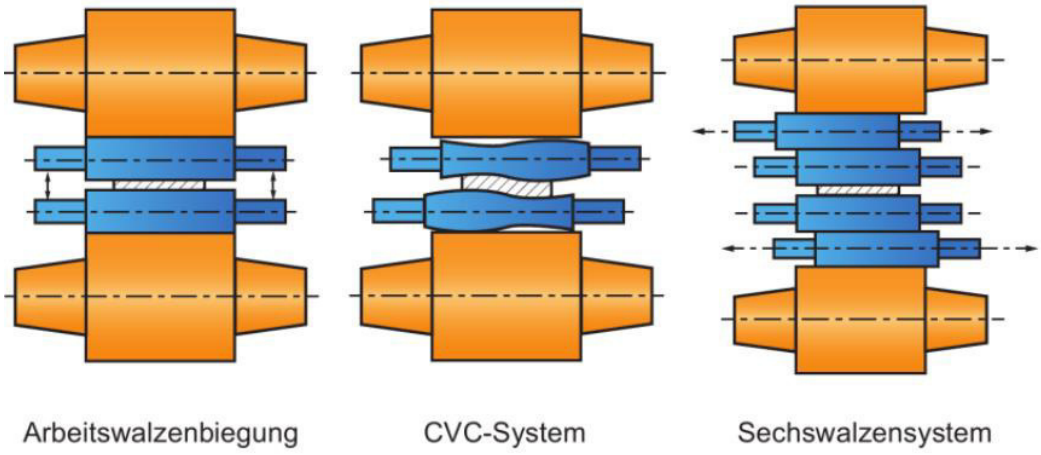


Abb. 10: Methoden zur Vermeidung von Planheitsfehlern [23]

3.3. Sensorik

Die Sensorik, eine Teildisziplin der Messtechnik, befasst sich mit der Anwendung und Entwicklung von Sensoren und dem Erfassen und Messen von physikalischen Größen, welche durch die Sensoren in elektrische Signale umgewandelt werden [10].

Ein Sensor besteht aus einem für die zu erfassende Messgröße geeigneten Messfühler, welcher durch physikalische Effekte und innerelektrische Gesetze die Messgröße in ein elektrisches Signal transformiert [10]. Sensoren können hinsichtlich der physikalischen Grundlage der Messung [10], dem sogenannten Messprinzip, unterschieden werden. Die Sensorkennline dient zur Herstellung des Zusammenhanges zwischen dem analogen Signal und dem digitalen, zur weiteren computergestützten Verarbeitung

benötigten Wert. Die Auflösung eines Sensors beschreibt die kleinste, messbare Zustandsänderung einer zu messenden physikalischen Größe, die vom Sensor wahrgenommen werden kann [10].

Die Linearität eines Sensors, oft auch als Nichtlinearität bezeichnet, gibt die höchste Abweichung von der Ideallinie, auch Kennlinie genannt, des Messwertes einer physikalischen Größe an. Die Folge ist ein Streuband um die Kennlinie, welches in Prozent des Messbereichs (% d.M.) angegeben wird, in welcher der Istwert der Messung liegt (Abb. 11) [14].

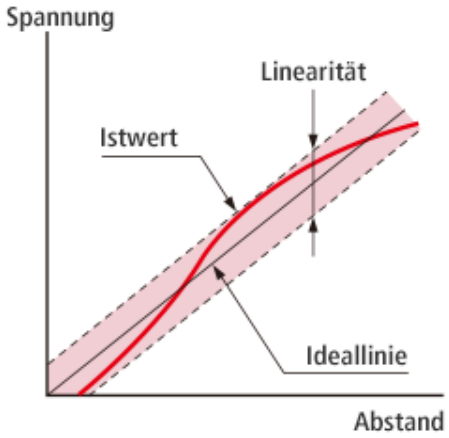


Abb. 11: Linearitätskurve [15]

Nach ISO IEC 2382-1 (1993) sind Daten als eine wieder interpretierbare Darstellung von Information zu verstehen, welche nach einer bestimmten Syntax kodiert sind. Informationen werden dann gewonnen, wenn Daten miteinander in Zusammenhang gebracht und interpretiert werden (auch bekannt als Semantik). Werden Informationen visualisiert oder anderweitig dargestellt, so spricht man von Signalen, welche sich in analoge und digitale Signale unterteilen lassen. Ein analoges Signal ist eine sich kontinuierlich veränderlich physikalische Größe, wie beispielsweise die Temperatur. Ein digitales Signal ist ein diskretes Signal, welches Informationen nur in gewissen Grenzen, der Auflösung, darstellen kann. Die Datenerfassung (engl. data acquisition (DAQ)) befasst sich mit der Aufnahme von analogen Signalen mithilfe geeigneter Systeme (Abb. 12). [10, 16]

Um analogen Signale zur weiteren Verarbeitung in computergestützten Systemen nutzen zu können, müssen diese zuerst in digitale Signale umgewandelt werden. Hierzu werden Analog-Digital-Wandler, kurz A/D-Wandler, eingesetzt, welche das analoge Signal des Sensors, eine elektrische Größe, interpretieren und in ein digitales Signal umwandeln. [10]

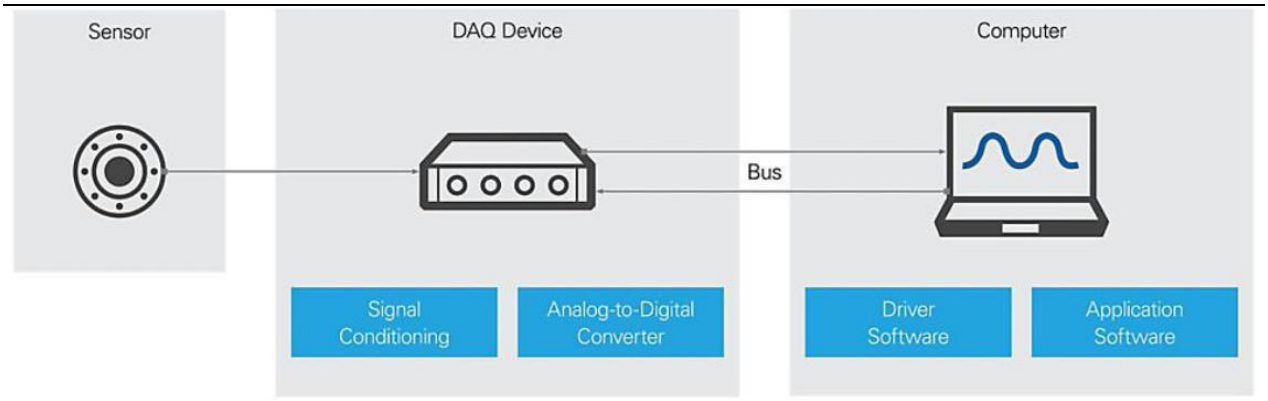


Abb. 12: Messdatenerfassung [17]

3.3.1. Wirbelstromsensor

Wirbelstromsensoren messen den Abstand zu elektrisch leitenden Messobjekten bzw. Werkstoffen mit ferromagnetischen und auch nicht ferromagnetischen Eigenschaften berührungslos und verschleißfrei, weshalb sie zur Gruppe der berührungslosen Abstandsensoren gezählt werden. [12, 13, 18, 21]

Das Messprinzip bedient sich der Induktivitätsänderung einer Spule, die auftritt, wenn ein elektrisch leitendes Objekt in das elektromagnetische Feld der Spule eintritt. Die Spule, welche sich im Sensorkopf des Wirbelstromsensors befindet, wird mit einem hochfrequenten Wechselstrom gespeist, welches ein elektromagnetisches Spulenfeld erzeugt. Befindet sich ein elektrisches Messobjekt im elektromagnetischen Feld der Spule, werden im Messobjekt Wirbelströme induziert, was eine Impedanzänderung der Spule verursacht. Diese Impedanzänderung ist proportional zum Abstand zwischen Messobjekt und Spule. [13, 18]

Durch die Fähigkeit der berührungslosen Abstandsmessung in Kombination mit der exklusiven Abstandsmessung zu elektrisch leitenden Objekten bzw. Werkstoffen ist es möglich, durch nichtmetallische Objekte hindurch den Abstand zum Messobjekt zu erfassen. Dies ermöglicht die Anwendung von Wirbelstromsensoren auch bei Verschmutzung, wie etwa durch Schmiermittel und Öle. Eine weitere Anwendung der Wirbelstromsensoren stellt die Schichtdickenmessung von elektrisch leitenden Objekten da, welche eine nicht elektrisch leitende Beschichtung besitzen. [13, 21]

Abb. 13: Wirbelstromsensor [12]

3.3.2. Linearpotentiometer

Linearpotentiometer, auch potentiometrische Wegaufnehmer genannt, dienen der Abstandsmessung und zählen zur Gruppe der tastenden Wegaufnehmer. Der Abstand wird ähnlich wie bei den Wirbelstromsensoren durch eine auf dem ohmschen Gesetz basierende Impedanzänderung gemessen. Eine Impedanzänderung erfolgt dann, wenn sich die Position des mechanischen Schleifkontaktes auf der Widerstandsbahn und damit auch der Schleifkontaktwiderstand R_x ändert (Abb. 14). Durch diese Änderung des Widerstands lässt sich gemäß Gl. 3.18 die Schleifkontaktspannung U_a, das analoge Signal des Sensors, in Abhängigkeit von der angelegten Spannung U_s, dem Bürdenwiderstand R_B, dem Widerstandsbahnwiderstand R_W und dem Schleifkontaktwiderstand R_x errechnen. Alternativ bieten Linearpotentiometer auch die Möglichkeit das analoge Signal in Form eines elektrischen Stromes auszugeben. [26]

$$U_a = U_s \cdot \frac{R_x \cdot R_B}{(R_W - R_x) \cdot R_x + R_B \cdot R_W} \rightarrow wenn\ R_B \gg R_W \rightarrow U_a \approx U_s \cdot \frac{R_x}{R_W} = U_s \cdot \frac{x}{x_{max}} \quad (3.18)$$

Folglich Gl. 3.18 ergibt sich eine lineare Kennlinie, wenn der Bürdenwiderstand R_B viel größer als der Widerstandsbahnwiderstand R_W ist. Würde die Spannungsmessung nicht hochohmig durchgeführt werden, könnten die in Gl. 3.18 getroffen Vereinfachungen nicht angewendet werden und es ergäbe sich eine degressive Kennlinie, was eine Linearisierung notwendig machen würde. Durch die hochohmige Spannungsmessung kann die Schleifkontaktspannung U_a als Produkt der angelegten Spannung U_s und dem Verhältnis von Scheifkontaktposition x zur Widerstandsbahnlänge x_{max} beschrieben werden [26].

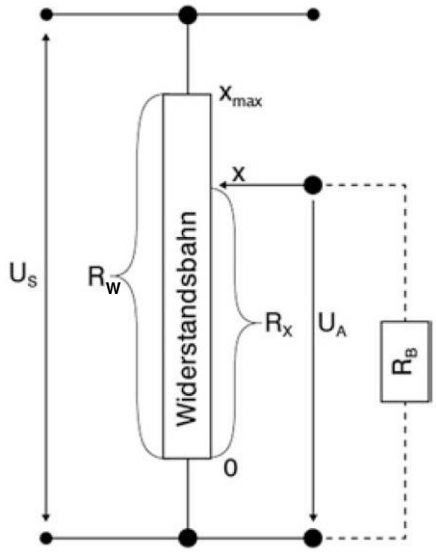


Abb. 14: Systematischer Aufbau eines potentiometrischen Wegaufnehmers [26]

3.3.3. Kraftmessdose

Kraftmessdosen, auch als Kraftaufnehmer bezeichnet, messen die auf sie wirkende Kraft mithilfe verschiedener Messprinzipien. Kraftaufnehmer werden nach dem verwendeten Messprinzip unterschieden (Abb. 15). [10]

Im Folgenden wird aufgrund der häufigen Verwendung nur auf die DMS-, piezoelektrischen und induktiven Kraftaufnehmer eingegangen.

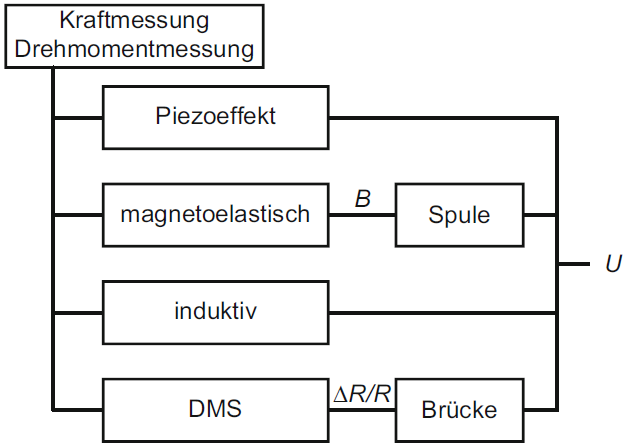


Abb. 15: Prinzipien der Kraft- und Drehmomentmessung [10]

Kraftaufnehmer, basierend auf dem Dehnungsmessstreifen (DMS)-Prinzip, bestehen aus einem Federkörper und den DMS-Streifen, welche auf dem Federkörper aufgebracht sind (Abb. 16). Durch eine auf den Federkörper wirkende Kraft kommt es zu einer elastischen Verformung des Federkörpers. Durch die elastische Verformung des Federkörpers werden auch die auf ihm aufgebrachten DMS gedehnt, was zu einer Impedanzänderung der DMS führt. Das Messprinzip der DMS beruht auf einer Impedanzänderung, hervorgerufen durch eine dehnungsinduzierte Querschnittsänderung. Hierfür kommt eine sogenannte Wheatstoneschen Messbrücke zum Einsatz, welche mindestens aus vier DMS besteht. Die Wheatstonesche Messbrücke liefert eine zur anliegenden Kraft proportionale Ausgangsspannung in Abhängigkeit von der Speisespannung der Messbrücke [10, 19, 27].

Aufgrund des sehr geringen Drifts, auch als Kriechen bezeichnet, einer zeitabhängigen, reversiblen Änderung des Ausgangssignals bei konstanter Belastung, eignen sich DMS-Kraftaufnehmer besonders für langfristige Messaufgaben [19, 31].

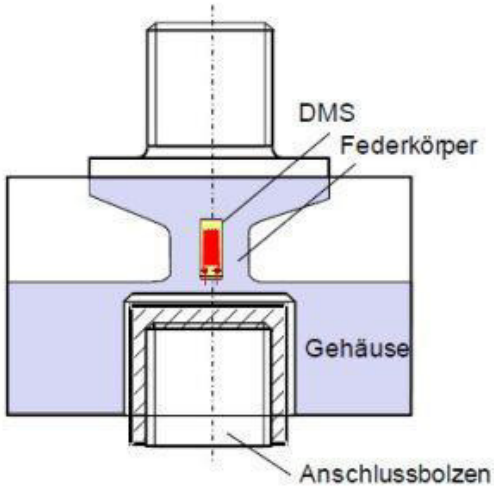


Abb. 16: Schema eines DMS-basierenden Kraftaufnehmers [20]

Piezoelektrische Kraftaufnehmer erzeugen durch piezoelektrische Kristallscheiben eine zur belastenden Kraft proportionale Ladung. Diese piezoelektrische Kristallscheiben werden durch eine Elektrodenfolie getrennt, die die unter Krafteinwirkung entstehende Ladung aufnimmt (Abb. 17). Zur weiteren Messung der Ladung wird ein Ladungsverstärker nachgeschalten. Der Drift von piezoelektrischen Kraftaufnehmern wird mit ungefähr 1 N/min beziffert, welcher abhängig von der Größe der zu messenden Kräfte mehr oder weniger ins Gewicht fällt. Bei großen Kräften verfälscht dieser Drift den wahren Wert der Messung daher deutlich geringer als bei einer Messung von sehr kleinen Kräften. [19]

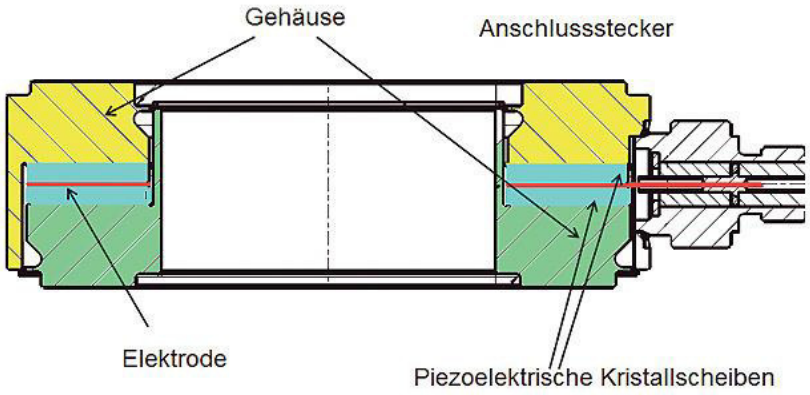


Abb. 17: Schema eines piezoelektrischen Kraftaufnehmers [20]

Induktive Kraftaufnehmer ermitteln die anliegende Kraft mithilfe eines Verformungs – bzw. Federkörpers. Das Messprinzip basiert auf der Abstandsänderung des Federkörpers zwischen zwei Punkten, wodurch die einwirkende Kraft ermittelt werden kann. Aufgrund der geringen Messtoleranz und hohen Genauigkeit eignen sie sich besonders zum Messen von sehr kleinen Kräften [4, 24].

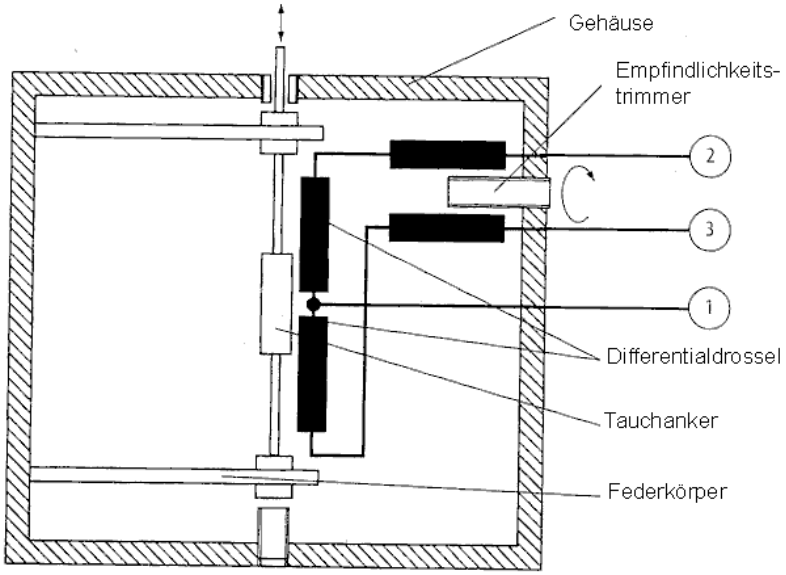


Abb. 18: Schema eines induktiven Kraftaufnehmers [25]

3.3.4. Drehmomentsensor

Ein Drehmomentsensor dient zur Erfassung des Drehmoments. Die verwendeten Messprinzipien können Abb. 15 entnommen werden und funktionieren analog zu den in Abschnitt 3.3.3. beschriebenen Prinzipien. Hierbei muss die gemessene Kraft zusätzlich mit dem orthogonal auf die Kraft und Rotationsachse stehenden Hebelarm multipliziert werden, um das vorhandene Drehmoment zu errechnen.

3.3.5. Drehzahlsensor

Es existieren zahlreiche Prinzipien zur Messung der Drehzahl, einige darunter sind nach [26] Wirbelstromdrehzahlsensoren, Stroboskope, Tacho-Generatoren, Wechselstrom-Generatoren, Unipolarmaschinen und Impulsdrehzahlsensoren. Eine günstige Variante der Drehzahlmessung kann mit einem Hodometer durchgeführt werden, jedoch ist hier aufgrund des ungleichen Durchmessers von Zählrad und Messobjekt eine Umrechnung der Drehzahl erforderlich. Aufgrund der rein theoretischen Implementierung eines Drehzahlsensors im Laufe der Arbeit wird hier nicht weiter auf die Messprinzipien der Drehzahlsensoren eingegangen.

3.3.6. Winkelsensor

Winkelsensoren, auch Drehgeber genannt, dienen zur Erfassung der winkelbezogenen Position einer Achse oder Welle. Abhängig von den zu erfassenden Umdrehungen spricht man bei einer einzigen Umdrehung von Single-Turn-Drehgebern und bei mehreren Umdrehungen von Multi-Turn-Drehgebern. Die Erfassung der Position bei optischen Encodern erfolgt durch die Verwendung einer Codierscheibe, welche nach [28] hell/dunkel-codierte Felder oder geätzte Schlitzstrukturen in Metallscheiben besitzt. Das optische Muster, welches abhängig von der Scheibenposition ist, wird durch einen Photodetektor-Array und eine Lichtquelle erfasst und erzeugt ein digitales Signal, welches zur weiteren computergestützten Weiterverarbeitung verwendet werden kann. Bei magnetischen Encodern werden mithilfe von Hallsensoren magnetische Pole im Sensor erfasst und durch die Modulation des Magnetfeldes die Position bestimmt. [28]

3.3.7. Drucksensor

Zur Messung des Drucks wird sich einer großen Anzahl an verschiedenen Prinzipien bedient, einige dieser Prinzipien sind nach [26] piezoresistive Drucksensoren, kapazitive Drucksensoren und Halleffekt Drucksensoren. Da im Laufe dieser Arbeit nur eine theoretische Implementierung eines Drucksensors erfolgt, wird hier nicht weiter auf die Messprinzipien der Drucksensoren eingegangen.

3.4. A/D-Wandler

Um analoge Signale, wie sie Sensoren liefern, in computergestützten, digital arbeitenden Systemen verarbeiten zu können müssen diese erst in digitale Signale umgewandelt werden. Hierfür kommen

Analog-Digital-Wandler, kurz A/D-Wandler, zum Einsatz um das analoge Signal in eine proportionale digitale Größe zu überführen. [10]

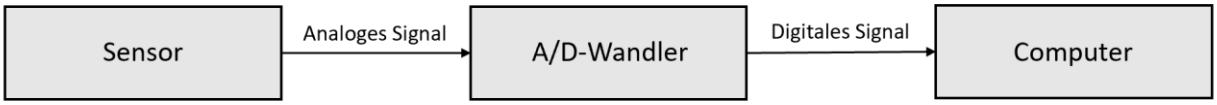


Abb. 19: Prinzip eines A/D-Wandlers

Die physikalische Messgröße wird durch das Messprinzip mithilfe eines physikalischen Effektes in eine elektrische Größe umgewandelt, welche durch innerelektrische Gesetze in einen elektrischen Strom oder eine elektrische Spannung umgewandelt werden. Dieser elektrische Strom oder diese elektrische Spannung sind das analoge Signal, welches in A/D-Wandler eingeht. Durch die Normierung befinden sich diese elektrischen Spannungen im Bereich von 0-10 V und die elektrischen Ströme im Bereich von 0-20 mA und 4-20 mA, welche in ein digitales Signal, hier Sensorausgangssignal genannt, umgewandelt werden können (Abb. 20). [10]

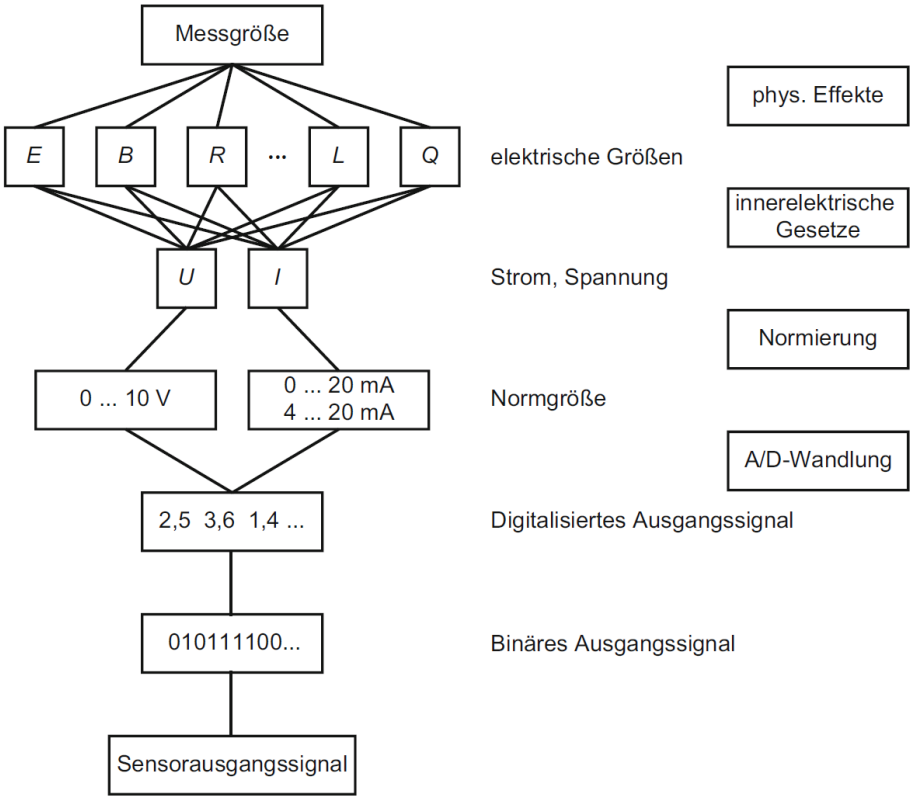


Abb. 20: Umwandlung einer Messgröße in ein digitales Signal [10]

A/D-Wandler arbeiten mit einer Abtastfrequenz, auch Taktung genannt, welche das analoge Signal mit einer definierten Abtastfrequenz abtasten und dieses so in ein proportionales, digitales Signal überführen. Je höher die Abtastfrequenz, mit der das analoge Signal abgetastet wird, desto besser spiegelt das digitale Signal den Verlauf des analogen Signals wider. [11]

Dennoch gilt, dass die Abtastrate nicht so hoch wie möglich gewählt werden sollte, sondern lediglich an den Prozess angepasst werden soll, da eine zu hohe Abtastrate kaum Vorteile mit sich bringt. [35]

Da für digitale Werte keine unendlich hohe Auflösung möglich ist, muss der Analogwertbereich, also jener Bereich in dem sich das Analogsignal bewegt, unterteilt werden. Diese Unterteilung des Analogwertbereichs wird auch Quantisierung genannt. Durch die Quantisierung wird der Digitalwertbereich mit einer Bitkombination in eine endliche Anzahl von Werten unterteilt, was eine Stufenfunktion bzw. Treppenkurve ergibt, durch deren Stufenmittelpunkte eine Gerade gelegt werden kann (Abb. 21). Schneidet man die Gerade mit der Treppenkurve erhält man an deren Schnittpunkten den Digitalwert, der alle in diesem Bereich liegenden Analogwerte darstellt. Je höher die Auflösung des Systems, desto geringer ist die Breite der Stufen, welche den kleinsten wahrnehmbaren Unterschied der Stufenfunktion repräsentieren [16].

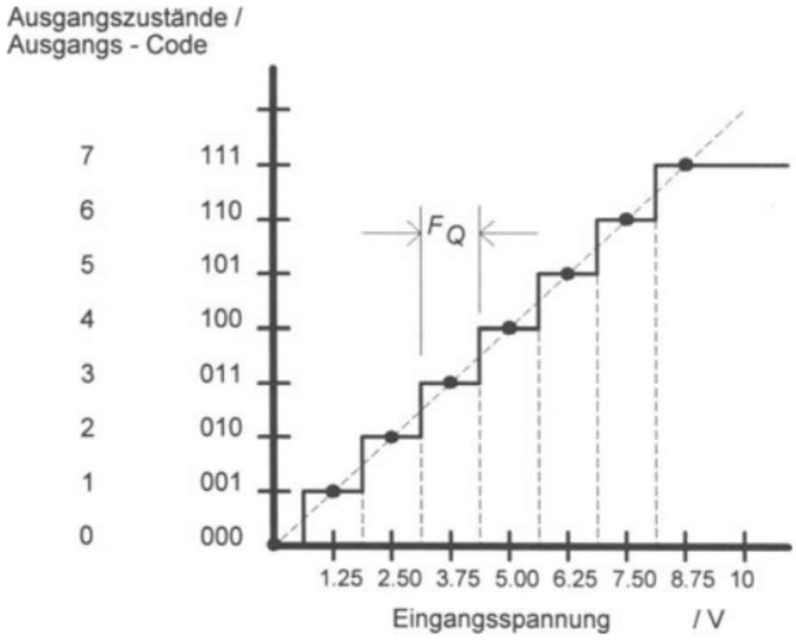


Abb. 21: Kennlinie eines 3-bit-A/D-Wandlers [16]

3.5. Industrie 4.0

Der Begriff Industrie 4.0 beschreibt das Vorhaben einer umfassenden Digitalisierung in der Industrie, vor allem in der Produktion. Im Zuge dessen kam es zu einer Erweiterung der gängigen Terminologie, auf die nachfolgend näher eingegangen wird. „Cyber Physical Systems“ (CPS) beschreiben ein System, welches Daten mit Internettechnologien („Internet of Things“ (IoT)) sammelt, speichert, analysiert und verarbeitet und diese Daten in der virtuellen und realen Welt integriert, um in weiterer Folge eine menschliche Interaktion zu ermöglichen. Die Erweiterung dieses Begriffs wird als „Cyber Physical Production Systems“ (CPPS) bezeichnet. Als Erweiterung zur Interaktion zwischen Computerwissenschaften, Informations- und Kommunikationstechnologie und menschlicher Interaktion durch grafische „User-Interfaces“ (GUI), welche bereits durch das CPS-Konzept bekannt sind, wird die Automatisierung miteinbezogen (siehe Abb. 22). Durch den Miteinbezug der Automatisierung in die Industrie 4.0 kommt es zur Kombination von Technologien der Industrie 3.0, wie Sensoren und Aktoren, mit moderner Informationstechnologie (Hard- und Software). [29]

Werden Sensoren, Aktoren und Maschinenkomponenten durch moderne IT und Software miteinander vernetzt, können diese über ein digitales Netzwerk miteinander kommunizieren, indem sie Daten erhalten und senden, um sich so untereinander abzustimmen. Dadurch wird eine standort- und unternehmensübergreifende Kommunikation von CPS ermöglicht [30].

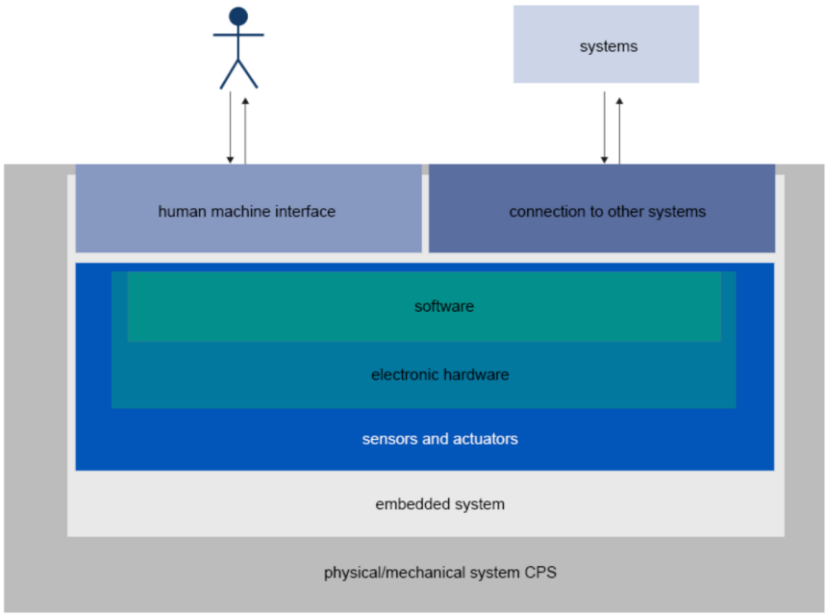


Abb. 22: Schema eines „Cyber Physical Systems“ (CPS) [40]

Ein „Digital Twin“ (DT) ist als eine digitale Repräsentanz eines physischen Produktes definiert. Für die metallverarbeitende Industrie ist es jedoch notwendig diesen Begriff zu erweitern. Die erste Unterscheidung ist auf Basis der Anwendung zu treffen. In diesem Industriezweig gibt es zwei Hauptanwendungsbereiche für das Konzept des „Digital Twin“: i.) als Repräsentation eines Produktionsprozesses über die im Laufe des Prozesses gefertigte Teile oder ii.) als gesamtheitlicher Prozess, welcher den realphysikalischen Einfluss auf das verarbeitende Werkstück näher betrachtet. In ii.) liegt der Fokus auf der numerischen Simulation (meist Finite Elemente Analyse (FEA)). [29]

Eine weitere Unterscheidung kann zwischen den Begriffen „Digital Twin“ (DT), „Digital Shadow“ (DS) und „Digital Model“ (DM) getroffen werden. Ein „Digital Model“ beschreibt eine digitale Kopie eines physischen Objekts, jedoch ohne automatisierten Datenaustausch zwischen dem digitalen und physischen Objekt. Ein „Digital Shadow“ beschreibt eine digitale Repräsentanz eines physischen Objektes mit Datenaustausch zwischen beiden Objekten, welcher aber nur unilateral automatisiert abläuft. Ein „Digital Twin“ ermöglicht, ähnlich wie ein „Digital Shadow“, einen Datenaustausch zwischen digitaler Repräsentanz und physischem Objekt, jedoch läuft dieser bilateral automatisiert ab (Abb. 23). [29]

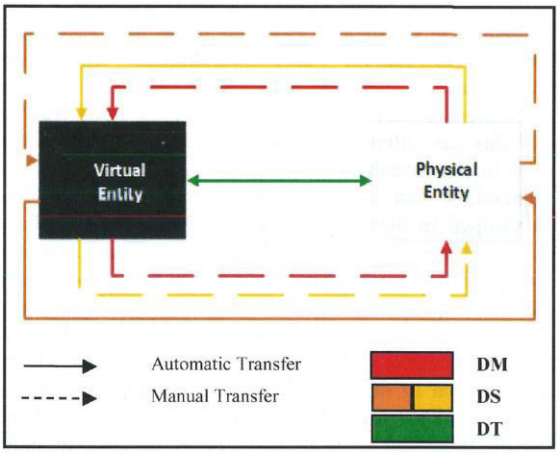


Abb. 23: Vergleich des Datenflusses zwischen DM, DS und DT [29]

„Retrofitting“ beschreibt im Kontext der Industrie 4.0 ein Upgrade einer bestehenden Anlage oder Maschine auf den Stand der Industrie 4.0. In wissenschaftlichen Publikationen beschränkt sich dieser Prozess im Wesentlichen auf die Konzipierung, Implementierung und Validierung von geeigneter Infrastruktur, Kommunikations- und Informationstechnologie. Das Ziel eines solchen „Retrofittings“ ist es eine Anlage oder Maschine mit älterer Technologie zu einem CPPS aufzurüsten, um dieses in weitere Folge in eine Industrie 4.0 Umgebung einzugliedern (auch als „brownfield-approach“ bezeichnet). [29]

4. Walzwerk

Das am Lehrstuhl für Umformtechnik der Montanuniversität Leoben vorhandene Walzwerk soll einem „Retrofitting“ unterzogen werden, um dieses auf den Stand der Industrie 4.0 zu bringen. Bei diesem Walzwerk handelt es sich um ein Duowalzwerk, ein Walzwerk mit zwei gleichläufigen Arbeitswalzen aus Werkzeugstahl (Abb. 24). Bei dem Walzwerk handelt es sich um eine Maßanfertigung einer externen Firma, weshalb kaum Standardbauteile verbaut wurden.

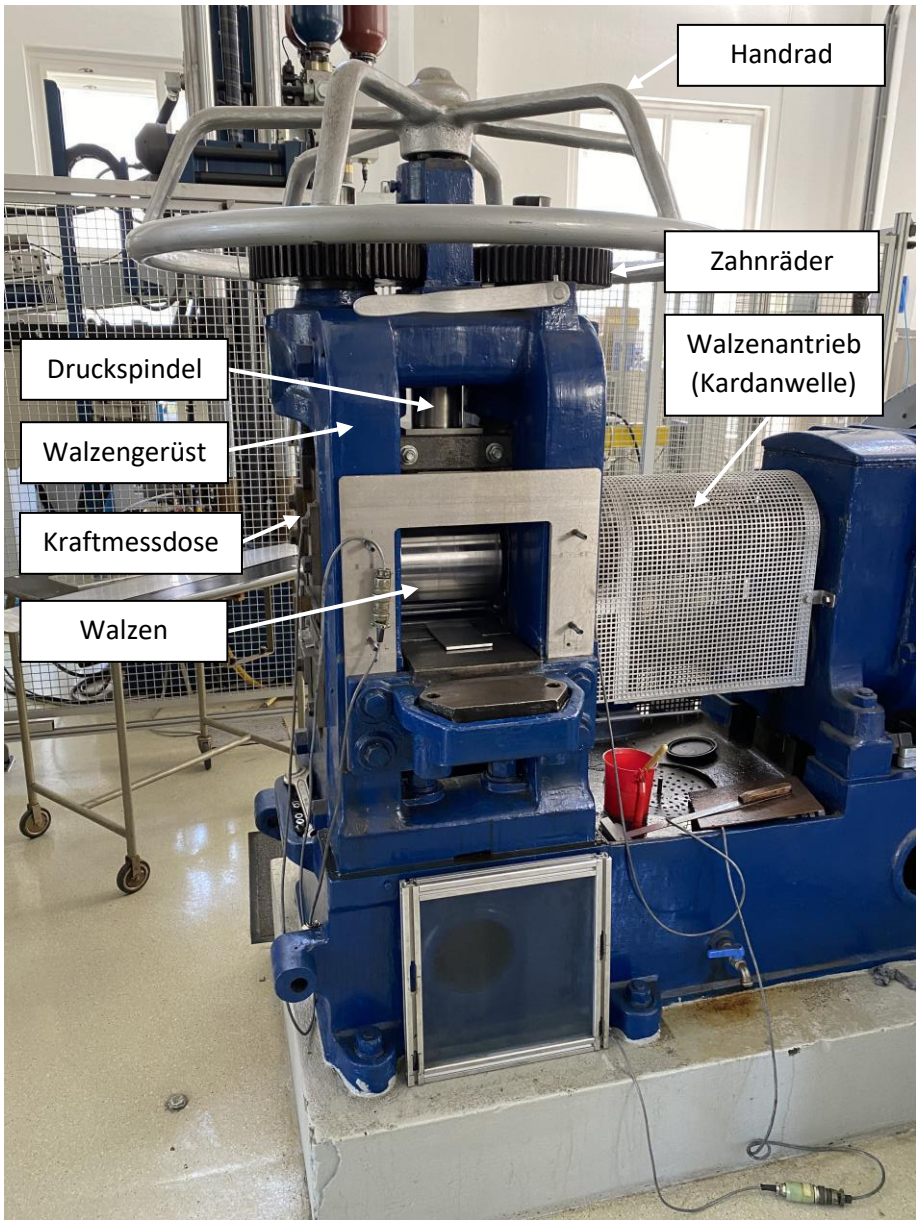


Abb. 24: Duowalzwerk am Lehrstuhl für Umformtechnik

Die obere Arbeitswalze ist bereits mit zwei Kraftmessdosen ausgestattet, welche nach dem DMS-Funktionsprinzip arbeiten (Abb. 25).

Abb. 25: Kraftmessdose 2 des Walzwerks

Durch das Handrad, welches sich auf dem Walzwerk befindet, ist es möglich durch eine Anstellspindel mit Übersetzung die Höhe des Walzspaltes zu justieren (Abb. 24). Ein Zahn an den Zahnrädern der Anstellspindel entspricht ungefähr 0,07mm Walzenzustellung.

Aus Tabelle 1 können die wichtigsten Kennwerte des Walzwerkes entnommen werden:

Tabelle 1: Kennwerte des Walzwerkes

Kennwert	Variable	Wert
Walzendurchmesser	d_W	203 mm
Maximale Walzkraft	F_{max}	300 kN
Maximales Walzmoment	M_{max}	2500 Nm
Ballenlänge der Walzen	l_B	220 mm
E-Modul des Walzenwerkstoffes	E	210000 N/mm²

5. Software

Zur Programmierung eines DS für das Walzwerk wurde die objektorientierte Programmiersprache Python verwendet. Ein Grund weshalb Python als Programmiersprache gewählt wurde, ist die mittlerweile weite Verbreitung. Python war nach dem TIOBE Index im September 2020 bereits die drittmeist verwendete Programmiersprache der Welt (siehe Abb. 26) [45]. Ein weiterer Grund Python zu verwenden war, dass es sich bei Python um eine „Open-Source“-Programmiersprache handelt die bereits eine Vielzahl verschiedenster „Frameworks“ bereitstellt, welche ständig erweitert und verbessert werden. Ein weiterer Grund für die Verwendung von Python war zu zeigen, dass „Open-Source“-Produkte wie Python anderen „Closed-Source“-Programmiersprachen gegenüber konkurrenzfähig sind und diese somit auch für kleine und mittelständische Unternehmen kostengünstige alternativen darstellen. Die Messdatenerfassung erfolgt durch einen Controller und der eCockpit-Software der Firma Wago.

Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	15.95%	+0.74%
2	1	↓	Java	13.48%	-3.18%
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%
6	6		Visual Basic	4.12%	+0.83%
7	7		JavaScript	2.54%	+0.41%
8	9	↑	PHP	2.49%	+0.62%
9	19	⇈	R	2.37%	+1.33%
10	8	↓	SQL	1.76%	-0.19%
11	14	↑	Go	1.46%	+0.24%
12	16	⇈	Swift	1.38%	+0.28%
13	20	⇈	Perl	1.30%	+0.26%
14	12	↓	Assembly language	1.30%	-0.08%
15	15		Ruby	1.24%	+0.03%
16	18	↑	MATLAB	1.10%	+0.04%

Abb. 26: TIOBE Index im September 2020 [45]

6. Konzipierung

Die Konzipierungen zur Digitalisierung des Walzwerks umfassen die Sensorik, Programmierung sowie deren Auswertung und Ergebnisse. In Abschnitt 6.1. und 6.2. werden, basierend auf den zu erfassenden Messgrößen, geeignete Sensoren gewählt und auf Basis dieser Sensorik ein Python-Programm zur Messdatenerfassung, -verarbeitung, -auswertung und -visualisierung verfasst. Im letzten Schritt wird jedes Konzept durch die Einspeisung exemplarischer Messdaten getestet.

6.1. Konzept 1

Das erste Konzept stellt den Versuch da, alle relevanten Messgrößen des Walzwerks zu erfassen. Hierzu kommt eine Vielzahl verschiedener Sensoren zum Einsatz, um diese Messgrößen zu erfassen. Durch eine eigens entwickelte Sensoranordnung soll die Durchbiegung der unteren Arbeitswalze erfasst werden (Abschnitt 6.1.1.). Die von der Sensorik erfassten Messdaten ermöglichen im Zusammenspiel mit der Python-Programmierung die Auswertung und Visualisierung der aufgenommenen Messdaten sowie die Visualisierung der Walzendurchbiegung (Abschnitt 6.1.2. & 6.1.3.).

6.1.1. Sensorik

Die Sensorik umfasst insgesamt 15 Sensoren, die 16 Messgrößen erfassen. Die Gesamtheit dieser Sensoren ermöglicht es Daten über die Durchbiegung der untere Arbeitswalze, die Größe und Auffederung des Walzspaltes zwischen den beiden Arbeitswalzen, das Walzendrehmoment, die Walzendrehzahl, die Walzkraft, den Winkel des Handrades sowie den Pumpendruck zu sammeln.
Um die Durchbiegung der Arbeitswalze abschätzen zu können wurde eine Vorberechnung zur Abschätzung der zu erwartenden Durchbiegung anhand eines Biegebalkens durchgeführt (Anhang A). Hierfür wurde als Biegeträger die Arbeitswalze mit ihren Abmessungen und die angreifende Kraft als maximale Walzkraft substituiert (Abb. 27).

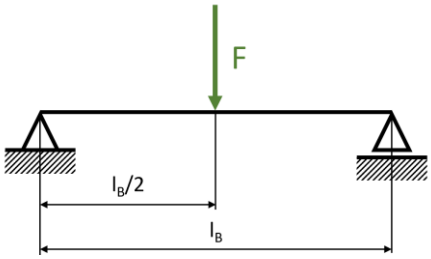


Abb. 27: Vereinfachung der Walze als Biegeträger

Aus der Literatur folgt die Gleichung für die maximale Durchbiegung w_{max} eines Biegeträgers, mit der Länge l, dem Flächenträgheitsmoment I und dem Elastizitätsmodul E, welcher mittig mit der Kraft F belastet wird:

$$w_{max} = \frac{F \cdot l^3}{48 \cdot E \cdot I} \tag{6.1}$$

Wobei sich das Flächenträgheitsmoment I für einen Kreisquerschnitt mit dem Durchmesser d wie folgt berechnet:

$$I = \frac{d^4 \cdot \pi}{64} \tag{6.2}$$

Aus den Kennwerten folglich Tab. 1 lassen sich alle für die Vorberechnung notwendigen Größen entnehmen. Setzt man Gl. 6.2 in Gl. 6.1 mit den Kennwerten aus Tabelle 1 ein, so erhält man:

$$w_{max} = \frac{F_{max} \cdot {l_B}^3}{48 \cdot E \cdot \frac{{d_W}^4 \cdot \pi}{64}} = \frac{300 \cdot 10^3 \cdot 220^3}{48 \cdot 210000 \cdot \frac{203^4 \cdot \pi}{64}} = 3{,}8\mu m \tag{6.3}$$

Aus der Vorberechnung (Gl. 6.3) geht hervor, dass an der Stelle der maximalen Durchbiegung der Walze eine Durchbiegung von 3,8µm zu erwarten ist. Hinzu kommen weiterer Anteile die zur Durchbiegung beitragen, wie z.B. durch die elastische Deformation der Walzenlager.

Durch die sehr geringe Durchbiegung im einstelligen µm-Bereich sind Sensoren mit hoher Genauigkeit unerlässlich. Der Begriff Genauigkeit setzt sich hierbei aus den Begriffen Präzision bzw. Linearität und Richtigkeit zusammen (Abb. 28). Die Genauigkeit ist ein Maß für die Übereinstimmung zwischen dem Messergebnis und dem wahren Wert der Messgröße. Eine hohe Genauigkeit kann man also nur erreichen, wenn die Präzision hoch und die Richtigkeit gut ist (Abb. 28) [22].

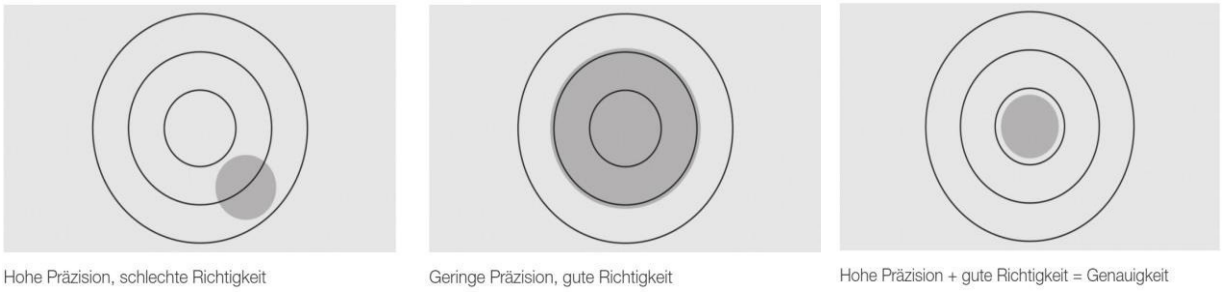


Abb. 28: Genauigkeit eines Sensors [22]

Für die richtige Wahl der Sensoren zur Messung der Walzendurchbiegung müssen die Arbeitsbedingungen mit in Betracht gezogen werden. Beim Kaltwalzen kommen Schmiermittel zum Einsatz, welche die Sensoren verschmutzen und diese in ihrer Funktion beeinträchtigen können. Die zu wählenden Sensoren müssen daher folgende Anforderungen erfüllen:

- hohe Genauigkeit (hohe Präzision + gute Richtigkeit)
- geringe Schmutzempfindlichkeit bis Schmutzunempfindlichkeit

Aufgrund dieser Anforderungen fällt die Wahl auf Wirbelstromsensoren, denn diese bieten eine hohe Auflösung, hohe Genauigkeit sowie Schmutzunempfindlichkeit gegenüber allen nichtmetallischen Stoffen, wie etwa Schmiermitteln.

Um die Durchbiegung der Arbeitswalze messen zu können wurde ein Sensoraufbau mit insgesamt neun Wirbelstromsensoren entwickelt. Jeweils drei Wirbelstromsensoren befinden sich in insgesamt drei Ebenen normal zur Walzenachse (Abb. 29).

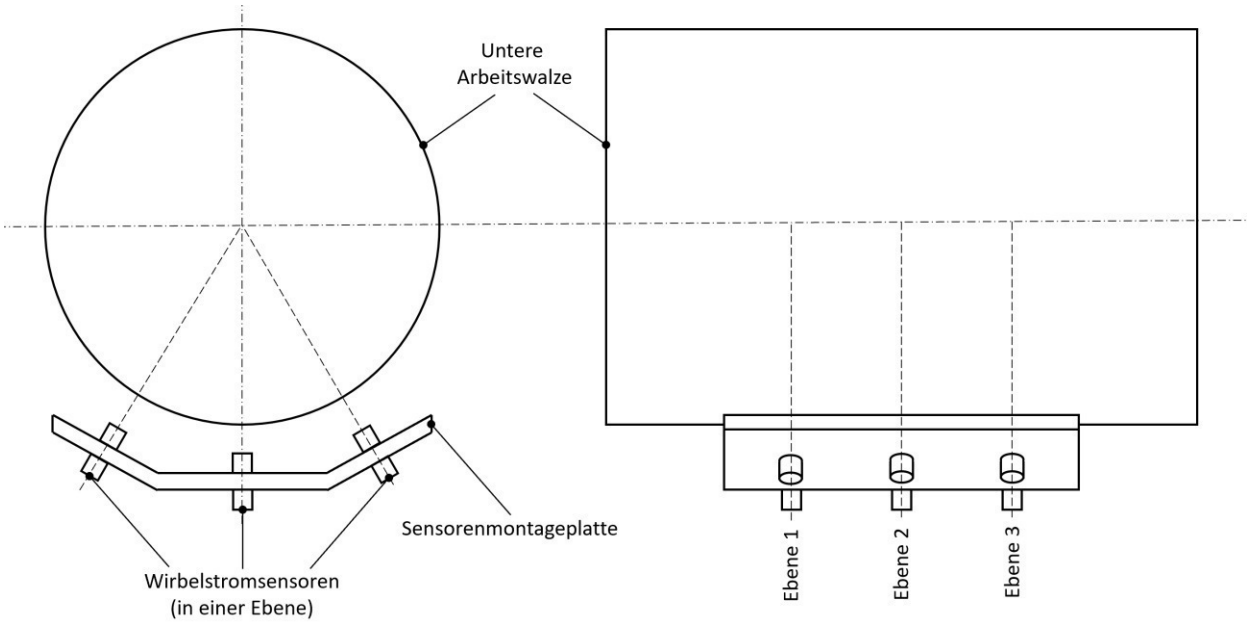


Abb. 29: Schematische Anordnung der Wirbelstromsensoren

Durch diesen Aufbau ist es möglich in jeder Ebene durch die Messpunkte in Kombination mit den bekannten Sensorpositionen drei Punkte zu ermitteln, die sich auf der Arbeitswalze befinden. Durch diese drei Punkte ist der Walzenquerschnitt, ein Kreis, eindeutig in seiner Lage und Form bestimmt. Es kann also die Lage im Raum über den Kreismittelpunkt und den Radius bzw. Durchmesser errechnet werden (siehe

Abb. 43). Die Montage der Wirbelstromsensoren erfolgt aus Platzgründen unter der unteren Arbeitswalze (Abb. 30).

Für die Messung der Größe des Walzspaltes und Auffederung zwischen den beiden Arbeitswalzen kommt aufgrund der hohen Auflösung ein Linearpotentiometer zum Einsatz. Die Montage des Sensors erfolgt zwischen den beiden Walzlagerdeckeln (Abb. 30). Die maximale Höhe des Walzspaltes beträgt ungefähr 35mm, weshalb das Linearpotentiometer eine Wegmessung von mindestens 35mm ermöglichen muss.

Das Drehmoment der Walze wird über einen Drehmomentsensor mit integriertem Drehzahlsensor ermittelt. Hierbei wurde am Walzwerk antriebsseitig eine Kardanwelle verbaut, was auf einen höheren Winkelversatz zwischen An- und Abtriebsseite schließen lässt, weshalb auf die Eignung bei höherem Winkelversatz des Sensors geachtet werden muss. Für die Montage des Drehmomentsensor wird dieser seriell zwischen Antriebs- und Abtriebsseite, also zwischen der Kardanwelle und der Walze, angebracht. Der Drehmomentsensor muss dabei dem maximalen Drehmoment des Walzwerkes von 250Nm (Tabelle 1) standhalten. Die Walzkraft wird mit zwei bereits vorhandenen DMS-Kraftmessdosen gemessen, welche sich über der oberen Arbeitswalze befinden. Die maximale Walzkraft des Walzwerkes beträgt 300kN. Der Pumpendruck, der zur hydraulischen Vorspannung der oberen Arbeitswalze benötigt wird, wird mit einem Drucksensor (barometer) gemessen, welcher nicht direkt am Walzwerk, sondern an einem externen Aggregat verbaut wird. Auf dem Walzwerk wird anstelle des Handrades ein Servomotor montiert, um mithilfe der aus der Programmierung erhaltenen Daten die Zustellung des Walzspaltes zu regeln (Abb. 30). Die insgesamt 15 verbauten Sensoren sind in Tabelle 2 zusammengefasst.

Tabelle 2: Sensortypen und Anzahl (Konzept 1)

Position in Abb. 30	Sensor	Anzahl	Messwert
1	Wirbelstromsensor	9	Abstand zur Walzenoberfläche [mm]
2	Linearpotentiometer	1	Auffederung zwischen den Walzen [mm]
3	Kraftmessdose	2	Walzkraft am Führungsholm [N]
4	Drehmomentsensor inkl. Drehzahlsensor	1	Drehmoment der Walze [Nm] + Drehzahl der Walze [1/s]
5	Servomotor inkl. Winkelsensor	1	Winkel des Handrades [°]
-	Drucksensor	1	Druck der hydraulischen Vorspannung [MPa]

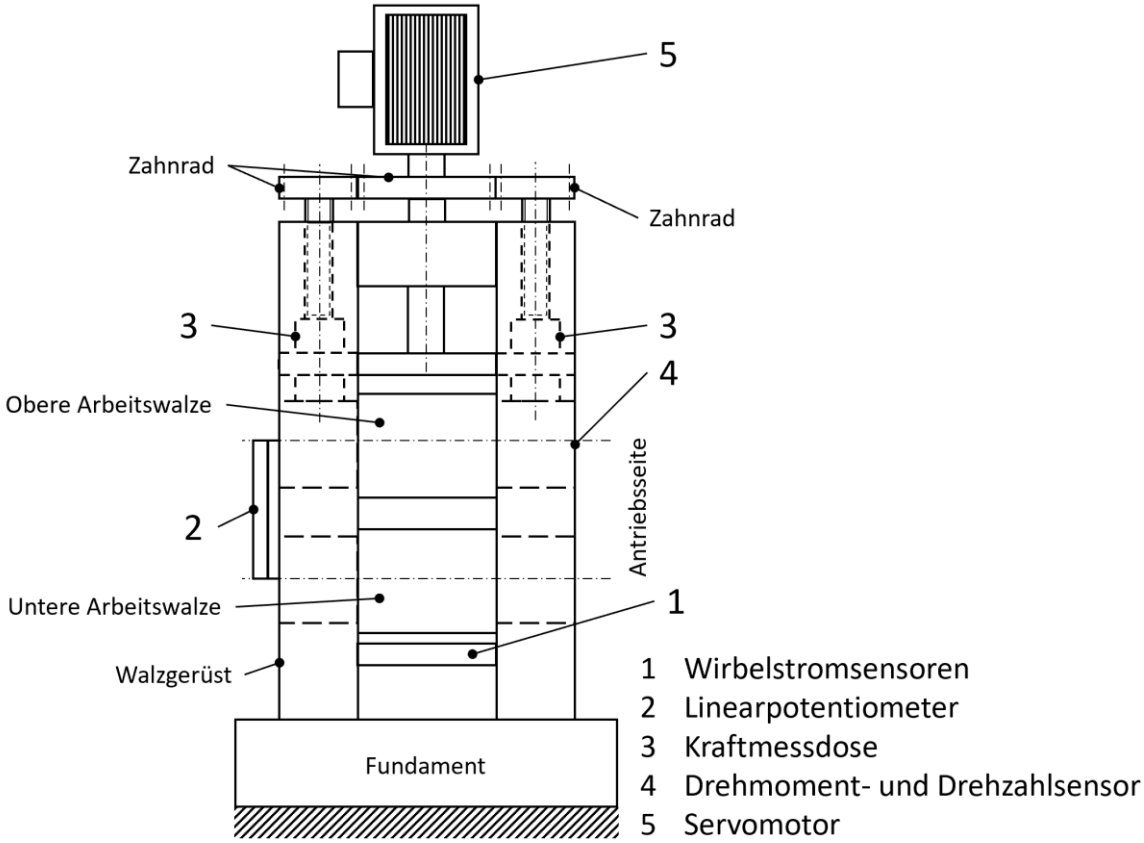


Abb. 30: Walzwerk mit Sensoren (Konzept 1)

Die analogen Messsignale der Sensoren werden über ein A/D-Wandler in digitale Messsignale umgewandelt und anschließend auf einem Rechner mit der dazugehörigen Programmierung (siehe Abschnitt 6.1.2.) ausgewertet. Um diese Daten für den Maschinenbediener leichter zugänglich zu machen, wird ein Tablet verwendet, welches als Bildschirm und Eingabemodul für diesen Rechner dient.

6.1.2. Programmierung

Zur Auswertung der Messdaten und zu deren Visualisierung wurde Python verwendet. Hierzu wurde ein Skript geschrieben, welches alle gewünschten Funktionen ausführt. Dabei greift das Skript auf ebenfalls selbst programmierte Module zu, welche diese Funktionen enthalten und dann ausführen. Abb. 31 zeigt ein Flussdiagramm der verschiedenen Module, die im Zuge des ausführenden Skripts verwendet werden. Der Quellcode kann Anhang A entnommen werden.

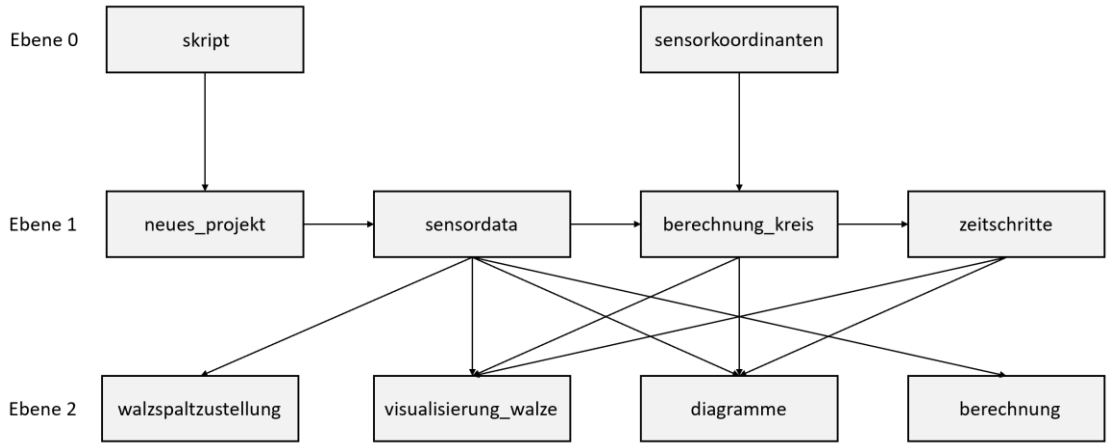


Abb. 31: Flussdiagramm der Module (Konzept 1)

Ebene 0 beschriebt jene Gruppe von Skripten und Modulen, die die Ausführung von Befehlen initialisieren („skript“) und vordefinierten Werte bzw. Daten liefern („sensorkoordinaten“). Ebene 1 beschreibt jene Gruppe von Modulen, die für Datensammlung und -filtrierung zuständig sind. Ebene 2 beschreibt jene Gruppe von Modulen, die für die Auswertung und Visualisierung der Daten, welche durch Module der Ebene 1 bereitgestellt werden, zuständig sind.

6.1.2.1. Modul „neues_projekt“

Das Modul „neues_projekt“ generiert einen neuen Ordner auf einem vordefinierten Pfad mit einem vom Anwender frei wählbaren Projektnamen. Die Benennung des Projektordners erfolgt standardisiert im Format „Jahr-Monat-Tag [Stunde-Minute-Sekunde] – Projektname“ um eine mehrfache Verwendung von Projektordnernamen zu verhindern und so auch Datenverlust vorzubeugen. Zur Generierung des Projektordnernamens wird auf die lokale Uhr des Rechners zugegriffen. Im Projektordner wird ein der Unterordner „csv-data“ generiert, welcher nach Wahl des Anwenders alle csv-Dateien mit den Messdaten der Sensoren und/oder eine Zusammenfassung aller Messdaten der Sensoren enthält.

6.1.2.2. Modul „sensorkoordinaten“

Das Modul „sensorkoordinaten“ stellt die grundlegenden Kennwerte bereit, die für die weitere Messdatenverarbeitung und Visualisierung benötigt werden. Das Modul liefert die x-, y- und z-Koordinaten und Anstellwinkel α_S der Wirbelstromsensoren (Abb. 32), die für die Berechnung und Visualisierung der Durchbiegung der unteren Arbeitswalze benötigt werden. Die Messwerte der Wirbelstromsensoren u_1, u_2 und u_3 werden zusammen mit dem Modul „berechnung_kreis“ und den von Modul „sensorkoordinaten“

gelieferten Werten umgerechnet (siehe Abschnitt 6.1.2.5.). Der Standardabstand im unbelasteten Zustand der Walze zwischen Wirbelstromsensor und Arbeitswalze a kann formal folgend dargestellt werden: $a = u_1 = u_2 = u_3 = 1mm$, dieser variiert jedoch in Abhängigkeit vom gewählten Wirbelstromsensor.

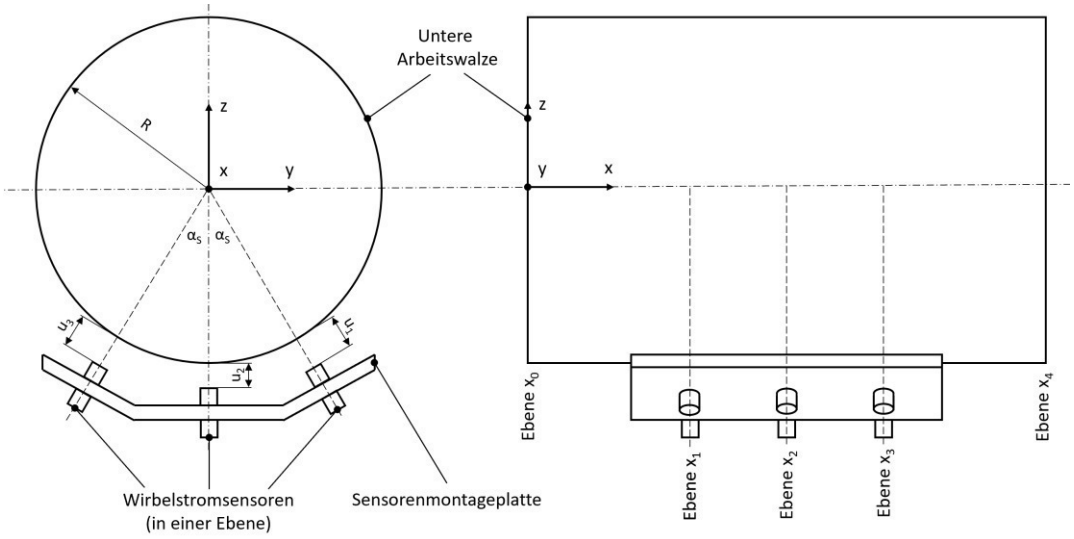


Abb. 32: Sensorkoordinaten

Die Wirbelstromsensoren werden hierbei mit einer fortlaufenden Nummerierung von eins bis neun versehen, um eine eindeutige Zuordnung der Sensorkoordinaten und Messdaten zu ermöglich und in weiterer Folge die Berechnung und Zuordnung der Ergebnisse zu erleichtern (Abb. 33).

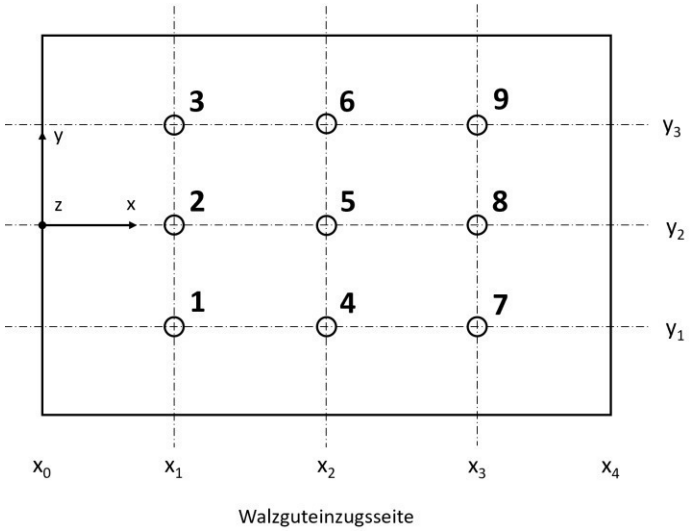


Abb. 33: Schematische Sensorpositionierung in der Draufsicht

Die Koordinate x_0 beschreibt hierbei die Anfangslänge der Walze (0mm) und x_4 auf die Ballenlänge der Walze (siehe Tab. 1).

6.1.2.3. Modul „sensordata“

Das Modul „sensordata“ liest die Messdaten aller Sensoren ein, welche in einem zuvor definierten Verzeichnis zu finden sind. Die Messdaten der Sensoren sind in einer csv-Datei gespeichert, die eingelesen und intern in Python als Liste gespeichert werden. Dieser Datensatz wird im Code allgemein als „all_data“ bezeichnet und ist eine „List of Lists“, eine Liste, die Listen mit den Messdaten zu jedem Messzeitpunkt enthält.

Exemplarisch wird hier ein Datensatz mit zwei Messzeitpunkten (t_1, t_2) und den Messdaten zum jeweiligen Messzeitpunt (s_1-s_{16}) angeführt:

all_data = [[t_1, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}], [t_2, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}]]

Tabelle 3: Sensorenbezeichnung im Datensatz "all_data" (Konzept 1)

Position im Datensatz	Sensor	Abkürzung	Messwert
1	Wirbelstromsensor 1	s_1	Abstand zur Walzenoberfläche [mm]
2	Wirbelstromsensor 2	s_2	Abstand zur Walzenoberfläche [mm]
3	Wirbelstromsensor 3	s_3	Abstand zur Walzenoberfläche [mm]
4	Wirbelstromsensor 4	s_4	Abstand zur Walzenoberfläche [mm]
5	Wirbelstromsensor 5	s_5	Abstand zur Walzenoberfläche [mm]
6	Wirbelstromsensor 6	s_6	Abstand zur Walzenoberfläche [mm]
7	Wirbelstromsensor 7	s_7	Abstand zur Walzenoberfläche [mm]
8	Wirbelstromsensor 8	s_8	Abstand zur Walzenoberfläche [mm]
9	Wirbelstromsensor 9	s_9	Abstand zur Walzenoberfläche [mm]
10	Linearpotentiometer	s_{10}	Auffederung zwischen den Walzen [mm]
11	Kraftmessdose 1	s_{11}	Walzkraft linker Führungsholm [N]
12	Kraftmessdose 2	s_{12}	Walzkraft rechter Führungsholm [N]
13	Drehmomentsensor	s_{13}	Drehmoment der Walze [Nm]
14	Drehzahlsensor	s_{14}	Drehzahl der Walze [1/s]
15	Winkelsensor	s_{15}	Winkel des Handrades [°]
16	Drucksensor	$s1_6$	Druck der hydraulischen Vorspannung [MPa]

Wie bereits in Abschnitt 6.1.2.1. beschrieben, kann dieser Datensatz in diesem Format als Zusammenfassung in eine csv-Datei in den Projektordner exportiert werden.

6.1.2.4. Modul „zeitschritte“

Das Modul „zeitschritte“ filtert aus dem von Modul „sensordata“ bereitgestellten Datensatz „all_data“ alle Zeitpunkte, an denen eine Messung durchgeführt wurde und liefert eine Liste, die alle Messzeitpunkte enthält. Zusätzlich liefert das Modul jenen Zeitpunkt, an dem die maximale Durchbiegung der Arbeitswalze im gesamten Messungszeitraum vorliegt.

6.1.2.5. Modul „berechnung_kreis“

Das Modul „berechnung_kreis“ berechnet mit dem von Modul „sensordata“ bereitgestellten Datensatz „all_data“ und den Sensorkoordinaten aus dem Modul „sensorkoordianten“ die auf der Arbeitswalze liegenden Messpunkte. Die Berechnung wird jeweils in jeder der drei x-Ebenen x_1, x_2 und x_3 (Abb. 32) zum Zeitpunkt t durchgeführt. Aus Abb. 32 können für die Ebene x_i folgende geometrische Zusammenhänge hergestellt werden, um die auf der Walze liegenden Punkte zu beschreiben. Für den Messpunkt von Sensor 1 (rechts) in der Ebene x_i gilt (siehe Abb. 32):

$$x_{1,i} = x_i \tag{6.4}$$

$$y_{1,i} = (R + a - u_1) \cdot \sin \alpha_s \tag{6.5}$$

$$z_{1,i} = -(R + a - u_1) \cdot \cos \alpha_s \tag{6.6}$$

Für den Messpunkt von Sensor 2 (mittig) in der Ebene x_i gilt (Abb. 32):

$$x_{2,i} = x_i \tag{6.7}$$

$$y_{2,i} = 0 \tag{6.8}$$

$$z_{2,i} = -(R + a) + u_2 \tag{6.9}$$

Für den Messpunkt von Sensor 3 (links) in der Ebene x_i gilt (Abb. 32):

$$x_{3,i} = x_i \tag{6.10}$$

$$y_{3,i} = -(R + a - u_3) \cdot \sin \alpha_s \tag{6.11}$$

$$z_{3,i} = -(R + a - u_3) \cdot \cos \alpha_s \tag{6.12}$$

Durch diese drei Punkte kann nun die Lage des Walzenquerschnittes im Raum ermittelt werden (siehe Anhang A). Wird dies für alle drei x-Ebenen x_1, x_2 und x_3 zum Zeitpunkt t durchgeführt, so können die drei Mittelpunkte und Radien des Walzenquerschnittes (Kreis) ermittelt werden. Durch die drei Mittelpunkte des Walzenquerschnittes kann die Durchbiegung bzw. die Biegelinie mittels einer Polynomapproximation ermittelt werden.

Dieses Modul liefert den Datensatz „data_tmr“, wobei „tmr“ für Zeit, Mittelpunkt und Radius steht, für alle Zeitpunkte t (exemplarisch für die Zeitpunkte t_1 und t_2) des Walzprozesses im Format:

data_tmr = $[[t_1, x_{1,1}, y_{1,1}, z_{1,1}, r_1], [t_1, x_{1,2}, y_{1,2}, z_{1,2}, r_2], [t_1, x_{1,3}, y_{1,3}, z_{1,3}, r_3], [t_2, x_{1,1}, y_{1,1}, z_{1,1}, r_1], [t_2, x_{1,2}, y_{1,2}, z_{1,2}, r_2], [t_2, x_{1,3}, y_{1,3}, z_{1,3}, r_3]]$

6.1.2.6. Modul „berechung“

Das Modul „berechnung“ führt eine Vorberechnung nach Gl. 6.3 durch, um die maximal auftretende Durchbiegung der Arbeitswalze abschätzen zu können. Zusätzlich werden ausgewählte Größen des Walzprozesses wie die gedrückte Länge l_d (nach Gl. 3.9), der Greifwinkel α (nach Gl. 3.14), die maximale Dickenabnahme des Walzgutes Δh_{max} (nach Gl. 3.15), sowie das Gerüstmodul C (nach Gl. 3.16) berechnet.

6.1.2.7. Modul „diagramme“

Das Modul „diagramme“ liefert auf Basis der Messdaten und deren Auswertung verschiedene Diagramme. Es werden Diagramme auf Basis der unverarbeiteten Messdaten, wie z.B. der Walzenkräfte, des Walzendrehmomentes, der Walzendrehzahl, des Handradwinkels und des Pumpendruckes ausgegeben. Durch eine Auswertung der Messdaten können Diagramme der Walzenradien über den gesamten Walzprozess, die Gerüstkennlinie des Walzwerkes, sowie die Geometrie des Walzspaltes visualisiert werden.

6.1.2.8. Modul „visualisierung_walze“

Das Modul „visualisierung_walze“ ermöglicht es die 3-D-Biegeline der unteren Arbeitswalze, die Kreis- bzw. Sensorebenen (aus Abschnitt 6.1.2.5.), die untere Arbeitswalze separat und beide Arbeitswalzen simultan zu visualisieren.

Die Visualisierung der 3-D-Biegelinie erfolgt über eine Polynomapproximation durch die aus Abschnitt 6.1.2.5. errechneten Mittelpunkte der Walzenquerschnitte aus dem Datensatz „data_tmr“. Diese Polynomapproximation geschieht hierbei in der x-y- und x-z- Ebene – dies liefert zwei zweidimensionale

Polynome. Diese Polynome beziehen ihre y- und z-Werte jeweils auf dieselben x-Koordinaten, was es ermöglicht diese als dreidimensionale Biegelinie zu visualisieren. Außerdem wurde ein sogenannter Verstärkungsfaktor eingeführt, welcher frei wählbar ist. Dies ermöglicht bei den erwartungsgemäß sehr kleinen Durchbiegungen eine bessere Darstellung.

Die Visualisierung der Kreis- bzw. Sensorebenen wird durch die aus Abschnitt 6.1.2.5 gewonnen Daten über die Mittelpunkte und Radien der Walzenquerschnitte aus dem Datensatz „data_tmr“ errechnet. Zur Visualisierung der unteren Arbeitswalze werden die Funktionalitäten der 3-D-Biegelinie und Kreis- bzw. Sensorebenen kombiniert. Im ersten Schritt wird aus dem Datensatz „data_tmr“ aus den Mittelpunkten der Walzenquerschnitte die 3-D-Biegelinie der unteren Arbeitswalze ermittelt. Im zweiten Schritt werden wie zuvor beschrieben die Kreis- bzw. Sensorebenen ermittelt. Im dritten Schritt wird aus den Daten über Mittelpunkte und Radien, welche jeweils einen eindimensionalen Vektor darstellen, ein dreidimensionales Vektorfeld erzeugt, um die Oberfläche der unteren Arbeitswalze beschreiben zu können. Auch hier besteht die Möglichkeit die Darstellung der Durchbiegung durch einen Verstärkungsfaktor zu verbessern.

Zur Visualisierung beider Walzen wird derselbe, wie zuvor beschriebene Ablauf angewendet. Zusätzlich wird die obere Arbeitswalze unter der Annahme einer spiegelsymmetrischen Biegung unter Miteinbezug der Auffederung zwischen den beiden Arbeitswalzen aus dem Datensatz „all_data“ visualisiert. Hierfür wird der grundlegende Ablauf der Visualisierung der unteren Arbeitswalze beibehalten, jedoch einige geometrisch bedingte Änderungen an den Daten vorgenommen (siehe Anhang A).

6.1.2.9. Modul „walzspaltzustellung“

Das Modul „walzspaltzustellung“ ermittelt aus dem Datensatz „all_data“ die Walzspaltöffnung s_0 und die Auslaufdicke des Walzgutes bzw. die Walzspaltöffnung mit Auffederung h_1 und ermittelt daraus die notwendige Zustellung. Die notwendige Zustellung wird durch die Übersetzung des Walzwerkes umgerechnet und in einen Zustellwinkel überführt, welcher an den Servomotor weitergegeben wird, der sich um diesen Zustellwinkel dreht und damit den definierten Walzspalt einstellt.

6.1.3. Auswertung und Ergebnisse

Für die Auswertung wurden repräsentative csv-Dateien mit Messdaten angelegt, welche Messwerte der Sensoren darstellen. Diese wurden mit realen Grenzwerten, basierend auf Maschinen- und Sensorkennwerten, generiert, um real auftretende Messwerte bestmöglich abzubilden. Diese können den beigelegten Dateien entnommen werden.

In Abb. 34 – 42 sind die Diagramme ausgewählter Messwerte und Kenngrößen zu sehen.

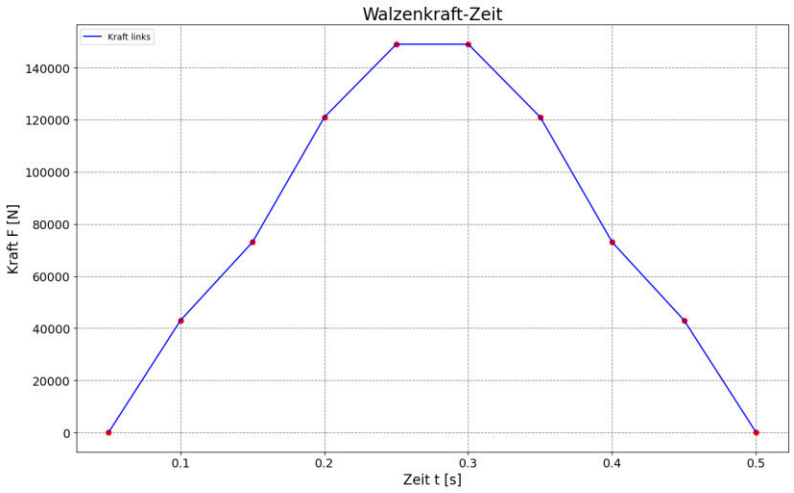


Abb. 34: Walzenkraft-Zeit-Diagramm der linken Kraftmessdose

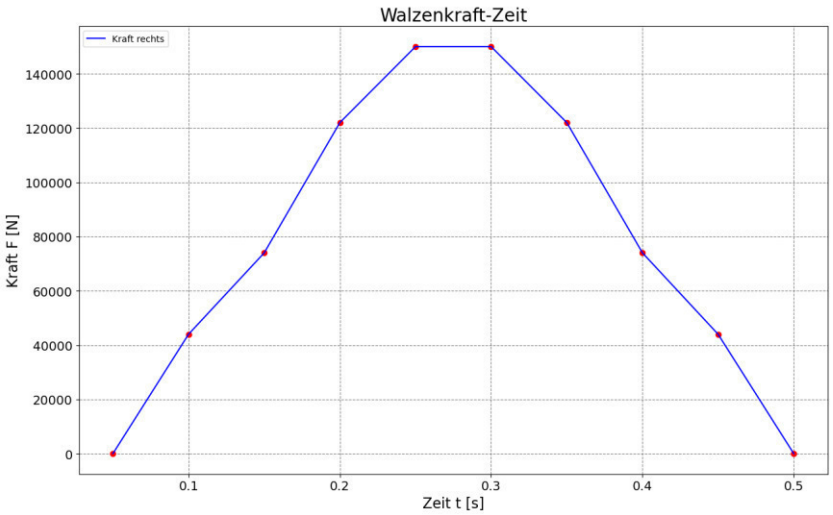


Abb. 35: Walzenkraft-Zeit-Diagramm der rechten Kraftmessdose

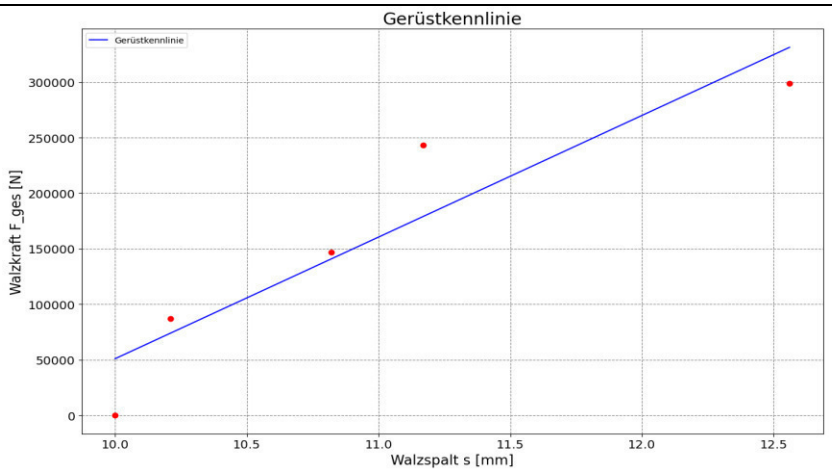


Abb. 36: Gerüstkennlinie des Walzwerkes

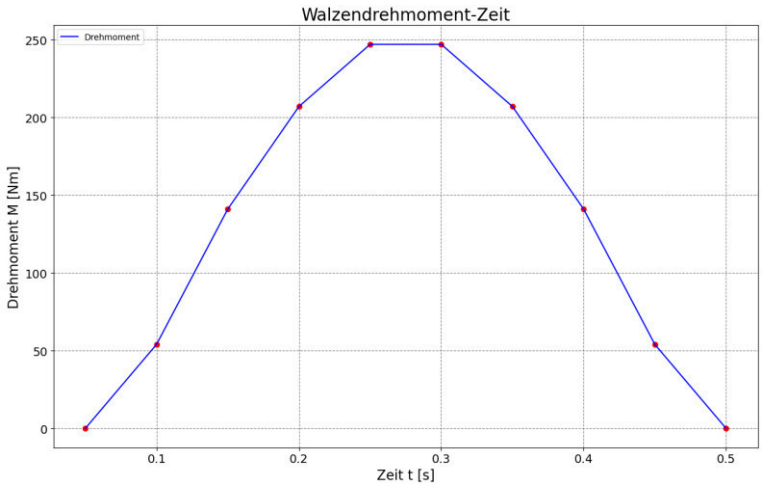


Abb. 37: Walzendrehmoment-Zeit-Diagramm

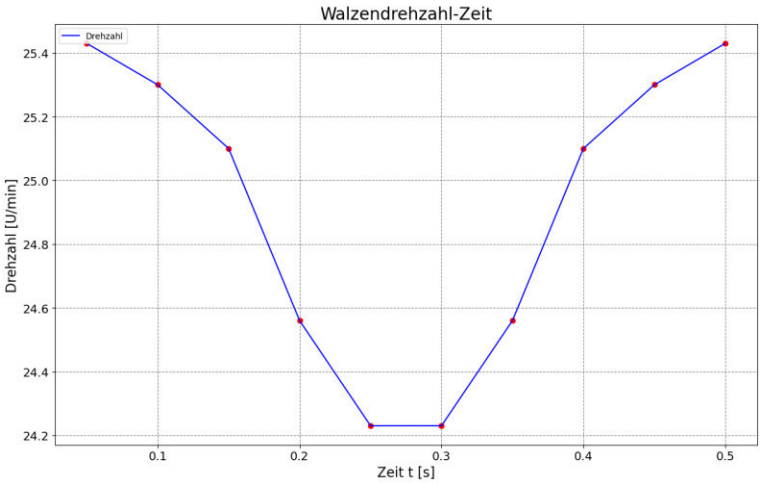


Abb. 38: Walzendrehzahl-Zeit-Diagramm

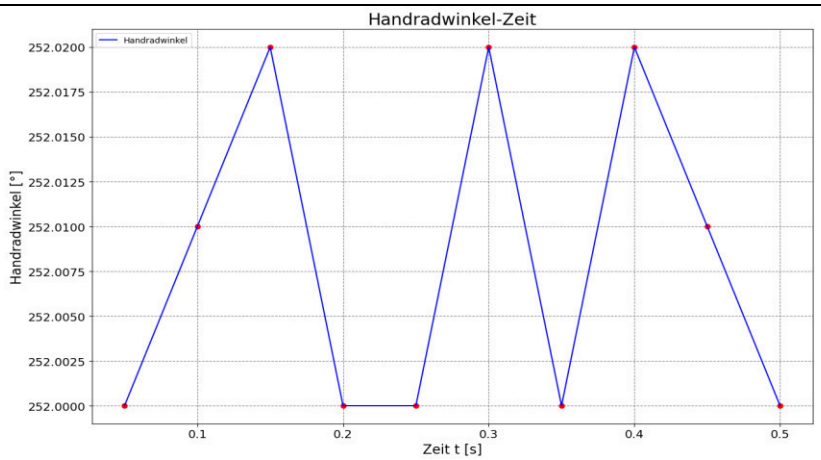


Abb. 39: Handradwinkel-Zeit-Diagramm

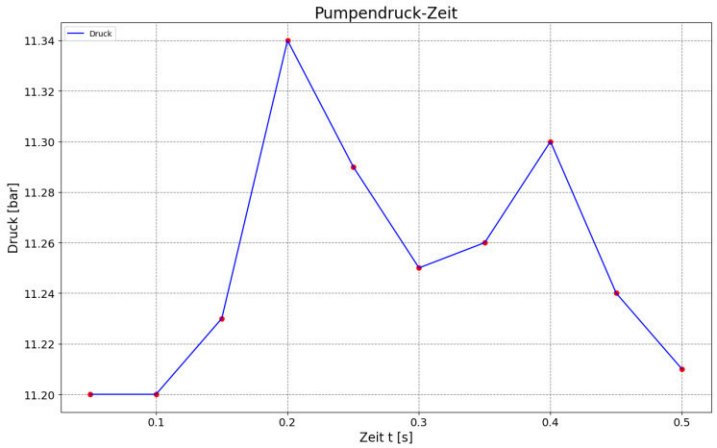


Abb. 40: Pumpendruck-Zeit-Diagramm

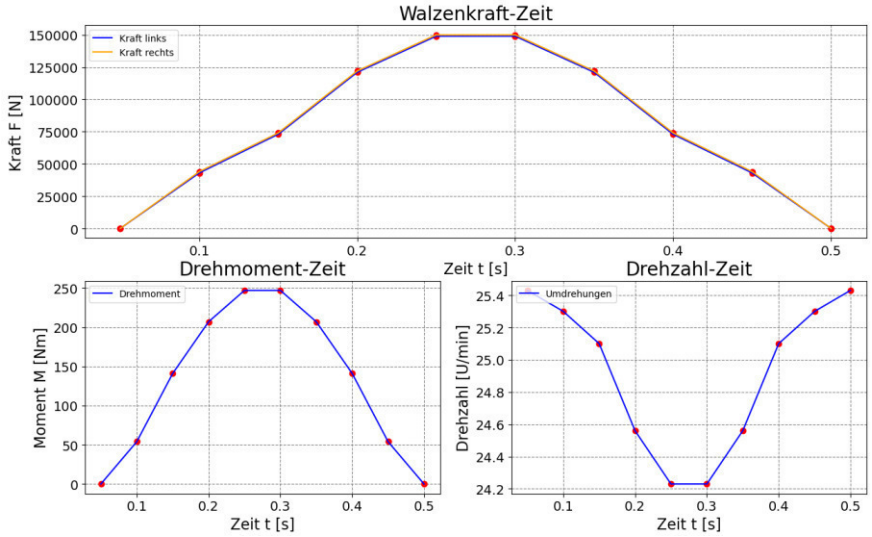


Abb. 41: Zusammenstellung von Kraft-Zeit-, Drehmoment-Zeit- und Drehzahl-Zeit-Diagramm

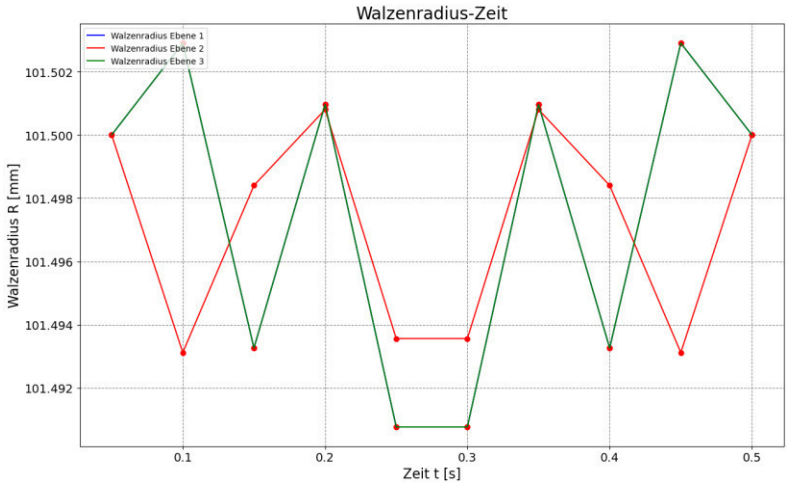


Abb. 42: Walzenradius-Zeit-Diagramm

In Abb. 43 ist die Visualisierung der Sensorebenen (Ebene 1 = blau, Ebene 2 = orange, Ebene 3 = grün) der Wirbelstromsensoren aus Abb. 32 zusehen. Darin werden die aus dem Datensatz „data_tmr“ berechneten Daten über die Lage bzw. Mittelpunkte und die Radien des Walzenquerschnittes in der jeweiligen Ebene x_i dargestellt. Durch diese Mittelpunkte wird durch eine Polynomapproximation die 3-D-Biegelinie (Abb. 45) ermittelt. Durch die Projektion der Biegelinie kann unter Berücksichtigung der Auffederung zwischen den beiden Arbeitswalzen auf die Walzspaltgeometrie geschlossen werden (Abb. 44). Aus der Walzspaltgeometrie lässt sich die maximale Durchbiegung ermitteln, welche zusätzlich in Kombination mit der Walzspaltgeometrie Auskunft über die Planheit des Walzgutes liefert. Entlang der ermittelten Biegelinie erfolgt die, wie in Abb. 46 und 48 zu sehen, Visualisierung der Arbeitswalze.

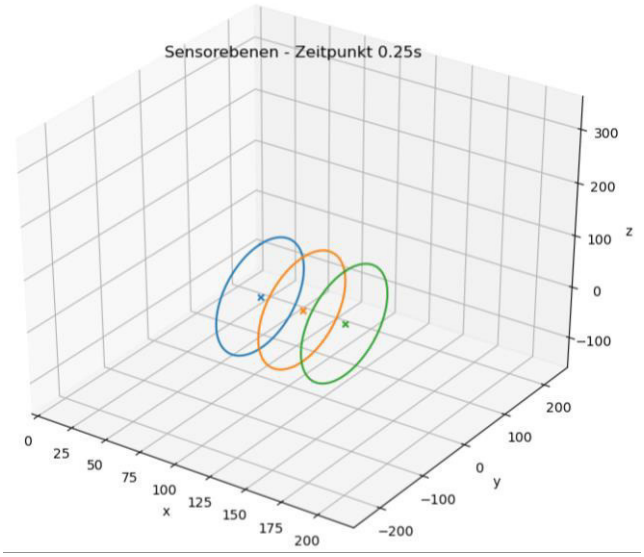


Abb. 43: Visualisierung der Wirbelstromsensorebenen

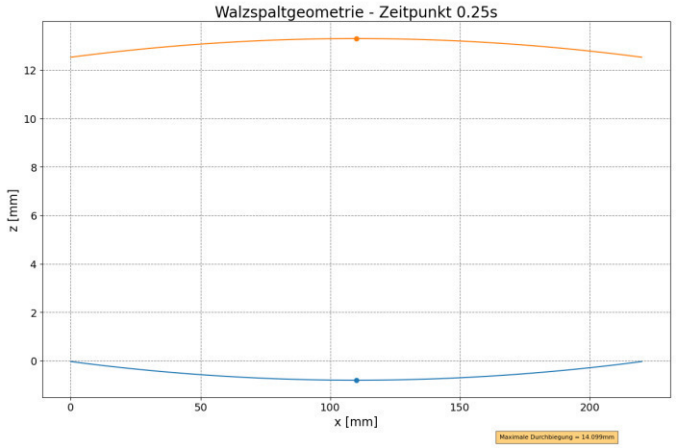


Abb. 44: Walzspaltgeometrie

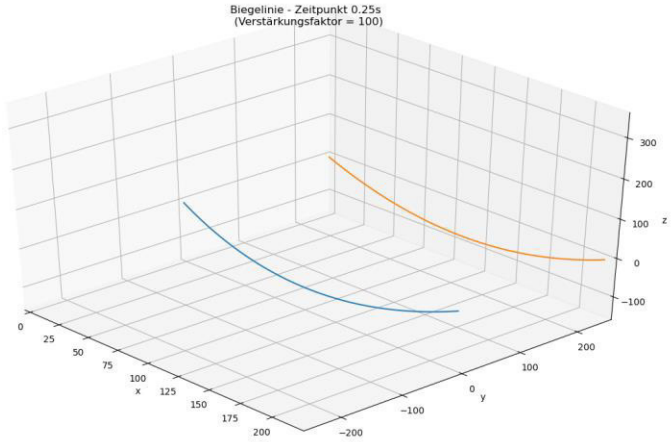


Abb. 45: 3-D-Biegelinie (blau) und 2-D-Biegelinie (orange)

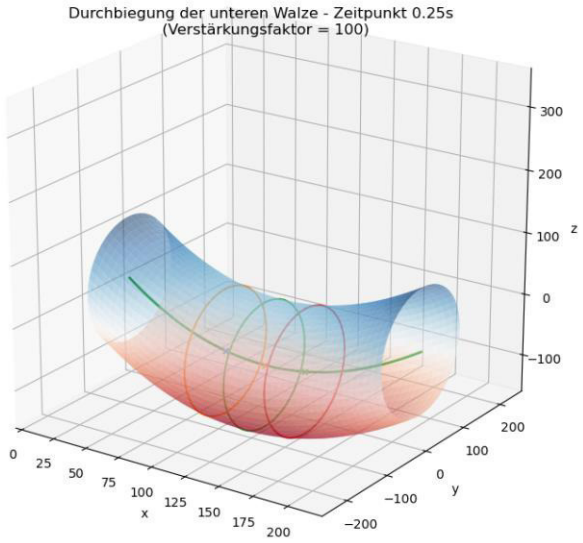


Abb. 46: Visualisierung der Durchbiegung der unteren Walze mit Verstärkungsfaktor

```
Projektordner '2020-07-08 [13-14-44] - Walzen 1.0038' wurde angelegt
Export aller .csv-Dateien in den Projektordner '2020-07-08 [13-14-44] - Walzen 1.0038' erfolgreich
Export der Sammlung aller .csv-Dateien in den Projektordner '2020-07-08 [13-14-44] - Walzen 1.0038' erfolgreich
t_max = 0.25 s, delta_z_max = 12.559999891892272 mm (= 12559.999891892272 µm)
Gedrückte Länge l_d = 14.213mm
Maximaler Greifwinkel alpha_0 = 7.971°
Maximale Dickenabnahme delta_h_max = 4.06mm
Gerüstmodul C = 42.777kN/mm
Auffederung 1/C = 0.023mm/kN
Notwendige Drehung der Zustellung: 1053.26° (entspricht 18.38 Umdrehungen)
```

Abb. 47: Ausgabe im Log

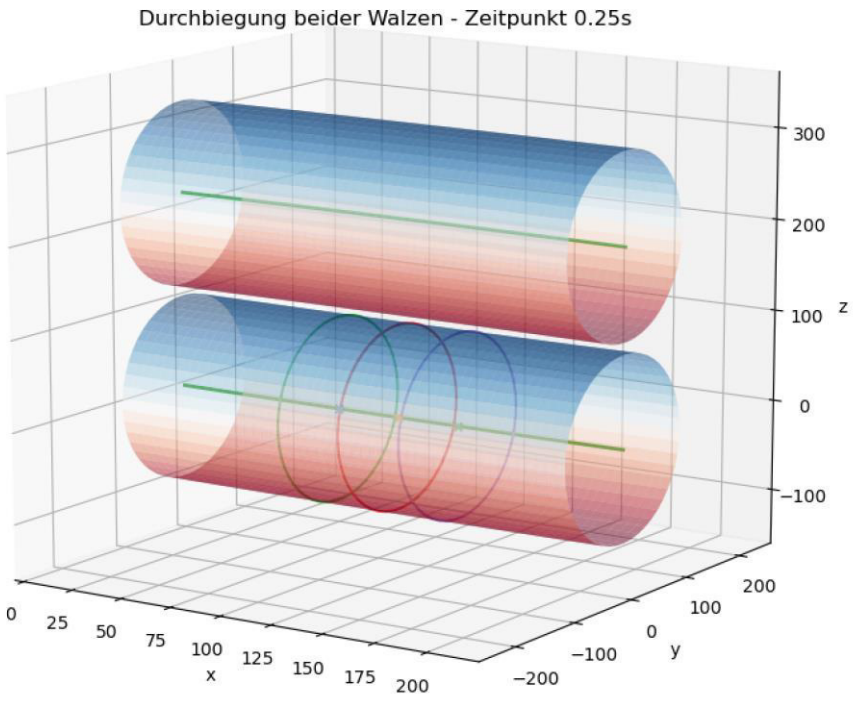


Abb. 48: Visualisierung der Durchbiegung der beiden Walzen

Das Konzept der Sensorik im Zusammenspiel mit der Programmierung durch Python liefert den „Proof of Concept“ und untermauert die Machbarkeit des Konzeptes. Unter den Gesichtspunkten der Wirtschaftlichkeit stehen die Kosten dem Nutzen gegenüber. Der Nutzen bestünde darin, die Daten für die Forschung und Lehre am Lehrstuhl zu sammeln, zu verarbeiten und zu visualisieren. Diese also für Mitarbeiter und Studenten zu veranschaulichen. Aufgrund der hohen Kosten der Sensorik ist dieses „Retrofitting“ in Bezug auf den dadurch gewonnen Nutzen jedoch nicht wirtschaftlich. Daher wurde im Zuge dieser Arbeit ein zweites Konzept (siehe Abschnitt 6.2.) erarbeitet.

6.2. Konzept 2

Das zweite Konzept wurde gegenüber dem ersten Konzept aus Abschnitt 6.1. aus wirtschaftlichen Gründen weiter modifiziert. Durch die Neudefinition der zu erfassenden, relevanten Messgrößen, der Walzspaltauffederung, der Zahnradwinkel zur Walzspaltzustellung und die Walzkräfte, wurde das Sensorkonzept (6.2.1.) sowie die Python-Programmierung zur Datenerfassung, -auswertung und Visualisierung (6.2.2. & 6.2.3.) modifiziert.

6.2.1. Sensorik

Die Sensorik umfasst insgesamt vier Sensoren die vier Messgrößen erfassen. Die Gesamtheit dieser Sensoren ermöglicht es Daten über die Größe und Auffederung des Walzspaltes zwischen den beiden Arbeitswalzen, die Walzkraft und den Winkel des Zahnrades zur Walzspaltzustellung zu sammeln.
Für die Messung der Größe des Walzspaltes und Auffederung zwischen den beiden Arbeitswalzen kommt aufgrund der hohen Auflösung ein Linearpotentiometer zum Einsatz. Die Montage des Sensors erfolgt zwischen den beiden Walzlagerdeckeln. Die maximale Höhe des Walzspaltes beträgt dabei ungefähr 35mm, weshalb das Linearpotentiometer eine Wegmessung von mindestens 35mm ermöglichen muss.
Die Walzkraft wird mit zwei bereits vorhandenen DMS-Kraftmessdosen gemessen, welche sich über der oberen Arbeitswalze befinden. Die maximale Walzkraft des Walzwerkes beträgt 300kN. Der Winkel des Zahnrades zur Verstellung des Walzspaltes wird mit einem sogenannten Multiturn-Encoder gemessen. Dieser erfasst auch nach mehreren Umdrehungen den Winkel des Zahnrades. Die Montage des Multiturn-Encoders erfolgt direkt unter den Speichen des Handrads und oberhalb des Zahnrades, welcher durch ein Reibrad mit dem Zahnrad in Kontakt gebracht wird. Das Reibrad weist nicht denselben Durchmesser wie jene Kontaktstelle des Zahnrades auf, mit dem das Reibrad in Kontakt steht. Durch die Ungleichheit dieser Durchmesser wird der Winkel nicht 1:1 vom Handrad auf den Multiturn-Encoder übertragen, was eine Umrechnung der Messwerte bzw. zusätzlich Kalibrierung erfordert (Abb. 49).

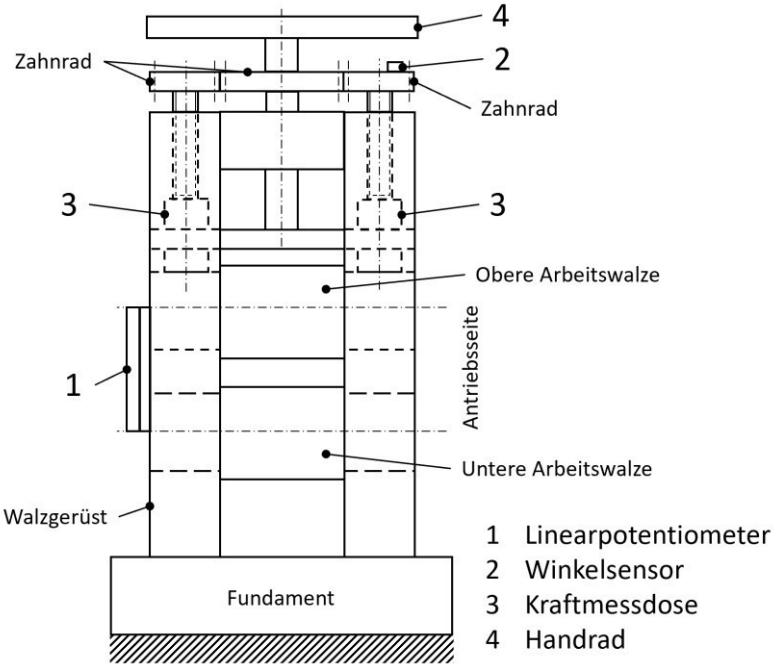


Abb. 49: Walzwerk mit Sensoren (Konzept 2)

Insgesamt sind vier Sensoren in das System eingebunden. Tabelle 4 bietet eine Übersicht über Art und Anzahl der gewählten Sensoren.

Tabelle 4: Sensortypen und Anzahl (Konzept 2)

Position in Abb. 49	Sensor	Anzahl	Messwert
1	Linearpotentiometer	1	Auffederung zwischen den Walzen [mm]
2	Winkelsensor	1	Winkel des Zahnrades [°]
3	Kraftmessdose	2	Walzkraft am Führungsholm [N]

Ein A/D-Wandler wandelt die analogen Signale der Sensoren in digitale Signale um, um diese für die computerunterstütze Weiterverarbeitung durch die Programmierung (6.2.2.) nutzbar zu machen. Für die leichtere Zugänglichkeit und Visualisierung für den Maschinenbediener wird ein Tablet verwendet.

6.2.2. Programmierung

Die Programmierung ähnelt jener von Konzept 1 (6.1.), jedoch musste aufgrund der Veränderung in der Sensorik auf einige Funktionalitäten aus dem ersten Konzept verzichtet werden. Für die Auswertung der Messdaten und zu deren Visualisierung wurde ein Python-Skript verwendet. Dabei greift das Skript auf

selbstgeschriebene Module zu, welche diese Funktionen enthalten und dann ausführen. Abb. 50 zeigt ein Flussdiagramm der verschiedenen Module, die im Zuge des ausführenden Skripts verwendet werden. Der Quellcode kann Anhang B entnommen werden.

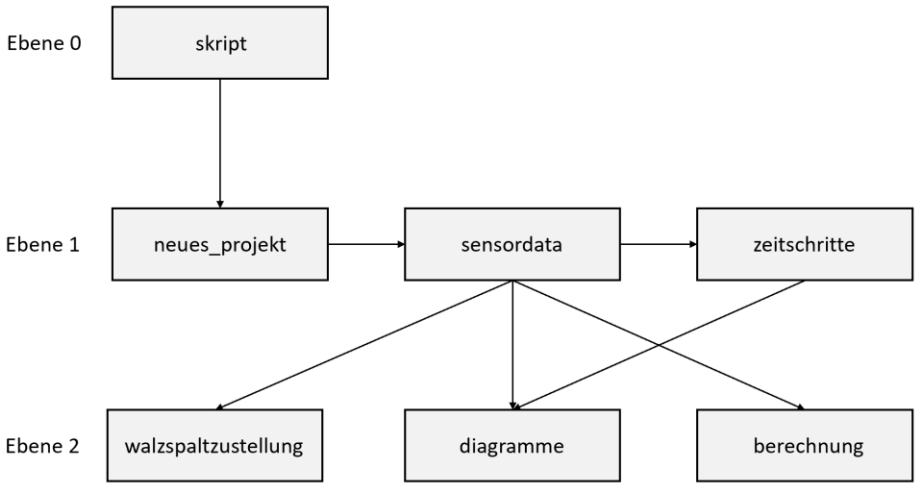


Abb. 50: Flussdiagramm der Module (Konzept 2)

Ebene 0 beschriebt jene Gruppe, die die Ausführung von Befehlen initialisieren („skript“). Ebene 1 beschreibt jene Gruppe von Modulen, die für Datensammlung und -filtrierung zuständig sind. Ebene 2 beschreibt jene Gruppe von Modulen, die für die Auswertung und Visualisierung der Daten, welche durch Module der Ebene 1 bereitgestellt werden, zuständig sind.

6.2.2.1. Modul „neues_projekt“

Das Modul „neues_projekt“ generiert, wie in Abschnitt 6.1.2.1, einen neuen Ordner auf einem vordefinierten Pfad mit einem vom Anwender frei wählbaren Projektnamen. Die Benennung des Projektordners erfolgt ebenfalls im standardisieren Format „Jahr-Monat-Tag [Stunde-Minute-Sekunde] – Projektname“ um eine mehrfache Verwendung von Projektordnernamen zu verhindern und so auch Datenverlust vorzubeugen. Zur Generierung des Projektordnernamens wird auf die lokale Uhr des Rechners zugegriffen. Im Projektordner wird ein der Unterordner „csv-data“ generiert, welcher nach Wahl des Anwenders alle csv-Dateien mit den Messdaten der Sensoren und/oder eine Zusammenfassung aller Messdaten der Sensoren enthält.

6.2.2.2. Modul „sensordata“

Das Modul „sensordata“ liest die Messdaten aller Sensoren ein, welche in einem zuvor definierten Verzeichnis zu finden sind. Die Messdaten der Sensoren sind in einer csv-Datei gespeichert, die eingelesen und intern in Python als Liste gespeichert werden. Dieser Datensatz wird im Code allgemein als „all_data“ bezeichnet und ist eine „List of Lists“, eine Liste, die Listen mit den Messdaten zu jedem Messzeitpunkt enthält.

Exemplarisch wird hier ein Datensatz mit zwei Messzeitpunkten (t_1, t_2) und den Messdaten zum jeweiligen Messzeitpunt (s_1-s_{16}) angeführt (Tabelle 5).

all_data = [[t_1, s_1, s_2, s_3, s_4], [t_2, s_1, s_2, s_3, s_4]]

Tabelle 5: Sensorenbezeichnung im Datensatz „all_data“ (Konzept 2)

Position im Datensatz	Sensor	Abkürzung	Messwert
1	Linearpotentiometer	s_1	Auffederung zwischen den Walzen [mm]
2	Kraftmessdose 1	s_2	Walzkraft linker Führungsholm [N]
3	Kraftmessdose 2	s_3	Walzkraft rechter Führungsholm [N]
4	Winkelsensor	s_4	Winkel des Zahnrades [°]

Wie bereits in 6.2.2.1. beschrieben, kann dieser Datensatz in diesem Format als Zusammenfassung in eine csv-Datei in den Projektordner exportiert werden.

6.2.2.3. Modul „zeitschritte“

Das Modul „zeitschritte“ filtert aus dem von Modul „sensordata“ bereitgestellten Datensatz „all_data“ alle Zeitpunkte, zu denen eine Messung durchgeführt wurde und liefert eine Liste, die alle Messzeitpunkte enthält. Zusätzlich liefert das Modul jenen Zeitpunkt, an dem die maximale Durchbiegung der Arbeitswalze im gesamten Messungszeitraum vorliegt.

6.2.2.4. Modul „berechung“

Das Modul „berechnung“ führt, wie in 6.1, eine Vorberechnung nach Gl. 6.3 aus, um die maximal auftretende Durchbiegung der Arbeitswalze abschätzen zu können. Zusätzlich werden ausgewählte Größen des Walzprozesses wie die gedrückte Länge l_d (nach Gl. 3.9), der Greifwinkel α (nach Gl. 3.14), die maximale Dickenabnahme des Walzgutes Δh_{max} (nach Gl. 3.15), sowie das Gerüstmodul C (nach Gl. 3.16) berechnet.

6.2.2.5. Modul „diagramme“

Das Modul „diagramme“ liefert auf Basis der Messdaten und deren Auswertung verschiedene Diagramme. Es werden Diagramme auf Basis der unverarbeiteten Messdaten, wie z.B. der Walzenkräfte, der Auffederung zwischen den beiden Arbeitswalzen und des Zahnradwinkels zur Walzspaltzustellung ausgegeben. Durch die Auswertung der Messdaten kann die Gerüstkennlinie des Walzwerkes visualisiert werden.

6.2.2.6. Modul „walzspaltzustellung“

Das Modul „walzspaltzustellung“ ermittelt aus dem Datensatz „all_data“ die Walzspaltöffnung s_0 und die Auslaufdicke des Walzgutes bzw. die Walzspaltöffnung mit Auffederung h_1 und ermittelt daraus die notwendige Zustellung. Die notwendige Zustellung wird durch die Übersetzung zwischen dem Zahnrad zur Walzspaltzustellung und dem Handrad des Walzwerkes umgerechnet und in einen Zustellwinkel überführt, welcher durch das Handrad nachjustiert werden kann.

6.2.3. Auswertung und Ergebnisse

Für die Auswertung wurden repräsentative csv-Dateien mit Messdaten angelegt, welche Messwerte der Sensoren darstellen. Diese wurden mit realen Grenzwerten, basierend auf Maschinen- und Sensorkennwerten, generiert, um real auftretende Messwerte bestmöglich abzubilden.
In Abb. 51 – 55 sind die Diagramme ausgewählter Messwerte und Kenngrößen zu sehen.

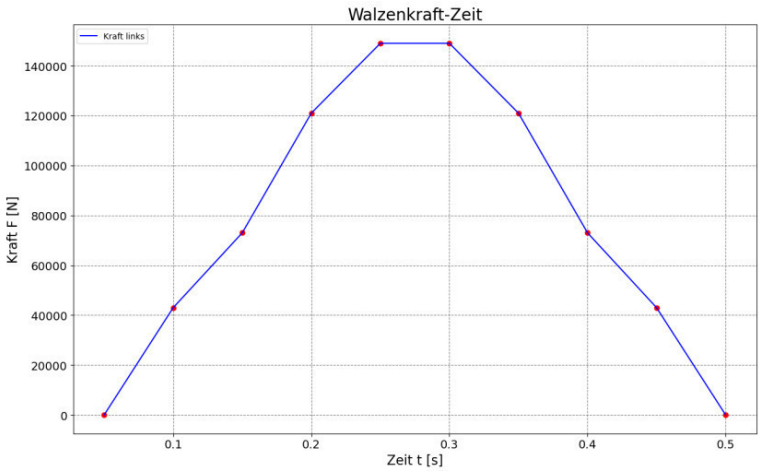


Abb. 51: Walzenkraft-Zeit-Diagramm der linken Kraftmessdose

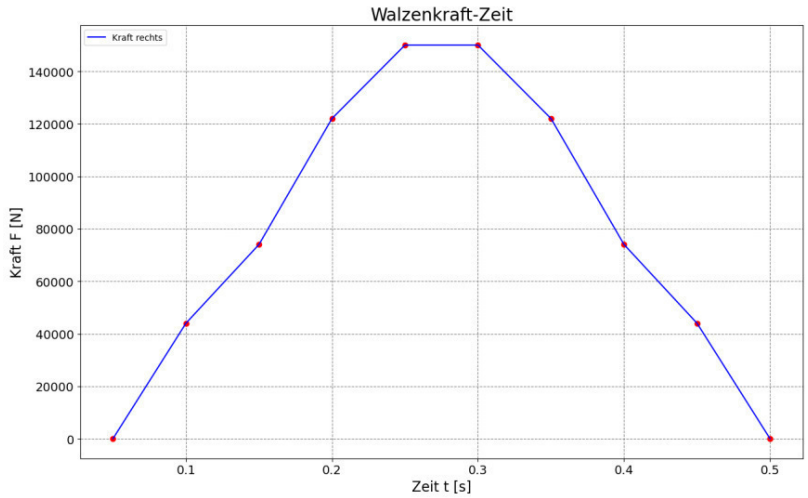


Abb. 52: Walzenkraft-Zeit-Diagramm der rechten Kraftmessdose

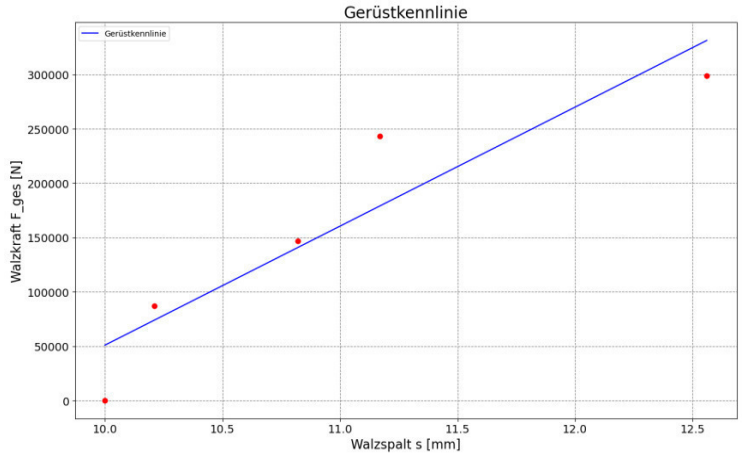


Abb. 53: Gerüstkennlinie des Walzwerkes

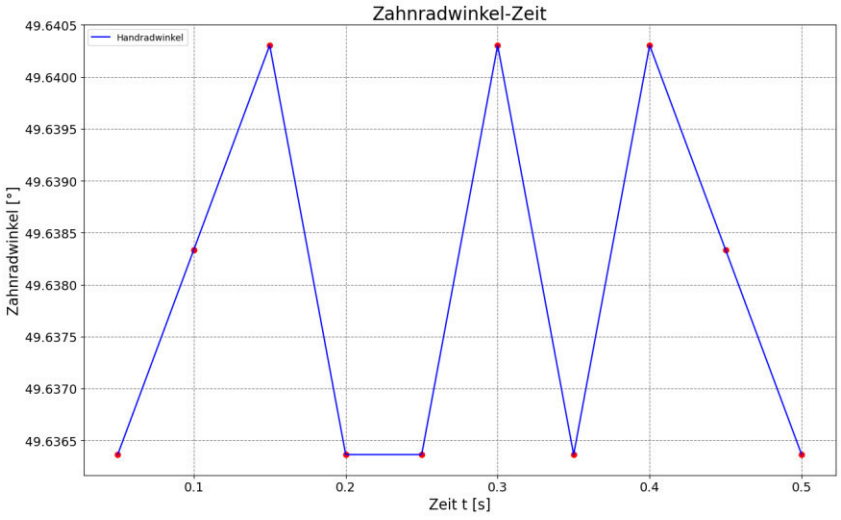


Abb. 54: Zahnradwinkel-Zeit-Diagramm

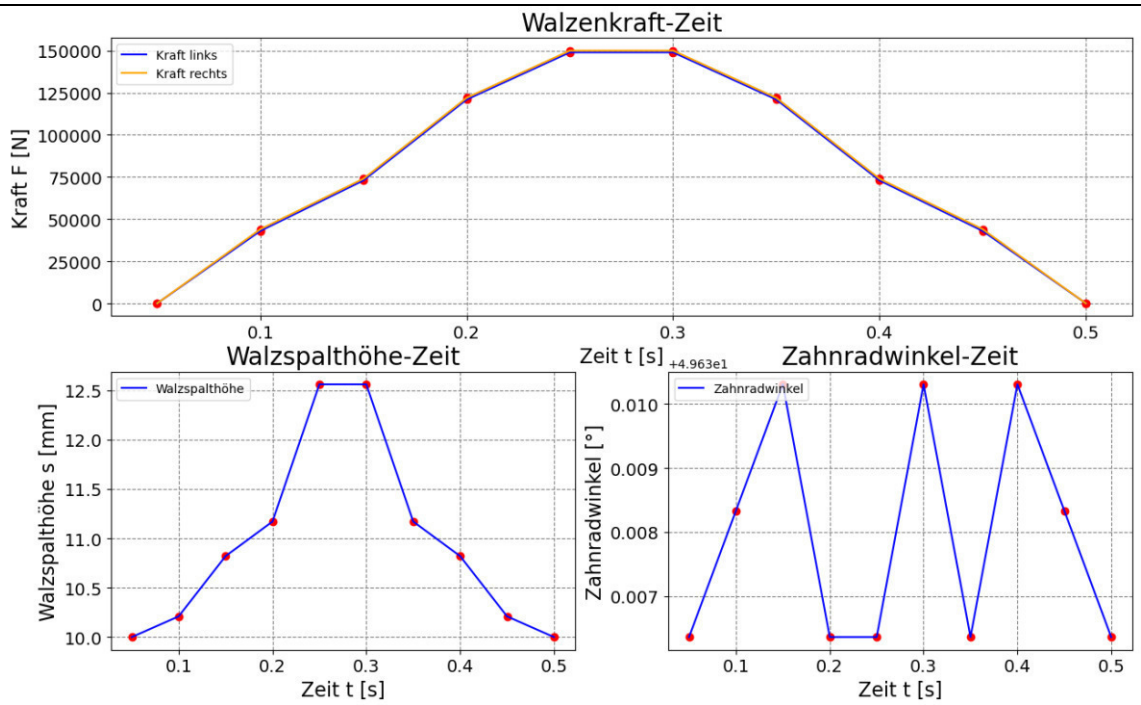


Abb. 55: Zusammenstellung von Kraft-Zeit-, Walzspalthöhe-Zeit- und Zahnradwinkel-Zeit-Diagramm

7. Implementierung

Im finalen Schritt, der Implementierung, wurde die Programmierung samt Sensoren dem in erläuterten 6.2. erarbeiteten Konzept nach am Walzwerk angebunden. Dafür wurde der Python-Code aus 6.2. (siehe Anhang B) geringfügig modifiziert, um ein fehlerfreies Laufverhalten im Betrieb zu erzielen (siehe Anhang C).

Aus Tab. 6 können die gewählten Sensoren (Datenblätter: siehe Anhang D) und Hardwarekomponenten (Datenblätter: siehe Anhang D) entnommen werden:

Tabelle 6: Gewählte Sensoren

Sensor	Bezeichnung	Anzahl	Messwert
Linearpotentiometer	Megatron RC13-75	1	Auffederung zwischen den Walzen [mm]
Winkelsensor	ASM posihall PH36-V01-31T-I1-CW-M12A5	1	Winkel des Zahnrades [°]
Kraftmessdose	-	2	Walzkraft am Führungsholm [N]

Tabelle 7: Gewählte Hardware

Hardware	Bezeichnung	Anzahl
mV-Transmitter	PR Electronics mV-Transmitter 2261	2
Analogausgang	Wago 4-Kanal-Analogeingang 750-453	1

Der mV-Transmitter dient zur Erfassung der sehr kleinen Spannungen der Kraftmessdosen. Der Analogausgang dient zur Erfassung der Messsignale der Sensoren und liefert diese an den Wago Controller, welcher zum Zeitpunkt der Implementierung bereits vorhanden war.

7.1. Linearpotentiometer

Das Linearpotentiometer Megatron RC-13-75 wurde auf dem Walzenlagerdeckel angebracht um die Walzspaltgröße und die Auffederung zwischen den Arbeitswalzen messen zu können (siehe Abb. 56).

Abb. 56: Montage des Linearpotentiometers

Tabelle 8. können die Anschlussparameter des Linearpotentiometer entnommen werden. Durch die Speisung mit 24 VDC entspricht der maximale Signalausgang von 20 mA nicht dem gesamten Messweg von 75 mm des Linearpotentiometers (siehe Datenblatt, Anhang D). Dennoch ist der Messweg beim maximalen Signalausgang von 20 mA größer als der maximale Walzspalt des Walzwerks, was die Messung der Walzspalthöhe möglich macht.

Tabelle 8: Anschlussparameter des Linearpotentiometers

Parameter	Parameterwert
Versorgungsspannung	24 VDC
Signalausgang	0 … 20 mA

Die Leiterbelegung des Linearpotentiometers kann Abb. 57 und dem Datenblatt in Anhang D entnommen werden. OUT+ und OUT- führen zum Wago Analogeingang 750-453, welcher an den Controller angebunden ist.

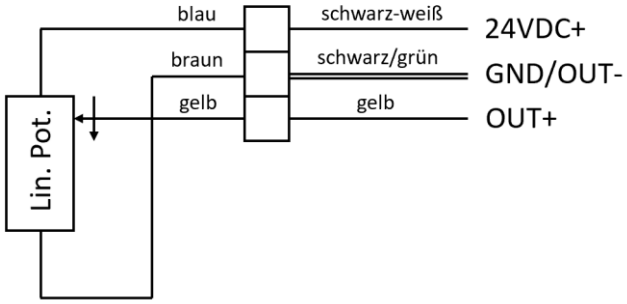


Abb. 57: Leiterbelegung des Linearpotentiometers

7.2. Winkelsensor

Der Winkelsensor ASM posihall PH36-V01-31T-I1- CW-M12A5 wurde unterhalb des Handrades angebracht. Er wurde durch ein Reibrad mit dem darunterliegenden Zahnrad verbunden (Abb. 58). Er dient zum Abgleich der Handradstellung bzw. Zahnradstellung mit dem Walzspalt.

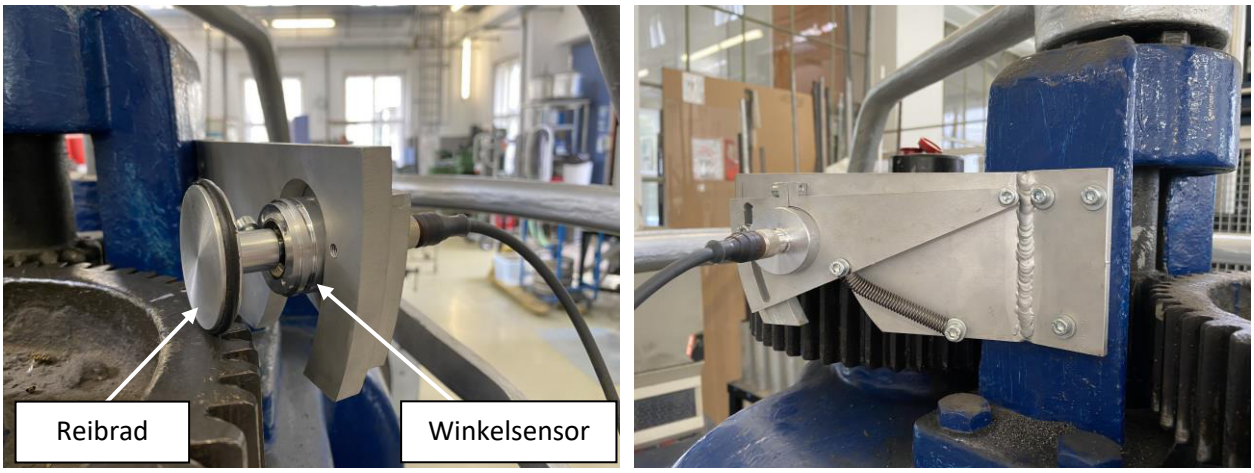


Abb. 58: Montage des Winkelsensors

Tabelle 9. können die Anschlussparameter des Winkelsensors entnommen werden.

Tabelle 9: Anschlussparameter des Winkelsensors

Parameter	Parameterwert
Versorgungsspannung	24 VDC
Signalausgang	4 … 20 mA

Die Leiterbelegung des Winkelsensors kann Abb. 59 und dem Datenblatt in Anhang D entnommen werden. OUT+ und OUT- führen zum Wago Analogeingang 750-453, welcher an den Controller angebunden ist.

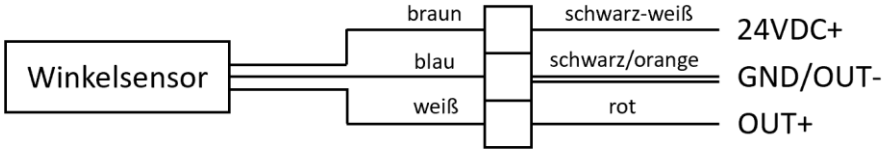


Abb. 59: Leiterbelegung des Winkelsensors

7.3. Kraftmessdosen

Die beiden vorhandenen Kraftmessdosen wurden ohne weitere Veränderungen beibehalten (siehe Abb. 60).

Abb. 60: Kraftmessdose (links) und die beiden mV-Transmitter der Kraftmessdosen (rechts)

Tabelle 10. können die Anschlussparameter des Winkelsensors entnommen werden.

Tab. 10: Anschlussparameter der Kraftmessdosen 1 und 2

Parameter	Parameterwert
Versorgungsspannung	5 VDC (durch PR Electronics mV-Transmitter 2261)
Signalausgang	0 … 20 mA

Durch das Ausmessen der Kraftmessdosen konnte ihr Aufbau rekonstruiert werden. Es handelt sich um vier Widerstände zu je 700Ω in Brückenschaltung (siehe Abb. 61). Es wurde, wie in Tabelle 10 zu sehen, eine Speispannung von 5VDC gewählt, welche durch den PR Electronics mV-Transmitter 2261 bereitgestellt wird, da die Kraftmessdosen bereits früher mit 5VDC gespeist wurden. Der Kennwert beträgt 2mV/V, was bedeutet das pro Volt Speisespannung mit einem Messausschlag von 2mV zu rechnen ist. Bei einer Speisespannung von 5VDC ist daher mit einem maximalen Messauschlag von 10mV zu rechnen.

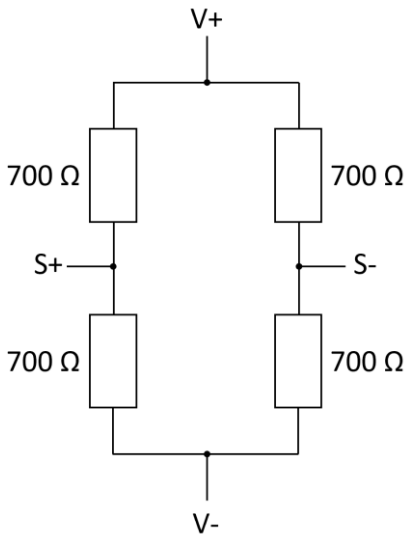


Abb. 61: Schematischer Aufbau der DMS-Kraftmessdosen

Beide Kraftmessdosen besitzen denselben Aufbau, jedoch ist auf die Ungleichheit der Leiterfarben zu achten (Abb. 62 & 63). Die Leiter SENSE+ und SENSE- besitzen hier keine Funktion.

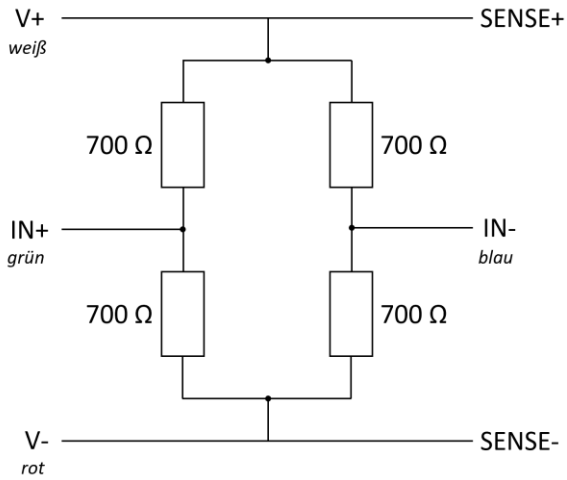


Abb. 62: Aufbau und Leiterbelegung der DMS-Kraftmessdose 1

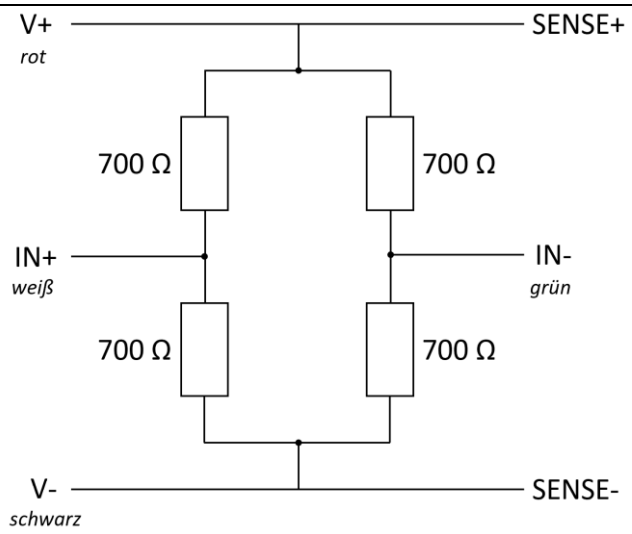


Abb. 63: Aufbau und Leiterbelegung der DMS-Kraftmessdose 2

Die Speisung einer Kraftmessdose wird durch jeweils einen PR Electronics mV-Transmitter 2261, auch Wägezellenverstärker genannt, gewährleistet. Die Belegung der Anschlüsse zur Versorgung des mV-Transmitters und der Kraftmessdose, sowie zum Signaleingang- und -ausgang des mV-Transmitters können Abb. 64 und Abb. 65 entnommen werden.

Für Kraftmessdose 1 und 2 gilt die gleiche Leiterbelegung am PR Electronics mV-Transmitter 2261, es bestehen jedoch Unterschiede in der Farbe der Leiter der Kraftmessdosen (siehe Abb. 64 und 65): Anschluss 7 und 11 liefern die Speisespannung von 5 VDC für die Kraftmessdose. Über Anschluss 9 und 10 erfolgt die Speisung des PR Electronics mV-Transmitter 2261 mit 24 VDC und 1 A (Tabelle 11). Anschluss 5 und 6 nehmen das Signal der Kraftmessdose in mV auf, welches vom mV-Transmitter interpretiert und in ein proportionales Signal im Bereich von 0 … 20 mA liefert. Anschluss 1 und 3 liefern das Ausgangsignal im Bereich von 0 … 20 mA zum Wago 4-Kanal-Analogeingang 750-453, welcher das Signal an den Wago Controller weitergibt.

Tabelle 11: Anschlussparameter der beiden PR Electronics mV-Transmitter 2261

Parameter	Parameterwert
Versorgungsspannung	24 VDC
Versorgungsstrom	1 A

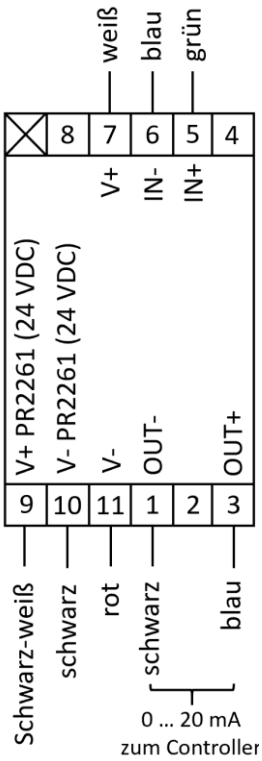


Abb. 64: Leiterbelegung des PR Electronics mV-Transmitters 2261 der DMS-Kraftmessdose 1

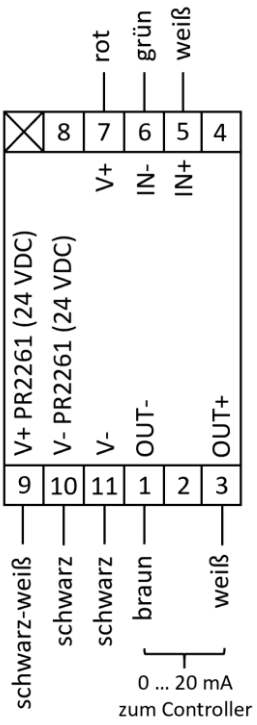


Abb. 65: Leiterbelegung des PR Electronics mV-Transmitters 2261 der DMS-Kraftmessdose 2

Aus Tab. 12 und 13 können die vorgenommenen Einstellungen an den PR Electronics mV-Transmitter 2261 entnommen werden. Eine Erklärung zur Funktion der einzelnen Kanäle und Einstellung können dem Datenblatt des PR Electronics mV-Transmitter 2261 in Anhang D entnommen werden. Der Wert, der vom mV-Transmitter angezeigt entspricht Prozent des Messbereichs. Im unbelasteten Zustand wird 000 und im vollbelasteten Zustand 100 am Digitaldisplay angezeigt.

Durch die Tarierung stellt der mV-Transmitter einen Signaleingang (Parameternummer 1.1) im unbelasteten Zustand fest. Durch ein minimales Rauschen dieses Signals kommt es zum Aufleuchten des Error-LEDs, da das Signal durch das Flackern immer wieder unter das minimale Eingangssignal fällt. Deshalb sollte hier der nächstgrößere Signaleingang gewählt werden.

Tabelle 12: Einstellung am PR Electronics mV-Transmitter 2261 für die Kraftmessdose 1

Parameternummer	Parameterbezeichnung	Parameterwert
0.0	-	Belastungsabhängig [% d. Messbereichs]
1.0	In (Eingang)	-
1.1	InL (Eingang 0%)	-3.0 [mV]
1.2	InH (Eingang 100%)	7.0 [mV]
1.3	In0 (Überbereich)	50.0 [%]
3.0	CAL (Kalibrierung)	-
3.1	CL0 (Kalibrierung niedrig %)	0.00
3.2	CH1 (Kalibrierung hoch %)	0.00
4.0	OUT (Analogausgang)	-
4.1	OL (Ausgang 0%)	0.00 [mA]
4.2	OH (Ausgang 100%)	20.0 [mA]
4.3	U1 (Strom oder Spannung)	002
4.4	REP (Ansprechzeit)	0.06 [s]
5.0	APP (Anwendungswahl)	-
5.1	TAR (Tarierung)	dtA
5.2	dIN (Digitaleingang Typ)	nPn
5.3	SUP (Umformerversorgung)	5.0 [V]
5.4	PAS (Programmzugang)	040
5.5	Frq (Frequenzunterdrückung)	50 [Hz]

Tabelle 13: Einstellung am PR Electronics mV-Transmitter 2261 für die Kraftmessdose 2

Parameternummer	Parameterbezeichnung	Parameterwert
0.0	-	Belastungsabhängig [% d. Messbereichs]
1.0	In (Eingang)	-
1.1	InL (Eingang 0%)	-3.2 [mV]
1.2	InH (Eingang 100%)	6.8 [mV]
1.3	In0 (Überbereich)	50.0 [%]
3.0	CAL (Kalibrierung)	-
3.1	CL0 (Kalibrierung niedrig %)	0.00
3.2	CH1 (Kalibrierung hoch %)	0.00
4.0	OUT (Analogausgang)	-
4.1	OL (Ausgang 0%)	0.00 [mA]
4.2	OH (Ausgang 100%)	20.0 [mA]
4.3	U1 (Strom oder Spannung)	002
4.4	REP (Ansprechzeit)	0.06 [s]
5.0	APP (Anwendungswahl)	-
5.1	TAR (Tarierung)	dtA
5.2	dIN (Digitaleingang Typ)	nPn
5.3	SUP (Umformerversorgung)	5.0 [V]
5.4	PAS (Programmzugang)	040
5.5	Frq (Frequenzunterdrückung)	50 [Hz]

Die Kalibrierung der Messdose wurde mit der „Servotest“, welche sich im Institut für Umformtechnik befindet, vorgenommen. Hierfür wurden die beiden Kraftmessdosen bis zu einer maximalen Prüfkraft von 150kN in 10kN-Schritten belastet und die ausgegeben Werte dokumentiert. Diese Werte wurden analysiert und im Wago eCockpit zur Kalibrierung verwendet, worauf hier aber nicht weiter eingegangen wird.

7.4. Datenerfassung

Die Anbindung der Sensorik an den Rechner wurde mittels geeigneter Hard- und Software der Firma Wago realisiert. Die Analogsignale der Sensoren werden vom Wago 4-Kanal-Analogeingang 750-453 aufgenommen (siehe Abb. 67 & 68) und an den Wago Controller weitergegeben. Der Controller, welcher als A/D-Wandler fungiert, wandelt die analogen Signale in digitale Werte um und gibt sie an das Netzwerk des Instituts weiter. Die Messwerte werden automatisiert als csv-Datei exportiert und mit Python ausgelesen, verarbeitet und im GUI visualisiert. Zusätzlich wurde ein GUI mi dem Wago eCockpit

entworfen über welches die Messung gestartet werden kann und die Messwerte der Sensoren in Echtzeit verfolgt werden können (Abb. 70).

Aus Abbildung 66 kann der schematische Aufbau aller implementierten Komponenten, deren Ein- und Ausgabeparameter und Verknüpfung zueinander, entnommen werden.

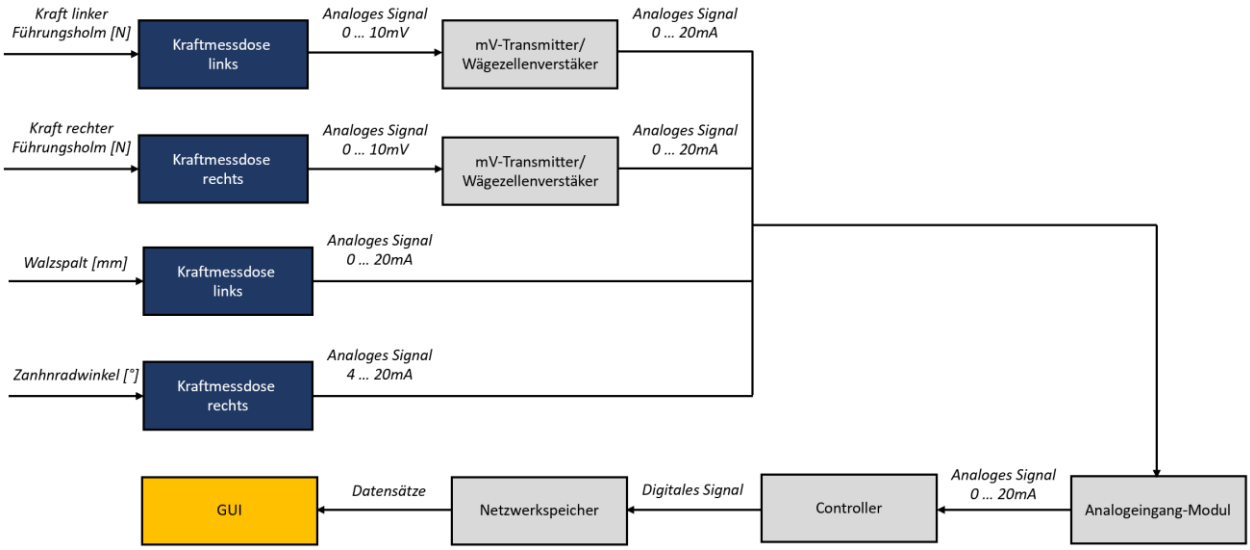


Abb. 66: Schematischer Aufbau des Messsystems

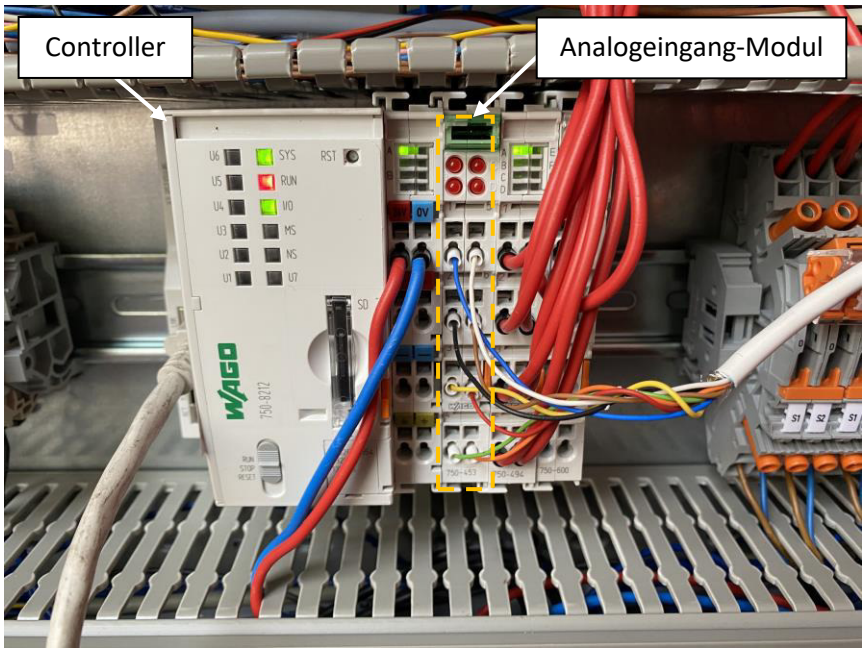


Abb. 67: Wago Controller und Wago 4-Kanal-Analogeingang 750-453

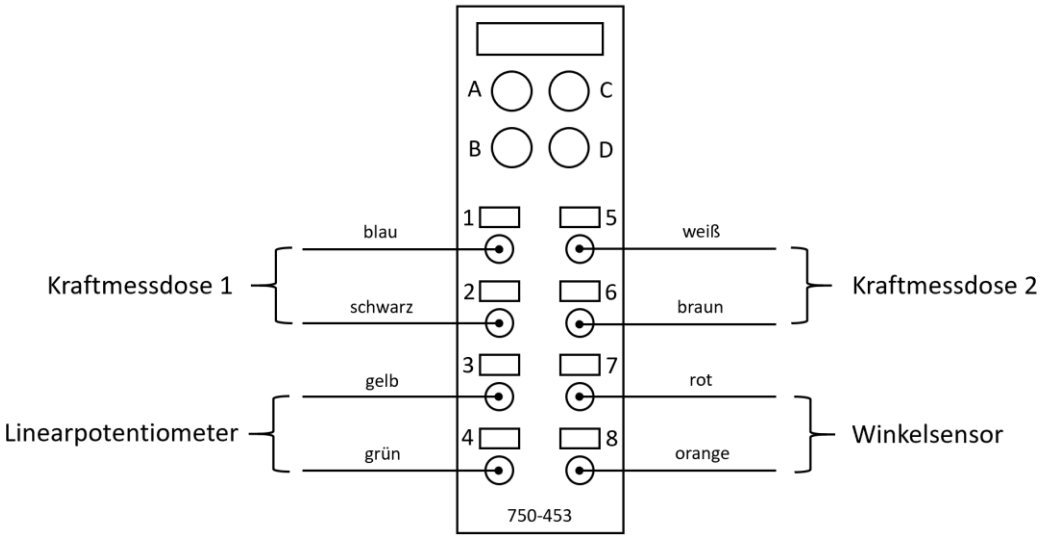


Abb. 68: Anschlussbelegung des Wago 4-Kanal-Analogeingang 750-453

Die Visualisierung für den Bediener erfolgt über ein Tablet über ein grafisches „User-Interface“ (GUI). Das GUI ermöglicht es, wie in Abschnitt 6.2. beschrieben, verschiedene Prozesskenngrößen und die Gerüstkennlinie für den Bediener zu visualisieren. Durch die Eingabe bestimmter Prozesskennwerte können ausgewählte Parameter des Walzprozesses im Voraus berechnet werden (Abb. 69).

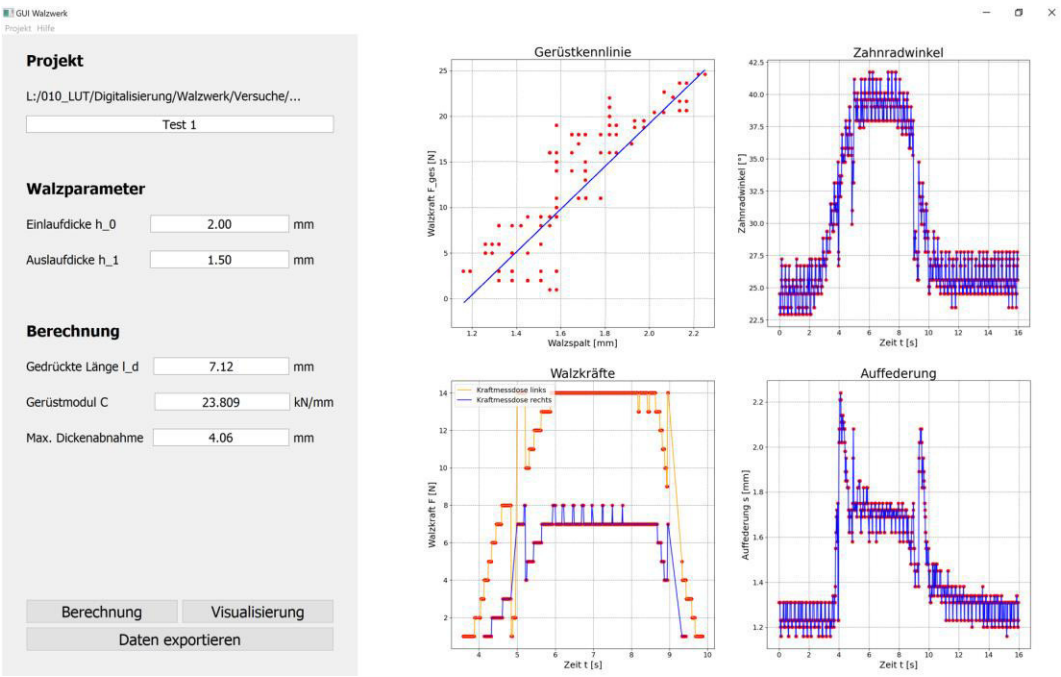


Abb. 69: Grafischer „User-Interface“ (entwickelt mit Python)

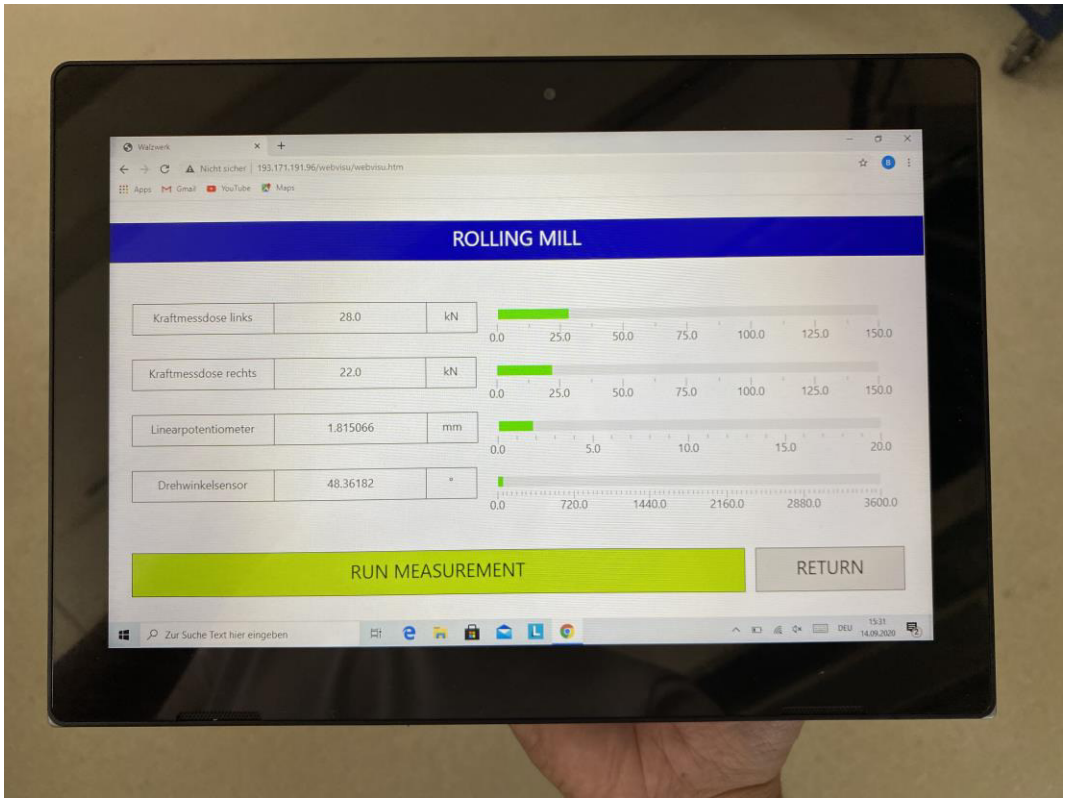


Abb. 70: Grafischer „User-Interface“ (entwickelt mit Wago eCockpit)

8. Auswertung und Ergebnisse

Durch die Implementierung eines geeigneten Sensorenkonzeptes in Kombination mit geeigneter Hard- und Software konnte das Walzwerk einem „Retrofitting“ unterzogen werden und ist nun auf dem Stand der Industrie 4.0. Es stellt nun ein CPPS mit einem DS auf der digitalen Seite des CPS dar.

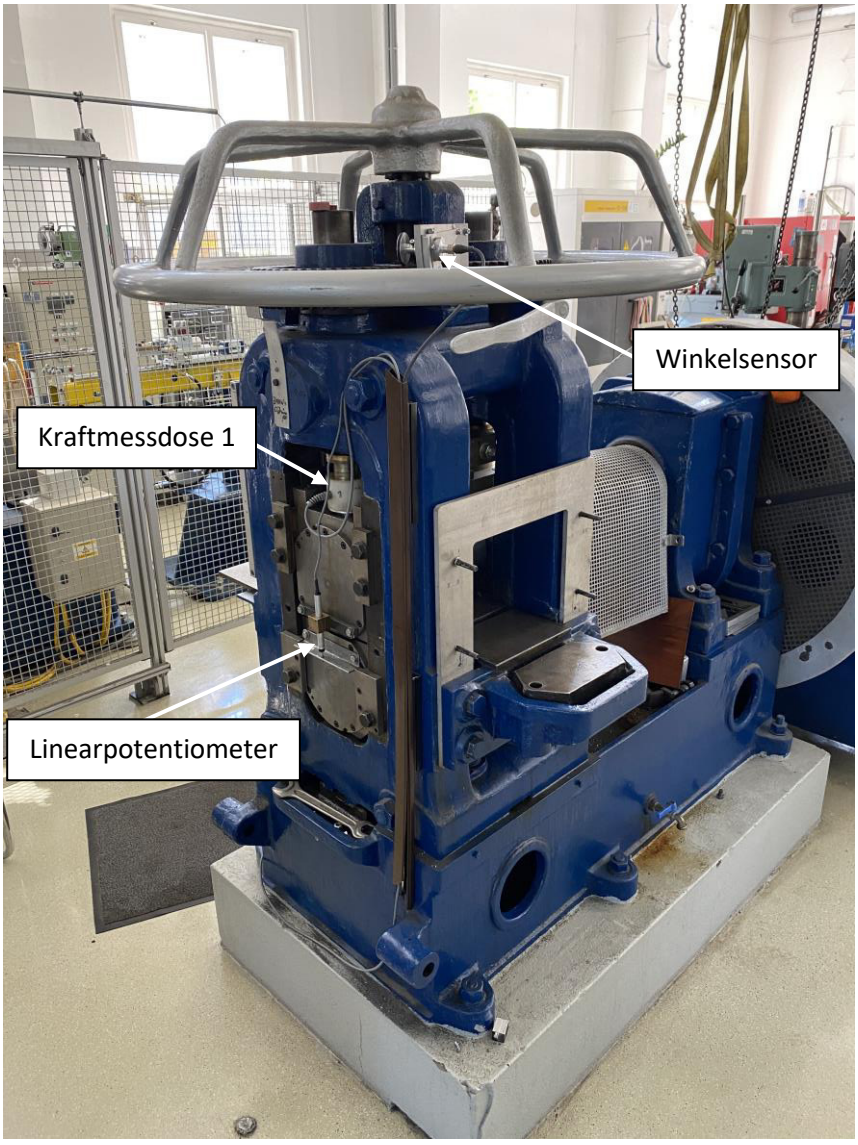


Abb. 71: Walzwerk mit implementierter Sensorik

Durch die Auswertung und Visualisierung auf einem Tablet kann dem Bediener Aufschluss über den Prozess, Prozesskenngrößen und deren Zusammenhang geboten werden, um im Betrieb eine schnelle Übersicht über diese zu ermöglichen und diese nach Bedarf auf Basis dieser Daten justieren zu können (siehe Abb. 69). Das GUI kann durch einen Klick auf die Verknüpfung am Desktop des Tablets gestartet werden.

9. Zusammenfassung und Ausblick

Durch die Modernisierung des Walzwerkes auf den Stand der Industrie 4.0 wurde gezeigt, dass durch die Kooperation geeigneter Sensorik mit „Open-Source“-Programmiersprachen sehr gute Ergebnisse erzielt werden können. Dies kann auf die Verbreitung und ständige Erweiterung bestehender, als auch neuer „Frameworks“ zurückgeführt werden. Aus diesem Grund könnte die Anwendung solcher „Open-Source“-Programmiersprachen auch in Zukunft in kleinen bis mittelständisches Unternehmen, im privaten als auch in akademischen Bereichen an Bedeutung gewinnen.

Ein wichtiger Aspekt einer solchen Modernisierung besteht darin im Vorhinein die zu erfassenden Mess- bzw. Systemgrößen und deren Größenordnung zu bestimmen und mit geeigneter Sensorik abzustimmen, um daraus die ungefähren Kosten ermitteln zu können. Diese Kosten sind dem erwarteten Nutzen gegenüberzustellen, um auf Basis dieser Überlegungen zur Wirtschaftlichkeit einer solchen Modernisierung durchführen zu können. Diese sind bei größeren Walzwerken bzw. Walzstraßen, bei denen beispielsweise die Planheit bzw. Formabweichung über die Walzspaltgeometrie kontrolliert und überwacht wird, einfacher zu rechtfertigen, da hier auch das Qualitätsmanagement (QM) ein große Rolle spielt, um in weiterer Folge Kosten für die Nachbearbeitung zu reduzieren. Bei kleineren Walzwerken, vor allem bei älteren, macht ein solches „Retrofitting“ zur Modernisierung auf den Stand der Industrie 4.0 nur bedingt Sinn. Hier stehen dem erwarteten Nutzen relativ hohe, abhängig von den zu erfassenden Mess- bzw. Systemgrößen, Kosten gegenüber.

Zukünftig werden vor allem neue Walzwerke und Walzstraßen kaum ohne eine Anbindung an eine digitale Repräsentanz auskommen, um wichtige Mess- bzw. Systemgrößen aufzeichnen, auswerten und analysieren zu können, um in weiterer Folge den Prozess flexibler und effektiver gestalten zu können. Diese These ist historisch gesehen belegbar, blickt man auf den relativ hohen Automatisierungsgrad von Walzstraßen in der metallverarbeitenden Industrie, den diese heutzutage aufweisen.

Abschließend bleibt zu sagen, dass wohl in nicht allzu ferner Zukunft die meisten Prozesse auch als „Cyber Physical System“ (CPS) existieren werden, nicht nur um die Prozesse besser verstehen und optimieren zu können, sondern um auch in einer schnelllebigen technischen Welt, die sich im stetigen Wandel befindet, konkurrenzfähig zu bleiben.

Weiterführend kann durch das „Retrofitting“ des Walzwerks der Gerüstmodul experimentell ermittelt werden, um zu beweisen ob dieser sich, wie in der Literatur angegeben, tatsächlich immer linear verhält. Weiters bietet sich im Zuge der Sensorkalibrierung eine Datenanalyse der Messwerte durch eine

Regressionsanalyse an, um die Messungen weiter zu verfeinern und so ihre Aussagekraft und -qualität zu erhöhen. Das GUI kann durch weitere Funktionen, wie eine Stichplan-KI erweitert werden, um im Sinne der Industrie 4.0 die Mensch-Maschine-Interaktion zu erhöhen.

Diese Arbeit hat somit gezeigt, dass auch mit geringem finanziellem Aufwand ältere Maschinen auf den Stand des heutigen, digitalisierten Produktionsumfeld zu bringen sind. Das in dieser Arbeit behandelte Projekt dient daher als Startschuss für weitere Digitalisierungsmaßnahmen am Lehrstuhl für Umformtechnik und trägt langfristig dazu bei, die Digitalisierung in der Umformtechnik weiter zu forcieren.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Doege, E., Behrens, B.A.: Handbuch Umformtechnik. Grundlagen, Technologien, Maschinen. 2. Aufl., Springer Verlag, Berlin/Heidelberg, 2010

[2] Dictionary of Production Engineering/Wörterbuch der Fertigungstechnik/Dictionnaire des Techniques de Production Mechanique Vol IV. 1. Auflage, Springer Verlag, Berlin Heidelberg, 2011

[3] Fritz A. H.: Fertigungstechnik. 12. Aufl., Springer Verlag, Berlin/Heidelberg, 2018

[4] Profos P., Pfeifer T.: Handbuch der industriellen Messtechnik. 6. Aufl., R. Oldenbourg Verlag, München/Wien, 1994

[5] Palkowski, H. (07.12.2017): Praktikum Metallurgie Master Wintersemester 2017/18. Versuch U1: Warmwalzen. Abgerufen 19.06.2020, von http://www2.imet.tu-clausthal.de/mp/main/lehre/praktika/tuc_imet_praktikum_warmwalzen.pdf

[6] Schwenzfeier, W., Herzog, A., Hohenwarter, J.: Walzwerktechnik. Ein Leitfaden für Studium und Praxis. 1. Aufl., Springer Verlag, Wien, 1979

[7] Bauer, H.G., Schadt, W.: Walzen von Flachprodukten. 1. Aufl., Springer Verlag, Berlin/Heidelberg, 2017

[8] Buchmayr, B.: Werkstoff- und Produktionstechnik mit Mathcad. Modellierung und Simulation in Anwendungsbeispielen. 1. Aufl., Springer Verlag, Berlin/Heidelberg, 2002

[9] Hoffman, H., Reimund, N., Spur, G.: Handbuch Umformen. Edition: Handbuch der Fertigungstechnik. 2.Aufl., Carl Hanser Verlag, München, 2012

[10] Heinrich, B., Linke, P., Glöckler, M.: Grundlagen Automatisierung. Sensorik, Regelung, Steuerung. 2. Aufl., Springer Vieweg Verlag, Wiesbaden, 2017

[11] Tieste K. D., Romberg O.: Keine Panik vor Regelungstechnik!. Erfolg und Spaß im Mystery-Fach des Ingenierstudiums. 3. Aufl., Springer Vieweg Verlag, Wiesbaden, 2015

[12] eddylab GmbH: Wirbelstromsensoren. In: https://www.wirbelstromsensor.de/ (2020), URL: https://www.wirbelstromsensor.de/ (zuletzt abgerufen am 19.06.2020)

[13] Micro-Epsilon Messtechnik GmbH & Co. KG: Wirbelstrom. In: https://www.micro-epsilon.de/ (2020), URL: https://www.micro-epsilon.de/service/glossar/Wirbelstrom.html (zuletzt abgerufen am 19.06.2020)

[14] Micro-Epsilon Messtechnik GmbH & Co. KG: Wirbelstrom. In: https://www.micro-epsilon.de/ (2020), URL: https://www.micro-epsilon.de/service/glossar/Linearitaet.html (zuletzt abgerufen am 19.06.2020)

[15] Keyence Deutschland GmbH: Wichtige Begriffe zur Auswahl von Messsystemen. In: https://www.keyence.de/ (2020), URL: https://www.keyence.de/ss/products/measure/measurement_library/basic/term/ (zuletzt abgerufen am 19.06.2020)

[16] Hoffmann J.: Messen nichtelektrischer Größen. Grundlagen der Praxis. 1. Aufl., Springer Verlag, Berlin/Heidelberg, 1996

[17] National Instruments: Data Acquisition (DAQ). In: https://www.ni.com/de-at.html (2020), URL: https://www.ni.com/en-us/shop/data-acquisition.html (zuletzt abgerufen am 19.06.2020)

[18] Heuer H., Schulze M., Klein M. (2012): Abbildende Wirbelstromsensoren zur hochauflösenden berührungslosen Abbildung von elektrischen Eigenschaften schlecht leitender Objekte. Fraunhofer Institut für zerstörungsfreie Prüfverfahren FhG IZFP Dresden

[19] Kleckers T.: Die Qual der Wahl: Piezoelektrische oder DMS-basierte Kraftaufnehmer?. In: https://www.hbm.com, URL:https://www.hbm.com/de/3719/piezoelektrische-oder-dms-basierte-kraftaufnehmer/ https://www.hbm.com/fileadmin/mediapool/files/technical-articles-technotes-white-papers/Piezoelektrische_oder_DMS-basierte_Kraftaufnehmer.pdf, (zuletzt abgerufen am 20.06.2020)

[20] https://wiki.polymerservice-merseburg.de/index.php/Elektro-Mechanischer_Kraftaufnehmer, Elektro-Mechanischer Kraftaufnehmer (Kraftmessdose) (zuletzt abgerufen am 20.06.2020)

[21] Andreeva E. (2005): Fertigung und Erprobung eines Mikro-Wirbelstromsensors zur Abstandsmessung. Fakultät Maschinenbau, Universität Hannover

[22] Balluff GmbH: Auflösung, Linearitätsabweichung, Genauigkeit. In: https://www.balluff.com (2020), URL: https://www.balluff.com/local/de/service/basics-of-automation/fundamentals-of-automation/resolution-non-linearity-accuracy/ (zuletzt abgerufen am 06.07.2020)

[23] Lange K., Liewald M.: Umformtechnik Handbuch für Industrie und Wissenschaft. Band 2: Massivumformung. 2. Aufl., Springer Verlag, Berlin, 1988

[24] Von Seggern F.: Was ist ein Kraftsensor. In: https://blog.trafag.de (23.11.2018), URL: https://blog.trafag.de/torque/was-ist-ein-kraftsensor#induktive (zuletzt abgerufen am 13.07.2020)

[25] https://homepages.thm.de/~hg7394/sns/Kraft1/Induktive_Kraftaufnehmer.htm, Induktive Kraftaufnehmer (zuletzt abgerufen am 13.07.2020)

[26] Tränkler H. R., Reindl L. M.: Sensortechnik. Handbuch für Praxis und Wissenschaft. 2. Auflage, Springer Vieweg Verlag, Berlin/Heidelberg, 2014

[27] Wagner M. (2017): Strukturelle und Methodische Untersuchung von Kraftaufnehmer-Systemen. Fakultät Architektur, Bauingenieurwesen und Umweltwissenschaften, Technische Universität Braunschweig

[28] Wenn es rau zugeht. Der Konstrukteur 6/2018, In: https://www.asm-sensor.com, URL: https://www.asm-sensor.com/files/asmTheme/pdf/fachartikel_konstrukteur_062018_sensortechnik_posihall.pdf, (zuletzt abgerufen am 13.07.2020)

[29] Ralph B. J., Stockinger M.: Digitalization and Digital Transformation in Metal Forming: Key Technologies, Challenges and Current Development of Industry 4.0 Application. Proceedings of the XXXIX. Colloquium on Metal Forming. Lehrstuhl für Umformtechnik, Montanuniversität Leoben, 2020

[30] Luber S., Litzel N (05.12.2017): Was ist ein Cyber-physisches System (CPS)?. In: https://www.bigdata-insider.de/, URL: https://www.bigdata-insider.de/was-ist-ein-cyber-physisches-system-cps-a-668494/ (zuletzt abgerufen am 13.07.2020)

[31] Bonfig K. W.: Technische Druck- und Kraftmessung. 2. Aufl., Expert Verlag, Renningen/Malmsheim, 1995

[32] Klocke F.: Fertigungsverfahren 4. Umformen. 6. Aufl., Springer Vieweg Verlag, Berlin, 2017

[33] Winkler H., Berger U., Mieke C., Schenk M.: Flexibilisierung der Fabrik im Kontext von Industrie 4.0. Anwendungsorientierte Beiträge zum Industriellen Management. 6. Band., Logos Verlag, Berlin 2017

[34] Bosch G., Bromberg T., Haipeter T., Schmitz J. (2017): Industrie und Arbeit 4.0 : Befunde zu Digitalisierung und Mitbestimmung im Industriesektor auf Grundlage des Projekts „Arbeit 2020“. Fakultät für Gesellschaftswissenschaften, Institut Arbeit und Qualifikation (IAQ), Universität Duisburg-Essen

[35] Thiede S., Juraschek M., Herrmann C. (2016): Implementing cyber-physical production systems in learning factories. Chair of Sustainable Manufacturing and Life Cycle Engineering, Institute of Machine Tools and Production Technology (IWF),
Technische Universität Braunschweig

[36] Bauer W., Schlund S., Marrenbach D., Ganschar O. (2014): Industrie 4.0 – Volkswirtschaftliches Potenzial für Deutschland.

[37] Lee E. A., Seshia S. A.: Introduction to Embedded Systems: A Cyber-Physical Systems Approach. 2. Aufl., MIT Press, Cambridge (MA), 2016

[38] Bory M.: Cyber-Physical Systems: Innovation durch softwareintensive eingebettete Systeme. Springer Verlag, Berlin, 2011

[39] Geisberger E., Broy M.: agendaCPS - Integrierte Forschungsagenda Cyber-Physical Systems (acatech STUDIE), Springer Verlag, Heidelberg 2012

[40] Brettel M., Friederichsen N., Keller M., Rosenberg N.: How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 Perspective. International Journal of Science, Engineering and Technology, 8 (2014) 37-88.

[41] Megatron Elektronik GmbH & Co. KG: Potentiometrischer Wegsensor RC13. In: https://www.megatron.de/ (2020), URL: https://www.megatron.de/fileadmin/user_upload/Datenblaetter/Wegsensoren/Potentiometrische/DS_RC13_dt.pdf (zuletzt abgerufen am 09.09.2020)

[42] ASM Automation Sensorik Messtechnik GmbH: posihall® Magnetische Multiturn-Encode. In: https://www.asm-sensor.com/de/# (2020), URL: https://www.asm-sensor.com/files/asmTheme/pdf/asm_posihall_ph36_de.pdf (zuletzt abgerufen am 09.09.2020)

[43] PR Electronics GmbH: mV-Transmitter 2261, In: https://www.prelectronics.com/de/ (2020), URL:
https://www.prelectronics.com/Umbraco/Surface/PdfGenerator/GeneratePdf?id=19097&langId=14368&culture=de-DE (zuletzt abgerufen am 09.09.2020)

[44] Bundesministerium für Bildung und Forschung (BMBF) Referat Zukunft von Arbeit und Wertschöpfung; Industrie 4.0 (2020): Industrie 4.0. Innovation im Zeitalter der Digitalisierung. In: https://www.bmbf.de/, URL: https://www.bmbf.de/upload_filestore/pub/Industrie_4.0.pdf (zuletzt abgerufen am 20.09.2020)

[45] TIOBE Software BV: TIOBE Index for September 2020. In: https://www.tiobe.com/ (2020), URL: https://www.tiobe.com/tiobe-index/ (zuletzt abgerufen am 20.09.2020)

Anhang

Anhang A: Python-Code Konzept 1

Dem folgenden Anhang kann der Quellcode des ersten Konzeptes entnommen werden.

Anhang A1: Quellcode „skript.py“

```
import os
from module import neues_projekt
from module import sensordata
from module import berechnung_kreis
from module import visualisierung_walze
from module import diagramme
from module import zeitschritte
from module import walzspaltzustellung
from module import berechnung

# ------------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 15.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Ablauf des Skripts:
# 1.) Neues Projekt erstellen mit Modul "create_new_project"
# 2.) Sensordaten mit Modul "sensordata" einlesen --> Output: all_data
# 3.) Kreisberechnung mit Modul "berechnung_kreis" --> Output: data_tmr
# 4.) Auswahl der Zeitschritte mit Modul "zeitschritte" --> Output: Zeitschritt der max. Durchbiegung oder alle Zeitschritte
# 5.) Berechung ausgewählter Größen
# 6.) Auswertung aller Diagramme durch das Modul "diagramme"
# 7.) Visualisierung der Walzen durch das Modul "visualisierung_walze"
# 8.) Zustellung des Walzspaltes durch das Modul "walzspaltzustellung"

# ------------------------------------------------------------------------------------------------------------------------

# 1.) neues_projekt
projekt = "Walzen 1.0038"
neues_projekt.Neues_Projekt(projekt).anlegen()
projektname = neues_projekt.Neues_Projekt.get_projektname()
#print(projektname)

# 2.) sensordata
all_data = sensordata.CsvReader().read()     # all_data = [t1 z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11 z12 z13 z14 z15 z16]
sensordata.CsvWriter().export_all()
sensordata.CsvWriter().export_summary()
#print(all_data)

# 3.) berechnung-keis
data_tmr = berechnung_kreis.Berechnung_Kreis(all_data).sensordata_zu_kreis()   # data_tmr = [t, mp_x, mp_y, mp_z, r]
#print(data_tmr)

# 4.) zeitschritte
t_all = zeitschritte.Zeitinkremente(all_data).get_timesteps()
t = zeitschritte.Zeitfilter(data_tmr, all_data).zeitpunkt_max_durchbiegung()
#print("Alle Zeitschritte t_all = " + str(t_all))
#print("Zeitschritt t zur max. Durchbiegung: " + str(t))

# 5.) berechnung
h_0 = 10        # Einlaufdicke des Walzgutes [mm]
h_1 = 8         # Auslaufdicke des Walzgutes [mm]

berechnung.Berechnung().gedrueckte_lange(h_0, h_1)
berechnung.Berechnung().greifwinkel(h_0, h_1)
berechnung.Berechnung().maximale_dickenabnahme()
berechnung.Berechnung().geruestmodul(all_data)

```

```
# 6.) diagramme
verstaerkung_1 = 1

diagramme.Walzspalt(t, data_tmr, all_data).walzspaltgeometrie(verstaerkung_1)
diagramme.Geruestkennlinie(all_data).geruestkennlinie()
diagramme.Diagramme(all_data).diagramm_kraft()
diagramme.Diagramme(all_data).diagramm_moment()
diagramme.Diagramme(all_data).diagramm_drehzahl()
diagramme.Diagramme(all_data).diagramm_handradwinkel()
diagramme.Diagramme(all_data).diagramm_pumpendruck()
diagramme.Diagramme(all_data).diagramm_all()
diagramme.Diagramme(data_tmr).diagramm_walzenradius()

# 7.) visualisierung_walze
verstaerkung_2 = 100

visualisierung_walze.Untere_Walze(t, data_tmr).biegeline_3d(verstaerkung_2)
visualisierung_walze.Untere_Walze(t, data_tmr).plot_surface(verstaerkung_2)
visualisierung_walze.Untere_Walze(t, data_tmr).plot_kreisebenen()
visualisierung_walze.Beide_Walzen(t, data_tmr, all_data).plot_surface()

# 8.) walzspaltzustellung
alpha = walzspaltzustellung.Zustellung(all_data).zustellung()
#print(alpha)

```

Anhang A2: Quellcode „neues_projekt.py“

```python
import os
from time import strftime, localtime

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 25.06.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# NewProject().new_folder()
    # Anlegen eines Projektordner auf einem vordefinierten Pfad mit einem vom Benutzer festgelegten Projektnamen.
    # Der Unterordner für die exportieren csv-Daten (aus dem Modul 'sensordata') wird automatisch angelegt.

# NewProject().new_folder()
    # Gibt den Projektordnernamen zurück.

# ----------------------------------------------------------------------------------------------------------------------

class Neues_Projekt():
    def __init__(self, projekt):
        self.projekt = projekt

    def anlegen(self):
        projektname = self.projekt
        date = strftime("%Y-%m-%d [%H-%M-%S]", localtime())     # Datum + Zeit zum Zeitpunkt des Erstellens
        global ordnername
        ordnername = date + " - " + projektname                 # gesamter Projektordername

        path = "./test_projects/" + ordnername                  # Pfad in dem der Projektordner angelegt werden soll

        if not os.path.exists(path):
            os.makedirs(path)                                   # Erstellen des Projektordners
            print("Projektordner '" + ordnername + "' wurde angelegt")
        else:
            print("Projektordner '" + ordnername + "' existiert bereits")

        os.makedirs(path + "/csv-data")                         # Unterordner für exportierte csv-Dateien der Sensoren

    def get_projektname():
        return ordnername

```

Anhang A3: Quellcode „sensorkoordianten.py“

```python
import numpy as np

# ------------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 25.06.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# get_sensorkoordianten():
    # Liefert die x- und y-Koordianten der Wirbelstromsensoren.

# ------------------------------------------------------------------------------------------------------------------------

# Sensoranordnung:
# _____________________________   (hinten)
#
#          3       6       9          y3
#          2       5       8          y2
#          1       4       7          y1
#  ____________________________
#  x0     x1      x2      x3     x4   (vorne)
#   Blecheinzugsseite

def get_sensorkoordinaten():
    # Grundlegende Abmessungen des Walzgerüstes:
    D = 203     # Walzendurchmesser [mm] (siehe Unterlagen des Institutes zur Anlage)
    a = 1       # Abstand zwischen Sensor und Walze (gemäß gewähltem Wirbelstromsensor)

    x0 = 0      # Anfang des Ballen der Walze   [mm]
    x4 = 220    # Länge des Ballen der Walze    [mm]

    alpha_deg = 20  # Winkel der Sensoren 1,3,4,6,7,9 [°] (Sensoranordnung: siehe oben, Skizze: siehe Masterarbeit)
    alpha = alpha_deg*np.pi/180

    # x-Koordinaten der Sensorebenen:
    x1 = x4/2-30      # Sensorebene 1 - links (= konst., da fix montiert)     [mm]
    x2 = x4/2         # Sensorebene 2 - mitte (= konst., da fix montiert)     [mm]
    x3 = x4/2+30      # Sensorebene 3 - rechts (= konst., da fix montiert)    [mm]

    # Koordinaten der Wirbelstromsensoren:
    y1 = (D/2+a)*np.sin(alpha)      # y-Koordinate des vorderen Sensors (= konst., da fix montiert)     [mm]
    y2 = 0                          # y-Koordinate des mittleren Sensors (= konst., da fix montiert)    [mm]
    y3 = -(D/2+a)*np.sin(alpha)     # y-Koordinate des hinteren Sensors (= konst., da fix montiert)     [mm]

    z1 = -(D/2+a)*np.cos(alpha)     # z-Koordinate des vorderen Sensors (= konst., da fix montiert)     [mm]
    z2 = -(D/2+a)                   # z-Koordinate des mittleren Sensors (= konst., da fix montiert)    [mm]
    z3 = -(D/2+a)*np.cos(alpha)     # z-Koordinate des hinteren Sensors (= konst., da fix montiert)     [mm]

    return [x0, x1, x2, x3, x4, y1, y2, y3, z1, z2, z3, alpha]

    #return [x1, y1, x2, y2, x3, y3, x4, x0]
```

Anhang A4: Quellcode „sensordata.py“

```
import csv
import os
from module import neues_projekt
from shutil import copyfile
import numpy as np

# ---------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 31.05.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# CsvReader().read()
    # Lesen der csv-Dateien der Sensoren und Ausgabe einer Liste im Format [[Zeit1, Sensor1, ..., Sensor 15], [...]]

# CsvWriter().export_all()
    # Exportieren aller csv-Dateien der Sensoren in den Projektordner in den Unterordner 'csv-data'

# CsvWriter().export_summary()
    # Exportieren einer Sammlung aller csv-Dateien der Sensoren als eine einzelne csv-Datei in den Projektordner
    # in den Unterordner 'csv-data'

# ---------------------------------------------------------------------------------------------------------------------

class CsvReader():
    def __init__(self):
        pass

    def read(self):
        n = 1
        number_files = len(os.listdir("sensordata"))    # WICHTIG! Pfad relativ zum ausführenden Skript (script.py)!

        global data_all_sensors
        data_all_sensors = []
        global data_all_sensors_transposed
        data_all_sensors_transposed = []

        # Aufbau der zu lesenden csv-Dateien:   Zeit; Sensor
        # Aufbau der erzeugten csv-Datei:       Zeit; Sensor1; Sensor2; ..., Sensor 15
        # Aufbau der erzeugten Liste:           [[Zeit1, Sensor1, Sensor2, ..., Sensor 15],
        # (data_all_sensors_transposed)          [Zeit2, Sensor1, Sensor2, ..., Sensor 15]]

        # Einlesen der Zeitschritte:
        with open("sensordata/" + "data_sensor" + str(n) + ".csv", "r", newline='', encoding='utf-8') as file:
            reader_timesteps = csv.reader(file, delimiter=';', quotechar='|')

            time = []
            for row in reader_timesteps:
                time.append(float(row[0]))
            data_all_sensors.append(time)

        # Einlesen der Sensordaten:
        for n in range(1, number_files + 1):
            csvfilename = "data_sensor" + str(n) + ".csv"   # Name der einzulesenden csv-Datein

            data_sensor = []
            with open ("sensordata/" + csvfilename, "r", newline='', encoding='utf-8') as file:
                reader_data = csv.reader(file, delimiter=';', quotechar='|')

                for row in reader_data:
                    data_sensor.append((float(row[1])))
                data_all_sensors.append(data_sensor)

        data_all_sensors_transposed = np.transpose(data_all_sensors)    # Transponieren auf oben erwähnte Datenstruktur

        return data_all_sensors_transposed


```

```
69  class CsvWriter():
70      def __init__(self):
71          pass
72
73      def export_all(self):
74          n = 1
75          number_files = len(os.listdir("sensordata"))
76
77          for n in range(1, number_files + 1):
78              csv_filename = "data_sensor" + str(n) + "_export.csv"
79              foldername = neues_projekt.Neues_Projekt.get_projektname()
80
81              source = "sensordata/data_sensor" + str(n) + ".csv"
82              destination = "test_projects/" + foldername + "/csv-data/" + csv_filename
83
84              copyfile(source, destination)
85
86          print("Export aller .csv-Dateien in den Projektordner '" + foldername + "' erfolgreich")
87
88      def export_summary(self):
89          number_files = len(os.listdir("sensordata"))
90          csv_data = CsvReader().read()
91
92          csv_filename = "data_sensors_summary.csv"
93          foldername = neues_projekt.Neues_Projekt.get_projektname()
94
95          with open("test_projects/" + foldername + "/csv-data/" + csv_filename, "w", newline='', encoding='utf-8') as file:
96              writer = csv.writer(file, delimiter=';', quotechar='|')
97
98              header = ["Zeit", "Sensor1", "Sensor2", "Sensor3", "Sensor4", "Sensor5", "Sensor6", "Sensor7", "Sensor8",
99                        "Sensor9", "Sensor10", "Sensor11", "Sensor12", "Sensor13", "Sensor14", "Sensor15", "Sensor16"]
100             writer.writerow(header)
101
102             # Reihen auf Spalten transponieren
103             for line in data_all_sensors_transposed:
104                 writer.writerow(line)
105
106         print("Export der Sammlung aller .csv-Dateien in den Projektordner '" + foldername + "' erfolgreich")
```

Anhang A5: Quellcode „zeitschritte.py“

```python
import numpy as np
from module import sensorkoordinaten
import scipy.optimize as opt

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 01.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Zeitinkremente(all_data).get_timesteps()
    # Gibt aus dem Datensatz "all_data", welcher alle Sensordaten enthält, alle Zeitschritte zurück.

# Zeitfilter(data_tmr, all_data).zeitpunkt_max_durchbiegung():
    # Gibt aus dem Datensatz "all_data", welcher alle Sensordaten enthält, und dem Datensatz "data_tmr", welcher alle
    # berechneten x-y-z-Koordianten der Biegelinie (sowie die errechneten Walzenradien) enthält, den Zeitschritt zurück,
    # zu dem die max. Durchbiegung der Walze vorherrscht.

# ----------------------------------------------------------------------------------------------------------------------

class Zeitinkremente():
    def __init__(self, all_data):
        self.all_data = all_data

    def get_timesteps(self):        # Erstellen einer Liste, die jeden Zeitschritt (nur) einmal enthält.
        timesteps = []              # Erstellen einer leeren Liste für die Zeitschritte
        for line in self.all_data:  # Auslesen der Zeitschritte aus dem Datensatz
            if line[0] not in timesteps:
                timesteps.append(line[0])
            else:
                pass

        return timesteps


class Zeitfilter():
    def __init__(self, data_tmr, all_data):
        self.data_tmr = data_tmr
        self.all_data = all_data

    def zeitpunkt_max_durchbiegung(self):
        timesteps = Zeitinkremente(self.data_tmr).get_timesteps()
        obere_grenze = sensorkoordinaten.get_sensorkoordinaten()[6]  # Länge/Ende der Walze (für Funktionsgrenze)
        untere_grenze = sensorkoordinaten.get_sensorkoordinaten()[7]  # Anfang der Walze (für Funktionsgrenze)

        t_max = 0           # Zeitpunkt der maximalen Durchbiegung
        delta_z_max = 0     # Maximale Durchbiegung in z

        for t in timesteps:
            x_m = []  # Liste der Mittelpunkte in x-Richtung
            z_m = []  # Liste der Mittelpunkte in z-Richtung

            # Dateneingang (tmr): [[t, mp_x, mp_y, mp_z, r]] von berechnung_kreis.sensordata_zu_kreis()
            for line in self.data_tmr:  # Auslesen des Datensatzes und Einfügen in die obrigen Listen
                if line[0] == t:
                    x_m.append(line[1])
                    z_m.append(line[3])

            # Polynom in z-Richtung --> z(x) = ...
            polynom_coeff_z = np.polyfit(x_m, z_m, 2)  # Polynomkoeffizienten durch Polynom-Fitting
            increments = 100
            x_mnew = np.linspace(untere_grenze, obere_grenze, increments)
            z_mnew = np.poly1d(polynom_coeff_z)  # Polynom z(x) = ...

            for line in self.all_data:
                if line[0] == t:
                    auffederung = line[10]

```

```
69              # Stelle der maximalen Durchbiegung:
70              # x-Koordinate der max. Durchbiegung
71              max_x_uW = opt.fminbound(lambda x: z_mnew(x), untere_grenze, obere_grenze)
72              # x- & z-Koordiante der max. Durchbiegung der unteren Walze
73              max_uW = [max_x_uW, z_mnew(max_x_uW)]
74              # x- & z-Koordiante der max. Durchbiegung der oberen Walze
75              max_oW = [max_x_uW, -z_mnew(max_x_uW) + (auffederung + 2 * polynom_coeff_z[-1])]
76              # Differenz der z-Koordianten = max. Planheitsabweichung des Walzgutes
77              max_durchbiegung_z = max_oW[1] - max_uW[1]
78
79              #print("unten: " + str(max_uW) + "       oben: " + str(max_oW))
80              #print("Maximales delta_z des Walzspaltes = " + str(max_durchbiegung_z) + " mm")
81
82              if max_durchbiegung_z > delta_z_max:      # Erfassung des neuen z_max und t_max
83                  delta_z_max = max_durchbiegung_z      # maximale Durchbiegung in z-Richtung
84                  t max = t                         # Zeitpunkt der maximalen Durchbiegung in z-Richtung
85                  #print("OVERWRITE: t_max = " + str(t_max) + " s, z_max = " + str(z_max) + " mm")
86              else:
87                  #print("Kein neues Maximum")
88                  pass
89
90          print("t_max = " + str(t_max) + " s, delta_z_max = " + str(delta_z_max) + " mm (= " + str(delta_z_max*10**3) +
91                " \u03BCm)")
92
93          return [t_max]
94
```

Anhang A6: Quellcode „berechnung_kreis.py“

```python
from math import sqrt
from module import sensorkoordinaten
import numpy as np

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 01.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Berechnung_Kreis(data).finde_kreis(t,x,z1,z2,z3)
    # Berechnung des Kreismittelpunktes in x-y-z und des Radius r mit den Sensordaten t,x,z1,z2,z3 die durch die
    # Funktion "sensordata_zu_kreis" übergeben werden.
    # Ausgeben wird eine Liste im Format [Zeit, Mittelpunkt x, Mittelpunkt y, Mittelpunkt z, Radius] für einen Kreis.

# Berechnung_Kreis(data).sensordata_zu_kreis()
    # Verwendet die Funktion 'finde_kreis(t,x,z1,z2,z3)' zur Berechnung und gibt eine Liste mit allen Ergebnisse im
    # Format [[Zeit, Mittelpunkt x, Mittelpunkt y, Mittelpunkt z, Radius], [...]] zurück.

# ----------------------------------------------------------------------------------------------------------------------

class Berechnung_Kreis():
    def __init__(self, all_data):
        self.all_data = all_data

        x0, x1, x2, x3, x4, y1, y2, y3, z1, z2, z3, alpha = sensorkoordinaten.get_sensorkoordinaten()
        self.x1 = x1         # Sensorebene 1 - links (= konst., da fix montiert)                    [mm]
        self.x2 = x2         # Sensorebene 2 - mitte (= konst., da fix montiert)                    [mm]
        self.x3 = x3         # Sensorebene 3 - rechts (= konst., da fix montiert)                   [mm]
        self.y1 = y1         # y-Koordinate des vorderen Sensors (= konst., da fix montiert)        [mm]
        self.y2 = y2         # y-Koordinate des mittleren Sensors (= konst., da fix montiert)       [mm]
        self.y3 = y3         # y-Koordinate des hinteren Sensors (= konst., da fix montiert)        [mm]
        self.z1 = z1         # z-Koordinate des vorderen Sensors (= konst., da fix montiert)        [mm]
        self.z2 = z2         # z-Koordinate des mittleren Sensors (= konst., da fix montiert)       [mm]
        self.z3 = z3         # z-Koordinate des hinteren Sensors (= konst., da fix montiert)        [mm]
        self.alpha = alpha   # Winkel der Sensoren 1,3,4,6,7,9 [°] (siehe Masterarbeit oder Modul "sensorkoordinaten")

    def finde_kreis(self, t, x, u1, u2, u3):
        self.t = t           # Zeitpunkt der Messung                                                [s]
        self.x = x           # x-Koordinate (Ebene) des Sensors                                     [mm]
        self.u1 = u1         # Messwert u1 des Sensors (vorne) in der Ebene x                       [mm]
        self.u2 = u2         # Messwert u2 des Sensors (mitte) in der Ebene x                       [mm]
        self.u3 = u3         # Messwert u3 des Sensors (hinten) in der Ebene x                      [mm]
        mittelpunkt_und_radius = []

        # Koordinaten der Sensorenmesspunkte in der Ebene x:
        # Koordianten Sensormesspunktes des Sensors (vorne):
        x1 = self.x                                 # x-Koordinate (Sensorebene) des Sensors und des Messpunktes
        y1 = self.y1-self.u1*np.sin(self.alpha)     # y-Koordinate des Sensors und des Messpunktes (= konst.)
        z1 = self.z1+self.u1*np.cos(self.alpha)     # z-Koordinate des Messpunktes

        # Koordianten Sensormesspunktes des Sensors (mitte):
        x2 = x1                                     # x-Koordinate (Sensorebene) des Sensors und des Messpunktes
        y2 = self.y2                                # y-Koordinate des Sensors und des Messpunktes (= konst.)
        z2 = self.z2+self.u2                        # z-Koordinate des Messpunktes

        # Koordianten Sensormesspunktes des Sensors (hinten):
        x3 = x1                                      # x-Koordinate (Sensorebene) des Sensors und des Messpunktes
        y3 = self.y3+self.u3*np.sin(self.alpha)     # y-Koordinate des Sensors und des Messpunktes (= konst.)
        z3 = self.z3+self.u3*np.cos(self.alpha)     # z-Koordinate des Messpunktes

        # Analytische Lösung zur Berechnung des Mittelpunktes (in x-y-z) und des Radius:
        # Quelle: https://www.geeksforgeeks.org/equation-of-circle-when-three-points-on-the-circle-are-given/
        y12 = y1 - y2
        y13 = y1 - y3

        z12 = z1 - z2
        z13 = z1 - z3

        z31 = z3 - z1
        z21 = z2 - z1
```

```
72
73          y31 = y3 - y1
74          y21 = y2 - y1
75
76          # x1^2 - x3^2
77          sy13 = pow(y1, 2) - pow(y3, 2)
78
79          # y1^2 - y3^2
80          sz13 = pow(z1, 2) - pow(z3, 2)
81
82          sx21 = pow(y2, 2) - pow(y1, 2)
83          sy21 = pow(z2, 2) - pow(z1, 2)
84
85          f = (((sy13) * (y12) + (sz13) *
86                (y12) + (sx21) * (y13) +
87                (sy21) * (y13)) / (2 *
88                ((z31) * (y12) - (z21) * (y13))))
89
90          g = (((sy13) * (z12) + (sz13) * (z12) +
91                (sx21) * (z13) + (sy21) * (z13)) /
92               (2 * ((y31) * (z12) - (y21) * (z13))))
93
94          c = (-pow(y1, 2) - pow(z1, 2) -
95               2 * g * y1 - 2 * f * z1)
96
97          # Kreisgleichung: x^2 + y^2 + 2*g*x + 2*f*y + c = 0
98          # Kreismittelpunkt ist (mp_y = -g, mp_z = -f) und Radius r ist r^2 = h^2 + k^2 - c
99          mp_y = -g                                   # Kreismittelpunkt in y
100         mp_z = -f                                   # Kreismittelpunkt in z
101         sqr_of_r = mp_y * mp_y + mp_z * mp_z - c
102
103         # Radius:
104         r = round(sqrt(sqr_of_r), 10)               # Radius
105
106         #print("Mittelpunkt = (" + str(x1) + ", " + str(mp_y) + ", " +  str(mp_z) + ")")
107         #print("Radius = " +  str(r) + " mm")
108
109         mittelpunkt_und_radius.extend([t, x1, mp_y, mp_z, r])  # [Zeit, Mittelpkt x, Mittelpkt y, Mittelpkt z, Radius]
110
111         return mittelpunkt_und_radius
112
113
114     def sensordata_zu_kreis(self):
115         mittelpunkte_und_radien = []    # Datenstruktur [[t, mp_x, mp_y, mp_z, r]]
116
117         for line in self.all_data:
118             t = line[0]                 # Zeitpunkt der Messung
119
120             #print("\n" + "----------- NEUE DATENREIHE - EBENE 1 - SENSOR 1-3 -----------")
121             u1 = line[1]                # z-Abstand (Messwert) Sensor vorne
122             u2 = line[2]                # z-Abstand (Messwert) Sensor mitte
123             u3 = line[3]                # z-Abstand (Messwert) Sensor hinten
124             ergebnis_ebene1 = Berechnung_Kreis(self.all_data).finde_kreis(t, self.x1, u1, u2, u3)
125             mittelpunkte_und_radien.append(ergebnis_ebene1)
126
127             #print("\n" + "----------- NEUE DATENREIHE - EBENE 2 - SENSOR 4-6 -----------")
128             u4 = line[4]                # z-Abstand (Messwert) Sensor vorne
129             u5 = line[5]                # z-Abstand (Messwert) Sensor mitte
130             u6 = line[6]                # z-Abstand (Messwert) Sensor hinten
131             ergebnis_ebene2 = Berechnung_Kreis(self.all_data).finde_kreis(t, self.x2, u4, u5, u6)
132             mittelpunkte_und_radien.append(ergebnis_ebene2)
133
134             #print("\n" + "----------- NEUE DATENREIHE - EBENE 3 - SENSOR 7-9 -----------")
135             u7 = line[7]                # z-Abstand (Messwert) Sensor vorne
136             u8 = line[8]                # z-Abstand (Messwert) Sensor mitte
137             u9 = line[9]                # z-Abstand (Messwert) Sensor hinten
138             ergebnis_ebene3 = Berechnung_Kreis(self.all_data).finde_kreis(t, self.x3, u7, u8, u9)
139             mittelpunkte_und_radien.append(ergebnis_ebene3)
140
141         return mittelpunkte_und_radien
142
143
```

Anhang A7: Quellcode „berechnung.py“

```python
import numpy as np

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 01.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Vorberechnung(walzendurchmesser, ballenlaenge, e_modul, kraft).durchbiegung():
    # Gibt die maximale Durchbiegung w_max der Walze in z-Richtung zurück.

# Berechnung().gedrueckte_laenge(einlaufdicke, auslaufdicke):
    # Gibt die gedrückte Länge l_d in Abhängigkeit der von Einlaufdicke h_0 und der Auslaufdicke h_1 des Walzgutes
    # zurück.

# Berechnung().greifwinkel(einlaufdicke, auslaufdicke):
    # Gibt den max. Greifwinkel alpha_0 zurück, der benötigt wird um das Walzgut einzuziehen.

# Berechnung().maximale_dickenabnahme():
    # Gibt die maximale Dickenabnahme delta_h_max des Walzgutes zurück.

# Berechnung().geruestmodul(all_data):
    # Gibt den Gerüstmodul C des Walzwerkes zurück. Dabei wird der Datensatz "all_data", die alle Sensordaten enthalten,
    # übergeben und daraus das Gerüstmodul C berechnet.

# ----------------------------------------------------------------------------------------------------------------------

class Vorberechnung():
    def __init__(self, walzendurchmesser, ballenlaenge, e_modul, kraft):
        self.d = walzendurchmesser                      # Walzendruchmesser [mm]
        self.l = ballenlaenge                           # Ballenlänge der Walze [mm]
        self.E = e_modul                                # E-Modul des Walzenwerkstoffes (Werkzeugstahl)[N/mm²]
        self.F = kraft                                  # Walzkraft [N]

    def durchbiegung(self):
        I = self.d**4*np.pi/64                          # Flächenträgheitsmoment der Walze [mm^4]
        w_max = (self.F*self.l**3)/(48*self.E*I)        # Maximale Durchbiegung in der Mitte der Walze [mm]
        w_max = round(w_max, 4)
        print("Max. Durchbiegung in z = " + str(w_max) + " mm (= " + str(w_max*10**3) + " \u03BCm)")

        return w_max


class Berechnung():
    def __init__(self):
        self.r = 203/2                                  # Walzenradius [mm]
        self.reibungskoeffizient = 0.2                  # Reibungskoeffizient zwischen Walzgut und Walze [-]

    def gedrueckte_lange(self, einlaufdicke, auslaufdicke):
        self.h_0 = einlaufdicke                         # Einlaufdicke des Walzgutes [mm]
        self.h_1 = auslaufdicke                         # Auslaufdicke des Walzgutes [mm]

        delta_h = self.h_0 - self.h_1                    # Höhenabnahme des Walzgutes [mm]
        l_d = np.sqrt(self.r*delta_h - delta_h**2/4)    # Gedrückte Länge [mm]
        print("Gedrückte Länge l_d = " + str(round(l_d,3)) + "mm")

        return l_d

    def greifwinkel(self, einlaufdicke, auslaufdicke):
        self.h_0 = einlaufdicke                                     # Einlaufdicke des Walzgutes [mm]
        self.h_1 = auslaufdicke                                     # Auslaufdicke des Walzgutes [mm]

        l_d = Berechnung().gedrueckte_lange(self.h_0, self.h_1)     # Gedrückte Länge [mm]
        alpha_0 = np.arctan(l_d/self.r)*180/np.pi                   # Greifwinkel [°]
        print("Maximaler Greifwinkel alpha_0 = " + str(round(alpha_0, 3)) + "°")

        return alpha_0

    def maximale_dickenabnahme(self):
        delta_h_max = self.reibungskoeffizient**2*self.r            # Maximale Dickenabnahme des Walzgutes [mm]
        print("Maximale Dickenabnahme delta_h_max = " + str(round(delta_h_max, 3)) + "mm")

        return delta_h_max

```

```
75      def geruestmodul(self, all_data):
76          self.all_data = all_data
77          s = []                                          # Walzspalthöhe [mm]
78          F_ges = []                                      # Walzkraft [N]
79
80          for line in self.all_data:
81              s.append(line[10])
82              F_ges.append(line[11] + line[12])
83
84          # Polynom durch Datenpunkte von s und F_ges:
85          poly_coeff = np.polyfit(s, F_ges, 1)            # Grad = 1, linear
86          s_new = np.linspace(min(s), max(s), 100)        # Inkrementierung der Walzspalthöhe
87          F_ges_new = np.poly1d(poly_coeff)               # Gerüstkennlinie als Polynom
88
89          delta_F = F_ges_new[1]                          # Steigung der Gerüstkennlinie [-]
90          delta_s = max(s) - min(s)                       # Differenz der min. und max. Walzspalthöhen [mm]
91          C = delta_F*10**-3/delta_s                             # Gerüstmodul [kN/mm]
92
93          #print(F_ges_new)
94          #print(delta_F)
95          #print(delta_s)
96          print("Gerüstmodul C = " + str(round(C, 3)) + "kN/mm")
97          print("Auffederung 1/C = " + str(round(1/C, 3)) + "mm/kN")
98
99          return C
100
```

Anhang A8: Quellcode „diagramme.py“

```python
import numpy as np
import matplotlib.pyplot as plt
from module import sensorkoordinaten
import scipy.optimize as opt
from module import zeitschritte

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 10.06.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Standardformat().diagramm(self, x_data, y_data, title, x_label, y_label, legend_label):
    # Definiert die Standardformatierung (z.B. Liniefarbe, Achsenbeschriftung) eines Diagrammes.

# Diagramme(all_data).diagramm_kraft():
    # Liefert je ein Diagramm der beiden Kraftmessdosen.

# Diagramme(all_data).diagramm_moment():
    # Liefert ein Diagramm des gemessenen Drehmomentes.

# Diagramme(all_data).diagramm_drehzahl():
    # Liefert ein Diagramm des gemessenen Drehzahl.

# Diagramme(all_data).diagramm_handradwinkel():
    # Liefert ein Diagramm des gemessenen Handradwinkels.

# Diagramme(all_data).diagramm_pumpendruck():
    # Liefert ein Diagramm des gemessenen Pumpendrucks.

# Diagramme(all_data).diagramm_all():
    # Liefert ein Diagramm der beiden Kraftmessdosen, des Drehmomentes und der Drehzahl.

# Diagramme(all_data).diagramm_walzenradius():
    # Liefert ein Diagramm der Walzenradien in allen 3 Sensorebenen.

# Geruestkennlinie(all_data).geruestkennlinie():
    # Liefert ein Diagramm der Gerüstkennline (Walzspaltgröße bzw. Auffederung über Walzkraft).

# Walzspalt(data_tmr, all_data).walzspaltgeometrie()
    # Liefert die Visualisierung des Walzspalzes unter Berücksichtigung der Walzendurchbiegung und Auffederung.

# ----------------------------------------------------------------------------------------------------------------------

# Dateneingang:
#             |Zeit|----------WIRBELSTROM-WALZE---------------|Auffed|F.li-----F.re|Moment|Drehz|Winkel|Pumpendruck|
#               0    1    2    3    4    5    6    7    8    9   10    11      12     13   14   15   16
# all_data = [[0.05, 2.3, 1.1, 2.3, 2.3, 1.1, 2.3, 2.3, 1.1, 2.3, 10, 100000, 101000, 1500, 50, 260, 250]]

class Standardformat():
    def __init__(self):
        pass

    def diagramm(self, x_data, y_data, title, x_label, y_label, legend_label):
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)

        ax.set_axisbelow(True)
        ax.xaxis.grid(color='gray', linestyle='dashed')
        ax.yaxis.grid(color='gray', linestyle='dashed')

        plt.title(title, fontsize=20)
        plt.xlabel(x_label, fontsize=16)
        plt.ylabel(y_label, fontsize=16)
        plt.tick_params(axis='x', labelsize=14)
        plt.tick_params(axis='y', labelsize=14)

        plt.scatter(x_data, y_data, color='r', zorder=1)
        plt.plot(x_data, y_data, color='b', zorder=2, label=legend_label)
        plt.legend(loc="upper left")


```

```
class Diagramme():
    def __init__(self, all_data):
        self.all_data = all_data

#    def get_timesteps(self):        # Erstellen einer Liste, die jeden Zeitschritt (nur) einmal enthält.
#        timesteps = []              # Erstellen einer leeren Liste für die Zeitschritte
#        for line in self.all_data:  # Auslesen der Zeitschritte aus dem Datensatz
#            if line[0] not in timesteps:
#                timesteps.append(line[0])
#            else:
#                pass
#        return timesteps

    def diagramm_kraft(self):
        t = []
        F_li = []
        F_re = []
        for line in self.all_data:
            t.append(line[0])
            F_li.append(line[11])
            F_re.append(line[12])
        Standardformat().diagramm(t, F_li, "Walzenkraft-Zeit", "Zeit t [s]", "Kraft F [N]", "Kraft links")
        Standardformat().diagramm(t, F_re, "Walzenkraft-Zeit", "Zeit t [s]", "Kraft F [N]", "Kraft rechts")
        plt.show()

        return [t, F_li, F_re]

    def diagramm_moment(self):
        t = []
        M = []
        for line in self.all_data:
            t.append(line[0])
            M.append(line[13])
        Standardformat().diagramm(t, M, "Walzendrehmoment-Zeit", "Zeit t [s]", "Drehmoment M [Nm]", "Drehmoment")
        plt.show()

        return [t, M]

    def diagramm_drehzahl(self):
        t = []
        u = []
        for line in self.all_data:
            t.append(line[0])
            u.append(line[14])
        Standardformat().diagramm(t, u, "Walzendrehzahl-Zeit", "Zeit t [s]", "Drehzahl [U/min]", "Drehzahl")
        plt.show()

        return [t, u]

    def diagramm_handradwinkel(self):
        t = []
        alpha = []
        for line in self.all_data:
            t.append(line[0])
            alpha.append(line[15])
        Standardformat().diagramm(t, alpha, "Handradwinkel-Zeit", "Zeit t [s]", "Handradwinkel [°]", "Handradwinkel")
        plt.show()

        return [t, alpha]

    def diagramm_pumpendruck(self):
        t = []
        p = []
        for line in self.all_data:
            t.append(line[0])
            p.append(line[16])
        Standardformat().diagramm(t, p, "Pumpendruck-Zeit", "Zeit t [s]", "Druck [bar]", "Druck")
        plt.show()

        return [t, p]

```

```
144     def diagramm_all(self):
145         # Kräfte
146         fig = plt.figure()
147         ax1 = fig.add_subplot(2,1,1)
148         ax1.set_axisbelow(True)
149         ax1.xaxis.grid(color='gray', linestyle='dashed')
150         ax1.yaxis.grid(color='gray', linestyle='dashed')
151         ax1.set_title("Walzenkraft-Zeit", fontsize=20)
152         ax1.set_xlabel("Zeit t [s]", fontsize=16)
153         ax1.set_ylabel("Kraft F [N]", fontsize=16)
154         ax1.tick_params(axis='x', labelsize=14)
155         ax1.tick_params(axis='y', labelsize=14)
156 
157         t = []
158         F_li = []
159         F_re = []
160         for line in self.all_data:
161             t.append(line[0])
162             F_li.append(line[11])
163             F_re.append(line[12])
164         ax1.scatter(t, F_li, color='r', zorder=1)
165         ax1.plot(t, F_li, color='b', zorder=2, label="Kraft links")
166         ax1.scatter(t, F_re, color='r', zorder=1)
167         ax1.plot(t, F_re, color='orange', zorder=2, label="Kraft rechts")
168         ax1.legend(loc="upper left")
169 
170         # Moment:
171         ax2 = fig.add_subplot(2,2,3)
172         ax2.set_axisbelow(True)
173         ax2.xaxis.grid(color='gray', linestyle='dashed')
174         ax2.yaxis.grid(color='gray', linestyle='dashed')
175         ax2.set_title("Drehmoment-Zeit", fontsize=20)
176         ax2.set_xlabel("Zeit t [s]", fontsize=16)
177         ax2.set_ylabel("Moment M [Nm]", fontsize=16)
178         ax2.tick_params(axis='x', labelsize=14)
179         ax2.tick_params(axis='y', labelsize=14)
180 
181         M = []
182         for line in self.all_data:
183             M.append(line[13])
184         ax2.scatter(t, M, color='r', zorder=1)
185         ax2.plot(t, M, color='b', zorder=2, label="Drehmoment")
186         ax2.legend(loc="upper left")
187 
188         # Drehzahl:
189         ax3 = fig.add_subplot(2,2,4)
190         ax3.set_axisbelow(True)
191         ax3.xaxis.grid(color='gray', linestyle='dashed')
192         ax3.yaxis.grid(color='gray', linestyle='dashed')
193         ax3.set_title("Drehzahl-Zeit", fontsize=20)
194         ax3.set_xlabel("Zeit t [s]", fontsize=16)
195         ax3.set_ylabel("Drehzahl [U/min]", fontsize=16)
196         ax3.tick_params(axis='x', labelsize=14)
197         ax3.tick_params(axis='y', labelsize=14)
198 
199         u = []
200         for line in self.all_data:
201             u.append(line[14])
202         ax3.scatter(t, u, color='r', zorder=1)
203         ax3.plot(t, u, color='b', zorder=2, label="Umdrehungen")
204         ax3.legend(loc="upper left")
205         plt.show()
206 
207         return [t, F_li, F_re, M, u]
208 
209     def diagramm_walzenradius(self):
210         fig = plt.figure()
211         ax1 = fig.add_subplot(1, 1, 1)
212         ax1.set_axisbelow(True)
213         ax1.xaxis.grid(color='gray', linestyle='dashed')
214         ax1.yaxis.grid(color='gray', linestyle='dashed')
215         ax1.set_title("Walzenradius-Zeit", fontsize=20)
216         ax1.set_xlabel("Zeit t [s]", fontsize=16)
217         ax1.set_ylabel("Walzenradius R [mm]", fontsize=16)
218         ax1.tick_params(axis='x', labelsize=14)
219         ax1.tick_params(axis='y', labelsize=14)
220         ax = plt.gca()
221         ax.ticklabel_format(useOffset=False)
```

```
222
223         # Dateneingang: [[t, mp_x, mp_y, mp_z, r]] von berechnung_kreis.sensordata_zu_kreis()
224         # data_tmr = [[t1, mp_x1, mp_y1, mp_z1, r_1],     Ebene 1 (in x-Richtung)
225         #             [t1, mp_x2, mp_y2, mp_z2, r_2],     Ebene 2 (in x-Richtung)
226         #             [t1, mp_x3, mp_y3, mp_z3, r_3],     Ebene 3 (in x-Richtung)
227
228         timesteps = zeitschritte.Zeitinkremente(self.all_data).get_timesteps()
229         x0, x1, x2, x3, x4, y1, y2, y3, z1, z2, z3, alpha = sensorkoordinaten.get_sensorkoordinaten()
230         r_1 = []  # Liste der Walzenradien in Ebene 1
231         r_2 = []  # Liste der Walzenradien in Ebene 2
232         r_3 = []  # Liste der Walzenradien in Ebene 3
233
234         for t in timesteps:
235             for line in self.all_data:  # Auslesen des Datensatzes und Einfügen in die obrigen Listen
236                 if line[0] == t and line[1] == x1:
237                     r_1.append(line[4])
238                 elif line[0] == t and line[1] == x2:
239                     r_2.append(line[4])
240                 elif line[0] == t and line[1] == x3:
241                     r_3.append(line[4])
242                 else:
243                     pass
244
245         ax1.scatter(timesteps, r_1, color='r', zorder=1)
246         ax1.plot(timesteps, r_1, color='b', zorder=2, label="Walzenradius Ebene 1")
247         ax1.scatter(timesteps, r_2, color='r', zorder=1)
248         ax1.plot(timesteps, r_2, color='r', zorder=2, label="Walzenradius Ebene 2")
249         ax1.scatter(timesteps, r_3, color='r', zorder=1)
250         ax1.plot(timesteps, r_3, color='g', zorder=2, label="Walzenradius Ebene 3")
251         ax1.legend(loc="upper left")
252         plt.show()
253         print(r_1)
254
255         return [timesteps, r_1, r_2, r_3]
256
257
258 class Geruestkennlinie():
259     def __init__(self, all_data):
260         self.all_data = all_data
261
262     def geruestkennlinie(self):
263         s = []
264         F_ges = []
265
266         for line in self.all_data:
267             s.append(line[10])
268             F_ges.append(line[11]+line[12])
269
270         # Polynom durch Datenpunkte von s und F_ges:
271         poly_coeff = np.polyfit(s, F_ges, 1)        # Grad = 1, anpassbar
272         s_new = np.linspace(min(s), max(s), 100)
273         F_ges_new = np.poly1d(poly_coeff)
274
275         fig = plt.figure()
276         ax = fig.add_subplot(1, 1, 1)
277         ax.set_axisbelow(True)
278         ax.xaxis.grid(color='gray', linestyle='dashed')
279         ax.yaxis.grid(color='gray', linestyle='dashed')
280
281         plt.title("Gerüstkennlinie", fontsize=20)
282         plt.xlabel("Walzspalt s [mm]", fontsize=16)
283         plt.ylabel("Walzkraft F_ges [N]", fontsize=16)
284         plt.tick_params(axis='x', labelsize=14)
285         plt.tick_params(axis='y', labelsize=14)
286         plt.scatter(s, F_ges, color='r', zorder=1)
287         plt.plot(s_new, F_ges_new(s_new), color='b', zorder=2, label="Gerüstkennlinie")
288         plt.legend(loc="upper left")
289         plt.show()
290
291         return [s, F_ges]
292
293
```

```python
class Walzspalt():
    def __init__(self, t, data_tmr, all_data):
        self.t = t
        self.data_tmr = data_tmr
        self.all_data = all_data

    def walzspaltgeometrie(self, verstaerkung):
        self.verstaerkung = verstaerkung
        timesteps = self.t
        obere_grenze = sensorkoordinaten.get_sensorkoordinaten()[4]         # Länge/Ende der Walze (für Funktionsgrenze)
        untere_grenze = sensorkoordinaten.get_sensorkoordinaten()[0]        # Anfang der Walze (für Funktionsgrenze)

        # Dateneingang (tmr): [[t, mp_x, mp_y, mp_z, r]] von berechnung_kreis.sensordata_zu_kreis()
        for t in timesteps:
            x_m = []  # Liste der Mittelpunkte in x-Richtung
            z_m = []  # Liste der Mittelpunkte in z-Richtung

            for line in self.data_tmr:  # Auslesen des Datensatzes und Einfügen in die obrigen Listen
                if line[0] == t:
                    x_m.append(line[1])
                    z_m.append(line[3])

            # Polynom in z-Richtung --> z(x) = ...
            polynom_coeff_z = np.polyfit(x_m, z_m, 2)  # Polynomkoeffizienten durch Polynom-Fitting
            increments = 100
            x_mnew = np.linspace(untere_grenze, obere_grenze, increments)
            z_mnew = np.poly1d(polynom_coeff_z)  # Polynom z(x) = ...

            for line in self.all_data:
                if line[0] == t:
                    auffederung = line[10]

            fig = plt.figure()
            ax = fig.add_subplot(1, 1, 1)
            ax.set_axisbelow(True)
            ax.xaxis.grid(color='gray', linestyle='dashed')
            ax.yaxis.grid(color='gray', linestyle='dashed')
            plt.title("Walzspaltgeometrie - Zeitpunkt " + str(t) + "s", fontsize=20)
            plt.xlabel("x [mm]", fontsize=16)
            plt.ylabel("z [mm]", fontsize=16)
            plt.tick_params(axis='x', labelsize=14)
            plt.tick_params(axis='y', labelsize=14)
            plt.plot(x_mnew, self.verstaerkung*z_mnew(x_mnew))
            plt.plot(x_mnew, -self.verstaerkung*z_mnew(x_mnew)+(auffederung + 2 * polynom_coeff_z[-1]))

            # Stelle der maximalen Durchbiegung
            max_x_uW = opt.fminbound(lambda x: z_mnew(x), untere_grenze, obere_grenze)
            max_uW = [max_x_uW, self.verstaerkung*z_mnew(max_x_uW)]
            max_oW = [max_x_uW, -self.verstaerkung*z_mnew(max_x_uW)+(auffederung + 2 * polynom_coeff_z[-1])]
            #print("unten: " + str(max_uW) + "       oben: " + str(max_oW))
            plt.scatter(max_uW[0], max_uW[1])
            plt.scatter(max_oW[0], max_oW[1])

            max_durchbiegung_z = max_oW[1] - max_uW[1]

            plt.figtext(0.76, 0.025, "Maximale Durchbiegung = " + str(round(max_durchbiegung_z, 3)) + "mm", ha="center",
                        fontsize=8, bbox={"facecolor": "orange", "alpha": 0.5, "pad": 5})
            #print("Max. Durchbiegung: " + str(max_durchbiegung_z))

            plt.show()
```

Anhang A9: Quellcode „visualisierung_walze.py“

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from module import sensorkoordinaten
from module import zeitschritte

# ---------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 01.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Untere_Walze(data_tmr).get_timesteps():
    # Erstellt eine chronologische Liste aller Zeitpunkte, die jeden Zeitpunkt nur einmal enthält

# Untere_Walze(data_tmr).biegelinie_3d(verstaerkung):
    # Liefert ein Diagramm der Biegelinie der Achse der unteren Arbeitswalze im Raum (3D-Biegelinie). Diese wird durch
    # durch zwei Polynome in der x-z-Ebene und y-z-Ebene mit den erechneten Mittelpunkten aus
    # berechnung_kreis.Berechnung_Kreis(data).sensordata_zu_kreis() ermittelt.
    # Der Parameter "verstaerkung" (verstaerkung = 1 = keine Verstärkung) multipliziert die Daten mit sich selbst, um
    # die Durchbiegung der Walze bzw. deren Biegelinie zu verdeutlichen.

    # Liefert außerdem eine Projektion der Biegelinie in der y-z-Ebene (2D-Biegeline), welche zur Ermittelung des
    # Walzprofies ermittelt.

# Untere_Walze(data_tmr).plot_surface(verstaerkung):
    # Lieftert eine Visualisierung der Walze zum Zeitpunkt t mit Biegeline und Sensorebenen. Zur Erstellung des Modells
    # der Walze wird auf die Funktion datenpunktvernetzung(center_y, center_z, radius, laenge_x, y_polynom, z_polynom)
    # zugegriffen
    # Der Parameter "verstaerkung" (verstaerkung = 1 = keine Verstärkung) multipliziert die Daten mit sich selbst, um
    # die Durchbiegung der Walze bzw. deren Biegelinie zu verdeutlichen.

# Untere_Walze.datenpunktvernetzung(center_y, center_z, radius, laenge_x, y_polynom, z_polynom):
    # Liefert die für das visualisierte Modell der Walze die benötigte Matrix mit den im Konstruktor angeführten Daten

# Untere_Walze(data_tmr).plot_kreisebenen():
    # Liefert eine Visualisierung der Kreisebenen im Raum, die durch die Sensordaten der unteren Arbeitswalze
    # errechnet wurden.

# Beide_Walzen(data_tmr, all_data).plot_surface():
    # Lieftert eine Visualisierung beider Walzen zum Zeitpunkt t mit Biegeline und Sensorebenen. Zur Erstellung des
    # Modells der Walze wird auf die Funktion datenpunktvernetzung(center_y, center_z, radius, laenge_x, y_polynom,
    # z_polynom) zugegriffen

# ---------------------------------------------------------------------------------------------------------------------

# Dateneingang von Modul "berechnung_kreis":
# data_tmr = [[Zeit, Mittelpunkt x, Mittelpunkt y, Mittelpunkt z, Radius], [...]]

class Untere_Walze():
    def __init__(self, timesteps, data_tmr):        # tmr = Time, Mittelpunkt, Radius
        self.data_tmr = data_tmr                    # [Zeit, Mittelpunkt x, Mittelpunkt y, Mittelpunkt z, Radius]
        self.timesteps = timesteps                  # Liste Zeitschritte (alle Zeitschritte oder einzelner Zeitschritt)
        self.obere_grenze = sensorkoordinaten.get_sensorkoordinaten()[4]   # Länge/Ende der Walze (für Funktionsgrenze)
        self.untere_grenze = sensorkoordinaten.get_sensorkoordinaten()[0]  # Anfang der Walze (für Funktionsgrenze)


    def biegeline_3d(self, verstaerkung):
        self.verstaerkung = verstaerkung
        for t in self.timesteps:
            x_m = []                    # Liste der Mittelpunkte in x-Richtung
            y_m = []                    # Liste der Mittelpunkte in y-Richtung
            z_m = []                    # Liste der Mittelpunkte in z-Richtung

            for line in self.data_tmr:  # Auslesen des Datensatzes und Einfügen in die obrigen Listen
                if line[0] == t:
                    x_m.append(line[1])
                    y_m.append(line[2])
                    z_m.append(line[3])

            # Polynom in z-Richtung --> z(x) = ...
            polynom_coeff_z = np.polyfit(x_m, z_m, 2)               # Polynomkoeffizienten durch Polynom-Fitting
            increments = 100
            # x_mnew = np.linspace(min(x_m), max(x_m), increments)  # Lineare Inkrementierung in x-Richtung
            x_mnew = np.linspace(self.untere_grenze, self.obere_grenze, increments)
            z_mnew = np.poly1d(polynom_coeff_z)                     # Polynom z(x) = ...
```

```
77
78              # Polynom in y --> y(x) = ...
79              polynom_coeff_y = np.polyfit(x_m, y_m, 2)               # Polynomkoeffizienten durch Polynom-Fitting
80              y_mnew = np.poly1d(polynom_coeff_y)                     # Polynom z(x) = ...
81
82              # Plot der Biegelinie:
83              ax = plt.figure().add_subplot(111, projection='3d')
84              ax.set_xlabel('x')
85             ax.set_ylabel('y')
86             ax.set_zlabel('z')
87             ax.set_xlim3d(self.untere_grenze, self.obere_grenze)    # Limit der x-Achsenskalierung
88             ax.set_ylim3d(-250, 250)                                # Limit der y-Achsenskalierung
89             ax.set_zlim3d(-150, 350)                                # Limit der z-Achsenskalierung
90             plt.title("Biegelinie - Zeitpunkt " + str(t) + "s \n (Verstärkungsfaktor = " + str(self.verstaerkung) + ")")
91             # Plot der Biegeliniie im Raum (3D)
92             plt.plot(x_mnew, self.verstaerkung*y_mnew(x_mnew), self.verstaerkung*z_mnew(x_mnew))
93             # Projektion der Biegelinie in der y-z-Ebene (2D-Biegeline) = Walzprofilierung
94             plt.plot(x_mnew, [250]*increments, self.verstaerkung*z_mnew(x_mnew))
95             plt.show()
96
97
98      def plot_surface(self, verstaerkung):
99          self.verstaerkung = verstaerkung
100         for t in self.timesteps:
101             # UNTERE WALZE: OBERFLÄCHE + BIEGELINIE:
102             x_m = []                                    # Liste der Mittelpunkte in x-Richtung
103             y_m = []                                    # Liste der Mittelpunkte in y-Richtung
104             z_m = []                                    # Liste der Mittelpunkte in z-Richtung
105             for line in self.data_tmr:                  # Auslesen des Datensatzes und Einfügen in die obrigen Listen
106                 if line[0] == t:
107                     x_m.append(line[1])
108                     y_m.append(line[2])
109                     z_m.append(line[3])
110
111             # Polynom in z-Richtung --> z(x) = ...
112             polynom_coeff_z = np.polyfit(x_m, z_m, 2)   # Polynomkoeffizienten durch Polynom-Fitting
113             increments = 100
114             x_mnew = np.linspace(self.untere_grenze, self.obere_grenze, increments) # Lin. Inkrementierung in x-Richtung
115             z_mnew = np.poly1d(polynom_coeff_z)         # Polynom z(x) = ...
116
117             # Polynom in y --> y(x) = ...
118             polynom_coeff_y = np.polyfit(x_m, y_m, 2)   # Polynomkoeffizienten durch Polynom-Fitting
119             y_mnew = np.poly1d(polynom_coeff_y)         # Polynom y(x) = ...
120
121             # Plot der Oberfläche:
122             fig = plt.figure(figsize=(8, 8))
123             ax = fig.add_subplot(111, projection='3d')
124             plt.title("Zeitpunkt: " + str(t) + "s")
125
126             Xc, Yc, Zc = Untere_Walze.datenpunktvernetzung(0, 0, line[4], self.obere_grenze, self.verstaerkung*y_mnew,
127                                                            self.verstaerkung*z_mnew)
128             ax.plot_surface(Xc, Yc, Zc, alpha=0.5, cmap=cm.RdBu)               # Visualisierung der Walze als Oberfläche
129             # ax.plot_wireframe(Xc,Yc,Zc, color='green', linewidth=0.5)        # Visualisierung der Walze als Netz
130
131             # Plot der Biegelinie (3D):
132             plt.plot(x_mnew, self.verstaerkung*y_mnew(x_mnew), self.verstaerkung*z_mnew(x_mnew), color='g', linewidth=2)
133
134             # Projektionen:
135             #cset = ax.contour(Xc, Yc, Zc, zdir='x', offset=-1, cmap=cm.RdBu)    # Projektion in x-Richtung
136             #cset = ax.contour(Xc, Yc, Zc, zdir='y', offset=4, cmap=cm.RdBu)     # Projektion in y-Richtung
137             #cset = ax.contour(Xc, Yc, Zc, zdir='z', offset=-1, cmap=cm.RdBu)    # Projektion in z-Richtung
138
139
140             # KREISEBENEN
141             for line in self.data_tmr:  # Vergleich mit den Daten
142                 if line[0] == t:        # Wenn der Datensatz zum Zeitpunkt t aufgezeichnet wurde, dann plotten.
143                     x_m = line[1]
144                     y_m = line[2]
145                     z_m = line[3]
146                     r = line[4]
147
148                     increments = 50     # increments -> Kreisinkremente, bestimmen die Auflösung des Kreises
149                     theta = np.linspace(0, 2 * np.pi, increments)
150
151                     x = [x_m] * increments
152                     y = self.verstaerkung*y_m + r * np.sin(theta)
153                     z = self.verstaerkung*z_m + r * np.cos(theta)
154
```

```
                    ax.plot(x, y, z)                                                    # Kreis
                    ax.scatter(x_m, self.verstaerkung*y_m, self.verstaerkung*z_m, marker='x')   # Mittelpunkt

                else:
                    pass

        ax.set_xlim3d(self.untere_grenze, self.obere_grenze)               # Limit der x-Achsenskalierung
        ax.set_ylim3d(-250, 250)                                           # Limit der y-Achsenskalierung
        ax.set_zlim3d(-150, 350)                                           # Limit der z-Achsenskalierung
        ax.set_xlabel('x')                                                 # Beschriftung der x-Achse
        ax.set_ylabel('y')                                                 # Beschriftung der y-Achse
        ax.set_zlabel('z')                                                 # Beschriftung der z-Achse
        plt.title("Durchbiegung der unteren Walze - Zeitpunkt " + str(t) + "s \n (Verstärkungsfaktor = " +
                  str(self.verstaerkung) + ")")
        plt.show()                                                         # Plotten aller Ergebnisse


    def datenpunktvernetzung(center_y, center_z, radius, laenge_x, y_polynom, z_polynom):
        increments = 50                                                    # Inkrementierung (für Auflösung)
        untere_grenze = sensorkoordinaten.get_sensorkoordinaten()[0]       # Anfang der Walze (für Funktionsgrenze)

        x = np.linspace(untere_grenze, laenge_x, increments)               # Inkrementierung entlang der x-Achse
        theta = np.linspace(0, 2 * np.pi, increments)                      # Inkrementierung des Rotationswinkel
        theta_grid, x_grid = np.meshgrid(theta, x)                         # Erstellen eines x-theta-Datenpunktnetzes

        y_poly = y_polynom(x_grid)                                         # Polynom y(x)
        y_grid = radius * np.cos(theta_grid) + center_y + y_poly           # Datenpunktnetz y_grid

        z_poly = z_polynom(x_grid)                                         # Polynom z(x)
        z_grid = radius * np.sin(theta_grid) + center_z + z_poly           # Datenpunktnetz z_grid

        return x_grid, y_grid, z_grid


    def plot_kreisebenen(self):
        # Plot zum Zeitpunkt t
        for t in self.timesteps:                                  # Schrittweises durchgehen der Zeitpunkte
            ax = plt.figure().add_subplot(111, projection='3d')  # 3D-Plot mit in x = 1 Plot, y = 1 Plot an Position 1
            ax.set_xlabel('x')
            ax.set_ylabel('y')
            ax.set_zlabel('z')
            plt.title("Zeitpunkt: " + str(t) + "s")
            for line in self.data_tmr:      # Vergleich mit den Daten
                if line[0] == t:            # Wenn der Datensatz zum Zeitpunkt t aufgezeichnet wurde, dann plotten.
                    x_m = line[1]
                    y_m = line[2]
                    z_m = line[3]
                    r = line[4]

                    increments = 180        # increments -> Kreisinkremente, bestimmen die Auflösung des Kreises
                    theta = np.linspace(0, 2*np.pi, increments)

                    x = [x_m]*increments
                    y = y_m + r*np.sin(theta)
                    z = z_m + r*np.cos(theta)

                    ax.plot(x, y, z)                        # Kreis
                    ax.scatter(x_m, y_m, z_m, marker='x')   # Mittelpunkt

                else:
                    pass

            ax.set_xlim3d(self.untere_grenze, self.obere_grenze)    # Limit der x-Achsenskalierung
            ax.set_ylim3d(-250, 250)                                # Limit der y-Achsenskalierung
            ax.set_zlim3d(-150, 350)                                # Limit der z-Achsenskalierung
            plt.title("Sensorebenen - Zeitpunkt " + str(t) + "s")
            plt.show()


class Beide_Walzen():
    def __init__(self, timesteps, data_tmr, all_data):
        self.data_tmr = data_tmr      # [Zeit, Mittelpunkt x, Mittelpunkt y, Mittelpunkt z, Radius]
        self.all_data = all_data
        self.timesteps = timesteps
        self.obere_grenze = sensorkoordinaten.get_sensorkoordinaten()[4]    # Länge/Ende der Walze (für Funktionsgrenze)
        self.untere_grenze = sensorkoordinaten.get_sensorkoordinaten()[0]   # Anfang der Walze (für Funktionsgrenze)

```

```
232     def plot_surface(self):
233         for t in self.timesteps:
234             # UNTERE WALZE: OBERFLÄCHE + BIEGELINIE:
235             x_m = []                                    # Liste der Mittelpunkte in x-Richtung
236             y_m = []                                    # Liste der Mittelpunkte in y-Richtung
237             z_m = []                                    # Liste der Mittelpunkte in z-Richtung
238             for line in self.data_tmr:                  # Auslesen des Datensatzes und Einfügen in die obrigen Listen
239                 if line[0] == t:
240                     x_m.append(line[1])
241                     y_m.append(line[2])
242                     z_m.append(line[3])
243 
244             # Polynom in z-Richtung --> z(x) = ...
245             polynom_coeff_z = np.polyfit(x_m, z_m, 2)   # Polynomkoeffizienten durch Polynom-Fitting
246             increments = 100
247             x_mnew = np.linspace(self.untere_grenze, self.obere_grenze, increments) # Lin. Inkrementierung in x-Richtung
248             z_mnew = np.poly1d(polynom_coeff_z)         # Polynom z(x) = ...
249 
250             # Polynom in y --> y(x) = ...
251             polynom_coeff_y = np.polyfit(x_m, y_m, 2)   # Polynomkoeffizienten durch Polynom-Fitting
252             y_mnew = np.poly1d(polynom_coeff_y)         # Polynom y(x) = ...
253 
254             # Plot der Oberfläche:
255             fig = plt.figure(figsize=(8, 8))
256             ax = fig.add_subplot(111, projection='3d')
257             plt.title("Zeitpunkt: " + str(t) + "s")
258 
259             radius = line[4]
260             Xc1, Yc1, Zc1 = Untere_Walze.datenpunktvernetzung(0, 0, radius, self.obere_grenze, y_mnew, z_mnew)
261             ax.plot_surface(Xc1, Yc1, Zc1, alpha=0.5, cmap=cm.RdBu)  # Visualisierung der Walze als Oberfläche
262             # ax.plot_wireframe(Xc1,Yc1,Zc1, color='green', linewidth=0.5)         # Visualisierung der Walze als Netz
263 
264             # Plot der Biegelinie (3D):
265             plt.plot(x_mnew, y_mnew(x_mnew), z_mnew(x_mnew), color='g', linewidth=2)
266 
267             # Projektionen:
268             # cset = ax.contour(Xc, Yc, Zc, zdir='x', offset=-1, cmap=cm.RdBu)    # Projektion in x-Richtung
269             # cset = ax.contour(Xc, Yc, Zc, zdir='y', offset=4, cmap=cm.RdBu)     # Projektion in y-Richtung
270             # cset = ax.contour(Xc, Yc, Zc, zdir='z', offset=-1, cmap=cm.RdBu)    # Projektion in z-Richtung
271 
272             # OBERE WALZE: OBERFLÄCHE + BIEGELINIE
273             for line in self.all_data:
274                 if line[0] == t:
275                     auffederung = line[10]
276 
277             plt.plot(x_mnew, y_mnew(x_mnew), -z_mnew(x_mnew) + (2 * radius + auffederung + 2 * polynom_coeff_z[-1]),
278                      color='g', linewidth=2)
279 
280             Xc2, Yc2, Zc2 = Untere_Walze.datenpunktvernetzung(0, 0, radius, self.obere_grenze, y_mnew, -z_mnew + (
281                         2 * radius + auffederung + 2 * polynom_coeff_z[-1]))
282             ax.plot_surface(Xc2, Yc2, Zc2, alpha=0.5, cmap=cm.RdBu)  # Visualisierung der Walze als Oberfläche
283             # ax.plot_wireframe(Xc2,Yc2,Zc2, color='green', linewidth=0.5)         # Visualisierung der Walze als Netz
284 
285             # KREISEBENEN
286             for line in self.data_tmr:      # Vergleich mit den Daten
287                 if line[0] == t:            # Wenn der Datensatz zum Zeitpunkt t aufgezeichnet wurde, dann plotten.
288                     x_m = line[1]
289                     y_m = line[2]
290                     z_m = line[3]
291                     r = line[4]
292 
293                     increments = 50         # increments -> Kreisinkremente, bestimmen die Auflösung des Kreises
294                     theta = np.linspace(0, 2 * np.pi, increments)
295 
296                     x = [x_m] * increments
297                     y = y_m + r * np.sin(theta)
298                     z = z_m + r * np.cos(theta)
299 
300                     ax.plot(x, y, z)                            # Kreis
301                     ax.scatter(x_m, y_m, z_m, marker='x')       # Mittelpunkt
302 
303                 else:
304                     pass
305 
```

```
306         ax.set_xlim3d(self.untere_grenze, self.obere_grenze)    # Limit der x-Achsenskalierung
307         ax.set_ylim3d(-250, 250)                                # Limit der y-Achsenskalierung
308         ax.set_zlim3d(-150, 350)                                # Limit der z-Achsenskalierung
309         ax.set_xlabel('x')                                      # Beschriftung der x-Achse
310         ax.set_ylabel('y')                                      # Beschriftung der y-Achse
311         ax.set_zlabel('z')                                      # Beschriftung der z-Achse
312         plt.title("Durchbiegung beider Walzen - Zeitpunkt " + str(t) + "s")
313         plt.show()                                              # Plotten aller Ergebnisse
314
```

Anhang A10: Quellcode „walzspaltzustellung.py“

```python
import numpy as np

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 01.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Zustellung().get_auffederung(all_data):
    # Gibt aus dem Datensatz "all_data", welcher alle Sensordaten enthält, die Walzspaltöffnung s_0 und die Auslaufdicke
    # h_1 des Walzugtes zurück.

# # Zustellung().get_auffederung(all_data):
    # Gibt über eine Umrechnung von Maschinenparametern (Übersetzung) die aufgrund der Auffederung notwendige
    # Nachstellung in Grad und Umdrehungen zurück.

# ----------------------------------------------------------------------------------------------------------------------

class Zustellung():
    def __init__(self, all_data):
        self.all_data = all_data

    def get_auffederung(self):
        s_0 = np.inf                    # Walzspaltöffnung des Gerüsts [mm]
        h_1 = 0                         # Auslaufdicke des Walzgutes bzw. Walzspalt mit Auffederung [mm]
        for line in self.all_data:
            if line[10] <= s_0:
                s_0 = line[10]
            elif line[10] > h_1:
                h_1 = line[10]
            else:
                pass
        #print(s_0, h_1)

        return s_0, h_1

    def zustellung(self):
        # 360 Grad Drehung des Handrades entsprechen 12,5 Zähnen, 1 Zahn enstpricht 0,07mm Zustellung
        s_0, h_1 = Zustellung(self.all_data).get_auffederung()
        delta_h = h_1 - s_0             # Zustellung [mm]
        z_360 = 12.5                    # Anzahl der Zähne bei 360° Drehung

        z = delta_h*1/0.07              # Anzahl Zähne zur Zustellung um delta_h (1 Zahn = 0.07mm)
        alpha = round(360*z/z_360, 2)   # Notwendige Drehung [°] zur Zustellung um delta_h
        umdrehungen = round(alpha*np.pi/180, 2)

        print("Notwendige Drehung der Zustellung: " + str(alpha) + "° (entspricht " + str(umdrehungen) + " Umdrehungen)")

        return alpha
```

Anhang B: Python-Code Konzept 2

Dem folgenden Anhang kann der Quellcode des zweiten Konzeptes entnommen werden.

Anhang B1: Quellcode „skript.py“

```
import os
from module import neues_projekt
from module import sensordata
from module import diagramme
from module import zeitschritte
from module import walzspaltzustellung
from module import berechnung

# ------------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 15.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Ablauf des Skripts:
# 1.) Neues Projekt erstellen mit Modul "create_new_project"
# 2.) Sensordaten mit Modul "sensordata" einlesen --> Output: all_data
# 3.) Auswahl der Zeitschritte mit Modul "zeitschritte"
# 4.) Berechung ausgewählter Größen
# 5.) Auswertung aller Diagramme durch das Modul "diagramme"
# 6.) Zustellung des Walzspaltes durch das Modul "walzspaltzustellung"

# ------------------------------------------------------------------------------------------------------------------------

# 1.) neues_projekt
projekt = "Walzen 1.0038"
neues_projekt.Neues_Projekt(projekt).anlegen()
projektname = neues_projekt.Neues_Projekt.get_projektname()
#print(projektname)

# 2.) sensordata
all_data = sensordata.CsvReader().read()     # all_data = [t1 z1 z2 z3 z4] = [t1 Auff. Kraft1 Kraft2 Winkel]
sensordata.CsvWriter().export_all()
sensordata.CsvWriter().export_summary()
#print(all_data)

# 3.) zeitschritte
t_all = zeitschritte.Zeitinkremente(all_data).get_timesteps()
#print("Alle Zeitschritte t_all = " + str(t_all))

# 4.) berechnung
h_0 = 10        # Einlaufdicke des Walzgutes [mm]
h_1 = 8         # Auslaufdicke des Walzgutes [mm]

berechnung.Berechnung().gedrueckte_lange(h_0, h_1)
berechnung.Berechnung().greifwinkel(h_0, h_1)
berechnung.Berechnung().maximale_dickenabnahme()
berechnung.Berechnung().geruestmodul(all_data)

# 5.) diagramme
diagramme.Geruestkennlinie(all_data).geruestkennlinie()
diagramme.Diagramme(all_data).diagramm_kraft()
diagramme.Diagramme(all_data).diagramm_handradwinkel()
diagramme.Diagramme(all_data).diagramm_all()

# 6.) walzspaltzustellung
alpha = walzspaltzustellung.Zustellung(all_data).zustellung()
#print(alpha)

```

Anhang B2: Quellcode „neues_projekt.py“

```python
import os
from time import strftime, localtime

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 15.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# NewProject().new_folder()
    # Anlegen eines Projektordner auf einem vordefinierten Pfad mit einem vom Benutzer festgelegten Projektnamen.
    # Der Unterordner für die exportieren csv-Daten (aus dem Modul 'sensordata') wird automatisch angelegt.

# NewProject().new_folder()
    # Gibt den Projektordnernamen zurück.

# ----------------------------------------------------------------------------------------------------------------------

class Neues_Projekt():
    def __init__(self, projekt):
        self.projekt = projekt

    def anlegen(self):
        projektname = self.projekt
        date = strftime("%Y-%m-%d [%H-%M-%S]", localtime())     # Datum + Zeit zum Zeitpunkt des Erstellens
        global ordnername
        ordnername = date + " - " + projektname                 # gesamter Projektordername

        path = "./test_projects/" + ordnername                  # Pfad in dem der Projektordner angelegt werden soll

        if not os.path.exists(path):
            os.makedirs(path)                                   # Erstellen des Projektordners
            print("Projektordner '" + ordnername + "' wurde angelegt")
        else:
            print("Projektordner '" + ordnername + "' existiert bereits")

        os.makedirs(path + "/csv-data")                         # Unterordner für exportierte csv-Dateien der Sensoren

    def get_projektname():
        return ordnername

```

Anhang B3: Quellcode „sensordata.py“

```python
import csv
import os
from module import neues_projekt
from shutil import copyfile
import numpy as np

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 15.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# CsvReader().read()
    # Lesen der csv-Dateien der Sensoren und Ausgabe einer Liste im Format [[Zeit1, Sensor1, ..., Sensor 4], [...]]

# CsvWriter().export_all()
    # Exportieren aller csv-Dateien der Sensoren in den Projektordner in den Unterordner 'csv-data'

# CsvWriter().export_summary()
    # Exportieren einer Sammlung aller csv-Dateien der Sensoren als eine einzelne csv-Datei in den Projektordner
    # in den Unterordner 'csv-data'

# ----------------------------------------------------------------------------------------------------------------------

class CsvReader():
    def __init__(self):
        pass

    def read(self):
        n = 1
        number_files = len(os.listdir("sensordata"))    # WICHTIG! Pfad relativ zum ausführenden Skript (script.py)!

        global data_all_sensors
        data_all_sensors = []
        global data_all_sensors_transposed
        data_all_sensors_transposed = []

        # Aufbau der zu lesenden csv-Dateien:   Zeit; Sensor
        # Aufbau der erzeugten csv-Datei:       Zeit; Sensor1; Sensor2; ..., Sensor 15
        # Aufbau der erzeugten Liste:           [[Zeit1, Sensor1, Sensor2, ..., Sensor 15],
        # (data_all_sensors_transposed)          [Zeit2, Sensor1, Sensor2, ..., Sensor 15]]

        # Einlesen der Zeitschritte:
        with open("sensordata/" + "data_sensor" + str(n) + ".csv", "r", newline='', encoding='utf-8') as file:
            reader_timesteps = csv.reader(file, delimiter=';', quotechar='|')

            time = []
            for row in reader_timesteps:
                time.append(float(row[0]))
            data_all_sensors.append(time)

        # Einlesen der Sensordaten:
        for n in range(1, number_files + 1):
            csvfilename = "data_sensor" + str(n) + ".csv"   # Name der einzulesenden csv-Datein

            data_sensor = []
            with open ("sensordata/" + csvfilename, "r", newline='', encoding='utf-8') as file:
                reader_data = csv.reader(file, delimiter=';', quotechar='|')

                for row in reader_data:
                    data_sensor.append((float(row[1])))
                data_all_sensors.append(data_sensor)

        data_all_sensors_transposed = np.transpose(data_all_sensors)    # Transponieren auf oben erwähnte Datenstruktur

        return data_all_sensors_transposed


class CsvWriter():
    def __init__(self):
        pass

```

```
73      def export_all(self):
74          n = 1
75          number_files = len(os.listdir("sensordata"))
76
77          for n in range(1, number_files + 1):
78              csv_filename = "data_sensor" + str(n) + "_export.csv"
79              foldername = neues_projekt.Neues_Projekt.get_projektname()
80
81              source = "sensordata/data_sensor" + str(n) + ".csv"
82              destination = "test_projects/" + foldername + "/csv-data/" + csv_filename
83
84              copyfile(source, destination)
85
86          print("Export aller .csv-Dateien in den Projektordner '" + foldername + "' erfolgreich")
87
88      def export_summary(self):
89          number_files = len(os.listdir("sensordata"))
90          csv_data = CsvReader().read()
91
92          csv_filename = "data_sensors_summary.csv"
93          foldername = neues_projekt.Neues_Projekt.get_projektname()
94
95          with open("test_projects/" + foldername + "/csv-data/" + csv_filename, "w", newline='', encoding='utf-8') as file:
96              writer = csv.writer(file, delimiter=';', quotechar='|')
97
98              header = ["Zeit", "Sensor1", "Sensor2", "Sensor3", "Sensor4"]
99              writer.writerow(header)
100
101             # Reihen auf Spalten transponieren
102             for line in data_all_sensors_transposed:
103                 writer.writerow(line)
104
105         print("Export der Sammlung aller .csv-Dateien in den Projektordner '" + foldername + "' erfolgreich")
```

Anhang B4: Quellcode „zeitschritte.py“

```python
import numpy as np

# ---------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 15.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Zeitinkremente(all_data).get_timesteps()
    # Gibt aus dem Datensatz "all_data", welcher alle Sensordaten enthält, alle Zeitschritte zurück.

# ---------------------------------------------------------------------------------------------------------------------

class Zeitinkremente():
    def __init__(self, all_data):
        self.all_data = all_data

    def get_timesteps(self):        # Erstellen einer Liste, die jeden Zeitschritt (nur) einmal enthält.
        timesteps = []              # Erstellen einer leeren Liste für die Zeitschritte
        for line in self.all_data:  # Auslesen der Zeitschritte aus dem Datensatz
            if line[0] not in timesteps:
                timesteps.append(line[0])
            else:
                pass

        return timesteps

```

Anhang B5: Quellcode „berechnung.py“

```
import numpy as np

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 15.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Vorberechnung(walzendurchmesser, ballenlaenge, e_modul, kraft).durchbiegung():
    # Gibt die maximale Durchbiegung w_max der Walze in z-Richtung zurück.

# Berechnung().gedrueckte_laenge(einlaufdicke, auslaufdicke):
    # Gibt die gedrückte Länge l_d in Abhängigkeit der von Einlaufdicke h_0 und der Auslaufdicke h_1 des Walzgutes
    # zurück.

# Berechnung().greifwinkel(einlaufdicke, auslaufdicke):
    # Gibt den max. Greifwinkel alpha_0 zurück, der benötigt wird um das Walzgut einzuziehen.

# Berechnung().maximale_dickenabnahme():
    # Gibt die maximale Dickenabnahme delta_h_max des Walzgutes zurück.

# Berechnung().geruestmodul(all_data):
    # Gibt den Gerüstmodul C des Walzwerkes zurück. Dabei wird der Datensatz "all_data", die alle Sensordaten enthalten,
    # übergeben und daraus das Gerüstmodul C berechnet.

# ----------------------------------------------------------------------------------------------------------------------

class Vorberechnung():
    def __init__(self, walzendurchmesser, ballenlaenge, e_modul, kraft):
        self.d = walzendurchmesser                      # Walzendruchmesser [mm]
        self.l = ballenlaenge                           # Ballenlänge der Walze [mm]
        self.E = e_modul                                # E-Modul des Walzenwerkstoffes (Werkzeugstahl)[N/mm²]
        self.F = kraft                                  # Walzkraft [N]

    def durchbiegung(self):
        I = self.d**4*np.pi/64                          # Flächenträgheitsmoment der Walze [mm^4]
        w_max = (self.F*self.l**3)/(48*self.E*I)        # Maximale Durchbiegung in der Mitte der Walze [mm]
        w_max = round(w_max, 4)
        print("Max. Durchbiegung in z = " + str(w_max) + " mm (= " + str(w_max*10**3) + " \u03BCm)")

        return w_max


class Berechnung():
    def __init__(self):
        self.r = 203/2                                  # Walzenradius [mm]
        self.reibungskoeffizient = 0.2                  # Reibungskoeffizient zwischen Walzgut und Walze [-]

    def gedrueckte_lange(self, einlaufdicke, auslaufdicke):
        self.h_0 = einlaufdicke                         # Einlaufdicke des Walzgutes [mm]
        self.h_1 = auslaufdicke                         # Auslaufdicke des Walzgutes [mm]

        delta_h = self.h_0 - self.h_1                   # Höhenabnahme des Walzgutes [mm]
        l_d = np.sqrt(self.r*delta_h - delta_h**2/4)    # Gedrückte Länge [mm]
        print("Gedrückte Länge l_d = " + str(round(l_d,3)) + "mm")

        return l_d

    def greifwinkel(self, einlaufdicke, auslaufdicke):
        self.h_0 = einlaufdicke                                 # Einlaufdicke des Walzgutes [mm]
        self.h_1 = auslaufdicke                                 # Auslaufdicke des Walzgutes [mm]

        l_d = Berechnung().gedrueckte_lange(self.h_0, self.h_1)     # Gedrückte Länge [mm]
        alpha_0 = np.arctan(l_d/self.r)*180/np.pi                   # Greifwinkel [°]
        print("Maximaler Greifwinkel alpha_0 = " + str(round(alpha_0, 3)) + "°")

        return alpha_0

    def maximale_dickenabnahme(self):
        delta_h_max = self.reibungskoeffizient**2*self.r            # Maximale Dickenabnahme des Walzgutes [mm]
        print("Maximale Dickenabnahme delta_h_max = " + str(round(delta_h_max, 3)) + "mm")

        return delta_h_max
```

```
74
75      def geruestmodul(self, all_data):
76          self.all_data = all_data
77          s = []                                          # Walzspalthöhe [mm]
78          F_ges = []                                      # Walzkraft [N]
79
80          for line in self.all_data:
81              s.append(line[1])
82              F_ges.append(line[2] + line[3])
83
84          # Polynom durch Datenpunkte von s und F_ges:
85          poly_coeff = np.polyfit(s, F_ges, 1)            # Grad = 1, linear
86          s_new = np.linspace(min(s), max(s), 100)        # Inkrementierung der Walzspalthöhe
87          F_ges_new = np.poly1d(poly_coeff)               # Gerüstkennlinie als Polynom
88
89          delta_F = F_ges_new[1]                          # Steigung der Gerüstkennlinie [-]
90          delta_s = max(s) - min(s)                       # Differenz der min. und max. Walzspalthöhen [mm]
91          C = delta_F*10**-3/delta_s                      # Gerüstmodul [kN/mm]
92
93          #print(F_ges_new)
94          #print(delta_F)
95          #print(delta_s)
96          print("Gerüstmodul C = " + str(round(C, 3)) + "kN/mm")
97          print("Auffederung 1/C = " + str(round(1/C, 3)) + "mm/kN")
98
99          return C
100
```

Anhang B6: Quellcode „diagramme.py“

```python
import numpy as np
import matplotlib.pyplot as plt
from module import zeitschritte

# ---------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 15.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Standardformat().diagramm(self, x_data, y_data, title, x_label, y_label, legend_label):
    # Definiert die Standardformatierung (z.B. Liniefarbe, Achsenbeschriftung) eines Diagrammes.

# Diagramme(all_data).diagramm_kraft():
    # Liefert je ein Diagramm der beiden Kraftmessdosen.

# Diagramme(all_data).diagramm_handradwinkel():
    # Liefert ein Diagramm des gemessenen Handradwinkels.

# Diagramme(all_data).diagramm_all():
    # Liefert ein Diagramm der beiden Kraftmessdosen, der Größe des Walzspaltes und des Handradwinkels/Einstellwinkels.

# Geruestkennlinie(all_data).geruestkennlinie():
    # Liefert ein Diagramm der Gerüstkennline (Walzspaltgröße bzw. Auffederung über Walzkraft).

# ---------------------------------------------------------------------------------------------------------------------

# Dateneingang:
#             |Zeit|Auffed|F.li|f.re|Winkel|
#                0    1    2    3    4

class Standardformat():
    def __init__(self):
        pass

    def diagramm(self, x_data, y_data, title, x_label, y_label, legend_label):
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)

        ax.set_axisbelow(True)
        ax.xaxis.grid(color='gray', linestyle='dashed')
        ax.yaxis.grid(color='gray', linestyle='dashed')

        plt.title(title, fontsize=20)
        plt.xlabel(x_label, fontsize=16)
        plt.ylabel(y_label, fontsize=16)
        plt.tick_params(axis='x', labelsize=14)
        plt.tick_params(axis='y', labelsize=14)

        plt.scatter(x_data, y_data, color='r', zorder=1)
        plt.plot(x_data, y_data, color='b', zorder=2, label=legend_label)
        plt.legend(loc="upper left")


class Diagramme():
    def __init__(self, all_data):
        self.all_data = all_data

    def diagramm_kraft(self):
        t = []
        F_li = []
        F_re = []
        for line in self.all_data:
            t.append(line[0])
            F_li.append(line[2])
            F_re.append(line[3])
        Standardformat().diagramm(t, F_li, "Walzenkraft-Zeit", "Zeit t [s]", "Kraft F [N]", "Kraft links")
        Standardformat().diagramm(t, F_re, "Walzenkraft-Zeit", "Zeit t [s]", "Kraft F [N]", "Kraft rechts")
        plt.show()

        return [t, F_li, F_re]

```

```
73      def diagramm_zahnradwinkel(self):
74          d_1 = 264       # Durchmesser des Zahnradkontaktkreises mit dem Reibrad     [mm]
75          d_2 = 52        # Durchmesser des Reibrades                                 [mm]
76          z_1 = 58        # Zähnezahl des Zahnrades zur Walzspaltverstellung          [-]
77          z_2 = 13        # Zähnetahl des Zahnrades des Handrades                     [-]
78          i = d_1 / d_2   # Übersetzung zwischen Zahnradkontaktkreis und Reibrad      [-]
79
80          t = []
81          alpha = []
82          for line in self.all_data:
83              t.append(line[0])
84              alpha.append(line[4]/i)
85
86          Standardformat().diagramm(t, alpha, "Zahnradwinkel-Zeit", "Zeit t [s]", "Zahnradwinkel [°]", "Handradwinkel")
87          plt.show()
88
89          return [t, alpha]
90
91
92      def diagramm_all(self):
93          # Kräfte
94          fig = plt.figure()
95          ax1 = fig.add_subplot(2,1,1)
96          ax1.set_axisbelow(True)
97          ax1.xaxis.grid(color='gray', linestyle='dashed')
98          ax1.yaxis.grid(color='gray', linestyle='dashed')
99          ax1.set_title("Walzenkraft-Zeit", fontsize=20)
100         ax1.set_xlabel("Zeit t [s]", fontsize=16)
101         ax1.set_ylabel("Kraft F [N]", fontsize=16)
102         ax1.tick_params(axis='x', labelsize=14)
103         ax1.tick_params(axis='y', labelsize=14)
104
105         t = []
106         F_li = []
107         F_re = []
108         for line in self.all_data:
109             t.append(line[0])
110             F_li.append(line[2])
111             F_re.append(line[3])
112         ax1.scatter(t, F_li, color='r', zorder=1)
113         ax1.plot(t, F_li, color='b', zorder=2, label="Kraft links")
114         ax1.scatter(t, F_re, color='r', zorder=1)
115         ax1.plot(t, F_re, color='orange', zorder=2, label="Kraft rechts")
116         ax1.legend(loc="upper left")
117
118         # Walzspalthöhe:
119         ax2 = fig.add_subplot(2,2,3)
120         ax2.set_axisbelow(True)
121         ax2.xaxis.grid(color='gray', linestyle='dashed')
122         ax2.yaxis.grid(color='gray', linestyle='dashed')
123         ax2.set_title("Walzspalthöhe-Zeit", fontsize=20)
124         ax2.set_xlabel("Zeit t [s]", fontsize=16)
125         ax2.set_ylabel("Walzspalthöhe s [mm]", fontsize=16)
126         ax2.tick_params(axis='x', labelsize=14)
127         ax2.tick_params(axis='y', labelsize=14)
128
129         s = []
130         for line in self.all_data:
131             s.append(line[1])
132         ax2.scatter(t, s, color='r', zorder=1)
133         ax2.plot(t, s, color='b', zorder=2, label="Walzspalthöhe")
134         ax2.legend(loc="upper left")
135
136         # Zahnradwinkel:
137         ax3 = fig.add_subplot(2,2,4)
138         ax3.set_axisbelow(True)
139         ax3.xaxis.grid(color='gray', linestyle='dashed')
140         ax3.yaxis.grid(color='gray', linestyle='dashed')
141         ax3.set_title("Zahnradwinkel-Zeit", fontsize=20)
142         ax3.set_xlabel("Zeit t [s]", fontsize=16)
143         ax3.set_ylabel("Zahnradwinkel [°]", fontsize=16)
144         ax3.tick_params(axis='x', labelsize=14)
145         ax3.tick_params(axis='y', labelsize=14)
146
```

```
147         d_1 = 264   # Durchmesser des Zahnradkontaktkreises mit dem Reibrad     [mm]
148         d_2 = 52    # Durchmesser des Reibrades                                 [mm]
149         z_1 = 58    # Zähnezahl des Zahnrades zur Walzspaltverstellung          [-]
150         z_2 = 13    # Zähnetahl des Zahnrades des Handrades                     [-]
151         i = d_1/d_2 # Übersetzung zwischen Zahnradkontaktkreis und Reibrad      [-]
152 
153         alpha = []
154         for line in self.all_data:
155             alpha.append(line[4]/i)
156         ax3.scatter(t, alpha, color='r', zorder=1)
157         ax3.plot(t, alpha, color='b', zorder=2, label="Zahnradwinkel")
158         ax3.legend(loc="upper left")
159         plt.show()
160 
161         return [t, F_li, F_re, s, alpha]
162 
163 
164 class Geruestkennlinie():
165     def __init__(self, all_data):
166         self.all_data = all_data
167 
168     def geruestkennlinie(self):
169         s = []
170         F_ges = []
171 
172         for line in self.all_data:
173             s.append(line[1])
174             F_ges.append(line[2]+line[3])
175 
176         # Polynom durch Datenpunkte von s und F_ges:
177         poly_coeff = np.polyfit(s, F_ges, 1)        # Grad = 1, anpassbar
178         s_new = np.linspace(min(s), max(s), 100)
179         F_ges_new = np.poly1d(poly_coeff)
180 
181         fig = plt.figure()
182         ax = fig.add_subplot(1, 1, 1)
183         ax.set_axisbelow(True)
184         ax.xaxis.grid(color='gray', linestyle='dashed')
185         ax.yaxis.grid(color='gray', linestyle='dashed')
186 
187         plt.title("Gerüstkennlinie", fontsize=20)
188         plt.xlabel("Walzspalt s [mm]", fontsize=16)
189         plt.ylabel("Walzkraft F_ges [N]", fontsize=16)
190         plt.tick_params(axis='x', labelsize=14)
191         plt.tick_params(axis='y', labelsize=14)
192         plt.scatter(s, F_ges, color='r', zorder=1)
193         plt.plot(s_new, F_ges_new(s_new), color='b', zorder=2, label="Gerüstkennlinie")
194         plt.legend(loc="upper left")
195         plt.show()
196 
197         return [s, F_ges]
198 
```

Anhang B7: Quellcode „walzspaltzustellung.py“

```python
import numpy as np

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 15.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Zustellung().get_auffederung(all_data):
    # Gibt aus dem Datensatz "all_data", welcher alle Sensordaten enthält, die Walzspaltöffnung s_0 und die Auslaufdicke
    # h_1 des Walzugtes zurück.

# # Zustellung().get_auffederung(all_data):
    # Gibt über eine Umrechnung von Maschinenparametern (Übersetzung) die aufgrund der Auffederung notwendige
    # Nachstellung in Grad und Umdrehungen zurück.

# ----------------------------------------------------------------------------------------------------------------------

class Zustellung():
    def __init__(self, all_data):
        self.all_data = all_data

    def get_auffederung(self):
        s_0 = np.inf                    # Walzspaltöffnung des Gerüsts [mm]
        h_1 = 0                         # Auslaufdicke des Walzgutes bzw. Walzspalt mit Auffederung [mm]
        for line in self.all_data:
            if line[1] <= s_0:
                s_0 = line[1]
            elif line[1] > h_1:
                h_1 = line[1]
            else:
                pass
        #print(s_0, h_1)

        return s_0, h_1

    def zustellung(self):
        # 360 Grad Drehung des Handrades entsprechen 12,5 Zähnen, 1 Zahn enstpricht 0,07mm Zustellung
        s_0, h_1 = Zustellung(self.all_data).get_auffederung()
        delta_h = h_1 - s_0             # Zustellung [mm]
        z_360 = 12.5                    # Anzahl der Zähne bei 360° Drehung

        z = delta_h*1/0.07              # Anzahl Zähne zur Zustellung um delta_h (1 Zahn = 0.07mm)
        alpha = round(360*z/z_360, 2)   # Notwendige Drehung [°] zur Zustellung um delta_h
        umdrehungen = round(alpha*np.pi/180, 2)

        print("Notwendige Drehung der Zustellung: " + str(alpha) + "° (entspricht " + str(umdrehungen) + " Umdrehungen)")

        return alpha

```

Anhang C: Implementierter Python-Code basierend auf Konzept 2

Dem folgenden Anhang kann der Quellcode, welcher implementiert wurde, entnommen werden.

Anhang C1: Quellcode „skript.py“

```
import os
from module import neues_projekt
from module import sensordata
from module import diagramme
from module import zeitschritte
from module import walzspaltzustellung
from module import berechnung

# ------------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 07.09.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Ablauf des Skripts:
# 1.) Neues Projekt erstellen mit Modul "create_new_project"
# 2.) Sensordaten mit Modul "sensordata" einlesen --> Output: all_data
# 3.) Auswahl der Zeitschritte mit Modul "zeitschritte"
# 4.) Berechung ausgewählter Größen
# 5.) Auswertung aller Diagramme durch das Modul "diagramme"
# 6.) Zustellung des Walzspaltes durch das Modul "walzspaltzustellung"

# ------------------------------------------------------------------------------------------------------------------------

# 1.) neues_projekt
projekt = "Walzen 1.0038"
neues_projekt.Neues_Projekt(projekt).anlegen()
projektname = neues_projekt.Neues_Projekt.get_projektname()
#print(projektname)

# 2.) sensordata
all_data = sensordata.CsvReader().read()     # all_data = [t1 z1 z2 z3 z4] = [t1 Auff. Kraft1 Kraft2 Winkel]
sensordata.CsvWriter().export_all()
sensordata.CsvWriter().export_summary()
#print(all_data)

# 3.) zeitschritte
t_all = zeitschritte.Zeitinkremente(all_data).get_timesteps()
#print("Alle Zeitschritte t_all = " + str(t_all))

# 4.) berechnung
h_0 = 10        # Einlaufdicke des Walzgutes [mm]
h_1 = 8         # Auslaufdicke des Walzgutes [mm]

berechnung.Berechnung().gedrueckte_lange(h_0, h_1)
berechnung.Berechnung().greifwinkel(h_0, h_1)
berechnung.Berechnung().maximale_dickenabnahme()
berechnung.Berechnung().geruestmodul(all_data)

# 5.) diagramme
diagramme.Geruestkennlinie(all_data).geruestkennlinie()
diagramme.Diagramme(all_data).diagramm_kraft()
diagramme.Diagramme(all_data).diagramm_zahnradwinkel()
diagramme.Diagramme(all_data).diagramm_all()

# 6.) walzspaltzustellung
alpha = walzspaltzustellung.Zustellung(all_data).zustellung()
#print(alpha)

```

Anhang C2: Quellcode „neues_projekt.py“

```python
import os
from time import strftime, localtime

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 15.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# NewProject().new_folder()
    # Anlegen eines Projektordner auf einem vordefinierten Pfad mit einem vom Benutzer festgelegten Projektnamen.
    # Der Unterordner für die exportieren csv-Daten (aus dem Modul 'sensordata') wird automatisch angelegt.

# NewProject().new_folder()
    # Gibt den Projektordnernamen zurück.

# ----------------------------------------------------------------------------------------------------------------------

class Neues_Projekt():
    def __init__(self, projekt):
        self.projekt = projekt

    def anlegen(self):
        projektname = self.projekt
        date = strftime("%Y-%m-%d [%H-%M-%S]", localtime())     # Datum + Zeit zum Zeitpunkt des Erstellens
        global ordnername
        ordnername = date + " - " + projektname                 # gesamter Projektordername

        #path = "./test_projects/" + ordnername                  # Pfad in dem der Projektordner angelegt werden soll
        path = "L:/010_LUT/Digitalisierung/Walzwerk/Versuche"

        if not os.path.exists(path):
            os.makedirs(path)                                   # Erstellen des Projektordners
            print("Projektordner '" + ordnername + "' wurde angelegt")
        else:
            print("Projektordner '" + ordnername + "' existiert bereits")

        os.makedirs(path + "/csv-data")                         # Unterordner für exportierte csv-Dateien der Sensoren

    def get_projektname():
        return ordnername

```

Anhang C3: Quellcode „sensordata.py“

```
import csv
import os
from module import neues_projekt
from shutil import copyfile
import numpy as np
from glob import glob

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 06.09.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# CsvReader().read()
    # Lesen der csv-Dateien der Sensoren und Ausgabe einer Liste im Format [[Zeit1, Sensor1, ..., Sensor 4], [...]]

# CsvWriter().export_all()
    # Exportieren aller csv-Dateien der Sensoren in den Projektordner in den Unterordner 'csv-data'

# CsvWriter().export_summary()
    # Exportieren einer Sammlung aller csv-Dateien der Sensoren als eine einzelne csv-Datei in den Projektordner
    # in den Unterordner 'csv-data'

# ----------------------------------------------------------------------------------------------------------------------

class CsvReader():
    def __init__(self):
        pass

    def read(self):
        # Anlegen der Datensätze:
        global data_all_sensors
        data_all_sensors = []
        global data_all_sensors_transposed
        data_all_sensors_transposed = []


        # Festlegen des Ordners in dem die Messdaten der Sensoren abgelegt werden:
        #pfad_absolut = "D:/Uni/Master/Master-Arbeit/Arbeit/Python Implementiert/sensordata"
        global pfad_absolut
        pfad_absolut = "L:/010_LUT/Digitalisierung/Walzwerk/2020/September"
        number_files = len(os.listdir(pfad_absolut))    # WICHTIG! Pfad relativ zum ausführenden Skript (script.py)!
        #print(number_files)


        # Auslesen der Files in diesem Ordner:
        filenames = sorted(os.listdir(pfad_absolut))    # Namen der Files in alphabetischer Reihenfolge
        #print(filenames)
        newest_files = filenames[number_files-4:number_files]   # Namen der neuesten (vier letzten) Files
        #print(newest_files)
        newest_files = [newest_files[3], newest_files[1], newest_files[2], newest_files[0]] # Neuordnung zum Auslesen
        #print(newest_files)

        # Aufbau der zu lesenden csv-Dateien:   Zeit, Channel 1, Channel 2, ..., Channel n
        #                                       (eine Zeile entspricht einem Interval von 1 Sekunde)
        # Aufbau der erzeugten csv-Datei:       Zeit; Sensor1; Sensor2; Sensor3; Sensor 4
        # Aufbau der erzeugten Liste:           [[Zeit1, Sensor1, Sensor2, Sensor3, Sensor 4],
        # (data_all_sensors_transposed)          [Zeit2, Sensor1, Sensor2, Sensor3, Sensor 4]]


        # Einlesen der Zeitschritte aus dem ersten File aus newest_files:
        with open(str(pfad_absolut) + "/" + str(newest_files[0]), "r", newline='', encoding='utf-8') as file:
            reader_timesteps = csv.reader(file, delimiter=',', quotechar='|')

            content_file = []                               # Auslesen des ersten Files
            for row in reader_timesteps:
                content_file.append(row)
            #print(content_file)

            global start_time
            start_time = (content_file[1][0])  # Startzeit des Versuchs (Jahr-Monat-Tag Stunde-Minute-Sekunde)

            timesteps_per_second = len(content_file[0])-1     # Zeitschritte pro Sekunde (-1 -> ohne Zeitspalte)
            timesteps = len(content_file)-1                   # Gesamtzeit (eine Zeile entspricht 1s, -1 -> ohne Header)
            #print(timesteps_per_second)
            #print(timesteps)
```

```
77
78              total_time = []                                 # Alle Zeitschritte des Prozesses in Sekunden
79              for i in range(0, timesteps):
80                  for j in range(0, timesteps_per_second):
81                      t = i+j/timesteps_per_second
82                      total_time.append(round(t,5))
83              #print(total_time)
84              data_all_sensors.append(total_time)
85
86
87          # Einlesen der Sensordaten:
88          for file in newest_files:
89              data_sensor = []
90              with open (str(pfad_absolut) + "/" + str(file), "r", newline='', encoding='utf-8') as file:
91                  reader_data = csv.reader(file, delimiter=',', quotechar='|')
92                  next(reader_data)                           # Überspringen der Kopfzeile
93
94                  for row in reader_data:                     # Auslesen aller Reihen
95                      for datapoint in row[1:]:               # Auslesen aller Datenpunkte einer Reihe (bis auf Zeit)
96                          data_sensor.append(float(datapoint))
97
98                  data_all_sensors.append(data_sensor)
99
100         #print(len(data_all_sensors[0]), len(data_all_sensors[1]), len(data_all_sensors[2]), len(data_all_sensors[3]),
101         #      len(data_all_sensors[4]))    # Überprüfen der Dimensionen der Daten -> alle Länge müssen gleich sein
102
103         data_all_sensors_transposed = np.transpose(data_all_sensors)    # Transponieren auf oben erwähnte Datenstruktur
104         #print(data_all_sensors_transposed)
105
106         return data_all_sensors_transposed
107
108
109 class CsvWriter():
110     def __init__(self):
111         pass
112
113     def export_all(self):
114         number_files = len(os.listdir(pfad_absolut))            # Anzahl der Files im Ordner
115         filenames = sorted(os.listdir(pfad_absolut))            # Namen der Files in alphabetischer Reihenfolge
116         newest_files = filenames[number_files-4:number_files]   # Namen der neuesten (vier letzten) Files
117
118         for file in newest_files:
119             csv_filename = file
120             foldername = neues_projekt.Neues_Projekt.get_projektname()
121
122             source = pfad_absolut + "/" + csv_filename
123             destination = "test_projects/" + foldername + "/csv-data/" + csv_filename
124
125             copyfile(source, destination)
126
127         print("Export aller .csv-Dateien in den Projektordner '" + foldername + "' erfolgreich")
128
129
130     def export_summary(self):
131         csv_data = CsvReader().read()
132
133         csv_filename = "data_sensors_summary.csv"
134         foldername = neues_projekt.Neues_Projekt.get_projektname()
135
136         with open("test_projects/" + foldername + "/csv-data/" + csv_filename, "w", newline='', encoding='utf-8') as file:
137             writer = csv.writer(file, delimiter=';', quotechar='|')
138
139             header = ["Zeit", "Linearpotentiometer", "KMD_links", "KMD_rechts", "Winkel"]
140             writer.writerow(header)
141
142             # Reihen auf Spalten transponieren
143             for line in data_all_sensors_transposed:
144                 writer.writerow(line)
145
146         print("Export der Sammlung aller .csv-Dateien in den Projektordner '" + foldername + "' erfolgreich")
```

Anhang C4: Quellcode „zeitschritte.py“

```python
import numpy as np

# ---------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 06.09.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Zeitinkremente(all_data).get_timesteps()
    # Gibt aus dem Datensatz "all_data", welcher alle Sensordaten enthält, alle Zeitschritte zurück.

# ---------------------------------------------------------------------------------------------------------------------

class Zeitinkremente():
    def __init__(self, all_data):
        self.all_data = all_data

    def get_timesteps(self):        # Erstellen einer Liste, die jeden Zeitschritt (nur) einmal enthält.
        timesteps = []              # Erstellen einer leeren Liste für die Zeitschritte
        for line in self.all_data:  # Auslesen der Zeitschritte aus dem Datensatz
            if line[0] not in timesteps:
                timesteps.append(line[0])
            else:
                pass

        return timesteps

```

Anhang C5: Quellcode „berechnung.py“

```
import numpy as np

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 06.09.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Vorberechnung(walzendurchmesser, ballenlaenge, e_modul, kraft).durchbiegung():
    # Gibt die maximale Durchbiegung w_max der Walze in z-Richtung zurück.

# Berechnung().gedrueckte_laenge(einlaufdicke, auslaufdicke):
    # Gibt die gedrückte Länge l_d in Abhängigkeit der von Einlaufdicke h_0 und der Auslaufdicke h_1 des Walzgutes
    # zurück.

# Berechnung().greifwinkel(einlaufdicke, auslaufdicke):
    # Gibt den max. Greifwinkel alpha_0 zurück, der benötigt wird um das Walzgut einzuziehen.

# Berechnung().maximale_dickenabnahme():
    # Gibt die maximale Dickenabnahme delta_h_max des Walzgutes zurück.

# Berechnung().geruestmodul(all_data):
    # Gibt den Gerüstmodul C des Walzwerkes zurück. Dabei wird der Datensatz "all_data", die alle Sensordaten enthalten,
    # übergeben und daraus das Gerüstmodul C berechnet.

# ----------------------------------------------------------------------------------------------------------------------

class Vorberechnung():
    def __init__(self, walzendurchmesser, ballenlaenge, e_modul, kraft):
        self.d = walzendurchmesser                      # Walzendruchmesser [mm]
        self.l = ballenlaenge                           # Ballenlänge der Walze [mm]
        self.E = e_modul                                # E-Modul des Walzenwerkstoffes (Werkzeugstahl)[N/mm²]
        self.F = kraft                                  # Walzkraft [N]

    def durchbiegung(self):
        I = self.d**4*np.pi/64                          # Flächenträgheitsmoment der Walze [mm^4]
        w_max = (self.F*self.l**3)/(48*self.E*I)        # Maximale Durchbiegung in der Mitte der Walze [mm]
        w_max = round(w_max, 4)
        print("Max. Durchbiegung in z = " + str(w_max) + " mm (= " + str(w_max*10**3) + " \u03BCm)")

        return w_max


class Berechnung():
    def __init__(self):
        self.r = 203/2                                  # Walzenradius [mm]
        self.reibungskoeffizient = 0.2                  # Reibungskoeffizient zwischen Walzgut und Walze [-]

    def gedrueckte_lange(self, einlaufdicke, auslaufdicke):
        self.h_0 = einlaufdicke                         # Einlaufdicke des Walzgutes [mm]
        self.h_1 = auslaufdicke                         # Auslaufdicke des Walzgutes [mm]

        delta_h = self.h_0 - self.h_1                   # Höhenabnahme des Walzgutes [mm]
        l_d = np.sqrt(self.r*delta_h - delta_h**2/4)    # Gedrückte Länge [mm]
        print("Gedrückte Länge l_d = " + str(round(l_d,3)) + "mm")

        return l_d

    def greifwinkel(self, einlaufdicke, auslaufdicke):
        self.h_0 = einlaufdicke                                 # Einlaufdicke des Walzgutes [mm]
        self.h_1 = auslaufdicke                                 # Auslaufdicke des Walzgutes [mm]

        l_d = Berechnung().gedrueckte_lange(self.h_0, self.h_1)     # Gedrückte Länge [mm]
        alpha_0 = np.arctan(l_d/self.r)*180/np.pi                   # Greifwinkel [°]
        print("Maximaler Greifwinkel alpha_0 = " + str(round(alpha_0, 3)) + "°")

        return alpha_0

    def maximale_dickenabnahme(self):
        delta_h_max = self.reibungskoeffizient**2*self.r            # Maximale Dickenabnahme des Walzgutes [mm]
        print("Maximale Dickenabnahme delta_h_max = " + str(round(delta_h_max, 3)) + "mm")

        return delta_h_max
```

```
74
75      def geruestmodul(self, all_data):
76          self.all_data = all_data
77          s = []                                          # Walzspalthöhe [mm]
78          F_ges = []                                      # Walzkraft [N]
79
80          for line in self.all_data:
81              s.append(float(line[1]))
82              F_ges.append(float(line[2]) + float(line[3]))
83
84          # Polynom durch Datenpunkte von s und F ges:
85          poly_coeff = np.polyfit(s, F_ges, 1)            # Grad = 1, linear
86          s_new = np.linspace(min(s), max(s), 100)        # Inkrementierung der Walzspalthöhe
87          F_ges_new = np.poly1d(poly_coeff)               # Gerüstkennlinie als Polynom
88
89          delta_F = F_ges_new[1]                          # Steigung der Gerüstkennlinie [-]
90          delta_s = max(s) - min(s)                       # Differenz der min. und max. Walzspalthöhen [mm]
91          C = delta_F*10**-3/delta_s                      # Gerüstmodul [kN/mm]
92
93          #print(F_ges_new)
94          #print(delta_F)
95          #print(delta_s)
96          print("Gerüstmodul C = " + str(round(C, 3)) + "kN/mm")
97          print("Auffederung 1/C = " + str(round(1/C, 3)) + "mm/kN")
98
99          return C
100
```

Anhang C6: Quellcode „diagramme.py“

```
import numpy as np
import matplotlib.pyplot as plt
from module import zeitschritte

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 06.09.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Standardformat().diagramm(self, x_data, y_data, title, x_label, y_label, legend_label):
    # Definiert die Standardformatierung (z.B. Liniefarbe, Achsenbeschriftung) eines Diagrammes.

# Diagramme(all_data).diagramm_kraft():
    # Liefert je ein Diagramm der beiden Kraftmessdosen.

# Diagramme(all_data).diagramm_handradwinkel():
    # Liefert ein Diagramm des gemessenen Handradwinkels.

# Diagramme(all_data).diagramm_all():
    # Liefert ein Diagramm der beiden Kraftmessdosen, der Größe des Walzspaltes und des Handradwinkels/Einstellwinkels.

# Geruestkennlinie(all_data).geruestkennlinie():
    # Liefert ein Diagramm der Gerüstkennline (Walzspaltgröße bzw. Auffederung über Walzkraft).

# ----------------------------------------------------------------------------------------------------------------------

# Dateneingang:
#             |Zeit|Auffed|F.li|f.re|Winkel|
#                0    1    2    3    4

class Standardformat():
    def __init__(self):
        pass

    def diagramm(self, x_data, y_data, title, x_label, y_label, legend_label):
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)

        ax.set_axisbelow(True)
        ax.xaxis.grid(color='gray', linestyle='dashed')
        ax.yaxis.grid(color='gray', linestyle='dashed')

        plt.title(title)
        plt.xlabel(x_label)
        plt.ylabel(y_label)

        plt.scatter(x_data, y_data, color='r', zorder=1)
        plt.plot(x_data, y_data, color='b', zorder=2, label=legend_label)
        plt.legend(loc="upper left")


class Diagramme():
    def __init__(self, all_data):
        self.all_data = all_data

     def diagramm_kraft(self):
         t = []
         F_li = []
         F_re = []
         for line in self.all_data:
             t.append(line[0])
             F_li.append(line[2])
             F_re.append(line[3])

         t_arr = np.array(t, dtype=object)             # Numpy-Array der Zeitschritte
         F_li_arr = np.array(F_li, dtype=object)       # Numpy-Array der Kräfte links
         F_re_arr = np.array(F_re, dtype=object)       # Numpy-Array der Kräfte rechts
         bool_arr_F_li = F_li_arr > 0.01               # Int-Bool um Datenpunkte kleiner 0.01 zu filtern
         bool_arr_F_re = F_re_arr > 0.01               # Int-Bool um Datenpunkte kleiner 0.01 zu filtern

         #print(len(t_arr[bool_arr_F_li]), len(F_li_arr[bool_arr_F_li]))
         #print(len(t_arr[bool_arr_F_re]), len(F_re_arr[bool_arr_F_re]))

```

```
75          Standardformat().diagramm(t_arr[bool_arr_F_li], F_li_arr[bool_arr_F_li], "Walzenkraft-Zeit", "Zeit t [s]",
76                                    "Kraft F [N]", "Kraft links")
77          Standardformat().diagramm(t_arr[bool_arr_F_re], F_re_arr[bool_arr_F_re], "Walzenkraft-Zeit", "Zeit t [s]",
78                                    "Kraft F [N]", "Kraft rechts")
79          plt.show()
80
81          return [t, F_li, F_re]
82
83      def diagramm_zahnradwinkel(self):
84          t = []
85          alpha = []
86          for line in self.all_data:
87              t.append(line[0])
88              alpha.append(line[4])
89
90          d_1 = 264   # Durchmesser des Zahnradkontaktkreises mit dem Reibrad     [mm]
91          d_2 = 52    # Durchmesser des Reibrades                                 [mm]
92          z_1 = 58    # Zähnezahl des Zahnrades zur Walzspaltverstellung          [-]
93          z_2 = 13    # Zähnetahl des Zahnrades des Handrades                     [-]
94          i = d_1/d_2 # Übersetzung zwischen Zahnradkontaktkreis und Reibrad      [-]
95
96          t_arr = np.array(t, dtype=object)               # Numpy-Array der Zeitschritte
97          alpha_arr = np.array(alpha, dtype=object)     # Numpy-Array des Zahnradwinkels
98          bool_arr_alpha = alpha_arr > 0.1  # Int-Bool um Datenpunkte kleiner 1 zu filtern
99
100         Standardformat().diagramm(t_arr[bool_arr_alpha], alpha_arr[bool_arr_alpha]/i, "Zahnradwinkel-Zeit",
101                                   "Zeit t [s]", "Zahnradwinkel [°]", "Handradwinkel")
102         plt.show()
103
104         return [t, alpha]
105
106
107     def diagramm_all(self):
108         # Kräfte
109         fig = plt.figure()
110         ax1 = fig.add_subplot(2,1,1)
111         ax1.set_axisbelow(True)
112         ax1.xaxis.grid(color='gray', linestyle='dashed')
113         ax1.yaxis.grid(color='gray', linestyle='dashed')
114         ax1.set_title("Walzenkraft-Zeit")
115         ax1.set_xlabel("Zeit t [s]")
116         ax1.set_ylabel("Kraft F [N]")
117
118         t = []
119         F_li = []
120         F_re = []
121         for line in self.all_data:
122             t.append(line[0])
123             F_li.append(line[2])
124             F_re.append(line[3])
125
126         t_arr = np.array(t, dtype=object)             # Numpy-Array der Zeitschritte
127         F_li_arr = np.array(F_li, dtype=object)       # Numpy-Array der Kräfte links
128         F_re_arr = np.array(F_re, dtype=object)       # Numpy-Array der Kräfte rechts
129         bool_arr_F_li = F_li_arr > 1                  # Int-Bool um Datenpunkte kleiner 1 zu filtern
130         bool_arr_F_re = F_re_arr > 1                  # Int-Bool um Datenpunkte kleiner 1 zu filtern
131
132         ax1.scatter(t_arr[bool_arr_F_li], F_li_arr[bool_arr_F_li], color='r', zorder=1)
133         ax1.plot(t_arr[bool_arr_F_li], F_li_arr[bool_arr_F_li], color='b', zorder=2, label="Kraft links")
134         ax1.scatter(t_arr[bool_arr_F_re], F_re_arr[bool_arr_F_re], color='r', zorder=1)
135         ax1.plot(t_arr[bool_arr_F_re], F_re_arr[bool_arr_F_re], color='orange', zorder=2, label="Kraft rechts")
136         ax1.legend(loc="upper left")
137
138         # Walzspalthöhe:
139         ax2 = fig.add_subplot(2,2,3)
140         ax2.set_axisbelow(True)
141         ax2.xaxis.grid(color='gray', linestyle='dashed')
142         ax2.yaxis.grid(color='gray', linestyle='dashed')
143         ax2.set_title("Walzspalthöhe-Zeit")
144         ax2.set_xlabel("Zeit t [s]")
145         ax2.set_ylabel("Walzspalthöhe s [mm]")
146
```

```
147         s = []
148         for line in self.all_data:
149             s.append(line[1])
150 
151         s_arr = np.array(s, dtype=object)       # Numpy-Array des Zahnradwinkels
152         bool_arr_s = s_arr > 0.1    # Int-Bool um Datenpunkte kleiner 0.1 zu filtern
153 
154         ax2.scatter(t_arr[bool_arr_s], s_arr[bool_arr_s], color='r', zorder=1)
155         ax2.plot(t_arr[bool_arr_s], s_arr[bool_arr_s], color='b', zorder=2, label="Walzspalthöhe")
156         ax2.legend(loc="upper left")
157 
158         # Zahnradwinkel:
159         ax3 = fig.add_subplot(2,2,4)
160         ax3.set_axisbelow(True)
161         ax3.xaxis.grid(color='gray', linestyle='dashed')
162         ax3.yaxis.grid(color='gray', linestyle='dashed')
163         ax3.set_title("Zahnradwinkel-Zeit")
164         ax3.set_xlabel("Zeit t [s]")
165         ax3.set_ylabel("Zahnradwinkel [°]")
166 
167         alpha = []
168         for line in self.all_data:
169             alpha.append(line[4])
170 
171         alpha_arr = np.array(alpha, dtype=object)       # Numpy-Array des Zahnradwinkels
172         bool_arr_alpha = alpha_arr > 0.1                # Int-Bool um Datenpunkte kleiner 1 zu filtern
173 
174         ax3.scatter(t_arr[bool_arr_alpha], alpha_arr[bool_arr_alpha], color='r', zorder=1)
175         ax3.plot(t_arr[bool_arr_alpha], alpha_arr[bool_arr_alpha], color='b', zorder=2, label="Zahnradwinkel")
176         ax3.legend(loc="upper left")
177         plt.show()
178 
179         return [t, F_li, F_re, s, alpha]
180 
181 
182 class Geruestkennlinie():
183     def __init__(self, all_data):
184         self.all_data = all_data
185 
186     def geruestkennlinie(self):
187         s = []
188         F_ges = []
189 
190         for line in self.all_data:
191             s.append(line[1])
192             F_ges.append(line[2]+line[3])
193 
194         s_arr = np.array(s)          # Numpy-Array des Zahnradwinkels
195         F_ges_arr = np.array(F_ges)
196         bool_arr = s_arr > 0.003     # Int-Bool um Datenpunkte kleiner 0.003 zu filtern
197 
198         # Polynom durch Datenpunkte von s und F_ges:
199         poly_coeff = np.polyfit(s_arr[bool_arr], F_ges_arr[bool_arr], 1)        # Grad = 1, anpassbar
200         s_new = np.linspace(min(s_arr[bool_arr]), max(s_arr[bool_arr]), 100)
201         F_ges_new = np.poly1d(poly_coeff)
202 
203         fig = plt.figure()
204         ax = fig.add_subplot(1, 1, 1)
205         ax.set_axisbelow(True)
206         ax.xaxis.grid(color='gray', linestyle='dashed')
207         ax.yaxis.grid(color='gray', linestyle='dashed')
208 
209         plt.title("Gerüstkennlinie")
210         plt.xlabel("Walzspalt s [mm]")
211         plt.ylabel("Walzkraft F_ges [N]")
212         plt.scatter(s, F_ges, color='r', zorder=1)
213         plt.plot(s_new, F_ges_new(s_new), color='b', zorder=2, label="Gerüstkennlinie")
214         plt.legend(loc="upper left")
215         plt.show()
216 
217         return [s, F_ges]
218 
```

Anhang C7: Quellcode „walzspaltzustellung.py“

```python
import numpy as np

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 15.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Zustellung().get_auffederung(all_data):
    # Gibt aus dem Datensatz "all_data", welcher alle Sensordaten enthält, die Walzspaltöffnung s_0 und die Auslaufdicke
    # h_1 des Walzugtes zurück.

# # Zustellung().get_auffederung(all_data):
    # Gibt über eine Umrechnung von Maschinenparametern (Übersetzung) die aufgrund der Auffederung notwendige
    # Nachstellung in Grad und Umdrehungen zurück.

# ----------------------------------------------------------------------------------------------------------------------

class Zustellung():
    def __init__(self, all_data):
        self.all_data = all_data

    def get_auffederung(self):
        s_0 = np.inf                    # Walzspaltöffnung des Gerüsts [mm]
        h_1 = 0                         # Auslaufdicke des Walzgutes bzw. Walzspalt mit Auffederung [mm]
        for line in self.all_data:
            if line[1] <= s_0 and line[1] > 0.001:
                s_0 = line[1]
            elif line[1] > h_1:
                h_1 = line[1]
            else:
                pass
        #print(s_0, h_1)

        return s_0, h_1

    def zustellung(self):
        # 360 Grad Drehung des Handrades entsprechen 12,5 Zähnen, 1 Zahn enstpricht 0,07mm Zustellung
        s_0, h_1 = Zustellung(self.all_data).get_auffederung()
        delta_h = h_1 - s_0             # Zustellung [mm]
        z_360 = 12.5                    # Anzahl der Zähne bei 360° Drehung

        z = delta_h*1/0.07              # Anzahl Zähne zur Zustellung um delta_h (1 Zahn = 0.07mm)
        alpha = round(360*z/z_360, 2)   # Notwendige Drehung [°] zur Zustellung um delta_h
        umdrehungen = round(alpha*np.pi/180, 2)

        print("Notwendige Drehung der Zustellung: " + str(alpha) + "° (entspricht " + str(umdrehungen) + " Umdrehungen)")

        return alpha
```

Anhang C8: Quellcode „diagramme_gui.py“

```python
import numpy as np
import matplotlib.pyplot as plt
from module import zeitschritte

# ----------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: 23.07.2020
# Version: PyCharm Community Edition 2019.3.4

# Beschreibung:
# Diagramme(all_data).kraft():
    # Liefert die Daten für die Visualisierung der beiden Kraftmessdosen an das GUI.

# Diagramme(all_data).handradwinkel():
    # Liefert die Daten für die Visualisierung des Handradwinkels an das GUI.

# Diagramme(all_data).auffederung():
    # Liefert die Daten für die Visualisierung der Auffederung an das GUI.

# Geruestkennlinie(all_data).geruestkennlinie():
    # Liefert ein Daten für die Visualisierung der Gerüstkennline (Walzspaltgröße bzw. Auffederung über Walzkraft).

# ----------------------------------------------------------------------------------------------------------------------

# Dateneingang:
#             |Zeit|Auffed|F.li|f.re|Winkel|
#                0    1    2    3    4

class Diagramme():
    def __init__(self, all_data):
        self.all_data = all_data

    def kraft(self):
        t = []
        F_li = []
        F_re = []
        for line in self.all_data:
            t.append(line[0])
            F_li.append(line[2])
            F_re.append(line[3])

        t_arr = np.array(t, dtype=object)             # Numpy-Array der Zeitschritte
        F_li_arr = np.array(F_li, dtype=object)       # Numpy-Array der Kräfte links
        F_re_arr = np.array(F_re, dtype=object)       # Numpy-Array der Kräfte rechts
        bool_arr_F_li = F_li_arr > 0.01  # Int-Bool um Datenpunkte kleiner 1 zu filtern
        bool_arr_F_re = F_re_arr > 0.01  # Int-Bool um Datenpunkte kleiner 1 zu filtern

        return t_arr[bool_arr_F_li], F_li_arr[bool_arr_F_li], t_arr[bool_arr_F_re], F_re_arr[bool_arr_F_re]

    def zahnradwinkel(self):
        t = []
        alpha = []
        for line in self.all_data:
            t.append(line[0])
            alpha.append(line[4])

        d_1 = 264   # Durchmesser des Zahnradkontaktkreises mit dem Reibrad     [mm]
        d_2 = 52    # Durchmesser des Reibrades                                 [mm]
        z_1 = 58    # Zähnezahl des Zahnrades zur Walzspaltverstellung          [-]
        z_2 = 13    # Zähnetahl des Zahnrades des Handrades                     [-]
        i = d_1/d_2 # Übersetzung zwischen Zahnradkontaktkreis und Reibrad      [-]

        t_arr = np.array(t, dtype=object)  # Numpy-Array der Zeitschritte
        alpha_arr = np.array(alpha, dtype=object)  # Numpy-Array des Zahnradwinkels
        bool_arr_alpha = alpha_arr > 0.1  # Int-Bool um Datenpunkte kleiner 1 zu filtern

        return t_arr[bool_arr_alpha], alpha_arr[bool_arr_alpha]/i

    def auffederung(self):
        t = []
        s = []
        for line in self.all_data:
            t.append(line[0])
            s.append(line[1])

```

```
76          t_arr = np.array(t, dtype=object)  # Numpy-Array der Zeitschritte
77          s_arr = np.array(s, dtype=object)       # Numpy-Array des Zahnradwinkels
78          bool_arr_s = s_arr > 0.1    # Int-Bool um Datenpunkte kleiner 0.1 zu filtern
79
80          return t_arr[bool_arr_s], s_arr[bool_arr_s]
81
82
83  class Geruestkennlinie():
84      def __init__(self, all_data):
85          self.all_data = all_data
86
87      def geruestkennlinie(self):
88          s = []
89          F_ges = []
90
91          for line in self.all_data:
92              s.append(line[1])
93              F_ges.append(line[2]+line[3])
94
95          s_arr = np.array(s)  # Numpy-Array des Zahnradwinkels
96          F_ges_arr = np.array(F_ges)
97          bool_arr = s_arr > 0.003  # Int-Bool um Datenpunkte kleiner 0.001 zu filtern
98
99          # Polynom durch Datenpunkte von s und F_ges:
100         poly_coeff = np.polyfit(s_arr[bool_arr], F_ges_arr[bool_arr], 1)        # Grad = 1, anpassbar
101         s_new = np.linspace(min(s_arr[bool_arr]), max(s_arr[bool_arr]), 100)
102         F_ges_new = np.poly1d(poly_coeff)
103
104         fig = plt.figure()
105         ax = fig.add_subplot(1, 1, 1)
106         ax.set_axisbelow(True)
107         ax.xaxis.grid(color='gray', linestyle='dashed')
108         ax.yaxis.grid(color='gray', linestyle='dashed')
109
110         plt.title("Gerüstkennlinie")
111         plt.xlabel("Walzspalt s [mm]")
112         plt.ylabel("Walzkraft F_ges [N]")
113         plt.scatter(s, F_ges, color='r', zorder=1)
114         plt.plot(s_new, F_ges_new(s_new), color='b', zorder=2, label="Gerüstkennlinie")
115         plt.legend(loc="upper left")
116         #plt.show()
117
118         return s_arr[bool_arr], F_ges_arr[bool_arr], s_new, F_ges_new
119
```

Anhang C9: Quellcode „build.py“ (GUI)

```
1    from qtpy import uic
2
3    uic.compileUiDir("gui")
```

Anhang C10: Quellcode „mainwindow.py“ (GUI)

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'gui\mainwindow.ui'
#
# Created by: PyQt5 UI code generator 5.15.0
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1366, 768)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)
        self.gridLayout.setObjectName("gridLayout")
        spacerItem = QtWidgets.QSpacerItem(2050, 20, QtWidgets.QSizePolicy.Ignored, QtWidgets.QSizePolicy.Minimum)
        self.gridLayout.addItem(spacerItem, 1, 2, 1, 1)
        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setObjectName("verticalLayout")
        self.label = QtWidgets.QLabel(self.centralwidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.verticalLayout.addWidget(self.label)
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        self.pfad = QtWidgets.QLabel(self.centralwidget)
        font = QtGui.QFont()
        font.setPointSize(13)
        self.pfad.setFont(font)
        self.pfad.setScaledContents(True)
        self.pfad.setObjectName("pfad")
        self.horizontalLayout_2.addWidget(self.pfad)
        self.verticalLayout.addLayout(self.horizontalLayout_2)
        self.projektname = QtWidgets.QLineEdit(self.centralwidget)
        self.projektname.setEnabled(True)
        font = QtGui.QFont()
        font.setPointSize(13)
        self.projektname.setFont(font)
        self.projektname.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)
        self.projektname.setInputMethodHints(QtCore.Qt.ImhNone)
        self.projektname.setText("")
        self.projektname.setAlignment(QtCore.Qt.AlignCenter)
        self.projektname.setObjectName("projektname")
        self.verticalLayout.addWidget(self.projektname)
        spacerItem1 = QtWidgets.QSpacerItem(20, 100, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Ignored)
        self.verticalLayout.addItem(spacerItem1)
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label 2.setEnabled(True)
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")
        self.verticalLayout.addWidget(self.label_2)
        self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_3.setObjectName("horizontalLayout_3")
        self.einlaufdicke_label = QtWidgets.QLabel(self.centralwidget)
        font = QtGui.QFont()
        font.setPointSize(13)
        self.einlaufdicke_label.setFont(font)
        self.einlaufdicke_label.setScaledContents(True)
```

```
73          self.einlaufdicke_label.setObjectName("einlaufdicke_label")
74          self.horizontalLayout_3.addWidget(self.einlaufdicke_label)
75          spacerItem2 = QtWidgets.QSpacerItem(85, 20, QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Minimum)
76          self.horizontalLayout_3.addItem(spacerItem2)
77          self.einlaufdicke = QtWidgets.QLineEdit(self.centralwidget)
78          font = QtGui.QFont()
79          font.setPointSize(13)
80          self.einlaufdicke.setFont(font)
81          self.einlaufdicke.setText("")
82          self.einlaufdicke.setAlignment(QtCore.Qt.AlignCenter)
83          self.einlaufdicke.setObjectName("einlaufdicke")
84          self.horizontalLayout_3.addWidget(self.einlaufdicke)
85          self.einlaufdicke_einheit = QtWidgets.QLabel(self.centralwidget)
86          font = QtGui.QFont()
87          font.setPointSize(13)
88          self.einlaufdicke_einheit.setFont(font)
89          self.einlaufdicke_einheit.setObjectName("einlaufdicke_einheit")
90          self.horizontalLayout_3.addWidget(self.einlaufdicke_einheit)
91          self.verticalLayout.addLayout(self.horizontalLayout_3)
92          self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
93          self.horizontalLayout_4.setObjectName("horizontalLayout_4")
94          self.auslaufdicke_label = QtWidgets.QLabel(self.centralwidget)
95          font = QtGui.QFont()
96          font.setPointSize(13)
97          self.auslaufdicke_label.setFont(font)
98          self.auslaufdicke_label.setScaledContents(True)
99          self.auslaufdicke_label.setObjectName("auslaufdicke_label")
100         self.horizontalLayout_4.addWidget(self.auslaufdicke_label)
101         spacerItem3 = QtWidgets.QSpacerItem(75, 20, QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Minimum)
102         self.horizontalLayout_4.addItem(spacerItem3)
103         self.auslaufdicke = QtWidgets.QLineEdit(self.centralwidget)
104         font = QtGui.QFont()
105         font.setPointSize(13)
106         self.auslaufdicke.setFont(font)
107         self.auslaufdicke.setText("")
108         self.auslaufdicke.setAlignment(QtCore.Qt.AlignCenter)
109         self.auslaufdicke.setObjectName("auslaufdicke")
110         self.horizontalLayout_4.addWidget(self.auslaufdicke)
111         self.auslaufdicke_einheit = QtWidgets.QLabel(self.centralwidget)
112         font = QtGui.QFont()
113         font.setPointSize(13)
114         self.auslaufdicke_einheit.setFont(font)
115         self.auslaufdicke_einheit.setObjectName("auslaufdicke_einheit")
116         self.horizontalLayout_4.addWidget(self.auslaufdicke_einheit)
117         self.verticalLayout.addLayout(self.horizontalLayout_4)
118         spacerItem4 = QtWidgets.QSpacerItem(20, 100, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Ignored)
119         self.verticalLayout.addItem(spacerItem4)
120         self.label_3 = QtWidgets.QLabel(self.centralwidget)
121         font = QtGui.QFont()
122         font.setPointSize(16)
123         font.setBold(True)
124         font.setWeight(75)
125         self.label_3.setFont(font)
126         self.label_3.setObjectName("label_3")
127         self.verticalLayout.addWidget(self.label_3)
128         self.horizontalLayout = QtWidgets.QHBoxLayout()
129         self.horizontalLayout.setObjectName("horizontalLayout")
130         self.gedrueckte_laenge_Label = QtWidgets.QLabel(self.centralwidget)
131         self.gedrueckte_laenge_Label.setEnabled(True)
132         font = QtGui.QFont()
133         font.setPointSize(13)
134         self.gedrueckte_laenge_Label.setFont(font)
135         self.gedrueckte_laenge_Label.setScaledContents(True)
136         self.gedrueckte_laenge_Label.setObjectName("gedrueckte_laenge_Label")
137         self.horizontalLayout.addWidget(self.gedrueckte_laenge_Label)
138         spacerItem5 = QtWidgets.QSpacerItem(30, 20, QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Minimum)
139         self.horizontalLayout.addItem(spacerItem5)
140         self.gedrueckte_laenge = QtWidgets.QLineEdit(self.centralwidget)
141         font = QtGui.QFont()
142         font.setPointSize(13)
143         self.gedrueckte_laenge.setFont(font)
144         self.gedrueckte_laenge.setText("")
145         self.gedrueckte_laenge.setAlignment(QtCore.Qt.AlignCenter)
146         self.gedrueckte_laenge.setObjectName("gedrueckte_laenge")
147         self.horizontalLayout.addWidget(self.gedrueckte_laenge)
148         self.gedrueckte_laenge_Einheit = QtWidgets.QLabel(self.centralwidget)
```

```
149         font = QtGui.QFont()
150         font.setPointSize(13)
151         self.gedrueckte_laenge_Einheit.setFont(font)
152         self.gedrueckte_laenge_Einheit.setObjectName("gedrueckte_laenge_Einheit")
153         self.horizontalLayout.addWidget(self.gedrueckte_laenge_Einheit)
154         self.verticalLayout.addLayout(self.horizontalLayout)
155         self.horizontalLayout_7 = QtWidgets.QHBoxLayout()
156         self.horizontalLayout_7.setObjectName("horizontalLayout_7")
157         self.handradwinkel_label = QtWidgets.QLabel(self.centralwidget)
158         font = QtGui.QFont()
159         font.setPointSize(13)
160         self.handradwinkel_label.setFont(font)
161         self.handradwinkel_label.setScaledContents(True)
162         self.handradwinkel_label.setObjectName("handradwinkel_label")
163         self.horizontalLayout_7.addWidget(self.handradwinkel_label)
164         spacerItem6 = QtWidgets.QSpacerItem(120, 20, QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Minimum)
165         self.horizontalLayout_7.addItem(spacerItem6)
166         self.gerustmodul = QtWidgets.QLineEdit(self.centralwidget)
167         font = QtGui.QFont()
168         font.setPointSize(13)
169         self.gerustmodul.setFont(font)
170         self.gerustmodul.setText("")
171         self.gerustmodul.setAlignment(QtCore.Qt.AlignCenter)
172         self.gerustmodul.setObjectName("gerustmodul")
173         self.horizontalLayout_7.addWidget(self.gerustmodul)
174         self.gerustmodul_einheit = QtWidgets.QLabel(self.centralwidget)
175         font = QtGui.QFont()
176         font.setPointSize(13)
177         self.gerustmodul_einheit.setFont(font)
178         self.gerustmodul_einheit.setObjectName("gerustmodul_einheit")
179         self.horizontalLayout_7.addWidget(self.gerustmodul_einheit)
180         self.verticalLayout.addLayout(self.horizontalLayout_7)
181         self.horizontalLayout_8 = QtWidgets.QHBoxLayout()
182         self.horizontalLayout_8.setObjectName("horizontalLayout_8")
183         self.dickenabnahme_label = QtWidgets.QLabel(self.centralwidget)
184         font = QtGui.QFont()
185         font.setPointSize(13)
186         self.dickenabnahme_label.setFont(font)
187         self.dickenabnahme_label.setScaledContents(True)
188         self.dickenabnahme_label.setObjectName("dickenabnahme_label")
189         self.horizontalLayout_8.addWidget(self.dickenabnahme_label)
190         spacerItem7 = QtWidgets.QSpacerItem(4, 20, QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Minimum)
191         self.horizontalLayout_8.addItem(spacerItem7)
192         self.dickenabnahme = QtWidgets.QLineEdit(self.centralwidget)
193         font = QtGui.QFont()
194         font.setPointSize(13)
195         self.dickenabnahme.setFont(font)
196         self.dickenabnahme.setText("")
197         self.dickenabnahme.setAlignment(QtCore.Qt.AlignCenter)
198         self.dickenabnahme.setObjectName("dickenabnahme")
199         self.horizontalLayout_8.addWidget(self.dickenabnahme)
200         self.dickenabnahme_einheit = QtWidgets.QLabel(self.centralwidget)
201         font = QtGui.QFont()
202         font.setPointSize(13)
203         self.dickenabnahme_einheit.setFont(font)
204         self.dickenabnahme_einheit.setObjectName("dickenabnahme_einheit")
205         self.horizontalLayout_8.addWidget(self.dickenabnahme_einheit)
206         self.verticalLayout.addLayout(self.horizontalLayout_8)
207         spacerItem8 = QtWidgets.QSpacerItem(20, 400, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Preferred)
208         self.verticalLayout.addItem(spacerItem8)
209         self.horizontalLayout_9 = QtWidgets.QHBoxLayout()
210         self.horizontalLayout_9.setObjectName("horizontalLayout_9")
211         self.berechnung = QtWidgets.QPushButton(self.centralwidget)
212         font = QtGui.QFont()
213         font.setPointSize(16)
214         self.berechnung.setFont(font)
215         self.berechnung.setObjectName("berechnung")
216         self.horizontalLayout_9.addWidget(self.berechnung)
217         self.visualisierung = QtWidgets.QPushButton(self.centralwidget)
218         font = QtGui.QFont()
219         font.setPointSize(16)
220         self.visualisierung.setFont(font)
221         self.visualisierung.setObjectName("visualisierung")
222         self.horizontalLayout_9.addWidget(self.visualisierung)
223         self.verticalLayout.addLayout(self.horizontalLayout_9)
224         self.datenexport = QtWidgets.QPushButton(self.centralwidget)
```

```
224         self.datenexport = QtWidgets.QPushButton(self.centralwidget)
225         font = QtGui.QFont()
226         font.setPointSize(16)
227         self.datenexport.setFont(font)
228         self.datenexport.setObjectName("datenexport")
229         self.verticalLayout.addWidget(self.datenexport)
230         self.gridLayout.addLayout(self.verticalLayout, 1, 1, 1, 1)
231         spacerItem9 = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Ignored)
232         self.gridLayout.addItem(spacerItem9, 0, 0, 1, 1)
233         spacerItem10 = QtWidgets.QSpacerItem(50, 20, QtWidgets.QSizePolicy.Ignored, QtWidgets.QSizePolicy.Minimum)
234         self.gridLayout.addItem(spacerItem10, 1, 0, 1, 1)
235         spacerItem11 = QtWidgets.QSpacerItem(20, 50, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Ignored)
236         self.gridLayout.addItem(spacerItem11, 2, 0, 1, 1)
237         MainWindow.setCentralWidget(self.centralwidget)
238         self.menubar = QtWidgets.QMenuBar(MainWindow)
239         self.menubar.setGeometry(QtCore.QRect(0, 0, 1366, 21))
240         self.menubar.setObjectName("menubar")
241         MainWindow.setMenuBar(self.menubar)
242         self.statusbar = QtWidgets.QStatusBar(MainWindow)
243         self.statusbar.setObjectName("statusbar")
244         MainWindow.setStatusBar(self.statusbar)
245 
246         self.retranslateUi(MainWindow)
247         QtCore.QMetaObject.connectSlotsByName(MainWindow)
248 
249     def retranslateUi(self, MainWindow):
250         _translate = QtCore.QCoreApplication.translate
251         MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
252         self.label.setText(_translate("MainWindow", "Projekt"))
253         self.pfad.setText(_translate("MainWindow", "pfad"))
254         self.label_2.setText(_translate("MainWindow", "Walzparameter"))
255         self.einlaufdicke_label.setText(_translate("MainWindow", "Einlaufdicke h_0"))
256         self.einlaufdicke_einheit.setText(_translate("MainWindow", "mm     "))
257         self.auslaufdicke_label.setText(_translate("MainWindow", "Auslaufdicke h_1"))
258         self.auslaufdicke_einheit.setText(_translate("MainWindow", "mm     "))
259         self.label_3.setText(_translate("MainWindow", "Berechnung"))
260         self.gedrueckte_laenge_Label.setText(_translate("MainWindow", "Gedrückte Länge l_d"))
261         self.gedrueckte_laenge_Einheit.setText(_translate("MainWindow", "mm     "))
262         self.handradwinkel_label.setText(_translate("MainWindow", "Gerüstmodul C"))
263         self.gerustmodul_einheit.setText(_translate("MainWindow", "kN/mm"))
264         self.dickenabnahme_label.setText(_translate("MainWindow", "Max. Dickenabnahme "))
265         self.dickenabnahme_einheit.setText(_translate("MainWindow", "mm     "))
266         self.berechnung.setText(_translate("MainWindow", "Berechnung"))
267         self.visualisierung.setText(_translate("MainWindow", "Visualisierung"))
268         self.datenexport.setText(_translate("MainWindow", "Daten exportieren"))
269 
```

Anhang C11: Quellcode „main.py“ (GUI)

```
import sys
import os
import subprocess
from qtpy import QtWidgets
from gui.mainwindow import Ui_MainWindow
from module import berechnung, diagramme_gui, sensordata, neues_projekt, walzspaltzustellung, zeitschritte
import matplotlib
matplotlib.use('Qt5Agg')
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from matplotlib.figure import Figure
from PyQt5.QtWidgets import QApplication, QMainWindow, QMenu, QVBoxLayout, QSizePolicy, QMessageBox, QWidget, QPushButton, QAction, QLineEdit, QI
from PyQt5.QtGui import QIcon
import random
import numpy as np

# --------------------------------------------------------------------------------------------------------------------
# Name: Marcel Sorger
# Datum der letzten Änderung: .07.2020
# Version: PyCharm Community Edition 2019.3.4, Qt Creator 4.11.2 (Community)

# Anmerkung:
# "build.py" ausführen um die Dateien im Ordner "gui" umzuwandeln, damit diese von "main.py" verwendet werden können.

# Beschreibung:
# Visualisierung():
    # Initialisiert alle Diagramme des GUI mit den aktuellen Daten.

# MainWindow().update_plot():
    # Intialisiert alle Diagramme erneut um sie im GUI zu aktualisieren.

# MainWindow().daten_exportieren():
    # Exportiert die im GUI angezeigten Daten in den vom Benutzer benannten Projektordner.

# MainWindow().auswertung():
    # Berechnet die gedrückte Länge, den Gerüstmodul und die maximale Dickenabnahme mit den im GUI vom Benutzer
    # eingegebenen Einlauf- und Auslaufhöhe.

# --------------------------------------------------------------------------------------------------------------------

class Visualisierung(FigureCanvasQTAgg):
    def __init__(self, parent = None):
        width, height = 21, 21.5
        fig = Figure(figsize=(width, height), dpi=100)
        FigureCanvasQTAgg.__init__(self, fig)
        self.setParent(parent)
        FigureCanvasQTAgg.setSizePolicy(self, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)
        FigureCanvasQTAgg.updateGeometry(self)
        self.move(1000, -150)
        self.fig = fig

        self.plot()

    def plot(self):
        global all_data
        all_data = sensordata.CsvReader().read()
        size_title = 25
        size_labels = 18
        size_ticks = 14

        # Gerüstkennlinie:
        s, F_ges, s_new, F_ges_new = diagramme_gui.Geruestkennlinie(all_data).geruestkennlinie()
        ax1 = self.figure.add_subplot(221)                          # Erstellen eines Subplots
        ax1.scatter(s, F_ges, color='r', zorder=1)                  # Plot der Datenpunkte als Scatterplot (Punktplot)
        ax1.plot(s_new, F_ges_new(s_new), color='b', zorder=2)      # Plot des Diagrammes
        ax1.set_axisbelow(True)
        ax1.xaxis.grid(color='gray', linestyle='dashed')            # Initialisieren des x-Gitters
        ax1.yaxis.grid(color='gray', linestyle='dashed')            # Initialisieren des y-Gitters
        ax1.set_title("Gerüstkennlinie", fontsize=size_title)       # Plottitel
        ax1.set_xlabel("Walzspalt [mm]", fontsize=size_labels)      # Beschriftung der x-Achse
        ax1.set_ylabel("Walzkraft F_ges [N]", fontsize=size_labels) # Beschriftung der y-Achse
        ax1.tick_params(axis='x', labelsize=size_ticks)             # Größe der x-Achsenbeschriftung
        ax1.tick_params(axis='y', labelsize=size_ticks)             # Größe der y-Achsenbeschriftung

        # Zahnradwinkel
        t, alpha = diagramme_gui.Diagramme(all_data).zahnradwinkel()
        ax2 = self.figure.add_subplot(222)
        ax2.scatter(t, alpha, color='r', zorder=1)
        ax2.plot(t, alpha, color='b', zorder=2)
        ax2.set_axisbelow(True)
        ax2.xaxis.grid(color='gray', linestyle='dashed')
        ax2.yaxis.grid(color='gray', linestyle='dashed')
        ax2.set_title("Zahnradwinkel", fontsize=size_title)
        ax2.set_xlabel("Zeit t [s]", fontsize=size_labels)
        ax2.set_ylabel("Zahnradwinkel [°]", fontsize=size_labels)
        ax2.tick_params(axis='x', labelsize=size_ticks)
        ax2.tick_params(axis='y', labelsize=size_ticks)

```

```
88          # Walzkräfte:
89          t_li, F_li, t_re, F_re = diagramme_gui.Diagramme(all_data).kraft()
90          ax3 = self.figure.add_subplot(223)
91          ax3.scatter(t_li, F_li, color='r', zorder=1)
92          ax3.plot(t_li, F_li, color='orange', zorder=2, label="Kraftmessdose links")
93          ax3.scatter(t_re, F_re, color='r', zorder=1)
94          ax3.plot(t_re, F_re, color='b', zorder=2, label="Kraftmessdose rechts")
95          ax3.set_axisbelow(True)
96          ax3.xaxis.grid(color='gray', linestyle='dashed')
97          ax3.yaxis.grid(color='gray', linestyle='dashed')
98          ax3.set_title("Walzkräfte", fontsize=size_title)
99          ax3.set_xlabel("Zeit t [s]", fontsize=size_labels)
100         ax3.set_ylabel("Walzkraft F [N]", fontsize=size_labels)
101         ax3.tick_params(axis='x', labelsize=size_ticks)
102         ax3.tick_params(axis='y', labelsize=size_ticks)
103         ax3.legend(loc="upper left", fontsize=size_ticks)
104
105         # Auffederung:
106         t, s = diagramme_gui.Diagramme(all_data).auffederung()
107         ax4 = self.figure.add_subplot(224)
108         ax4.scatter(t, s, color='r', zorder=1)
109         ax4.plot(t, s, color='b', zorder=2)
110         ax4.set_axisbelow(True)
111         ax4.xaxis.grid(color='gray', linestyle='dashed')
112         ax4.yaxis.grid(color='gray', linestyle='dashed')
113         ax4.set_title("Auffederung", fontsize=size_title)
114         ax4.set_xlabel("Zeit t [s]", fontsize=size_labels)
115         ax4.set_ylabel("Auffederung s [mm]", fontsize=size_labels)
116         ax4.tick_params(axis='x', labelsize=size_ticks)
117         ax4.tick_params(axis='y', labelsize=size_ticks)
118
119         self.draw()
120
121
122 class MainWindow(QtWidgets.QMainWindow):
123     def __init__(self, parent = None):
124         super().__init__(parent)
125
126         self.ui = Ui_MainWindow()
127         self.ui.setupUi(self)
128
129         # Visualisierung der Daten/Visualisierung
130         Visualisierung(self)
131
132         # BUTTONS & BEFEHLE:
133         self.ui.berechnung.clicked.connect(self.auswertung)
134         self.ui.visualisierung.clicked.connect(self.update_plot)
135         self.ui.datenexport.clicked.connect(self.daten_exportieren)
136
137         # Navigationsleiste:
138         self.statusBar().showMessage('Ready')
139         mainMenu = self.menuBar()
140         mainMenu.setNativeMenuBar(False)
141         fileMenu = mainMenu.addMenu('Projekt')
142         helpMenu = mainMenu.addMenu('Hilfe')
143
144         # Menüpunkt "Projekt wählen"
145         projektauswahl =  QAction(QIcon('exit.png'), 'Projekt wählen', self)
146         projektauswahl.setShortcut('Ctrl+P')
147         projektauswahl.triggered.connect(self.projekt_auswahl)
148         fileMenu.addAction(projektauswahl)
149
150         # Menüpunkt "Schließen"
151         exitButton = QAction(QIcon('exit24.png'), 'Schließen', self)
152         exitButton.setShortcut('Ctrl+Q')
153         exitButton.setStatusTip('Exit application')
154         exitButton.triggered.connect(self.close)
155         fileMenu.addAction(exitButton)
156
157         #self.setMinimumSize(1366, 768)              # Größe des Fensters beim Start der GUI (Lenovo: 1366, 768)
158         self.setWindowTitle("GUI Walzwerk")         # Titel des Fensters
159         self.ui.pfad.setText("L:/010_LUT/Digitalisierung/Walzwerk/Versuche/...") # Ablagepfad des Projektes
160         self.showMaximized()
161
162     def update_plot(self):
163         path_to_interpreter = sys.executable
164         path_to_file = "D://Uni/Master/Master-Arbeit/Arbeit/Python Implementiert/main.py"
165
166         subprocess.call([path_to_interpreter, path_to_file])
167         sys.exit(app.exec_())
168
169     def projekt_auswahl(self):
170         file = str(QFileDialog.getExistingDirectory(self, "Projekt wählen"))
171
```

```
172     def daten_exportieren(self):
173         if self.ui.projektname.text() != "" and self.ui.projektname.text() != "Bitte Projektnamen wählen":
174             # Modul "neues_projekt"
175             projekt = str(self.ui.projektname.text())
176             neues_projekt.Neues_Projekt(projekt).anlegen()
177             #projektname = neues_projekt.Neues_Projekt.get_projektname()
178 
179             # Modul "sensordata"
180             #all_data = sensordata.CsvReader().read()  # all_data = [t1 z1 z2 z3 z4] = [t1 Auff. Kraft1 Kraft2 Winkel]
181             sensordata.CsvWriter().export_all()
182             sensordata.CsvWriter().export_summary()
183             #print(all_data)
184         else:
185             self.ui.projektname.setText("Bitte Projektnamen wählen")
186 
187     def auswertung(self):
188         def is_float(string):
189             try:
190                 float(string)
191                 return True
192             except ValueError:
193                 return False
194 
195         if (is_float(self.ui.einlaufdicke.text()) and is_float(self.ui.auslaufdicke.text())) is True and \
196                 (float(self.ui.einlaufdicke.text()) and float(self.ui.auslaufdicke.text())) > 0 and \
197                 float(self.ui.einlaufdicke.text()) > float(self.ui.auslaufdicke.text()):
198             # Modul "berechnung"
199             h_0 = round(float(self.ui.einlaufdicke.text()), 3)
200             h_1 = round(float(self.ui.auslaufdicke.text()), 3)
201 
202             l_d = round(berechnung.Berechnung().gedrueckte_lange(h_0, h_1), 3) # h_0 und h_1 noch als String erkannt
203             #alpha = round(berechnung.Berechnung().greifwinkel(h_0, h_1), 3)
204             delta_h_max = round(berechnung.Berechnung().maximale_dickenabnahme(), 3)
205             C = round(berechnung.Berechnung().geruestmodul(all_data), 3)
206 
207             self.ui.gedrueckte_laenge.setText(str(l_d))
208             self.ui.gerustmodul.setText(str(C))
209             self.ui.dickenabnahme.setText(str(delta_h_max))
210 
211             # Modul "Walzspaltzustellung"
212             delta_alpha = walzspaltzustellung.Zustellung(all_data).zustellung()
213 
214         else:
215             self.ui.einlaufdicke.setText("")
216             self.ui.auslaufdicke.setText("")
217             self.ui.gedrueckte_laenge.setText("")
218             self.ui.gerustmodul.setText("")
219             self.ui.dickenabnahme.setText("")
220 
221 
222 
223 app = QtWidgets.QApplication(sys.argv)
224 window = MainWindow()
225 window.show()
226 
227 sys.exit(app.exec_())
```

Anhang D: Datenblätter

Dem folgenden Anhang können die Datenblätter der verwendeten Hardware entnommen werden.

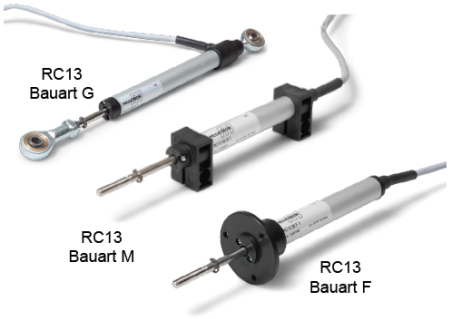
Anhang D1: Linearpotentiometer

Das Datenblatt des Linearpotentiometers Megatron RC13-75 [41] kann dem Anhang D1 entnommen werden.

Datenblatt für Wegsensoren

MEGATRON

Linearpotentiometer (Leitplastik) — **Serie RC13**



Die Serie RC13 in Schutzart IP60 wird in Applikationen eingesetzt, die eine Wegsensor mit einseitig geführter Schubstange mit Messlängen von 25 bis 250 mm benötigen. Die drei Bauarten erschließen ein breites Anwendungsfeld.

- Linearpotentiometer (Leitplastikelement) mit nahezu unendlicher Auflösung
- Messlängen von 25 mm bis 250 mm
- Hohe Lebensdauer (100 Mio. Achsbewegungen)

Die Variante mit Gelenkköpfen gleicht Bewegungen quer zur Schubstange aus, so dass auch nicht lineare Bewegungen einfach und spannungsfrei angekoppelt werden können.

Elektrische Daten	RC13-25	RC13-50	RC13-75	RC13-100	RC13-125	RC13-150	RC13-200	RC13-250
Elektrisch wirksamer Einstellweg (+1 - 0 mm) 1.)	25	50	75	100	125	150	200	250
Gesamter elektrisch Einstellweg (mm) 1.)	26 ±1	51 ±1	76 ±1	101 ±1	126 ±1	151 ±1	201 ±1	251 ±1
Gesamtwiderstand 1.)	1 kOhm	2 kOhm	3 kOhm	4 kOhm	5 kOhm	6 kOhm	8 kOhm	6 kOhm
Widerstandstoleranz	±20 %							
Unabhängige Linearität (beste Gerade) 1.)	±0,2 %	±0,1 %			±0,05 %			
Theoretische Auflösung 1.)	Nahezu unendlich							
Max. / empfohlener Schleiferstrom 1.)	1 mA (@40 °C, 1 min im Fehlerfall) / <1 µA							
Nennbelastbarkeit @40 °C (0 W @120 °C)	≤ 0,5 W	≤ 1 W	≤ 1,5 W	≤ 2 W	≤ 2,5 W	≤ 3 W		
Isolationsspannung 1.)	<100 µA @500 VAC, 1bar, 2s							
Isolationswiderstand 1.)	100 MOhm @ 500 VDC, 1bar, 2s							

Mechanische Daten, Umgebungsdaten, sonstiges	RC13-25	RC13-50	RC13-75	RC13-100	RC13-125	RC13-150	RC13-200	RC13-250
Mechanischer Einstellweg (mm) 1.)	25 +5	50 +5	75 +5	100 +5	125 +5	150 +5	200 +5	250 +5
Lebensdauer (90 % el. wirksamer Einstellweg) 2.)	>25 Mio. Meter oder 100 Mio. Bewegungen (es gilt der jeweils kleinere Wert)							
Max. Betätigungsgeschwindigkeit	< 10 m/s							
Betätigungskraft @ RT 1.) 2.)	≤ 0,5 N							
Betriebstemperaturbereich	-30..+100 °C							
Lagertemperaturbereich	-50..+120 °C							
Schutzart (IEC60529)	IP60							
Vibration (IEC 68-2-6, Test Fc)	20 g (10..2000 Hz, 0,75 mm)							
Schock (IEC 68-2-27, Test Ea)	50 g, Halbsinus, 11 ms							
Gehäuselänge Bauart M (mm)	74,5	99,5	124,5	149,5	174,5	199,5	249,5	299,5
Gehäuselänge Bauart G (mm)	102	127	152	177	202	227	277	327
Gehäuselänge Bauart F (mm)	74,5	99,5	124,5	149,5	174,5	199,5	249,5	299,5

Anhang D2: Winkelsensor

Das Datenblatt des Winkelsensors ASM posihall PH36-V01-31T-I1-CW-M12A5 [42] kann dem Anhang D2 entnommen werden.

posihall®
PH36

ASM®

V / mA

PH36 - Magnetischer Multiturn-Encoder
Variante mit Analog-Ausgang

Technische Daten

			Bestellvarianten
Welle	Vollwelle 6 mm Vollwelle 10 mm	1	**V01** **V02**
Messbereich	Bis 31 x 360° (31 Umdrehungen) 1 Umdrehung, 2 Umdrehungen bis 31 Umdrehungen	2	 **1T** **2T ...** **31T**
Ausgang	Spannung 0,5 ... 10 V Spannung 0,5 ... 4,5 V Spannung 0,5 ... 4,5 V Strom 4 ... 20 mA, 3-Leiter-Technik	3	**U2** **U6** **U8** **I1**
Auflösung	Bis 16 Bit		
Wiederholgenauigkeit	0,1°		
Linearität	±(2° + 0,015% vom Messbereich)		
Gehäusematerial	Aluminium (Gehäuse), Edelstahl (Welle)		
Befestigung	Befestigungsexzenter, Befestigungsplatte		
Schutzart	IP67 wellenseitig IP67/69 gehäuseseitig (nur mit IP69-Gegenstecker)		
Signalverlauf	Signal rechtsdrehend ansteigend Signal linksdrehend ansteigend	4	**CW** **CCW**
Elektrischer Anschluss	M12-Stecker, axial, 5-polig	5	**M12A5**
Maximale Drehzahl (mech.)	Max. 10.000 U/min		
Zulässige Wellenbelastung	20 N radial, 10 N axial		
Lagerlebensdauer	1,5 x 10^{10} Umdrehungen (4500 h bei 6000 U/min)		
Schockbelastung	DIN EN 60068-2-27:2010, 100 g/11 ms, 100 Schocks		
Vibration	DIN EN 60068-2-6:2008, 20 g 10 Hz-2 kHz, 10 Zyklen		
Temperaturbereich	-40 ... +85°C		
Gewicht	ca. 120 g		
EMV	DIN EN 61326-1:2013		

Bestellcode

PH36 – 1 – 2 – 3 – 4 – 5

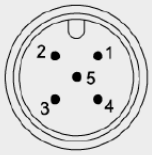
Bestellbeispiel: PH36 – V01 – 31T – I1 – CW – M12A5

posihall®
PH36

ASM®

I1 Stromausgang 4 … 20 mA, Dreileiter 	Versorgungsspannung	8 … 36 V DC
	Stromaufnahme	typisch 36 mA bei 24 V DC typisch 70 mA bei 12 V DC max. 120 mA
	Bürde R_L	500 Ω max.
	Ausgangsstrom	4 … 20 mA
	Messrate	1 kHz Standard
	Stabilität (Temperatur)	±50 x 10^{-6} / °C vom Messbereich (typisch)
	Elektrischer Schutz	Gegen Verpolung, Kurzschluss
	Arbeitstemperatur	siehe Modellspezifikation
	EMV	DIN EN 61326-1:2013

Anschlussbelegung
Stecker M12, 5-polig



Sicht auf die Stecker-kontakte des Sensors

Signal	Stecker PIN	Kabeladerfarbe
Versorgung +	1	braun
Signal	2	weiß
GND	3	blau
Nicht anschließen!	4	schwarz
Nicht anschließen!	5	(grau)

Anhang D3: mv-Transmitter

Das Datenblatt des Winkelsensors PR Electronics mV-Transmitter 2261 [43] kann dem Anhang D3 entnommen werden.

PR electronics

PERFORMANCE MADE SMARTER

Millivolt Signalgeber

2261

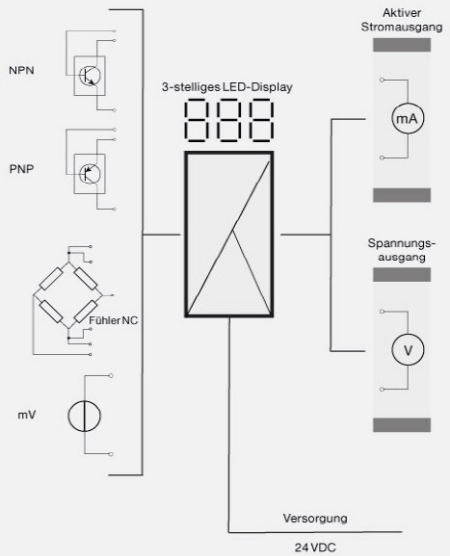
- Wägezellenverstärker
- mV für Strom- / Spannungsumformung
- Frontprogrammierbar / LED-Display
- Verhältniskalibrierung der Eingangsmessspanne
- NPN- / PNP-Eingang für externe Tarierung
- Versorgung für Standard-Umformer

Erweiterte Merkmale
- Konfiguration über Benutzerschnittstelle mit einem 3-ziffrigen Display und 3 Funktionstasten in der Gerätefront.

Anwendung
- 2261 verwandelt bipolare mV-Signale von Umformern, die direkt vom Gerät versorgt werden, in Standard-Strom- / Spannungssignale.
- Der 2261 eignet sich für Wägezellenanwendungen sowie für andere Anwendungen wie Tankentleerung /-füllung, Wiegung mit Selbsttarierung, Kabelzugkraftmessung, Niveaumessung, Signalumsetzung / -verstärkung usw.

Technische Merkmale
- Leuchtdiode für Fehlanzeige in der Gerätefront.
- Der Analogeingang kann auf Spannungen im Bereich -40...100 mVDC frei programmiert werden.
- Das digitale Signal kann als NPN (Kurzschluss mit Masse) oder PNP (+24 VDC) gewählt werden.
- Die Tarierung kann entweder über den Digitaleingang erfolgen oder von der Gerätevorderseite aus gewählt werden.
- Der Analogausgang kann auf Strom im Bereich 0...20 mA oder Spannung im Bereich 0...10 VDC frei programmiert werden.
- Umformerversorgung, die von der Gerätefront aus auf 5...13 VDC programmiert werden kann. Der Benutzer muss sicherstellen, dass die Versorgung mit maximal 230 mA belastet werden (z. B. 6 Stück 350-Ω-Wägezellen in Parallelschaltung).
- Fühler-Eigang (wenn die Umformerversorgung benutzt wird) zur Kompensation des Leiterwiderstandes zum Umformer.
- Montierbar auf ein Standard 11-poligen Sockel, welcher auf DIN-Schiene oder der Montageplatte montiert werden kann, mit PR 7023 Adapter und 7024 Kodier-Ring. Bei starken Vibrationen kann der PR 7002 Sicherungsbügel für die Serie 2200 auf Relaissockel verwendet werden.

Anwendungen



Bestellangangaben:

Typ
2261

Umgebungsbedingungen

Betriebstemperatur	-20°C bis +60°C
Kalibrierungstemperatur	20...28°C
Relative Luftfeuchtigkeit	< 95% RF (nicht kond.)
Schutzart	IP50

Mechanische Spezifikationen

Abmessungen (HxBxT)	80,5 x 35,5 x 84,5 mm (T ohne Kontaktstifte)
Gewicht, ca.	130 g

Allgemeine Spezifikationen

Versorgung

Versorgungsspannung	19,2...28,8 VDC
Leistungsbedarf, max.	7,2 W
Verlustleistung	2,2 W

Ansprechzeit

Ansprechzeit (programmierbar)	0,06...999 s
Signal- / Rauschverhältnis	Min. 60 dB
Aktualisierungszeit	20 ms
Signaldynamik, Eingang	17 Bit
Signaldynamik, Ausgang	16 Bit
Einfluss von Änderung der Versorgungsspannung	< ±0,002% d. Messsp. / %V
Temperaturkoeffizient	< ±0,01% d. Messsp. / °C
Linearitätsfehler	< 0,1% d. Messsp.
Hilfsspannung: Umformerversorgung	5...13 VDC
Belastung (max.)	230 mA
EMV-Immunitätswirkung	< ±0,5% d. Messsp.

Eingangsspezifikazionen

Allgemeine Eingangsspezifikationen

Max. Nullpunktverschiebung (Offset)	70% d. gew. Max.-Wertes

Spannungseingang

Messbereich	-40...100 mV
Min. Messereich (Spanne)	10 mV
Eingangswiderstand	> 10 MΩ
Überbelastung	0...999% der gewählten Messspanne
NPN, Digitaleingang	Pull up 24 VDC / 6,9 mA
PNP, Digitaleingang	Pull down 0 VDC / 6,9 mA
Trig-Niveau NIEDRIG, NPN/PNP	< 6 VDC
Trig-Niveau HOCH, NPN/PNP	> 10,5 VDC
Impulslänge	> 30 ms

Ausgangsspezifikationen

Stromausgang

Signalbereich	0...20 mA
Min. Signalbereich	5 mA
Belastung (bei Stromausgang)	≤ 600 Ω
Belastungsstabilität	≤ 0,01% d. Messsp. / 100 Ω
Strombegrenzung	< 23 mA
Spannungsausgang über internen Shunt (1)	Siehe Manual
d. Messspanne	= der gewählten Messspanne (1)

Eingehaltene Behördenvorschriften

EMV	2014/30/EU
EAC	TR-CU 020/2011

2261-052820