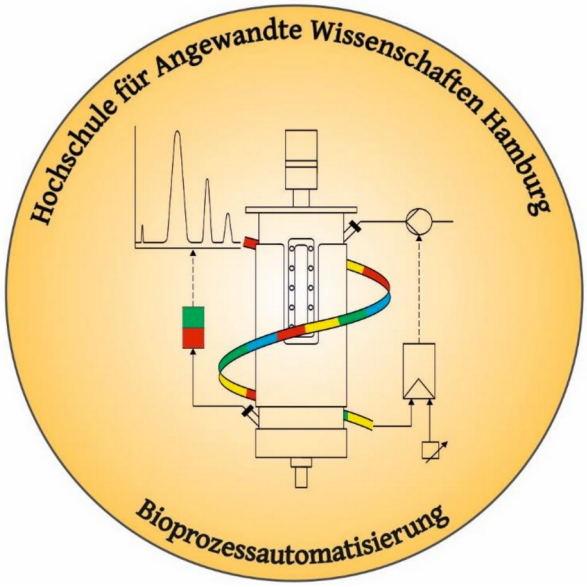
Fakultät Life Sciences
Department
Biotechnologie

Bachelorthesis

im Studiengang Biotechnologie

Entwicklung und Implementierung eines Kompensationsalgorithmus zur Methanolmessung in Bioreaktionsprozessen



Labor für Bioprozessautomatisierung

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

Philipp Schmidt
[redacted]

Hamburg, den 06.07.2023

1. Gutachter Prof. Dr. Gesine Cornelissen (HAW Hamburg)
2. Gutachter Prof. Dr. Christian Kaiser (HAW Hamburg)

Danksagung

Ich möchte mich an dieser Stelle herzlich bei Allen bedanken, die mich bei der Anfertigung dieser Arbeit unterstützt und motiviert haben. Besonderer Dank gilt Frau Prof. Dr. Gesine Cornelissen, die mir die Möglichkeit der Mitarbeit im Labor für Bioprozessautomatisierung gab, sowie für die Bereitstellung des Themas.
Weiterhin geht mein Dank an Herrn Prof. Dr. Christian Kaiser für die Übernahme der Zweitkorrektur. Ebenso geht mein Dank an Herrn Ullrich Scheffler für die Anregungen und Ratschläge, sowie für die Hilfestellungen in Problemsituationen. Herrn Alexander Thoma möchte ich für die Einarbeitung in die Thematik und die Unterstützung bei der Durchführung der ersten Versuche danken. Hans-Peter Bertelsen, Lisa Michel, sowie dem gesamten Team des Labors für Bioprozessautomatisierung danke ich für die ausgezeichnete Unterstützung und großartige Zusammenarbeit.
Ganz besonderer Dank gilt auch Frau Laura Amler, die mir die Motivation gab, diese Arbeit zu schreiben und mich stets bei jeglichen Fragen unterstützte. Mein Dank gilt auch meinen Eltern und Freunden, die mir während meines Studiums stets zur Seite standen.

Inhaltsverzeichnis

Nomenklatur und Abkürzungen

Tabelle 1: Nomenklatur

Symbol		Beschreibung	Einheit
AH	:=	absolute Feuchtigkeit (im Trägergas)	$[g\,m^{-3}]$
XL	:=	Änderung der Biomasse	$[g\,h^{-1}]$
$S1, M$	:=	Änderung der Substratmenge im Medium	$[g\,h^{-1}]$
$F_{G,AIR}$	:=	Begasungsrate	$[L\,h^{-1}]$
c_{S1}	:=	Konzentration des Substrats 1	$[g\,L^{-1}]$
$\mu_{\max}$	:=	maximale spezifische Wachstumsrate	$[h^{-1}]$
θ_{med}	:=	Medientemperatur	$[°C]$
c_{MeOH}	:=	Methanolvolumenkonzentration	[Vol%]
R_{Norm}	:=	normierter Widerstand	[]
OD_{VK}	:=	optische Dichte der Vorkultur bei 600 nm	[AU]
f_G	:=	Proportionalitätsfaktor	$[-]$
$K_{X/OD}$	:=	Proportionalitätsfaktor Biomasse/optische Dichte	$[g\,\mathrm{AU}^{-1}]$
t	:=	Prozesszeit	$[h]$
RH	:=	relative Feuchtigkeit	$[\%]$
RH_{gas}	:=	relative Trägergasfeuchtigkeit	$[\%]$
pO_2	:=	relativer Sauerstoffpartialdruck in der Flüssigkeit	$[\%]$
$c_{S1,R1}$	:=	Reservoirkonzentration des Substrat 1	$[g\,L^{-1}]$
R_S	:=	Sensorwiderstand	[]
μ_w	:=	Sollwachstumsrate	$[h^{-1}]$
F_{Rw}	:=	Sollwert der Zufütterrate	$[L\,h^{-1}]$
$Y_{X/S1}$	:=	Substratausbeutekoeffizient	$[g\,g^{-1}]$
θ_{gas}	:=	Trägergastemperatur	$[°C]$
V_L	:=	Volumen der Flüssigphase	$[L]$
Q_{S1R1}	:=	volumetrische Substrateintragsrate	$[g\,L^{-1}\,h^{-1}]$
c_X	:=	Zellkonzentration	$[g\,L^{-1}]$
$q_{S1/X}$	:=	zellspezifische Substrataufnahmerate	$[h^{-1}]$
$q_{S1/Xm}$	:=	zellspezifische Substrataufnahmerate (Zellerhaltung)	$[h^{-1}]$
μ	:=	zellspezifische Wachstumsrate	$[h^{-1}]$
F_R	:=	Zufütterrate	$[L\,h^{-1}]$
F_{S1R1}	:=	Zufütterrate des Substrat 1 aus einem Reservoir	$[L\,h^{-1}]$

Tabelle 2: Abkürzungen

Abkürzung	**Beschreibung**
P. pastoris	*Pichia pastoris*
AOX	Aldehydoxidasen
DCU	digitale Kontrolleinheit (engl. *digital control unit*)
OPC	*open platform communication protocol*
GUI	Benutzeroberfläche (engl. *graphical user interface*)
MFCS/win	Multi Fermenter Control System for Windows
MFC	*mass flow controller*
rpm	Umdrehungen pro Minute (engl. *revolutions per minute*)
VE-Wasser	vollentsalztes Wasser
WCB	Working-Zellbank (engl. *working cell bank*)

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Kultivierung von Mikroorganismen in Bioreaktoren nimmt eine zentrale Stellung in der modernen Biotechnologie ein. Durch die gezielte Züchtung und das Wachstum von Mikroorganismen in kontrollierten Umgebungen können Substanzen wie Enzyme, Impfstoffe oder weitere Zielproteine hergestellt werden, die in verschiedensten Bereichen Anwendung finden. Methylotrophe Organismen spielen in der biotechnologischen Produktion eine wichtige Rolle. Diese Organismen sind in der Lage, Methanol als einzige Kohlenstoff- und Energiequelle zu nutzen. Ein Beispiel für einen methylotrophen Organismus ist *Komagataella phaffii*, im Allgemeinen als *Pichia pastoris* (*P. pastoris*) bekannt. *P. pastoris* hat sich aufgrund seiner Fähigkeit zur Produktion von rekombinanten Proteinen als ein geschätztes Expressionssystem etabliert. Darüber hinaus wird es als Modellorganismus für verschiedenste Forschungsansätze genutzt.

Problemstellung

Für eine erfolgreiche Kultivierung von *P. pastoris* in Bioreaktoren ist eine präzise Kontrolle der Prozessparameter erforderlich. Ein besonders wichtiger Parameter ist dabei die Konzentration von Methanol im Kultivierungsmedium. Methanol dient nicht nur als Kohlenstoff- und Energiequelle, sondern kann auch gezielt als Induktor zur Aktivierung der Genexpression verwendet werden. Die Messung der Methanolkonzentration geschieht beispielsweise über die Abluft [14] oder über eine Tauchsonde im Medium. Eine im Hause konstruierte Methanolsonde wurde bereits für die Messung der Methanolkonzentration in Bioreaktoren verwendet[12]. Die Sonde arbeitet mit zwei Sensoren, um potenzielle Messabweichungen, die durch Quereinflüsse entstehen könnten, auszugleichen. Die bisherige Sonden-Software basiert jedoch auf einer veralteten Python Version, die nicht mehr gewartet wird und Sie bietet keine Funktion zur automatischen Kompensation dieser Querempfindlichkeiten.

Zielsetzung

Das Ziel dieser Arbeit besteht darin, die vorhandene Software zu aktualisieren, die Benutzeroberfläche zu optimieren und einen Algorithmus zur Kompensation von Querempfindlichkeiten zu integrieren. In dieser Bachelorarbeit liegt der Fokus auf der Entwicklung und Optimierung der Software, um die Verwendung der Methanolsonde in Bioprozessen für nachfolgende Studenten zu ermöglichen und die Sondensoftware auf eine mögliche Regelung des Methanolgehaltes über ein externes Prozessleitsystem vorzubereiten. Hierfür wird im Laborfermenter der dreistufige Fermentationsprozess mit einem methylotrophen Hefestamm durchgeführt, um die Differenz zwischen dem Ist-Zustand der Software und den Anforderungskriterien zu ermitteln. Im Anschluss wird der Programmcode an die neue Python-Umgebung angepasst und erweitert. Daraufhin wird die Software getestet und die Methanolsonde unter simulierten Prozessbedingungen mit verschiedenen Parametern kalibriert und die Daten ausgewertet.

2 Theoretischer Hintergrund

2.1 Modellorganismus

In dieser Arbeit wird ein transgener, methylotropher *Pichia pastoris* Stamm zur Produktion eines Modellproteins (*enhanced green fluorescent protein*, eGFP) verwendet. Methylotrophe Organismen besitzen spezifische Aldehydoxidasen (AOX-Gene), die den Metabolismus von Methanol als Kohlenstoffquelle ermöglichen. Sie können hierbei unterschiedliche Wachstumsraten aufweisen, die als ”*slow*” und ”*fast*” Metabolismus klassifiziert werden. Durch gentechnische Methoden wird ein Gen für das Modellprotein mit dem Promotor der AOX-Gene verknüpft, wodurch die Zugabe von Methanol eine Induktion des Zielgens bewirkt.

Methylotrophe Hefen

Methylotrophe *Pichia pastoris* Stämme sind als eukaryotisches Expressionssystem aus der modernen Biotechnologie nicht mehr wegzudenken und zählen neben der Bäckerhefe (*Saccharomyces cerevisiae*) zu den am meisten erforschten Hefestämmen [10]. Entdeckt wurde die Hefe im Jahr 1920 von Alexandre Guilliermond und wurde erstmals unter dem Namen *„Zygosacchormyces pastori“* publiziert. Im Jahr 1950 isolierte man verschiedene Anhänger der Gattung aus Eichenbäumen mit anschließender Umbenennung in *P. pastoris*. 35 Jahre später, im Jahr 1985 wurde das methanolinduzierte Expressionssystem auf Basis des stark exprimierten Gens für das Enzym Aldehydoxidase (AOX1) entwickelt [11]. Nach der Veröffentlichung zweier Gensequenzen im Jahre 2009, wurde das Genom kartiert und dient seitdem als Basis für Analysen und die Entwicklung von neuen Expressionssystemen. Die industriell, vielfältige Anwendung und Beliebtheit von *P. pastoris* Stämmen verdanken Sie unter Anderem der Fähigkeit rekombinante Proteine sowohl intra- als auch extrazellulär zu produzieren, sowie diese im Anschluss posttranslationell zu modifizieren [1]. Die Methoden für die Erstellung von *P. pastoris*-basierten Expressionssystemen sind heutzu-

tage gut dokumentiert und leicht umzusetzen [2]. Im Gegensatz zur Bäckerhefe verfügen die *P. pastoris* Stämme über einen Stoffwechselweg, welcher die Metabolisierung von Methanol erlaubt, welches somit als einzige Kohlenstoffquelle für Wachstum und Produktbildung vom Organismus genutzt werden kann. Bei Anwesenheit von Methanol im umgebenden Medium, wird der Promoter für die AOX1-Expression in *P. pastoris* aktiviert [8]. Das Methanol wird im ersten Schritt zu Formaldehyd oxidiert und anschließend durch die Dihydroxyaceton-Synthase in Xylose-5-Phosphat verstoffwechselt. Dies wird wiederum separat im Peroxisom zu Glyceralaldehyd-3-Phosphat und Diydroxyaceton metabolisiert [11]. Im Vergleich zu prokaryontischen Expressionssystemen bieten *P. pastoris*-basierte Expressionssysteme neben posttranslationellen Modifikationen auch die Möglichkeit zu höheren Produktkonzentrationen im Gramm-pro-Liter-Bereich, was eine industrielle Anwendung sehr attraktiv macht [2]. Die Anzucht bis hin zu hohen Zelldichten mit Glycerin eignet sich hierfür besonders, da der Glucose-Stoffwechsel im Vergleich zur Bäckerhefe um ein zehnfaches langsamer abläuft [5]. Der verwendete Stamm zählt zu den Mut^S-Stämmen, die besser für die Expression rekombinanter Proteine geeignet sind, als Mut^+-Stämme [4]. In Bioprozessen mit methylotrophen *P. pastoris* Stämmen, muss die Methanolkonzentration nach Induktion sorgfältig überwacht werden, da zu hohe Konzentrationen cytotoxisch sind. Bei einer zu geringen Methanolkonzentration im Medium kann es dazu kommen, dass die Expression des gewünschten Proteins aussetzt und die Ausbeute in der Produktionsphase sinkt.

2.2 Bioprozessmodell

Das Bioprozessmodell dient zur Anschauung komplexer biologischer Prozesse in Kombination mit technischen Aspekten (Bioreaktor). Dabei können die drei Betriebsweisen Satzverfahren (*batch*), Zulaufverfahren (*fed-batch*) und kontinuierliches Verfahren unterschieden werden. Ein dreistufiger Fermentationsprozess, welcher für die Kultivierung von *P. pastoris* verwendet wird, kann in die Phasen *batch*, *fed-*

batch und Produktion unterteilt werden. Jede Phase ist durch spezifische Wachstumsraten gekennzeichnet, die durch Anpassung der Prozessbedingungen moduliert werden können. In der *batch*-Phase erfolgt ein exponentielles Wachstum der Zellen bei einem bestimmten Reglerprofil, bis das Substrat aufgebraucht ist. Das Ende dieser Phase wird durch charakteristische Änderungen im Reglerverhalten signalisiert, woraufhin automatisch die *fed-batch*-Phase eingeleitet wird. In der *fed-batch*-Phase wird das Zellwachstum durch eine substratlimitierte Wachstumsrate gesteuert, bis eine vorab festgelegte Zelldichte erreicht ist. Anschließend beginnt die Produktionsphase durch Zugabe eines Induktionsmittels oder alternativen Substrats (z.B. Methanol), was zu einer veränderten Genexpression führt und die Produktion des Zielproteins ermöglicht.

Dreistufiger Fermentationsprozess

Im Folgenden wird der theoretische Ablauf des in dieser Arbeit vorliegenden Fermentationsprozesses erläutert. Dieser gliedert sich in drei Phasen, welche sich durch unterschiedliche Wachstumsraten des zu kultivierenden Organismus definieren.

Phase 0: Vorkultur

Für die Inokulation eines Bioreaktors wird aus einer Arbeitszellbank (*working cell bank*, WCB) eine Vorkultur im Schüttelkolben angesetzt. Dies bietet den Vorteil, den Bioreaktor mit einer gewünschten Zellkonzentration inokulieren zu können. Des Weiteren ermöglicht dies eine Inokulation des Bioreaktors mit Zellen in der exponentiellen Wachstumsphase, um die Anlaufphase (*lag*-Phase) zu verringern. Eine Kultivierung im Satzbetrieb beschreibt die Anzucht von Mikroorganismen in einem Bioreaktor mit einem vorgegebenen Ende, sobald das vorliegende Substrat vollständig aufgebraucht ist.

Phase 1: *batch*-Phase

Die *batch*-Phase beginnt mit der Inokulation, bei der die Vorkultur in den autoklavierten Bioreaktor überführt wird. Nach einer *lag*-Phase, in der sich der Metabolismus der Zellen an die vorliegenden Gegebenheiten (z.B. Temperatur und Begasung) des Bioreaktors adaptiert, läuft das Wachstum unter optimalen Bedingungen und Substratüberschuss exponentiell bei maximaler Wachstumsrate ab. Der hierbei ebenfalls exponentiell ansteigende Sauerstoffverbrauch wird durch eine Regelung des Sauerstoffpartialdrucks gedeckt, wobei die Rührerdrehzahl als Stellgröße fungiert. Das Ende der *batch*-Phase lässt sich erkennen an einem sprunghaften Anstieg des pO_2-Wertes, sobald die Zellen ihren Metabolismus in Abwesenheit von Glycerol verlangsamen. Allgemein lässt sich das Wachstum von Mikroorganismen in der *batch*-Phase durch Gl. (1) beschreiben:

$$\dot{m}_{\mathrm{XL}} = \mu \cdot c_{\mathrm{XL}}(t) \cdot V_{\mathrm{L}} \tag{1}$$

Mit:

Tabelle 3: Variablen der Gl. (1), Biomasseänderungsrate

Symbol	Beschreibung	Einheit
$\dot{m}_{\mathrm{XL}}$	Änderung der Biomasse	[g h^{-1}]
μ	zellspezifische Wachstumsrate	[h^{-1}]
c_{XL}	Zellkonzentration	[g L^{-1}]
V_{L}	Volumen der Flüssigphase	[L]

Die zeitliche Änderung der Masse ist proportional zur Wachstumsrate und verläuft in der *batch*-Phase exponentiell bei μ_{max}. Unter Vernachlässigung der Volumenänderung (Zugabe von Korrekturmitteln und Probenahmen) und bei bekannter Zellkonzentration zum Zeitpunkt der Inokulation kann die Zellkonzentration zu einem beliebigen Zeitpunkt innerhalb der Phase beschrieben werden mit Gl. (2):

$$c_{\mathrm{XL}}(t) = c_{\mathrm{XL},0} \cdot e^{\mu_{\max} \cdot (t - t_0)} \quad \text{for} \quad t \leq t_{\mathrm{end}} \tag{2}$$

Mit:

Tabelle 4: Variablen der Gl. (2), Exponentielles Zellwachstum

Symbol	Beschreibung	Einheit
$c_{\mathrm{XL}}(t)$	Zellkonzentration zum Zeitpunkt t	[g L^{-1}]
$c_{\mathrm{XL},0}$	Startzellkonzentration zum Zeitpunkt t_0	[g L^{-1}]
$\mu_{\max}$	Maximale spezifische Wachstumsrate	[h^{-1}]
t_0	Startzeitpunkt der *batch*-Phase (Inokulation)	[h]

Das Ende der *batch*-Phase tritt ein, sobald das gesamte Substrat im Medium verbraucht wurde. Der Zeitpunkt der Substratlimitierung lässt sich unter Vernachlässigung der *lag*-Phase mit Gl. (3) berechnen:

$$t_{\mathrm{end}} = \frac{\ln\left(\frac{c_{\mathrm{XL,end}}}{c_{\mathrm{XL},0}}\right)}{\mu_{\max}} \tag{3}$$

Mit:

Tabelle 5: Variablen der Gl. (3), *batch*-Phase-Endzeitpunkt

Symbol	Beschreibung	Einheit
t_{end}	Endzeitpunkt der *batch*-Phase	[h]
$c_{\mathrm{XL,end}}$	Zellkonzentration zum Ende der *batch*-Phase	[g L^{-1}]
$c_{\mathrm{XL},0}$	Zellkonzentration im Medium nach Inokulation	[h^{-1}]
$\mu_{\max}$	maximale spezifische Wachstumsrate	[h]

Die Zellkonzentration zu Beginn der *batch*-Phase lässt sich mit Gl. (4) und dem Proportionalitätsfaktor $K_{x/OD}$ bestimmen.

$$c_{\mathrm{XL},0} = \frac{OD_{\mathrm{VK}} \cdot V_{\mathrm{L,VK}}}{V_{\mathrm{L,R}}} \cdot K_{\frac{X}{OD}} \tag{4}$$

Tabelle 6: Variablen der Gl. (4), *batch*-Phase-Start-Zellkonzentration

Symbol	Beschreibung	Einheit
$c_{\mathrm{XL},0}$	Zellkonzentration zum Zeitpunkt t_0	[g L^{-1}]
$\mathrm{OD_{VK}}$	optische Dichte der Vorkultur bei 600 nm	[AU]
$V_{\mathrm{L,VK}}$	Volumen der Vorkultur	[L]
$V_{\mathrm{L.R}}$	Volumen des Mediums im Reaktor	[L]
$K_{X/\mathrm{OD}}$	Proportionalitätsfaktor	[g AU^{-1}]

Des Weiteren lässt sich die Zellkonzentration zum Ende der *batch*-Phase mit Hilfe des Substratausbeutekoeffizienten und Gl.(3) berechnen. Die Substratkonzentration $c_{S1,0}$ zum Zeitpunkt t_0 ist dabei bekannt.

$$c_{\mathrm{XL,end}} = c_{\mathrm{XL},0} + c_{S1,0} \cdot y_{\frac{X}{S1}} \tag{5}$$

Mit den Variablen:

Tabelle 7: Variablen der Gl. (5), *batch*-Phase-End-Zellkonzentration

Symbol	Beschreibung	Einheit
$c_{\mathrm{XL,end}}$	Zellkonzentration zum Ende der *batch*-Phase	[g L^{-1}]
$c_{\mathrm{XL},0}$	Zellkonzentration im Medium nach Inokulation	[g L^{-1}]
$c_{\mathrm{S1},0}$	Substratkonzentration zu Beginn der *batch*-Phase	[g L^{-1}]
$y_{X/S1}$	Substratausbeutekoeffizient	[g_X g_S^{-1}]

Phase 2: *fed-batch*-Phase

Unter einem *fed-batch* versteht man einen diskontinuierlichen Prozessschritt, bei dem Substrat aus einem externen Reservoir dem Bioreaktor zugeführt wird. Dies ermöglicht beispielsweise Die Einstellung einer gewünschten Wachstumsrate. Die *fed-batch*-Phase folgt auf eine *batch*-Phase und dient der kontrollierten Zellanreicherung bis hin zu hohen

Zelldichten. Im *fed-batch* wird über eine Zufütterpumpe frisches Substrat dem Reaktor zugeführt. Die Zufütterung erfolgt dabei über eine Pumpe, welche mit oder ohne Regelung betrieben werden kann. Für ein Zufütterungsprofil ohne Regelung wird bereits vor der Fermentation eine exponentielle Zufütterrate F_{Rw} bestimmt. Dies ermöglicht die Einhaltung einer konstanten Wachstumsrate μ_w unterhalb der maximalen Wachstumsrate μ_{max}. Da der maximal mögliche Sauerstoffeintrag ins Medium technisch begrenzt durch beispielsweise die Rührerdrehzahl und die Begasungsrate ist, wird die Wachstumsrate so gewählt, dass diese Faktoren keine Limitierung darstellen und das Wachstum auf Substratlimiterung durch die Zufütterung geregelt wird. Die zeitliche Änderung der Substratmenge im Medium wird beschrieben durch Gl. (6):

$$\dot{m}_{S_{1,M}} = q_{\frac{S_1}{X}}(t) \cdot c_{\mathrm{XL}}(t) \cdot V_{\mathrm{L}}(t) + F_{S_1R_1}(t) \cdot c_{S_1R_1} \tag{6}$$

Mit:

Tabelle 8: Variablen der Gl. (6, 7), Zellwachstum im *fed-batch*

Symbol	Beschreibung	Einheit
$\dot{m}_{\mathrm{S1,M}}$	Änderung der Substratmenge im Medium	[g h^{-1}]
$q_{S1/X}$	zellspezifische Substrataufnahmerate	[h^{-1}]
c_{XL}	Zellkonzentration in der Flüssigphase	[g L^{-1}]
V_{L}	Flüssigkeitsvolumen im Bioreaktor	[L]
F_{S1R1}	Zufütterrate des Substrat 1 aus einem Reservoir	[L h^{-1}]
c_{S1R1}	Reservoirkonzentration des Substrat 1	[g L^{-1}]

Zur Vereinfachung wird die Annahme getroffen, dass unter substratlimitierenden Bedingungen, das zugeführte Substrat (hier Glycerin) direkt verbraucht wird und somit vernachlässigt werden kann. Die Zufütterrate lässt sich dann für jeden Zeitpunkt t mit Gl.(7) beschreiben.

$$F_{S_1R_1}(t) = \frac{q_{\frac{S_1}{X}}(t)}{c_{S_1R_1}} \cdot c_{\mathrm{XL}}(t) \cdot V_{\mathrm{L}}(t) \tag{7}$$

Das Zellwachstum lässt sich in der fed-*batch*-Phase mit Gl.(8) beschreiben. Dazu wird ein Sollwert für die Zufütterrate gewählt, welcher ein exponentielles Wachstum mit einem gewünschten Sollwert ermöglicht.

$$F_{\mathrm{RW}}(t) = F_{R,i} \cdot e^{\mu_W \cdot (t - t_i)} \tag{8}$$

Tabelle 9: Variablen der Gl. (8), Zufütterrate

Symbol	Beschreibung	Einheit
F_{RW}	Sollwert der Zufütterrate	[L h^{-1}]
$F_{\mathrm{R},i}$	Zufütterrate zum Zeitpunkt t_i	[L h^{-1}]
μ_{W}	Sollwachstumsrate	[h^{-1}]
t_i	Zeitpunkt i (hier Ende *batch* / Start *fed-batch*)	[h]

Die Zufütterrate $F_{R,i}$ zum Zeitpunkt t_i lässt sich mit Gl.(9) bestimmen.

$$F_{R,i} = \frac{(\mu_w + q_{\frac{S_1}{X_m}} \cdot y_{\frac{X}{S_1}}) \cdot V_{L,i} \cdot c_{XL,i}}{y_{\frac{X}{S_1}} \cdot c_{S_1R_1}} \tag{9}$$

Tabelle 10: Variablen der Gl. (9), Zellerhaltungsstoffwechsel

Symbol	Beschreibung	Einheit
$q_{\mathrm{S1/Xm}}$	zellspezifische Substrataufnahmerate (Zellerhaltung)	[h^{-1}]
$V_{\mathrm{L},i}$	Reaktionsvolumen zum Zeitpunkt t_i	[L]
$c_{\mathrm{XL},i}$	Zellkonzentration zum Zeitpunkt t_i	[g L^{-1}]

Die volumetrische Substrateintragsrate lässt sich mit Gl.(12) bestimmen, welche die momentane Zufütterrate $F_{R,1}$, die Substratkonzentration $c_{S1,R1}$ und das Reaktionsvolumen V_L beinhaltet.

$$Q_{\mathrm{S1R1}}(t) = \frac{F_{\mathrm{R}}(t) \cdot c_{\mathrm{S1R1}}}{V_{\mathrm{L}}(t)} \tag{10}$$

Tabelle 11: Variablen der Gl. (10), Volumetrische Substrateintragsrate $Q_{S1,R1}$

Symbol	Beschreibung	Einheit
$Q_{\mathrm{S1R1}}(t)$	volumetrische Substrateintragsrate	[g l^{-1} h^{-1}]
$F_{\mathrm{R}}(t)$	momentane Zufütterungsrate	[l h^{-1}]
c_{S1R1}	Substratkonzentration im Reservoir	[g l^{-1}]
$V_{\mathrm{L}}(t)$	Gesamtvolumen im Reaktor	[l]

Phase 3: Produktbildungsphase

Sobald die gewünschte Zelldichte im Reaktor erreicht wird, kann das Substrat (hier Glycerin) zum Induktor (hier Methanol) gewechselt werden, welcher die Produktbildungsphase einleitet. Die Produktbildungsphase ist ebenfalls eine *fed-batch* Phase, die sich von der vorherigen Phase im Substrat unterscheidet. Die Induktion der Produktbildung wird durch Zufütterung mit Methanol eingeleitet. Dies geschieht automatisch durch Aktivierung einer Zufütterpumpe, die mit dem Reservoir des Induktormediums verbunden ist. Die dafür nötige Pumprate wird so gewählt, dass sich eine stabile Induktorkonzentration im Medium einstellt, welche durch eine Sonde (hier Methanolsonde) erfasst werden kann. Nach der Induktion der Zellen kommt es zunächst zu einer *lag*-Phase, welche abgewartet wird, bevor erneut Induktor zugegeben wird. Für die Induktion ist eine Methanolkonzentration von 0,1 % bis 0,3 % üblich [14] und hängt hauptsächlich von dem verwendeten Stamm ab (Mut^S oder Mut^+). Während der Induktionsphase können die Zellen ihren Metabolismus dem Induktor entsprechend anpassen, was abgeschlossen ist, sobald der Sauerstoffbedarf der Zellen anfängt zu steigen. Nach Anpassung an das neue Substrat wird die automatische Zufütterung mit einer konstanten Zufütterrate initiiert. Anschließend wird die Zufütterrate mit einer Regelung der Methanolkonzentration im Reaktor auf dem gewünschten Wert gehalten. Für die Regelung der Methanolkonzentration kommen verschiedene Ansätze in Frage [3].

2.3 Bioreaktorsystem

Bioreaktor BIOSTAT A

Der Bioreaktor BIOSTAT® A der Firma Sartorius dient der Kultivierung von Mikroorganismen und ermöglicht ein kontrolliertes Wachstum durch umfassende Messinstrumentalisierung und eine Prozessleitsystem (*multi-fermenter control system*; MFCS). Der BIOSTAT® A ist ein Rührkesselreaktor, welcher mit Reglern für verschiedene Parameter wie Temperatur, Sauerstoffversorgung, pH-Wert, Schaumniveau, und Substratzufuhr ausgestattet ist. Des Weiteren sind Instrumente zur Messung von Füllstand, Methanolkonzentration und Abluftzusammensetzung vorhanden. Die Zusammenarbeit des MFCS mit der digitalen Kontrolleinheit (*digital control unit*; DCU) ermöglicht die Kommunikation und Datenerfassung von Prozessvariablen.

Aufbau des Bioreaktors

Die im Rahmen dieser Arbeit durchgeführten Fermentationen wurden im Bioreaktor BIOSTAT A der Firma Sartorius durchgeführt. Der Fermenter besteht aus einem einwandigen Glasgefäß mit Edelstahldeckel, welches in einem dreibeinigen Edelstahlgestell eingehängt ist. An dem Gestell befindet sich eine Vorrichtung zum Einhängen eines Probenahmegefäßes, sowie drei Bohrungen für die Befestigung des Reaktordeckels. Das Fassungsvermögen des Fermenters beträgt 3 L, wobei ein Arbeitsvolumen von 0,6 L bis 2,0 L sich empfiehlt. Die Temperierung erfolgt über eine von außen befestigte Heizmanschette und einen ins Kulturmedium eintauchenden Kühlfinger, welcher mit einem Umlaufkühler verbunden ist. Das Rührwerk befindet sich in der Mitte des Deckels und wird durch eine einfache Gleitringdichtung abgedichtet. Durch die insgesamt zwölf Öffnungen im Reaktordeckel werden die Messtechnik (pO_2-Sonde, pH-Sonde, Füllstandsonde), sowie das Probenahmerohr, Korrekturmittelzugabeports (Säure, Base, Antischaum), Zulaufreservoir, Begasungsring und Zugabeport für das Inokulum und den Induktor angebracht. Der gesamte Bioreaktor befin-

det sich auf einer Waage, um Gewichtsänderungen durch beispielsweise Korrekturmittel- und Substratzugabe zu erfassen. Dies ermöglicht die gravimetrische Bestimmung von Korrekturmittel- und Substratverbrauch in den jeweiligen Fermentationsphasen.

Messtechnik

Die Kontrolleinheit (*digital control unit*, DCU) dient der Regelung und Kontrolle von Prozessparametern. Die Messinstrumente für pH-Wert, Sauerstoffpartialdruck (pO_2) sowie die Füllstandssonde können an der DCU angeschlossen werden. Desweiteren ist in der DCU ein Massendurchflussregler (*mass flow controller*, MFC) eingebaut, welcher die Möglichkeit liefert, einen Massenstrom auf einen Sollwert zu regeln. So kann die Begasungsrate individuell eingestellt werden. Der Bioreaktor kann mit wahlweise gefilterter Druckluft oder Stickstoff begast werden. Dazu befindet sich außerhalb des Bioreaktors ein Wechselventil. An der Vorderseite der DCU befinden sich zwei Schwebekörperdurchflussmesser und drei peristaltische Pumpenköpfe, welche die Zufuhr von Korrekturmitteln (Säure, Base, Antischaum) ermöglichen.

Temperierung

Die Temperatur des Mediums wird von innen, mit einem, im Medium eingetauchten, Kühlfinger und von außen mit einer Heizmanschette geregelt. Die Messung der Temperatur erfolgt im Medium über die pH-Elektrode, welche über einen Halbleitertemperatursensor verfügt.

Begasung

Für eine Kultivierung unter aeroben Bedingungen wird eine konstante Begasung mit gefilterter Druckluft bereitgestellt. Für die im Rahmen der Arbeit durchgeführten Kultivierungen wird eine spezifische Begasungsrate von 2 vvm eingestellt, was umgerechnet zu einer Begasung mit $2{,}8\,\mathrm{l\,min^{-1}}$ führt. Diese Begasungsrate sorgt für ausreichend Gaszufuhr in allen Phasen des Prozesses. Die Begasungsrate errechnet sich

aus dem Volumen der Flüssigphase im Bioreaktor und der spezifischen Begasungsrate mit der Gl. (11):

$$F_{G,\mathrm{AIR}} = f_G \cdot V_L \tag{11}$$

Mit:

Tabelle 12: Variablen der Gleichung (11), Berechnung der Begasungsrate

Symbol	Beschreibung	Einheit
$F_{G,\mathrm{AIR}}$	Begasungsrate	[l/min]
f_G	Proportionalitätsfaktor	[–]
V_L	Volumen Flüssigphase	[L]

Für eine Kultivierung unter aeroben Bedingungen wird eine konstante Begasung mit gefilterter Druckluft bereitgestellt. Diese erfolgt über einen Begasungsring, welcher sich unterhalb des Rührwerks/des Schreibenrührers befindet. Die Zerstäubung der entstehenden Gasblasen durch den Rührer erhöht den Sauerstoffeintrag ins Medium und lässt sich mit Veränderung der Umdrehungszahl dem Sauerstoffbedarf der Mikroorganismen anpassen. Das Abgas wird durch einen Abluftkühler geleitet, um den Volumenverlust durch Verdunstung möglichst gering- und um den Abgasfilter trocken zu halten. Ein Abgassensor erfasst die Zusammensetzung des Abgases.

Prozessleitsystem MFCS/win

Für eine automatisierte und kontrollierte Prozessführung ist ein System zur Datenerfassung und Systemsteuerung nötig. Das Prozessleitsystem *MFCS/win* (*Multi Fermentor Control System for Windows*), entwickelt von *Sartorius Stedim System GmbH*, erfüllt diese Anforderung als Server/Client-PC-Leitrechnersystem. Zusammen mit der DCU setzt das Prozessleitsystem programmierte Abläufe um.

Das Prozessleitsystem *MFCS/win* hat mehrere Funktionen, auf die über die *Shell* zugegriffen werden kann:

- **Operator Service:** Im Operator Service werden *online*-Variablen des laufenden Prozesses angezeigt, wie etwa Pumpraten, Rühr-

erdrehzahl und pH-Wert. Hier sind auch manuelle Einstellungen möglich.

- **Sample Data Management:** Hier werden *offline*-Daten verwaltet, einschließlich der Eintragung von Messwerten wie der optischen Dichte.
- **Batch Management:** Ermöglicht das Definieren neuer Kultivierungen und die Auswahl von Rezepten für den Kultivierungsprozess. Sowohl geplante als auch abgeschlossene Kultivierungen sind hier einsehbar.
- **Reporting:** Das Reporting-Tool ermöglicht den Export von im Prozess erfassten Daten für weitere Analysen.
- **Plotting:** Bietet eine visuelle Darstellung ausgewählter Prozessparameter des laufenden Prozesses.
- **Configuration Management:** Erlaubt das Erstellen, Modifizieren und Löschen von Rezepten sowie das Definieren neuer Variablen. Diese Rezepte repräsentieren zeitliche Abläufe im Prozess und können im Batch Management ausgewählt werden.

2.4 Methanolsonde

Die in dieser Arbeit verwendete Sonde für die Messung der Methanolkonzentration taucht mit einer Membran in die Kulturbrühe im Rührkessel ein. Methanolmoleküle diffundieren durch die Membran in die Gasphase und werden durch ein Trägergas an einem Alkoholsensor (TGS2620) und einem Kombisensor für relative Feuchtigkeit, Temperatur und Druck (BM260) vorbeigeführt. Der Alkoholsensor besteht aus einem erhitzten Filament, an dem Gasmoleküle adsorbieren können, wodurch sich die Leitfähigkeit verändert und als proportionale Messwertänderung erfasst werden kann. Ein Quereinfluss von Wasser im Trägergas wird durch den Kombisensor und eine nachfolgende algorithmische Kompensation berücksichtigt, ebenso wie die Temperatur des Mediums.

In dieser Arbeit wird eine speziell konstruierte Methanolsonde verwendet, die in Anlehnung an die Methanolsonde der Firma Biotechnologie Kempe GmbH entwickelt wurde. Diese Sonde enthält einen externen Gasraum (Abb 1), der außerhalb des Reaktors angeordnet ist und in dem der TGS2620 Alkoholsensor zusammen mit dem BME280 Kombisensor platziert ist. Der externe Gasraum ist über ein Rohr mit der Fermentationsbrühe verbunden. Innerhalb dieses Rohres befindet sich eine Polytetrafluorethylen-Membran, die durch einen Silikonschlauch geschützt ist. Diese Membran ermöglicht das Eindringen flüchtiger organischer Stoffe in die Messkammer.

Der TGS2620 Sensor ist ein präzises Instrument zur Detektion von Alkoholen und basiert auf der Metalloxid-Halbleitertechnologie. Im Besonderen verwendet dieser Sensor Zinndioxid (SnO_2) als Sensormaterial auf einer Aluminiumgrundlage. Vor der Gasdetektion muss das Sensormaterial durch ein internes Heizelement auf eine Betriebstemperatur von etwa 400 °C erhitzt werden, wodurch es zur Adsorption von Sauerstoffmolekülen an der Oberfläche der Halbleiterschicht kommt. Dies führt zu einem Übergang von Elektronen des Halbleiters auf die Sauerstoffmoleküle und bewirkt einen Anstieg des Sensorwiderstands, da weniger freie Elektronen an der Oberfläche des Halbleiters verfügbar sind.

Wenn ein Methanolmolekül auf ein adsorbiertes Sauerstoffmolekül trifft, wird es oxidiert. Dies erhöht die Elektronendichte in der Halbleiterschicht und führt zu einer Abnahme des Sensorwiderstands. Die Änderung des Sensorwiderstands ist proportional zur Gaskonzentration und kann zur Messung des Methanolgehalts im Medium genutzt werden.

Da der Widerstand der Halbleiterschicht auch durch den Gasdruck, die Temperatur und die Luftfeuchtigkeit des Trägergases beeinflusst werden kann, ist es wichtig, diese Quereinflüsse zu berücksichtigen. Der BME280 Sensor, der zusammen mit dem TGS2620 im externen Gasraum platziert ist, erfasst diese Parameter. Durch programmiertechnische Kompensation werden diese Quereinflüsse ausgeglichen, um genaue Messungen zu gewährleisten.

Die Messkammer wird von einem Trägergas durchflutet, welches die durch die Membran pervaporierten Bestandteile der Fermentationsbrühe an den Sensor heranführt. Dieses Prinzip ermöglicht eine präzise und zuverlässige Messung der Konzentration flüchtiger organischer Verbindungen, insbesondere von Methanol.

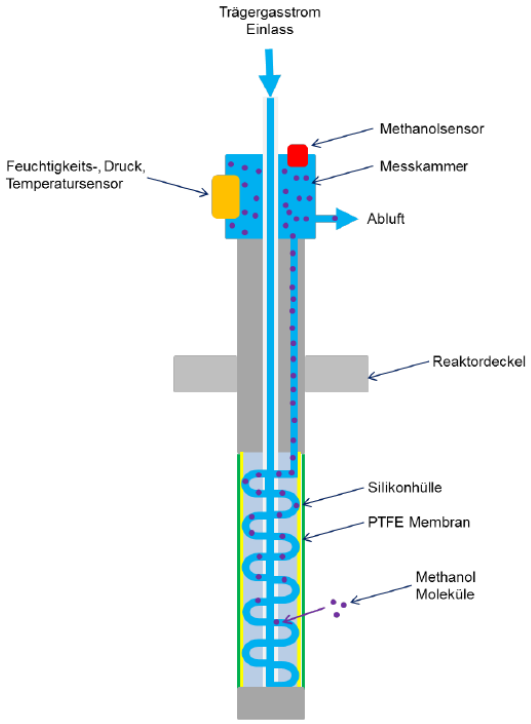


Abbildung 1: **Schematische Darstellung der Methanolsonde** Übernommen nach Lampert [9] und Struck [13].

2.4.1 Kompensation der Trägergaseigenschaften

Die gemessenen Widerstandswerte der Methanolsonde werden normiert auf einen Wert von 65 % Luftfeuchtigkeit bei 20-°C, indem zunächst die absolute Luftfeuchtigkeit aus der temperaturabhängigen Sättigungsmenge berechnet wird. Anschließend wird der Quotient aus gemessenem und normiertem Widerstand gebildet. Zunächst wird hierfür die relative Feuchtigkeit mit der Trägergastemperatur zu einer absoluten Feuchtigkeit umgerechnet. Für die Bestimmung der absoluten Feuchtigkeit gilt die Gl. (12).

$$\mathrm{AH}(RH, \vartheta_\mathrm{U}) = \frac{RH \cdot (0.0259 \cdot \vartheta_\mathrm{U}^2 + 0.0608 \cdot \vartheta_\mathrm{U} + 5.7252)}{100\%} \tag{12}$$

Tabelle 13: Variablen der Gl. 12, AH aus RH und ϑ_{U}

Symbol	Beschreibung	Einheit
AH	absolute Luftfeuchtigkeit	[g m^{-3}]
RH	relative Luftfeuchtigkeit	[%]
ϑ_{Gas}	Trägergastemperatur	[°C]

Nach dem technischen Datenblatt des TGS2620 Alkoholsensors [6] ergibt sich folgende Formel für die Berechnung des normierten Widerstandes auf 20°C und 65% Luftfeuchtigkeit Gl. (13):

$$R_{\mathrm{SM,komp}}(R_{\mathrm{SM}}, AH) = R_{\mathrm{SM}} + (1 - 2.7786 \cdot AH^{-0.428}) \cdot R_{\mathrm{SM}} \tag{13}$$

Tabelle 14: Variablen der Gl. (13), $\mathrm{R_{Norm}}$ aus AH

Symbol	Beschreibung	Einheit
$R_{\mathrm{SM,Norm}}$	normierter Widerstand	[Ω]
R_{SM}	unkompensierter Widerstandsmesswert	[Ω]
AH	absolute Feuchtigkeit im Trägergas	[g m^{-3}]

2.4.2 Kompensation der Medientemperatur

Die Temperatur in der Flüssigphase hat ebenfalls Einfluss auf den Widerstandswert. Es wird der Ansatz untersucht, ob es möglich ist die Koeffizienten einer Kalibrierfunktion einer bestimmten Temperatur, durch lineare Interpolation in Koeffizienten einer anderen Temperatur zu übersetzen. Hierbei gilt für Kalibrierpolynome generell folgender Zusammenhang für die Abhängigkeit des Sensorwiderstandes von der Methanolkonzentration Gl. (14) (n=1):

$$\ln(c_{\mathrm{MeOH}}) = a_1 \cdot \ln(R_S) + a_0 \tag{14}$$

Tabelle 15: Variablen der Gl. 14, Kalibriergerade

Symbol	Beschreibung	Einheit
c_{MeOH}	Methanolvolumenkonzentration	[Vol%]
a_1	Steigung der Kalibriergerade	[–]
a_0	Y-Achsenabschnitt	[–]
R_S	Sensorwiderstand	[Ω]

Es gilt die Annahme, dass auch eine Querempfindlichkeit gegenüber

der Medientemperatur θ_L besteht für die Steigung und den Achsenversatz. Somit gilt folgender, beispielsweise linearer Zusammenhang zwischen den Koeffizienten a_n für zwei bekannte Temperaturen θ_1 und θ_2. Eine Lineare Interpolation aus zwei Punkten ergibt Gl. (15):

$$a_n(\theta_\mathrm{L}) = a_{n,1} + \frac{\theta_\mathrm{L} - \theta_{\mathrm{L},1}}{\theta_{\mathrm{L},2} - \theta_{\mathrm{L},1}}(a_{n,2} - a_{n,1}) \tag{15}$$

Tabelle 16: Variablen der Gl. (15), Lineare Interpolation

Symbol	Beschreibung	Einheit
a_{n,θ_L}	Koeffizient bei n bei θ_L	[–]
$a_{\mathrm{n},1}$	Koeffizient bei Temperatur θ_1	[–]
θ_L	Medientemperatur	[°C]
$\theta_{\mathrm{L},1}, \theta_{\mathrm{L},2}$	Kalibriertemperaturen 1,2	[Ω]

Mit diesem Ansatz lässt sich jeweils die Steigung und der Achsenversatz eines Kalibriermodells um einen möglichen Einfluss der Medientemperatur kompensieren. Aus Gl. (13), Gl. (14) und Gl. (15) lässt sich somit folgendes Modell für die medientemperaturkompensierte Konzentrationsberechnung aufstellen:

$$c(R_{Norm}, \theta_\mathrm{L}) = e^{\sum_{n=0}^{N} a_{n,\theta} \cdot \ln(R_\mathrm{Norm})^n} \tag{16}$$

Tabelle 17: Variablen der Gl. (16), Allgemeine Form Kalibriergerade

Symbol	Beschreibung	Einheit
c	Methanolkonzentration	Vol%
R_S	Sensorwiderstand	[Ω]
θ_L	Medientemperatur	[°C]
RH_Gas	Relative Trägergasfeuchtigkeit	[%]
θ_Gas	Trägergastemperatur	[°C]
$a_{n,\theta}$	Temperaturkompensierter Koeffizient	[–]
R_Norm	Normierter Widerstand	[–]

Für die Kalibrierung der Methanolsonde ist es daher nötig, die gemessenen Widerstandswerte mit Gl.(13) um die Eigenschaften des Trägergases zu kompensieren. Anschließend werden diese doppelt-logarithmisch gegen die zugehörige Methanolkonzentration aufgetragen. Der Konzentrationsbereich wird hierfür zwischen 0,5 und 1,5 Vol% gewählt. Dieser Konzentrationsbereich erlaubt eine Aktivierung des AOX1-Gens

bei gleichzeitiger Vermeidung einer cytotoxischen Konzentration. Dabei ist 1 Vol% Methanol in der Bioprozessautomatisierung ein üblicher Zielwert für eine Feedregelung während der Produktionsphase methylotropher Organismen.

Kompensation durch Offset-Korrektur

Bei der Kalibrierung ist es nicht möglich einen Nullpunkt zu definieren weil der natürliche Logarithmus von null nicht definiert ist. Deshalb besteht ein Offset zwischen der berechneten Konzentration unterhalb des Kalibrierbereichs. Die Offset-Korrektur durch Näherung mit $c_{0,1\ \%} \approx c_{0\ \%}$ wird in den Programmcode integriert. Die Korrektur eines Kalibrierpolynoms über Anpassung des Offsets lässt sich mit folgenden Gleichungen (17, 18,19) beschreiben:

$$c_{\text{pred}} = \exp\left(a_0 \cdot \log(r) + a_1\right) \tag{17}$$

$$\text{offset} = \log(c) - \log\left(c_{\text{pred}}\right) \tag{18}$$

$$a_1' = a_1 + \text{offset} \tag{19}$$

Tabelle 18: Variablen der Gleichung (17) bis (19), Kompensation des Offsets

Symbol	Beschreibung	Einheit
c_{pred}	Vorhergesagte Konzentration	
a_0	Steigung der Kalibriergeraden	
a_1	Achsenabschnitt der Kalibriergeraden	
a_1'	Angepasster Achsenabschnitt der Kalibriergeraden	
$R_{(Norm)}$	Abgelesener normierter Widerstand	
c	Tatsächliche Konzentration	
offset	Abweichung zwischen $\log(c)$ und $\log(c_{\text{pred}})$	

2.4.3 Berechnung der Verzögerungszeit

Dem Datenblatt des TGS2620 Sensorelements ist die Verzögerungszeit nach Änderung der Ethanolkonzentration in der Luft zu entnehmen:

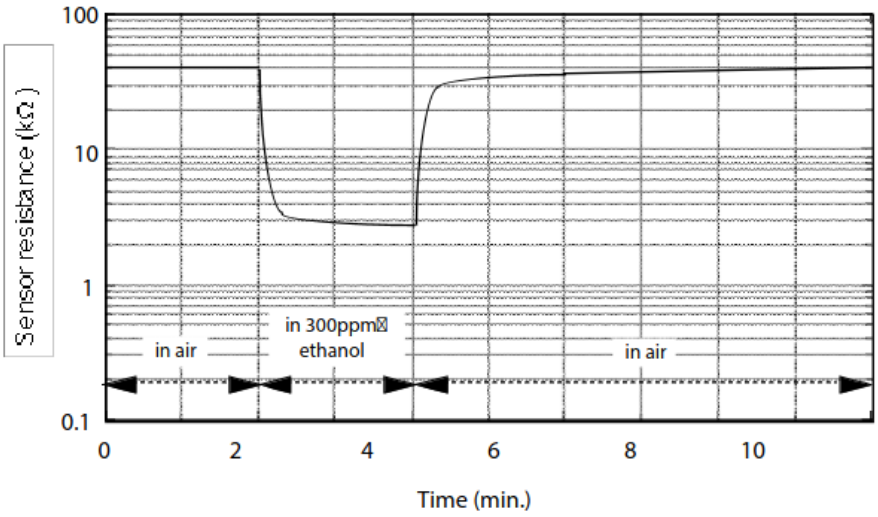


Abbildung 2: **Verzögerungszeit des TGS26260 Alkoholsensors** Übernommen nach [6], Anspruchverhalten des Sensors auf Konzentrationsänderungen

Es wird ein Algorithmus entwickelt, der einen gleitenden Mittelwert der vergangenen 10 Sensormesswerte bildet und die Steigung für das Intervall berechnet. Die Verzögerungszeit wird definiert als der Zeitpunkt δ T, an dem die Steigung positiv wird [12]. Eine erneute Umkehrung durch Sensorrauschen kann durch Anpassung der Messwertspanne angepasst werden.

2.5 Sondensoftware

Die Sensordaten werden durch einen Microcontroller (Arduino), welcher die Möglichkeit zur Steuerung von elektrischen Systemen gibt, erfasst und über eine USB-verknüpfte serielle Schnittstelle an einen Computer übermittelt. Ein Python-Skript liest die seriellen Daten aus und stellt sie über eine grafische Benutzeroberfläche (*graphical user interface*, GUI) für die Aufzeichnung und Analyse zur Verfügung. Das Skript verfügt unter anderem über Funktionen für die Kompensation von Querempfindlichkeiten, der Aufnahme von Kalibrierkurven, der Speicherung und Auslesung von Sensordaten, der Verbindung mit dem MFCS-OPC Server und für die automatische Berechnung der Methanolkonzentration aus den Sensordaten.

3 Material

Im Folgenden wird auf die in dieser Arbeit verwendeten Laborgeräte, Chemikalien und Software eingegangen.

3.1 Laborgeräte

Tabelle 19: Verwendete Geräte

Bezeichnung	**Hersteller**	**Modell**
Bioreaktor	Sartorius Stedim Biotech GmbH	BIOSTAT Aplus 2
Temperaturmodul	Sartorius Stedim Biotech GmbH	BB-8822002
Basiseinheit	Sartorius Stedim Biotech GmbH	-
Abgassensor	BlueSens gas Sensor GmbH	-
pH Sonde	Endres+Hauser	Memosens CPS171
pO_2 Sonde	Endres+Hauser	Memosens COS81D
Niveausonde	Sartorius Stedim Biotech GmbH	-
Autoklav	Systec GmbH	VX-150
Magnetrührer	Janke und Kunkel KG IKA Wert	IKA-COMBIMAG REO/RCO
pH-Messgerät	Sartorius Stedim Biotech GmbH	PB-20
Sicherheitswerkbank	Heraeus Instruments	HERA Safe
Schüttelinkubator	Gesellschaft für Labortechnik GmbH	3032
Zentrifuge	Eppendorf	Centrifuge 5417R
Trockenschrank	Heraeus Instruments	vacutherm
Analysenwaage	Sartorius Stedim Biotech GmbH	BA110S
Photometer	Thermo Fisher Scientific Inc	Genesys20
Vortexmischer	IKA Works Inc.	MS2 Minishaker

3.2 Chemikalien

Tabelle 20: Medienzusammensetzung

Name	**Einsatzstoff**	**Konzentration [g/L]**
PTM4 Stammlösung	Calciumsulfat Pentahydrat	2,00
	Schwefelsäure	1,00
	Natriumiodid	0,08
	Mangansulfat Monohydrat	3,00
	Borsäure	0,02
	Natriummolybdat Dihydrat	0,20
	Cobalt(II)-chlorid Hexahydrat	0,50
	Zinksulfat Heptahydrat	7,00
	Eisensulfat Heptahydrat	22,0
Biotin Stammlösung	Biotin >99%	0,20
Vorkultur Medium	Glycerin	15
	Natriumcitrat Dihydrat	1,47
	Magnesiumsulfat-Heptahydrat	4,10
	Calciumsulfat Dihydrat	0,35
Hauptkulturmedium	Kaliumsulfat	2,15
	Ammoniumsulfat	1,25
	Kaliumdihydrogenphosphat	6,43
	Glycerin	25
	Natriumcitrat Dihydrat	1,47
	Magnesiumsulfat-Heptahydrat	4,10
	Calciumsulfat Dihydrat	0,34
	Kaliumsulfat	2,15
	Ammoniumsulfat	1,25
	Kaliumdihydrogenphosphat	6,43

Tabelle 21: Verwendete Chemikalien

Bezeichnung	**Hersteller**	**CAS**	**Artikel Nr.**
Glycerin	Carl Roth GmbH	56-81-5	3783.5
Methanol 99%	VWR	67-56-1	20864.320
Phosphorsäure 85%	Carl Roth GmbH	7664-38-2	6366.1
Ammoniakwasser 25%	Carl Roth GmbH	1336-21-6	5460.2
Tri-Natriumcitrat Dihydrat	Carl Roth GmbH	6132-04-3	3580.2
Magnesiumsulfat-Heptahydrat	Carl Roth GmbH	10034-99-5	T888.3
Calciumsulfat Dihydrat	Carl Roth GmbH	10101-41-4	P741.2
Kaliumsulfat	Carl Roth GmbH	7778-80-5	CN79.1
Ammoniumsulfat	Carl Roth GmbH	7783-20-2	3746.4
Kaliumdihydrogenphosphat	Carl Roth GmbH	7778-77-0	3904.3
Calciumsulfat Pentahydrat	-	10101-41-4	-
Schwefelsäure	Riedel-de Hähn	7664-93-9	35347
Natriumiodid	-	7681-82-5	-
Mangansulfat Monohydrat	-	10034-96-5	-
Borsäure	Riedel-de Hähn	10043-35-3	31146
Natriummolybdat Dihydrat	-	10102-40-6	-
Cobalt(II)-chlorid Hexahydrat	Sigma-Aldrich	7791-13-1	C2644
Zinksulfat Heptahydrat	-	7746-20-0	-
Eisensulfat Heptahydrat	-	7782-63-0	-
Biotin >99% (HPLC)	-	58-85-5	-

3.3 Software-Komponenten

Python

Python ist eine flexible und dynamische Programmiersprache, die sich durch ihre elegante Syntax und Lesbarkeit auszeichnet. Diese Eigenschaften fördern die Entwicklung von klarem, wartungsfreundlichem Code. Python unterstützt mehrere Programmierstile, einschließlich objektorientierter, funktionaler und prozeduraler Programmierung und erleichtert dadurch ein modulares Software-Design. Durch die Integration von eigenständigen oder importierten Modulen können bestehende Programme um neue Funktionen erweitert werden. Desweiteren verfügt Python über eine reichhaltige Standardbibliothek und eine aktive Entwicklergemeinschaft, die eine Vielzahl von Drittanbieter-Paketen bereitstellt.

OpenOPC-python3x

OpenOPC-python3x ist ein Python-Paket, das die Kommunikation über das *open platform communication protocol* (OPC) ermöglicht, welches in der industriellen Automatisierung weit verbreitet ist und in Kombination mit dem Prozessleitsystem MFCS/win für die Prozessautomatisierung genutzt werden kann [7]. OPC ist ein gruppenbasiertes Client/ Server-Kommunikationsprotokoll, das ursprünglich für Windows-Plattformen entwickelt wurde und den Austausch Prozessdaten in Echtzeit zwischen Softwareanwendungen und Prozesssteuerungsgeräten, wie beispielsweise MFCS erleichtert. OpenOPC-python3x erweitert die Fähigkeit von Python, als OPC-Client zu fungieren, und ermöglicht somit eine effiziente und zuverlässige Datenkommunikation mit OPC Servern.

PyQT5

PyQt5 ist ein umfassendes Python-Paket für das Qt-Framework, das ermöglicht, plattformübergreifende grafische Benutzeroberflächen und

Desktop-Anwendungen zu erstellen. Mit PyQt5 können die Leistungsfähigkeit der Qt-Bibliothek genutzt und gleichzeitig die Einfachheit und Eleganz der Python-Programmiersprache beibehalten werden. Es bietet ein breites Spektrum an Funktionalitäten, das durch den integrierten QT Designer ermöglicht wird. Das manuelle Programmieren einer grafischen Nutzeroberfläche ist ebenfalls möglich.

PySerial

PySerial ist ein Python-Paket zur Kommunikation über serielle Schnittstellen und ermöglicht das Auslesen der Sensordaten des Arduino.

Pandas

Pandas ist ein Python-Paket zur effizienten Manipulation und Organisation von Daten. Es verbindet die schnellen Berechnungsfunktionen von Numpy mit tabellarischen Datenstrukturen und erweitert die Sondensoftware um Funktionen zur Verwaltung der Sensordaten und Implementierung von Rechenoperationen.

4 Methoden

4.1 Labormethoden

Probenahme

Das Wachstum der Hefekultur wurde *offline* verfolgt. Zur sterilen Probenahme diente ein Probenahmesystem mit dessen Hilfe ein Aliquot der Fermentationsbrühe in ein Probennahmegefäß überführt werden kann. Die Probenahme erfolgt über ein Tauchrohr, welches in die Fermentationsbrühe reicht. Über einen abklemmbaren Schlauch ist das Probenahmerohr mit einem Probenahmegefäß aus Glas verbunden. Am Probenahmegefäß sind zwei weitere Schläuche befestigt. Zum einen ist ein Schlauch mit einem Sterilfilter versehen, der es ermöglicht eine Spritze anzuschließen, um die Fermentationsbrühe aus dem Reaktor in das Probenahmegefäß zu befördern. Der zweite Schlauch ermöglicht die Überführung der Probe vom Probenahmegefäß zum Auffanggefäß (z.B. 15 mL Sammelgefäß) Mittels Spritze wird ein Unterdruck in dem Probenahmegefäß erzeugt, was dazu führt, dass Kulturbrühe in das Gefäß befördert wird. Nachdem das Probenahmegefäß hinreichend gefüllt ist, wird die reaktorseitige Schlauchklemme geschlossen und die andere Seite, die zum Probensammelgefäß führt, geöffnet. Mithilfe der Spritze wird anschließend ein Überdruck im Probenahmegefäß erzeugt, sodass der Inhalt in das beschriftete Sammelgefäß befördert wird.

Biotrockenmassebestimmung

Das Gewicht der Biomasse wurde nach der Fermentation bestimmt und stellt im Allgemeinen nach der Umrechnung in die Zellkonzentration (c_x) einen prozessübergreifenden Vergleichswert in der Biotechnologie dar. Hierfür wurden 1,5 mL Mikroreaktionsgefäße beschriftet und über Nacht getrocknet, bevor die Leergewichte notiert wurden. Während des Fermentationsprozesses, beginnend mit der Inokulation des Bioreaktors, wurde in regelmäßigen Abständen ca. 5 mL der Zellsuspension

über ein Tauchrohr, mit Hilfe einer Spritze entnommen. Davon wurden je 2 x 1 mL auf die beschrifteten Mikroreaktionsgefäße aufgeteilt und drei Tage geöffnet im Trockenschrank bei 65 °C gelagert, um eine vollständige Evaporation des restlichen Wassers sicherzustellen. Im Anschluss wurden die Mikroreaktionsgefäße geschlossen und im Exsikkator auf Raumtemperatur abgekühlt, bevor das Trockengewicht der Biomasse nach dem Wiegen der Gefäße notiert wurde.

Messung der optischen Dichte

Das Zellwachstum im Bioreaktor wurde *offline* über eine regelmäßige Probennahme mit Hilfe eines Photometers bei 600 nm verfolgt. Dazu wurde eine Küvette mit 1 mL Zellsuspension befüllt, bei der entsprechenden Wellenlänge durchleuchtet und die Intensitätsabnahme als Absorption gemessen. Beim Überschreiten einer optischen Dichte von 0,6 wurde die Probe entsprechend (mit VE-Wasser) verdünnt und erneut gemessen. Die Wellenlänge wurde hierbei so gewählt, dass die Intensitätsabnahme durch, in der Suspension befindliche, Zellen bewirkt wird und nicht durch das Medium, dessen Zusammensetzung sich im zeitlichen Verlauf einer Fermentation ändern kann. Nach dem Lambert-Beerschen Gesetz besteht ein linearer Zusammenhang für Absorptionen (zwischen 0,1 und 0,6) und der Zellkonzentration, sodass sich diese mit folgender Formel errechnen lässt:

$$c_{xL,OD} = K_{\frac{X}{OD}} \cdot OD_{600} \tag{20}$$

Tabelle 22: Variablen für Gleichung 20, Berechnung der Zellkonzentration aus der optischen Dichte

Symbol	Beschreibung	Einheit
$c_{xL,OD}$	Zellkonzentration	[g L^{-1}]
$K_{\frac{X}{OD}}$	Proportionalitätsfaktor	[g L^{-1} AU^{-1}]
OD_{600}	optische Dichte bei 600 nm	[AU]

Ein Wert von $K_{\frac{X}{OD}} = 0,53 gl^{-1} AU^{-1}$ wurde empirisch bestimmt.

4.2 Vorversuche

Dreistufiger Fermentationsprozess

Für die Fermentationen wurde der Reaktor vorbereitet und mit 1,2 Liter des in Tab. 20 gezeigtem Medium angesetzt. Anschließend wurde der Fermenter für 20 min bei 120 °C und 2 bar autoklaviert. Die Supplemente wurden aufgrund ihrer Thermoinstabilität sterilfiltriert und nach dem Autoklavieren über einen Sterilport zugegeben. Der Reaktor wurde mit 200 mL der Vorkultur inokuliert, was den dreistufigen Fermentationsprozess gestartet hat. In der Produktionsphase wurde über einen Sterilport automatisiert Methanol zugegeben.

Schwebekörperdurchflussmesser

Der verwendete Schwebekörperdurchflussmesser verfügt über eine einheitenlose Skala. Zur Bestimmung des Trägergasstroms wurde daher ein Wasserbad vorbereitet, in dem ein 100 mL Messzylinder vollständig eingetaucht wurde.Anschließend wurde der Messzylinder senkrecht positioniert, sodass sich die Wassersäule im Messzylinder oberhalb des Füllstandes des Wasserbads befindet. Der Ausgangsschlauch des Rotameters wurde so positioniert, dass entweichende Gasblasen in den Messzylinder aufsteigen und die sich darin befindende Wassersäule verdrängen. Gemessen wurde die Zeit, bis 100 mL verdrängt wurden. Insgesamt wurde der Versuch pro Skala-Einstellung dreifach wiederholt. Es wurden verschiedene Einstellungen (50, 80, 100, 120) bei einem konstanten Trägergas-Vordruck von 2,0 bar miteinander verglichen, um eine Formel zu entwickeln, die eine Angabe des Trägergasstroms in Liter pro Minute ermöglicht.

4.3 Kalibrierung der Methanolsonde

Kalibrierung im Bioreaktor

Um den Zusammenhang zwischen Sensordaten und der Methanolkonzentration bestimmen zu können, wurden Kalibrierungen der Metha-

nolsonde bei unterschiedlichen Prozessbedingungen durchgeführt. Bei einem konstanten Trägergasvordruck von 2,0 bar wurde mittels Nadelventil des Schwebekörperdurchflussmessers ein Wert von 80 auf der Skala am Schwebekörperdurchflussmesser eingestellt. Dazu wurde der bestückte Bioreaktor mit ausreichend vollentsalztem Wasser (VE-Wasser) befüllt, damit die Membran der Methanolsonde vollständig in die Flüssigkeit eintauchte. Anschließend wurden dieselben Reglerparameter, wie während einer Fermentation eingestellt. Nachdem die Messkammer der Sonde sich equilibriert hatte und das Wasser im Reaktor auf 30 °C aufgeheizt ist, erfolgte die erste Zugabe einer abgewogenen Menge Methanol mithilfe einer Spritze über ein Inokulationsport im Deckel. Der Messwertszeitpunkt wurde protokolliert, nachdem ein scheinbar stabiler Endwert für den Messwiderstand erreicht wurde. Insgesamt erfolgten sechs Zugaben, um die Volumenkonzentrationen 0,5 %, 0,8 %, 0,9 %, 1,0 %, 1,1 % und 1,3 % einzustellen. Die Wahl dieser Konzentrationswerte erfolgte mit der Absicht, den in der Produktionsphase des Modellorganismonzentration angestrebten Regelwert für den Methanolgehalt von 1,0 % bestmöglich abzubilden. Die Auswertung der Sensordaten und die Berechnung der Regressionsmodelle erfolgte mit den entwickelten Software-Tools. Für die Berechnung der Konzentration wurden lineare Kalibriermodelle gewählt.

Kalibrierung im Becherglas

Damit die Umsetzbarkeit einer Kalibrierung außerhalb des Reaktors und dessen Übertragbarkeit auf Prozessbedingungen ermittelt werden kann, wurde ein 1 L Becherglas mit einem Rührfisch versehen und mit VE-Wasser befüllt und die Sonde wurde mittels Stativ in das Becherglas abgesenkt, bis die Membran vollständig in das Wasser eintauchte. Bei einem konstanten Trägergasvordruck von 2,0 bar wurde mittels Nadelventil des Schwebekörperdurchflussmessers ein Wert von 80 auf der Skala am Schwebekörperdurchflussmesser eingestellt. . Nachdem die Messkammer sich equilibriert hatte, erfolgten wie zuvor beschrieben sechs Zugaben um die Konzentrationen 0,5 %, 0,8 %, 0,9 %, 1,0 %,

1,1 % und 1,3 % einzustellen. Der Messwertszeitpunkt wurde jeweils protokolliert, nachdem ein stabiler Endwert für den Messwiderstand erreicht wurde. Die Temperatur des Wassers wurde mithilfe eines Pt-1000 Thermometers nach dem Ablesen des jeweiligen Messwertes erfasst.

Kalibrierung vor und nach der Autoklavierung

Um den Einfluss des Autoklavierens auf das Messerverhaltens zu untersuchen, wurde zunächst eine Kalibrierung im Bioreaktor wie zuvor beschrieben durchgeführt. Anschließend wurde der Bioreaktor gereinigt und erneut mit derselben Menge an VE-Wasser befüllt. Daraufhin wurde der bestückte Bioreaktor bei 121 °C für 20 min autoklaviert und anschließend auf Raumtemperatur abgekühlt. Im Anschluss wurde der Kalibrierungsversuch wiederholt, um die Reproduzierbarkeit einer Kalibrierung vor einer Fermentation zu untersuchen. Die Kalibrierungen im Becherglas, sowie die Kalibrierung vor dem Autoklavieren werden anschließend zur rückwirkenden Berechnung der Konzentration aus dem Sensordatenverlauf genutzt, um eine Abweichung zu berechnen und die Sondenkalibrierung zu bewerten.

Einfluss des Trägergasstromes auf das Messverhalten

Um den Einfluss der Trägergasströmungsgeschwindigkeit auf das Messverhalten der Sonde zu untersuchen, wurde zunächst der Bioreaktor mit der Sonde aufgebaut und die Betriebsbedingungen wurden eingestellt. Bei einem konstanten Trägergasvordruck von 2,0 bar wurden drei verschiedene Einstellungen (50, 80, 120) mithilfe der Skala am Schwebekörperdurchflussmesser eingestellt. Nachdem sich konstante Sensorwerte für die Temperatur und relative Feuchtigkeit des Trägergases eingestellt haben und der Messwiderstand einen stabilen Endwert anzeigte, wurde eine vorher abgewogene Menge an Methanol mit einer Spritze über einen Inokulationsport im Deckel in die vorgelegte Flüssigkeit injiziert. Nachdem die Messkammer sich erneut equilibriert hat, wurde der entsprechende Messwertszeitpunkt proto-

kolliert und eine erneute Zugabe erfolgte. Es wurden je Trägergasströmungsgeschwindigkeit sechs Aufkonzentrierungen durchgeführt, um die Volumenkonzentrationen 0,5 %, 0,8 %, 0,9 %, 1,0 %, 1,1 % und 1,3 % einzustellen.

Einfluss der Medientemperaturen auf das Messverhalten

Um eine Medientemperaturkompensation zu entwickeln, ist es zu Beginn notwendig die Intensität der Querempfindlichkeiten zu erfassen. Es wurde hierfür eine Konzentration von 0,5 Vol% im auf 18 °C temperierten Bioreaktor vorgelegt und im ersten Schritt die Temperatur auf 21 °C und nach Erreichen eines stabilen Messwertes anschließend auf 25 °C erhöht. Es wurden ebenfalls Kalibrierkurven bei unterschiedlichen Temperaturen unter sonst identischen Bedingungen aufgezeichnet und anschließend miteinander verglichen.

4.4 Aktualisierung der Sondensoftware

Stabilitätstest der Sondensoftware

Für den Stabilitätstest wurde der Reaktor vorbereitet und auf Prozessbedingungen eingestellt. Anschließend wurde dieser Zustand für mindestens fünf Tage gehalten. Die Funktionalität wurde im Nachgang bewertet

Aktualisierung der Sondensoftware

Da die Wartung für Python 2 eingestellt wurde, musste eine entsprechende Aktualisierung des Programmcodes nach Umstellung auf Python 3 stattfinden, damit auswertbare Experimente mit der Sonde stattfinden können. Im Anschluss wurde die Software um sämtliche Komponenten erweitert und die Bedienungsoberfläche für zukünftige Experimente angepasst. Hierfür wurde ein Skript für die automatisierte Auswertung und anschließende grafische Darstellung erstellt.

Automatisierte Sondenqualifizierung

Es wurde ein Skript (s. Anhang) geschrieben, um nach Angabe von Zeitpunkten und injizierten Mengen an Methanol eine Verzögerungszeit und prozentuale Widerstandsänderung zu berechnen. Diese Werte werden als Anhaltspunkte für die Messempfindlichkeit und Ansprechzeit der Sonde genutzt. Die Berechnung der Verzögerungszeit erfolgt durch Bildung eines Mittelwert der 10 vergangenen Einträge in der log-Datei und der anschließenden Berechnung der Steigung über den Zeitraum. Sobald die Steigung null oder positiv wird, wird dies als Verzögerungszeit erkannt. Die prozentuale Widerstandsänderung wird durch den Quotienten des normierten Widerstandes vor Injektion und nach Verstreichen der Verzögerungszeit berechnet.

5 Ergebnisse

5.1 Vorversuche

Schwebekörperdurchflussmesser

Die Untersuchung des einheitenlosen Rotameters bei den Skalenwerteinstellungen 50, 80, 100, 120 führte zu folgender Gleichung (21) für die Umrechnung des Skalenwertes x in eine Trägergasströmungsrate F_G in mL min^{text-1}:

$$F_G(x) = 0.5 \cdot x - 9.6 \tag{21}$$

Fermentation im Bioreaktor

Es wurden vier Fermentationen im Bioreaktor durchgeführt. Zwei wurden während der *fed-batch*-Phase beendet (F3, F4) und zwei Fermentationen wurden einschließlich Produktionsphase durchgeführt (F2.F3). Eine Fermentation ist vor Beginn der *batch*-Phase beendet worden.

Tabelle 23: Wachstumsraten der Vorkulturen aus den Fermentationen

Kürzel	**Wachstumsrate μ [h^{-1}]**
F2	0.119
F3	0.194
F4	0.318
F5	0.179

Die folgende Tabelle fasst die ermittelten Wachstumsraten der jeweiligen Fermentationen in ihren einzelnen Phasen zusammen. Dabei sind nicht für jede Phase Daten vorhanden.

Tabelle 24: Wachstumsraten der Fermentationen in unterschiedlichen Phasen

	***batch* [h^{-1}]**	***fed-batch*[h^{-1}]**	**Produktion [h^{-1}]**
F2	0.2820	-	0.0009
F3	-	0.0735	0.0002
F4	0.2954	-	-
F5	0.2648	0.1784	-

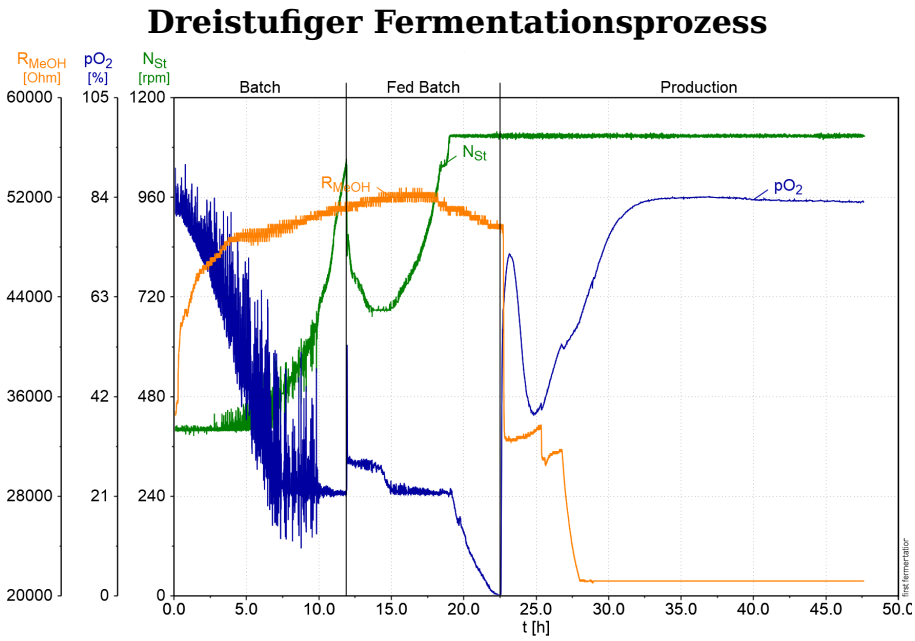


Abbildung 3: **Zeitlicher Verlauf der Prozessparameter während einer dreistufigen Fermentation** Abgebildet sind der gemessene Sondenwiderstand ohne Kompensation R_S (hier: R_{MeOH}), die partiale Sauerstoffsättigung pO_2 und die Rührerdrehzahl: N_{st} während einer dreistufigen Fermentation (F2).

Während der Fermentation (Abb. 3) verlief die *batch*-Phase wie erwartet, bis das Glycerin im Medium aufgebraucht war. Dies machte sich an einem Anstieg des pO_2-Wertes bemerkbar, da der Metabolsimus der Zellen durch Abwesenheit eines Substrats eingeschränkt wurde. Als Resultat sinkt die Rührerdrehzahl, was die algorithmische Erkennung des *batch*-Endes auslöst. Das Ende der *batch*-phase erfolgte um t = 12 h und ab t = 22,5 h wurde das Ende der *fed-batch*-Phase manuell eingeleitet. Die Produktionsphase begann bei t = 30 h. und verlief für ca. 20 h. Der Verlauf des $p0_2$-Wertes zeigt, wie die Zellen über den Verlauf von ca. 3 h ihren Metabolismus an das neue Substrat anpassen, woraufhin die Methanolkonzentration im Medium erhöht wird. Es ist zu beobachten, wie der Sensorwiderstand R_{MeOH} nach Einstellung der Induktionskonzentration bei ca. 35 kΩ verbleibt. Im Verlauf der Produktionsphase ist zu erkennen, wie der Messwert bei jeder Zugabe von Methanol sinkt auf einen Endwert von ca. 21 $k\Omega$.

5.2 Kalibrierung der Methanolsonde

Im Nachfolgenden sind die Abbildungen 4 bis 21b zu sehen. Sie zeigen die Sondenkalibrierung in verschiedenen Umgebungen, sowie die Empfindlichkeit und Verzögerungszeit bei verschiedenen Prozessbedingungen und die Benutzeroberfläche der Methanolsondensoftware

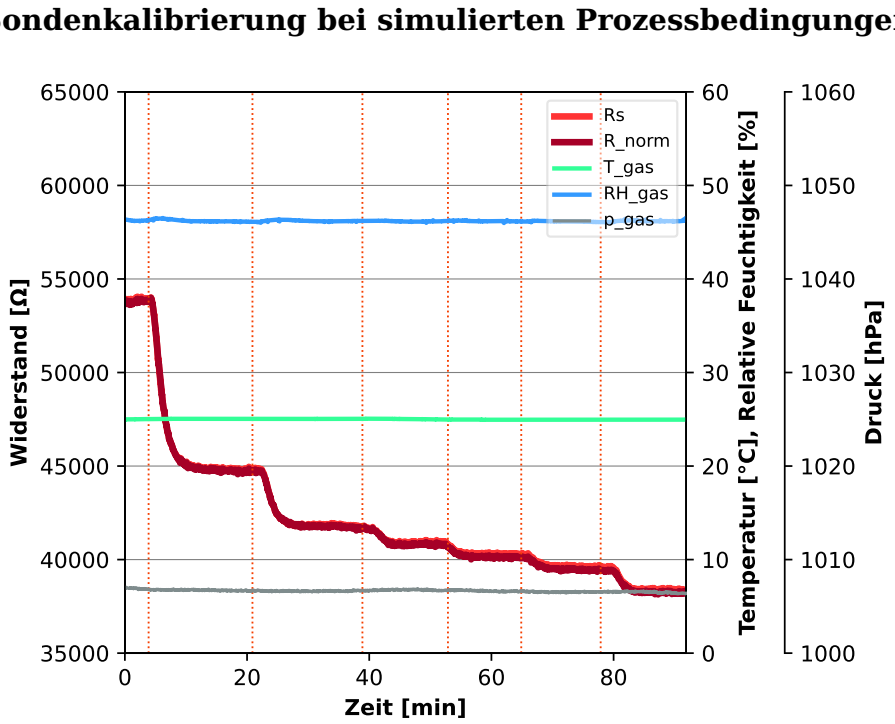


Abbildung 4: **Sondenkalibrierung bei simulierten Prozessbedingungen** Darstellung der Sensordaten für den Alkoholsensor (links: Sensorwiderstand R_S, normierter Sensorwiderstand R_{Norm}) und den Kombisensor (rechts: Trägergastemperatur θ_{gas}, relative Trägergasfeuchtigkeit RH_{gas}, und der Trägergasdruck p_{gas})

In Abb. 4 ist der zeitliche Verlauf der Sensordaten während einer Kalibrierung unter simulierten Prozessbedingungen (F_{gas}: 28 ml/min, θ_{med}: 30°C, N_{st}: 400 rpm, F_{air}: 1,5 L/min, d: 0,3 mm) dargestellt. Die Injektionen wurden jeweils steril über einen Zugabeport durch ein Septum durchgeführt und sind als vertikale Linien eingezeichnet. Es erfolgten sechs Aufstockungen um Volumenkonzentrationen von 0,5 %, 0,8 %, 0,9 %, 1,0 %, 1,1 %, 1,3 % Methanol einzustellen. Der gemessene Sensorwiderstand R_S verläuft identisch mit dem auf 20 °C und 65 % relative Feuchtigkeit normierten Widerstandswert R_{Norm}. Über den Verlauf der Kalibrierung zeigen die Umgebungsbedingungen tg_{gas}, RH_{gas} und p_{gas} einen konstanten Verlauf.

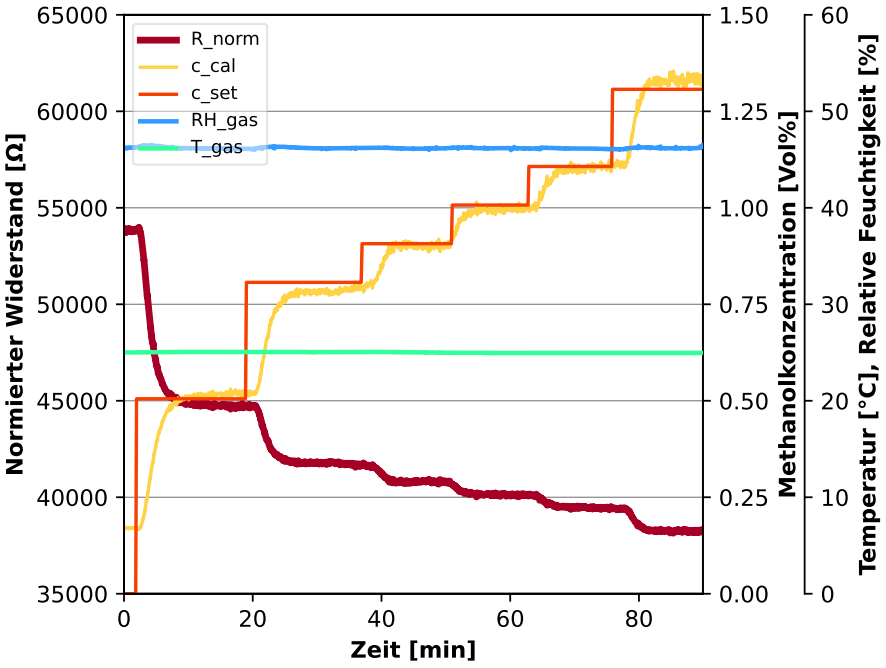


Abbildung 5: **Gemeinsame Darstellung zeitlicher Verläufe von Sensor- und Kalibrierdaten** Prozessbedingungen während der Kalibrierung: Trägergasstrom F_tgas = 28 ml/min, Medientemperatur θ_{med} = 30 °C, Rührerdrehzahl N_{st} = 400 rpm, Begasungsrate F_{air} = 1,5 L/min, Silikonschlauchdicke d = 0,3 mm.

In Abb. 5 werden die Sensordaten im zeitlichen Verlauf mit der vorgelegten Konzentration überlagert dargestellt. Aufgetragen sind der normierte Widerstand und die Trägergasparameter, sowie die vorgelegte Konzentration. Ebenfalls abgebildet wird die errechnete Konzentration aus dem normierten Widerstand über dieselbe Kalibrierungskurve, die aus dem Experiment hervorging. Nach der algorithmisch bestimmten Verzögerungszeit der Methanolsonde zu dem Kalibrierzeitpunkt von $\bar{\Delta t}$ = 10,15 min, nimmt die errechnete Konzentration einen stabilen Endwert an. Die errechnete Messabweichung schließt die Verzögerungszeit ein und liegt durchschnittlich 0,05 Vol% MeOH (0,4 g/L) unterhalb der vorgelegten Konzentration.

Sondenkalibrierung bei im 0,5 L Becherglas bei Raumtemperatur

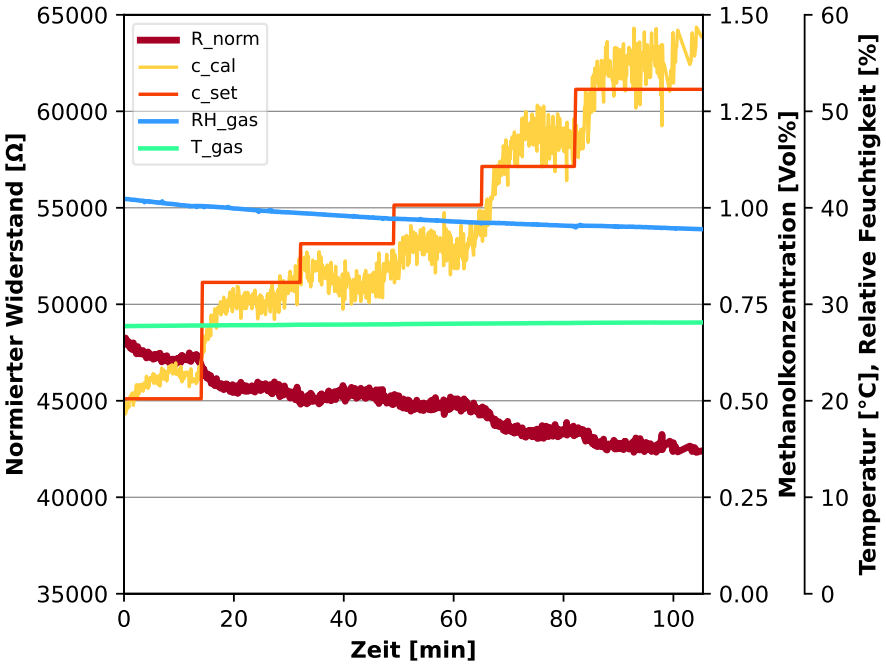


Abbildung 6: **Berechnete Konzentration während einer Kalibrierung im 0,5 L Becherglas bei Raumtemperatur** Die erste Dosis Methanol wurde hinzupipettiert, bevor die Programmroutine gestartet war. Messbedingungen während der Kalibrierung: θ_{med} = 26,5 °C, d = 0,5 mm, V_L = 0,2 L, schwache Durchmischung. Folgende Messparameter wurden algorithmisch ermittelt: $\bar{\Delta}t$ = 10,47 min, $\Sigma\Delta\ R$ = 12,5%, $\Sigma\Delta\ R/\Sigma\Delta$ c = 9,54 %/%.

Um die Möglichkeit einer externen Kalibrierung vor einem Kultivierungsprozess zu testen, wurde die Kalibrierungsmessung in einem mit Magnetrührer versehenem 0.5 L Becherglas durchgeführt. Es ist in Abb. 6 zu erkennen, wie die Kalibrierergebnisse einer externen Kalibrierung bei Raumtemperatur in der rückwirkenden Berechnung der Methanolkonzentration aus den Sensordaten während der Kalibrierung genutzt wurden. Der Messwert zeigt ein verrauschtes Signal, mit Signalunterbrechungen bei t = 110 min und t = 125 min. Die Temperatur wurde bei jeder Zugabe gemessen lag bei θ_{med} = 25.5 °C ±0.2 °C.

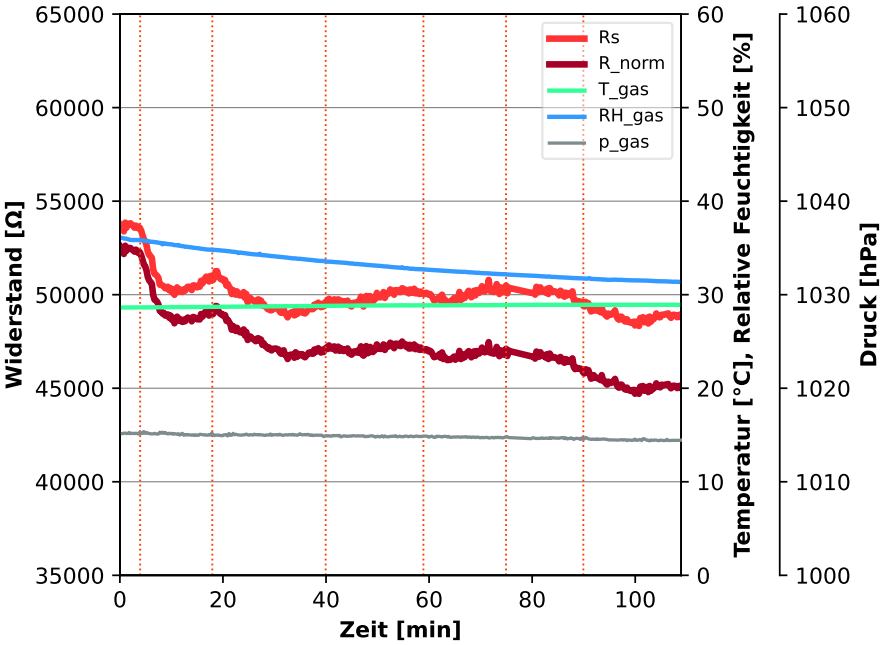


Abbildung 7: **Sensordatenverlauf während einer externen Kalibrierung im 1,0 L Becherglas** Messbedingungen während der Kalibrierung: d: 0,5 mm, V_L = 1 L, starke Durchmischung. Folgende Messparameter wurden algorithmisch ermittelt: $\bar{\Delta} t$ = 16,34 min, $\Sigma\Delta R$ = 14,69 %, $\Sigma\Delta R/\Sigma\Delta c$ = 11,21 %/%.

In Abb. 7 wird der Sensordatenverlauf des Kalibrierversuchs im 1.0 L Becherglas bei starker Durchmischung gezeigt. Es lässt sich beobachten, wie der Messwiderstand nach Injektion wieder beginnt zu steigen, nachdem eine Verzögerungszeit verstrichen ist. Bei der dritten Aufstockung liegt der Messwert R_{norm} bei 47000 $\Omega \pm 500\ \Omega$ und zeigte keine Veränderung durch die Methanolzugabe, die sich von dem Messrauschen hervorhebt. Die Temperatur wurde bei jeder Zugabe mit einem Thermometer gemessen und lag bei θ_{med}=25.5 ° C ±0.2 ° C.

Empfindlichkeit und Verzögerungszeit bei simulierten Prozessbedingungen

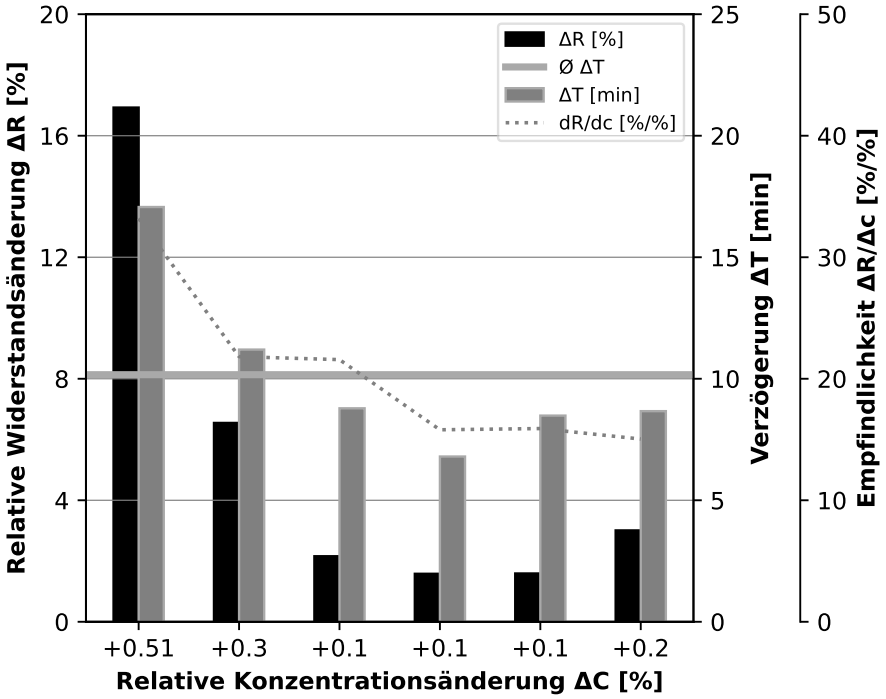


Abbildung 8: **Verzögerungszeit und Empfindlichkeit bei simulierten Prozessbedingungen**
Prozessbedingungen während der Kalibrierung: F_{gas}: 28 ml/min, θ_{med}: 30° C, N_{st}: 400 rpm, F_{air}: 1,5 L/min, d: 0.3 mm.

Dargestellt in Abb. 8 sind die relativen Widerstandsänderungen ΔR und die Verzögerungszeiten Δt bei den relativen Änderungen der Methanolkonzentrationen von je 0.5 %, 0.8 %, 0.9 %, 1.0 %, 1.1 %, 1.3 % v/v während eines Kalibrierexperiments unter simulierten Prozessbedingungen. Das Verhältnis der Widerstands- zur Konzentrationsänderung $\Delta R/\Delta c$ beträgt bei der ersten Injektion $\Delta R_{0.5}/\Delta c_{0.5} = 35\%/\%$ und sinkt bis zu $\Delta\ R_{0.2}/\Delta c_{0.2}$ = 15%/% bei der letzten Injektion. Das Gesamtverhältnis $\Sigma\Delta R/\Sigma\Delta c$ beträgt bei einer Endkonzentration von 1.3% $\Sigma\Delta R/\Sigma\Delta c = 24.56\%/\%$.

Nach einer sprungförmigen Änderung der Konzentration beträgt die algorithmisch ermittelte Verzögerungszeit $\bar{\Delta t}\ =\ 10.15$ min.

5.2.1 Trägergaskonfiguration

Sensordatenverlauf während einer Kalibrierung bei verringertem Trägergasstrom

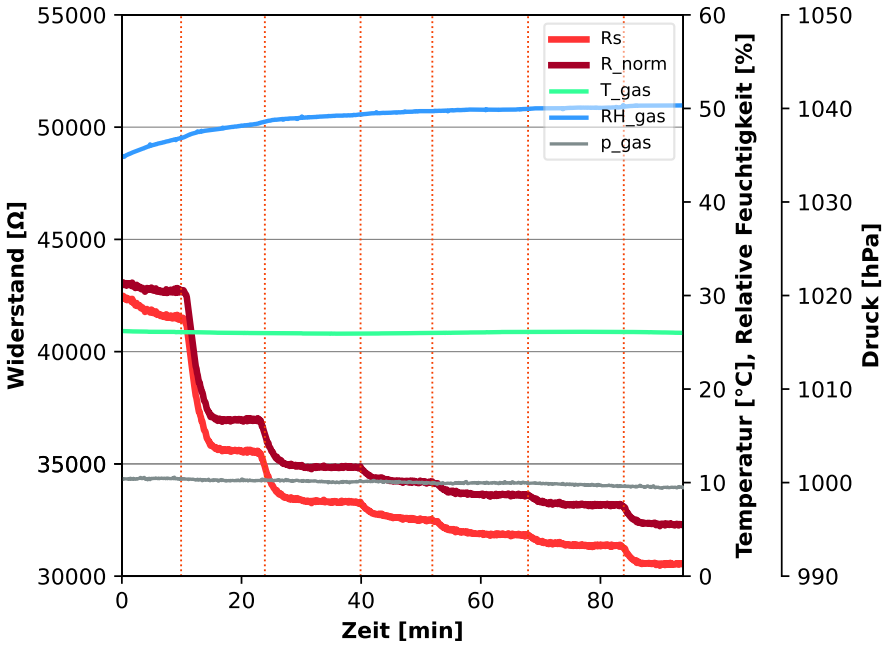
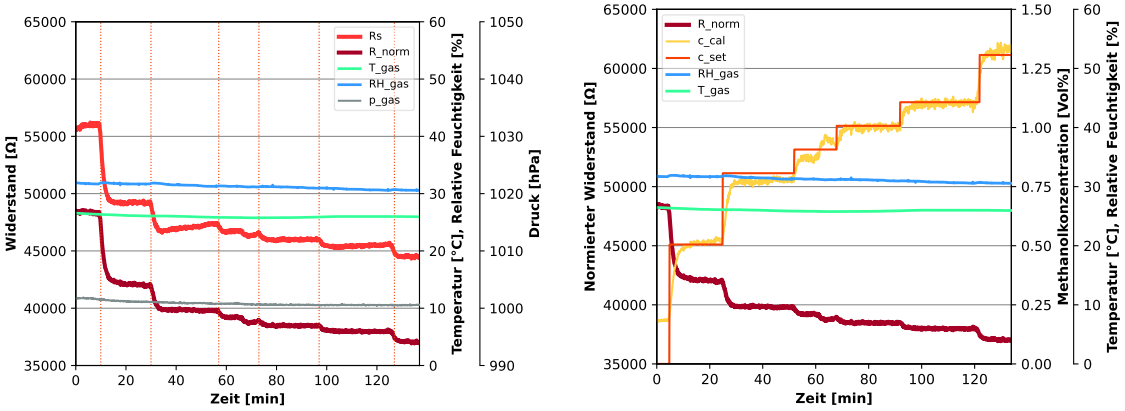


Abbildung 9: **Sondenkalibrierung bei Rotametereinstellung 50** Prozessbedingungen während der Kalibrierung: F_{gas}: 16,8 mL/min, θ_{med}: 30 °C, N_{st}: 400 rpm, F_{air}: 1,5 L/min, d: 0.3 mm Folgende Messparameter wurden algorithmisch als Maß für die Empfindlichkeit und Verzögerungszeit ermittelt: $\overline{\Delta t}$ = 12,45 min, $\Sigma\Delta\ R$= 26,41%, $\Sigma\Delta\ R/\Sigma\Delta\ c$ = 20,16 %/%

In Abb. 9 wird der Sensordatenverlauf während einer Kalibrierung bei einem Trägergasstrom von 16,8 mL/min dargestellt. Die Injektionen wurden steril über einen Zugabeport mit Septum durchgeführt. Es erfolgten sechs Aufstockungen um Volumenkonzentrationen von 0,5 %, 0,8 %, 0,9 %, 1,0 %, 1,1 %, 1,3 % Methanol einzustellen. Über die Dauer des Experiments steigt die relative Feuchtigkeit des Trägergases von 45 % auf 50 %. Das Verhältnis der Widerstands- zur Konzentrationsänderung sinkt von $\Delta R_{0.5}/\Delta c_{0.5}$ = 30 %/% auf $\Delta R_{0.2}/\Delta c_{0.2}$ = 15 %/%. Der Messbereich verschiebt sich im Vergleich zu einem höheren Trägergasstrom (Abb. 8) nach unten und beginnt für c_{MeOH} = 0 Vol% bei ca. 43 $k\Omega$ und endet bei ca. 31 $k\Omega$ für eine Konzentration von 1,3 Vol%.

Sensordatenverlauf während einer Kalibrierung bei erhöhtem Trägergasstrom



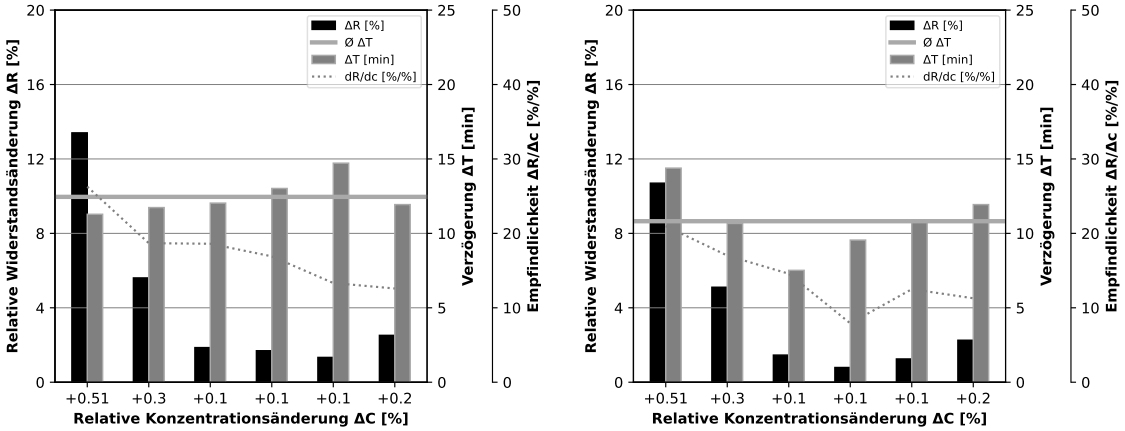
(a) Sensordaten

(b) Konzentrationsverlauf

Abbildung 10: **Kalibrierungsverlauf bei einem Trägergasstrom von 52,2 mL/min** Prozessbedingungen während der Kalibrierung: F_{gas} = 52,2 mL/min, θ_{med} = 30 °C, N_{st} = 400 rpm, F_{air} = 1,5 L/min, d = 0,3 mm.

Dargestellt in Abb. 10 wird der Sensordatenverlauf während einer Kalibrierung bei einem Trägergasstrom von 52,2 mL/min, gemeinsam mit der rückwirkend-berechneten Konzentrationen während der Kalibrierung. Die Injektionen wurden steril über einen Zugabeport mit Septum durchgeführt. Es erfolgten wie zuvor beschrieben sechs Aufstockungen. Der gemessene Sensorwiderstand nimmt im Gegensatz zum normierten Widerstand zwischen t = 0,5 h und t = 1 h eine positive Steigung an (Abb. 10a). Das Verhältnis der Widerstands- zur Konzentrationsänderung sinkt über den Verlauf der Kalibrierung von $\Delta\ R_{0,5}/\Delta\ c_{0,5}$ = 22 %/% auf $\Delta\ R_{0,2}/\Delta\ c_{0,2}$ = 10 %/%.

Empfindlichkeit und Verzögerungszeit bei veränderten Trägergasströmen



(a) 16,8 mL/min

(b) 52,2 mL/min

Abbildung 11: **Vergleich algorithmisch ermittelter Sondenparameter bei verschiedenen Trägergasströmen** (11a): Prozessbedingungen während der Kalibrierung: F_{gas}: 16,8 mL/min, θ_{med}: 30 °C, N_{st}: 400 rpm, F_{air}: 1,5 L/min, d: 0,3 mm. (11b): Prozessbedingungen während der Kalibrierung: Prozessbedingungen während der Kalibrierung: F_{gas}: 52,2 mL/min, θ_{med}: 30 °C, N_{st}: 400 rpm, F_{air}: 1,5 L/min, d: 0,3 mm.

Dargestellt in Abb. (11) sind die relativen Widerstandsänderungen Δ R und Verzögerungszeiten Δ t bei den Trägergasströmen 16,8 mL/min und 52,2 mL/min. Die ermittelte Verzögerungszeit $\bar{\Delta}t$ beträgt für im ersten Fall $\bar{\Delta}t = 12$ min und für 52,2 mL/min. Die totale Widerstandssänderung ΔR liegt bei ΔR = 26,41 % für den verringerten Trägergasstrom und sinkt auf Δ R = 21,56 % bei der höheren Trägergasströmungsgeschwindigkeit. Der Quotient aus Δ R und Δ c nimmt für zunehmende Methanolkonzentrationen in beiden Fällen ab. Es ist zu beobachten, dass die automatisch-erkannten Verzögerungszeiten nicht konstant sind.

Vergleich der Kalibriergeraden unterschiedlicher Trägergasströme

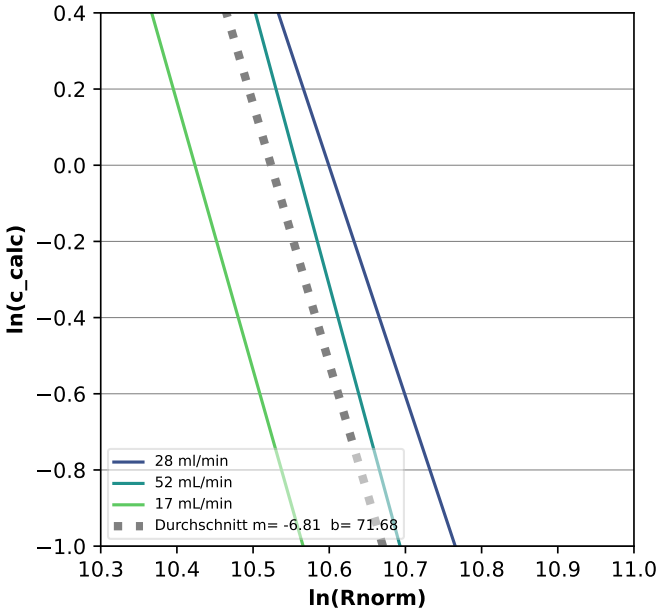
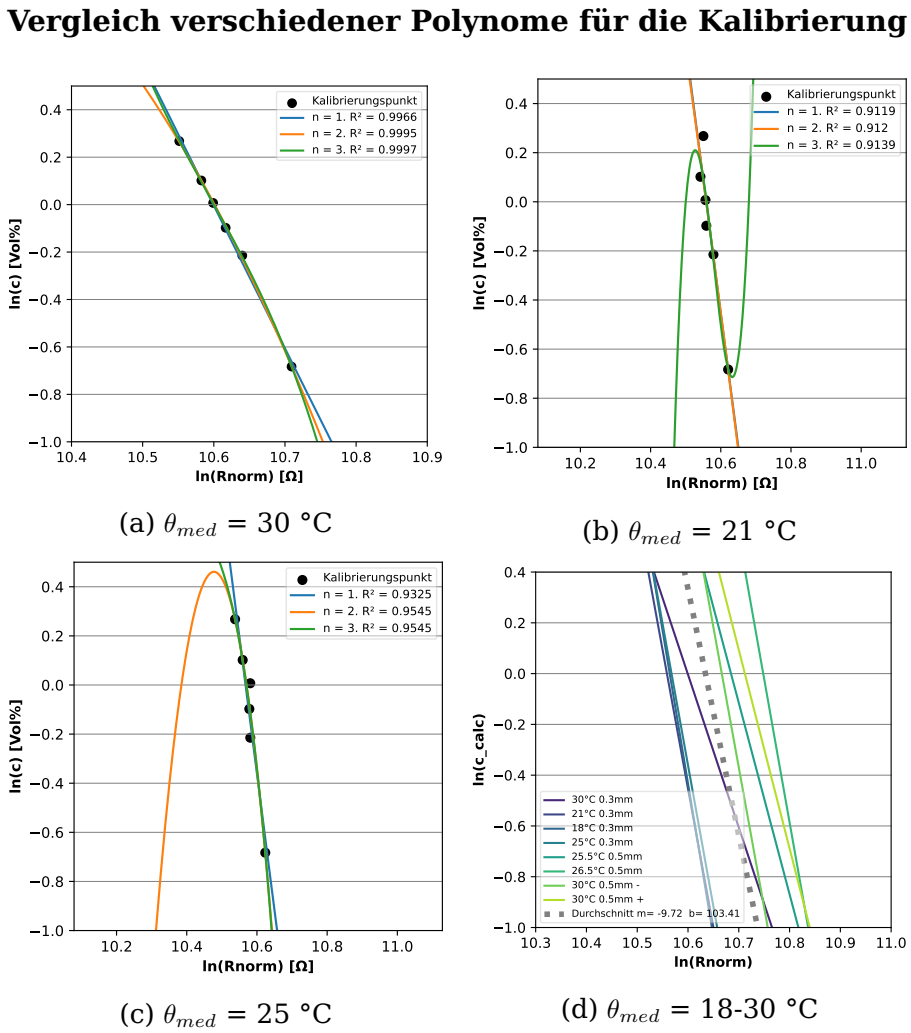


Abbildung 12: **Vergleich der Kalibrierpolynome bei verschiedenen Trägergasströmungsgeschwindigkeiten** Doppelt-logarithmische Auftragung der Kalibrierungsgeraden für drei unterschiedliche Trägergasströme. Prozessbedingungen während der Kalibrierung: F_{gas} = 28 mL/min, θ_{med} = 30 °C, N_{st} = 400 rpm, F_{air} = 1,5 L/min, d = 0.3 mm.

In Abb. 12 ist der natürliche Logarithmus der berechneten Konzentration $ln(c_{calc})$ gegen den natürlichen Logarithmus des normierten Widerstands $ln(R_{Norm})$ für die Kalibriergeraden der verschiedenen Trägergasströme aufgetragen. Abgebildet wird ein prozessrelevanter Konzentrationsbereich zwischen 0,36 Vol% und 1,5 Vol% (bzw. 2,8 g/L bis 11,7 g/L), sowie der Messwertbereich für den normierten Messwiderstand der Methanolsonde von 35 $k\Omega$ bis 55 $k\Omega$. Es ist zu sehen, dass die Steigungen der Kalibrierungsgeraden für die drei Trägergasströme F_{gas} = 18 mL/min, F_{gas} = 28 mL/min und F_{gas} = 52 mL/min nahezu identisch sind.

Die Steigung m für die durchschnittliche Kalibriergerade beträgt m = -6,81 bei einem Achsenversatz b von b = 71,68.

5.2.2 Auswahl des Polynoms



(a) θ_{med} = 30 °C

(b) θ_{med} = 21 °C

(c) θ_{med} = 25 °C

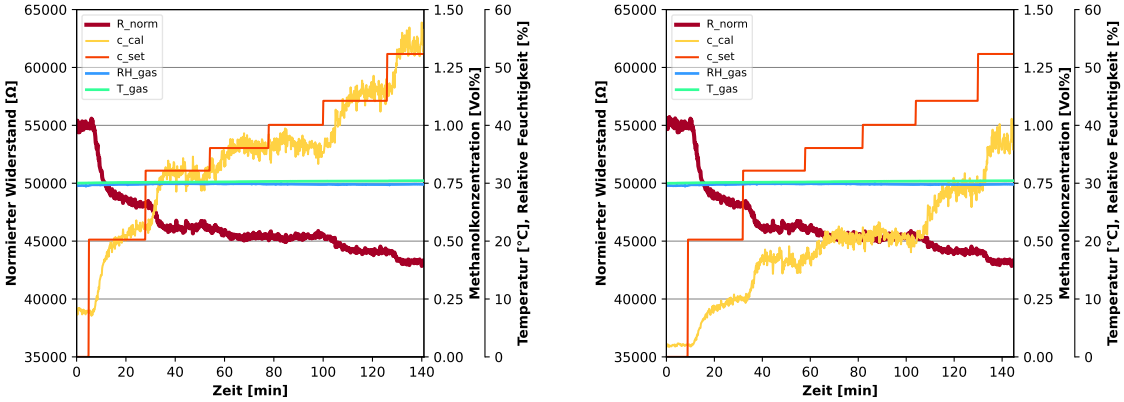
(d) θ_{med} = 18-30 °C

Abbildung 13: **Vergleich verschiedener Polynome für die Kalibrierung** Gemeinsame Darstellung der Polynome ersten bis dritten Grades, unter verschiedenen Bedingungen in doppelt-logarithmischer Auftragung. Abgebildet wird ein prozessrelevanter Konzentrationsbereich zwischen 0.36 Vol% und 1.5 Vol% (bzw. 2.8 g/L bis 11.7 g/L), sowie der Messwertbereich für den normierten Messwiderstand der Methanolsonde von 35 $k\Omega$ bis 55 $k\Omega$. Prozessbedingungen während der Kalibrierungen: $F_{\text{gas}} = 28$ ml/min, $N_{\text{st}} = 400$ rpm, $F_{\text{air}} = 1,4$ L/min, $d = 0.3$ mm.

Abb. (13a) ist zu entnehmen, dass grundsätzlich ein linearer Zusammenhang aus den Kalibrierdatenpunkten $\ln(c)$ zu $\ln(R_{\text{norm}})$ hervorgeht. Die Polynome zweiter (Abb. 13c) und dritter (Abb. 13b) Ordnung weisen unter den jeweiligen Kalibrierbedingungen eine Inversion der Steigung außerhalb des vorgelegten Konzentrationsbereichs auf. Eine gemeinsame Darstellung aller linearen Kalibriergeraden zeigt, dass die Steigungen der Kalibriergeraden sehr ähnlich sind (Abb. 13d).

5.2.3 Reproduzierbarkeit der Kalibrierung

Anwendung der Kalibriergeraden vor und nach Autoklavierung



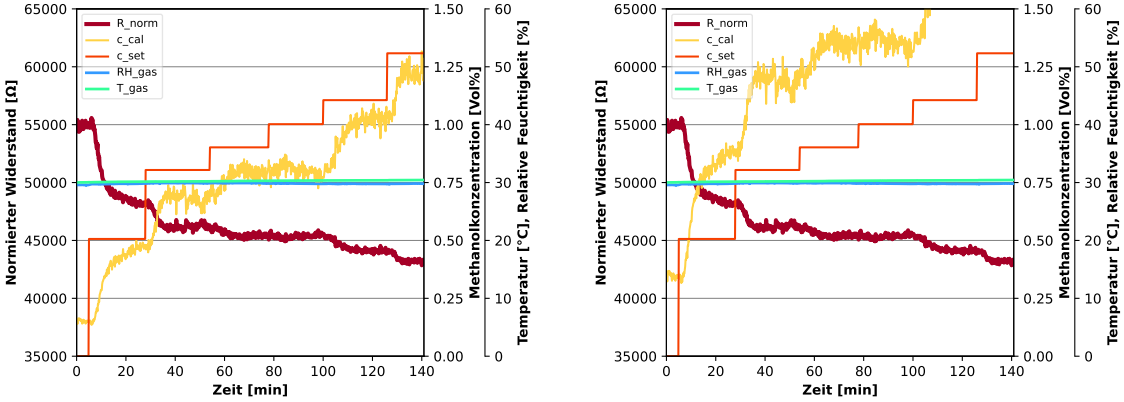
(a) Referenzkalibrierung

(b) Kalibrierung vor Autoklavieren

Abbildung 14: **Simulierte Anwendung von Kalibrierkoeffizienten nach Autoklavieren** Sensor- und Kalibrierdaten nach einer Kalibrierung nach Autoklavierung des Bioreaktors. Dargestellt ist der zeitliche Verlauf der berechneten Konzentration aus den Sensordaten, sowie der Verlauf der tatsächlichen Konzentration. Zusätzlich sind die gemessenen Sensorwerte der Methanolsonde abgebildet. 14a: Diese Kalibrierung misst in der rückwirkenden Berechnung des Kalibrierdatensatzes mit einer durchschnittlichen Abweichung von -0,07 Vol% bzw 0,55 g/L unter der vorgelegten Konzentration. Prozessbedingungen während der Kalibrierung: F_{gas}: 28 ml/min, θ_{med}: 30 ° C, N_{st}: 400 rpm, F_{air}: 1,2 L/min, d: 0,5 mm. Folgende Messparameter wurden algorithmisch für den Zustand der Methanolsonde ermittelt: $\overline{\Delta t}$ = 17,77 min, $\Sigma\Delta$ R = 23,66 %, $\Sigma\ \Delta\ R/\Sigma\ \Delta$ c = 18,06 %/% m: -8,062, b: 86,356. 14b:Berechnung der Konzentration durch Koeffizienten, die am Tag vor der Autoklavierung aufgezeichnet wurden: m = -12,05, b = 128,544 bei θ_{cal}: 30° C.Der errechnete Konzentrationswert liegt für den gesamten Sensordatenverlauf durchschnittlich -0,41 Vol% (3,2 g/L) unterhalb der vorgelegten Konzentration.

Um die Qualität der aufgezeichneten Kalibrierkoeffizienten im simulierten Fermentationsprozess zu ermitteln, wurde unmittelbar vor Autoklavieren eine Kalibrierung unter Prozessbedingungen durchgeführt. Anschließend wurden im Bioreaktor durch Injektionen über ein Septum die Volumenkonzentrationen 0.5%, 0.8%, 0.9%, 1.0%, 1.1%, 1.3% eingestellt. Der errechnete Konzentrationswert liegt für den gesamten Sensordatenverlauf durchschnittlich -0.41 Vol% (3.2 g/L) unterhalb der vorgelegten Konzentration.

Anwendung extern aufgezeichneter Kalibriergeraden



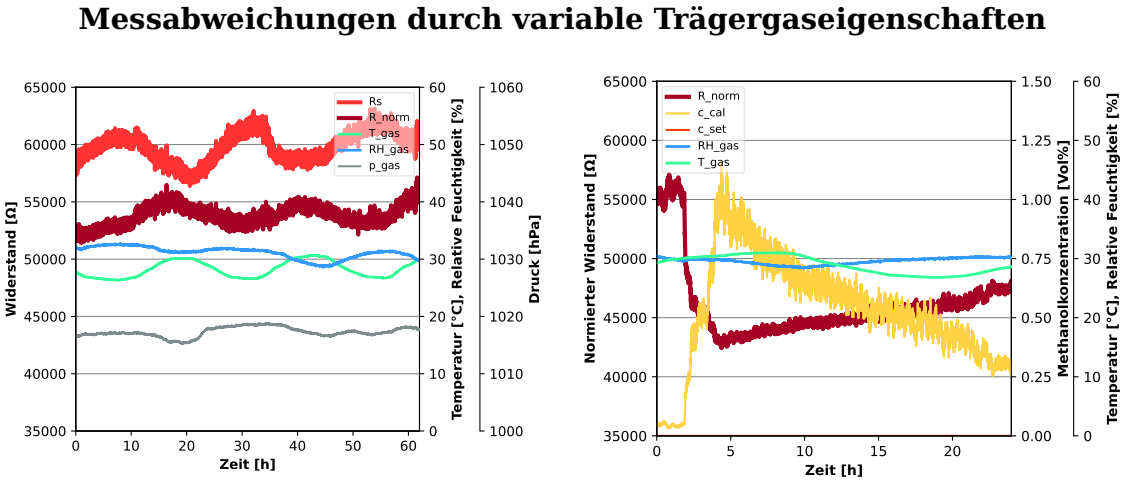
(a) 0,5 L Becherglas, θ_{med} = 26,5 ° C

(b) 1,0 L Becherglas, θ_{med} = 25,5 ° C

Abbildung 15: **Anwendung extern aufgezeichneter Kalibriergeraden** Dargestellt wird die rückwirkende Berechnung der Methanolkonzentration während einer Kalibrierung, die nach dem Autoklavieren aufgezeichnet wurde. Die Aufzeichnung der Kalibrierkurven erfolgte vor dem Autoklavieren im Becherglas. 15a: Die Kalibrierung im Becherglas misst in der rückwirkenden Berechnung des Sensordatensatzes mit einer durchschnittlichen Abweichung von -0.12 Vol% bzw 0.94 g/L von der vorgelegten Konzentration. 15b: Die errechnete Konzentration verläuft mit einem durchschnittlichen Offset von +0.39 Vol% oder 3.05 g/L oberhalb der vorgelegten Konzentration.

Um die Anwendbarkeit einer externen Methanolsondenkalibrierung während eines Bioprozesses zu untersuchen, wurde der Sensordatenverlauf während einer späteren Kalibrierung genutzt zur Berechnung einer Methanolkonzentration. Dargestellt wird die berechnete Konzentration aus dem normierten Widerstandswert und die vorgelegten Konzentrationen. Der errechnete Konzentrationsverlauf verläuft annähernd parallel zu den eingestellten Konzentrationen. . Die errechnete Konzentration mit den extern aufgezeichneten Koeffizienten weicht um -0,12 Vol% (Abb. 15a) und +0.39 Vol% (Abb. 15b) von der eingestellten Konzentration ab.

5.3 Kompensation der Trägergaseigenschaften



(a) c_{MeOH} =0,0 Vol%

(b) c_{MeOH} = 1,3 Vol%

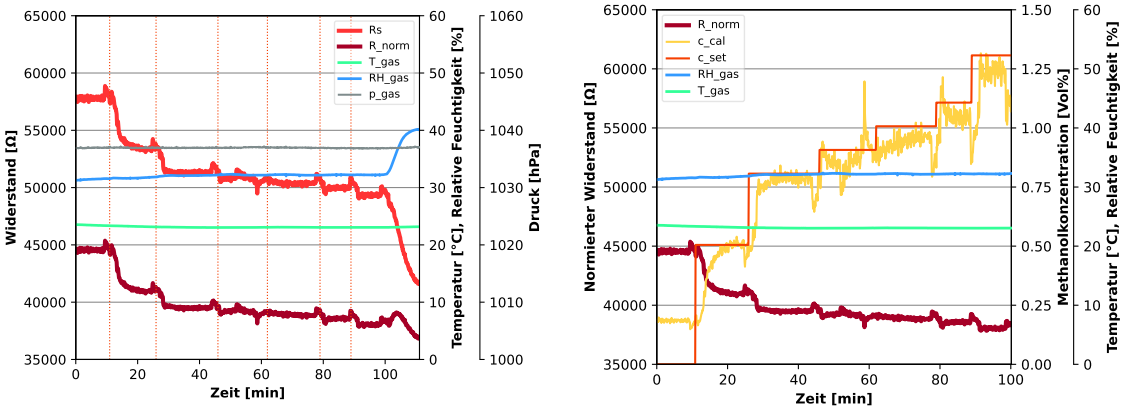
Abbildung 16: **Langzeitbeobachtung des normierten Widerstandes bei wechselnden Trägergaseigenschaften** Dargestellt werden der Messwiderstand R_s, sowie die Trägergaseigenschaften Druck, Temperatur und relative Luftfeuchtigkeit bei zwei unterschiedlichen Methanolkonzentrationen. Ebenfalls eingezeichnet ist der auf 20° C und 65 % rel. Feuchtigkeit normierte Widerstand 16a: Über den Verlauf von 70 h ist eine periodische Tag-Nacht-Oszillation der Trägergaseigenschaften zu sehen. Der Sensorwiderstand R_s zeigt eine Korrelation mit den Variationen der Trägergaseigenschaften und schwingt um einen Wert von $R_s = 60000\ \Omega \pm 3000\ \Omega$ (±5%). Durch Normierung des Widerstandes wird die Intensität der Oszillation verringert. Der normierte Widerstand R_{norm} liegt über die Dauer der Messung bei 54000 Ω ± 2000 Ω (±3.7%). Prozessbedingungen während des Experiments: F_{gas} = 28 ml/min, th_{med} = 30°C, N_{st} = 400 rpm, F_{air} = 1,2 L/min, d = 0.5 mm. 16b: Der normierte Messwiderstand steigt linear über die Dauer von 20 h von 42500 Ω auf 47500 Ω. Im Gegensatz dazu weist der rohe Messwiderstand eine periodische Schwankung auf, die sich zeitlich überschneidet, mit den Variationen der Trägergaseigenschaften. Prozessbedingungen für diesen Zeitraum: F_{gas}: 28 ml/min, th_{med}: 30°C, N_{st}: 400 rpm, F_{air}: 1,2 L/min, d: 0.5 mm

R_{norm}.

Um die Effektivität der Normierung des Widerstandes zu untersuchen, wurde der Reaktor über mehrere Stunden nach einer Kalibrierung (Abb. 16b) und im leeren Zustand mit VE-Wasser (Abb. 16a) auf Prozessbedingungen gehalten. Das Trägergas weist über diesen Zeitraum periodische Tag-Nacht-Schwankungen auf, die sich im leeren Bioreaktor stärker auf den normierten Widerstand R_{Norm} auswirken, als in der Anwesenheit von Methanol.

5.4 Kompensation der Medientemperatur

Vergleich der Kalibrierungsqualität bei Medientemperatur von 18 °C



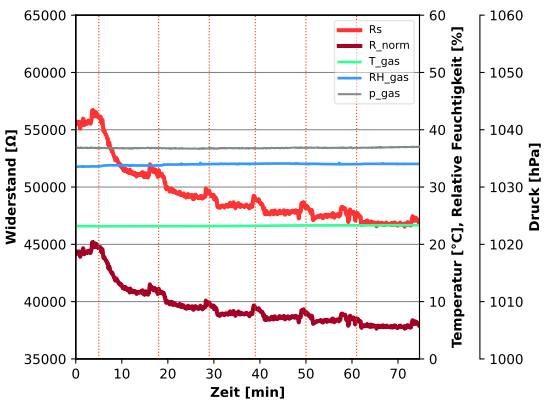
(a) Sensordaten bei θ_{med} = 18 °C

(b) Kalibrierdaten bei θ_{med} = 18 °C

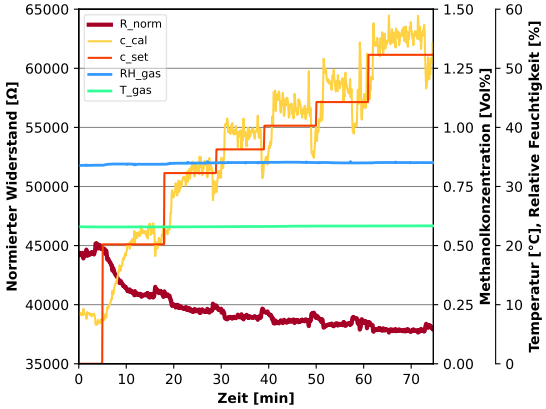
Abbildung 17: **Kalibrierungsqualität nach Kalibrierung bei Medientemperatur von 18 °C** Dargestellt wird der Sensordatenverlauf während einer Kalibrierung im Bioreaktor bei einer Medientemperatur von θ_{med} = 18 °C. Prozessbedingungen während der Kalibrierung: F_{gas}: 28 ml/min, th_{med} : $18°C$, N_{st}: 400 rpm, F_{air}: 1,4 L/min, d: 0.3 mm. Folgende Messparameter wurden algorithmisch ermittelt: $\overline{\Delta t}$ = 10.41 min, $\Sigma\Delta$ R = 14.43%, $\Sigma\Delta R/\Sigma\Delta$ c = 11.1%/%

In Abb. 17 sind die Kalibrierergebnisse eine Kalibrierung bei 18 °C dargestellt.Die Injektionen erfolgten hierbei unsteril über ein Tauchrohr ohne Septum. Der Messwert steigt vor jedem Injektionszeitpunkt bei Öffnen der Sterilkupplung an. Nach der dritten Injektion sinkt der Sensorwiderstand nicht wie erwartet auf einen stabilen Endwert, sondern schwingt in einer Spanne von $\pm$ 1000 Ω. Ab t = 100 min wurde zum leeren des Reaktors eine Pumpe eingeschaltet, was zu einem Anstieg der relativen Feuchtigkeit des Trägergases führte, sobald der Füllstand die Einbauhöhe der Sondenmembran unterschritten hatte. Der normierte Messwiderstand ändert sich dabei um 5%, während der unkompensierte Sensorwiderstand sich um 17% verändert. Nach der algorithmisch bestimmten Verzögerungszeit der Methanolsonde zu dem Kalibrierzeitpunkt von $\overline{\Delta t}$ = 10.41 min, schwingt die berechnete Konzentration mit einem durchschnittlichen Offset von 0.01 Vol% (0.08 g/L) unterhalb der eingestellten Konzentration. Die Schwankungen des Messwiderstands nach der dritten Injektion äußern sich in Konzentrationsschwankungen von -0.2 Vol% bis +0.5+ Vol%.

Vergleich der Kalibrierungsqualität bei Medientemperatur von 21 °C



(a) Sensordaten bei θ_{med} = 21 °C



(b) Kalibrierdaten bei θ_{med} = 21 °C

Abbildung 18: **Kalibrierungsqualität nach Kalibrierung bei 21 °C** Aufgetragen sind der normierte Widerstand und die Trägergasparameter (18a), sowie die im Medium vorgelegte Konzentration. Ebenfalls abgebildet wird die errechnete Konzentration aus dem normierten Widerstand über dieselbe Kalibrierungskurve, die aus dem Experiment hervorging. Prozessbedingungen während der Kalibrierung: F_{gas}: 28 ml/min, th_{med}: 21°C, N_{st}: 400 rpm, F_{air}: 1,4 L/min, d: 0.3 mm. Folgende Messparameter wurden algorithmisch ermittelt: $\bar{\Delta}t$ = 9.31 min, $\Sigma\Delta$ R = 17.29%, $\Sigma\Delta R/\Sigma\Delta$ c = 13.30%/%.

In Abb. 18 wird der Sensordatenverlauf während einer Kalibrierung im Bioreaktor bei einer Medientemperatur von θ_{med} = 21 °C. Die Injektionen erfolgten hierbei insteril über ein Tauchrohr ohne Septum. Der Messwert steigt vor jedem Injektionszeitpunkt bei Öffnen der Sterilkupplung an. Der normierte Messwiderstand verläuft bei konstanten Trägergaseigenschaften parallel-versetzt zum Sensorwiderstand. Nach der algorithmisch bestimmten Verzögerungszeit der Methanolsonde zu dem Kalibrierzeitpunkt von $\bar{\Delta}t = 9.31$ min, schwingt die berechnete Konzentration oberhalb der eingestellten Konzentration. Die errechnete Messabweichung, liegt durchschnittlich 0.08 Vol% (0.6 g/L) oberhalb der eingestellten Konzentration. Das Messwertrauschen des normierten Widerstands führt bei einer vorgelegten Konzentration von 1.0 Vol% (4 g/L) zu einer Messungenauigkeit von ± 0.05 Vol% (0.4 g/L).

Betrachtung der Kalibrierungsgeraden unterschiedlicher Temperaturen

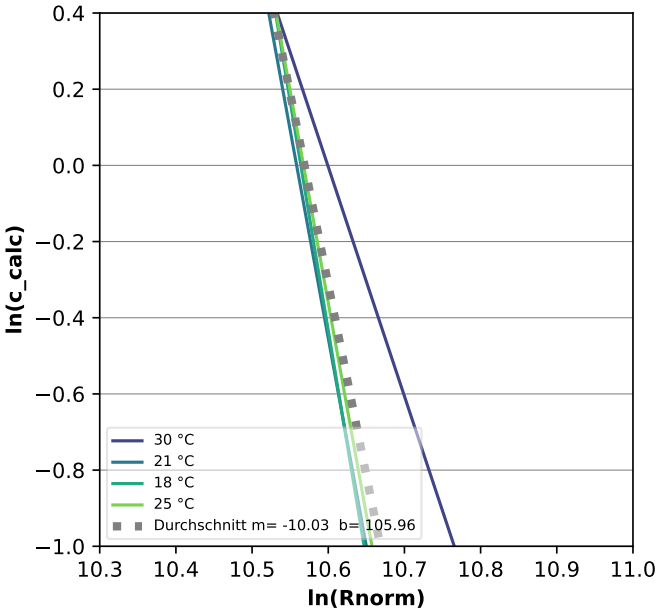


Abbildung 19: **Kalibriergeraden verschiedener Medientemperaturen im Vergleich** Es werden die Kalibriergeraden bei unterschiedlichen Medientemperaturen doppelt-logarithmisch aufgetragen. Die Kalibrierkoeffizienten wurden gemittelt und eine Durchschnittsgerade wurde erstellt und ebenfalls eingezeichnet.

In Abb. 19 sind die Kalibriergeraden von vier unterschiedlichen Medientemperaturen aufgetragen. Man erkennt, dass die Koeffizienten der Temperaturen $\theta_{18°C}$, $\theta_{21°C}$ und $\theta_{25°C}$ sich nur geringfügig voneinander unterscheiden, während die Kalibriergerade $\theta_{30°C}$ sowohl eine andere Steigung, als auch einen Offset besitzt.

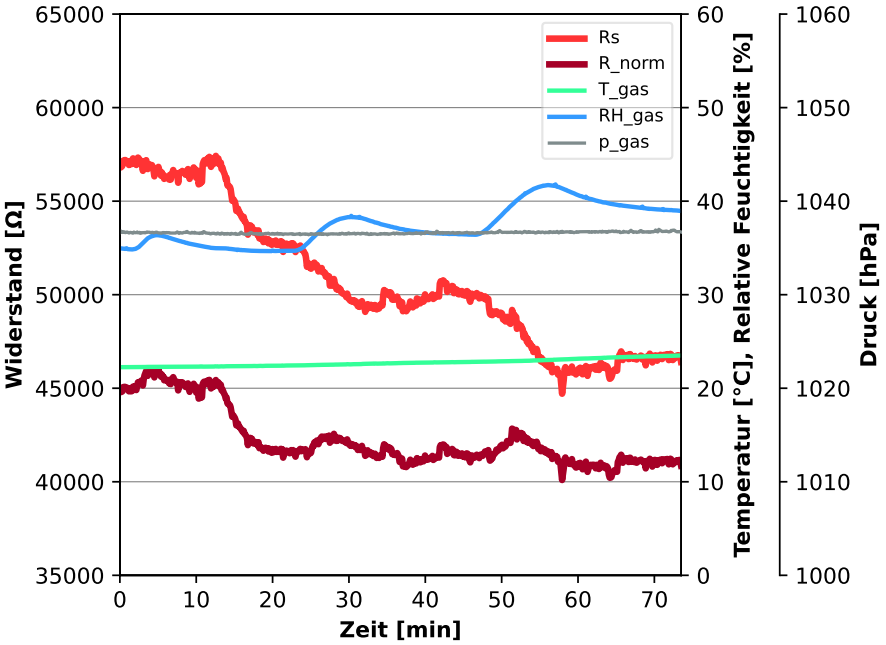


Abbildung 20: **Sondenmessverhalten bei steigender Temperatur und c_{MeOH} = 0.5 Vol%** Die Medientemperatur wurde nach initialer Zugabe von Methanol stufenweise erhöht und der Sensordatenverlauf aufgezeichnet. Nach Temperierung des Bioreaktors auf 18 °C wurde eine Konzentration von 0.5 Vol% eingestellt. Ab t = 20 min wurde die Temperatur des Mediums auf 21°C erhöht. Bei t = 50 min wurde die Medientemperatur erneut erhöht auf 25 °C. Die relative Feuchtigkeit steigt jeweils von 35 % bei 18 °C auf 37 % bei 21 °C und am Ende auf 39 % bei 25 °C. Der normierte Messwiderstand hat bei einer Konzentration von c = 0,5 Vol% über die Dauer des Experiments einen Durchschnittswert von R_{norm} = 41500 Ω ± 1000 Ω. Der unkompensierte Sensorwiderstand ändert sich nach der ersten Temperaturänderung von ΔT = 3 °C um 2500 Ω und bei der zweiten Änderung ΔT = 4 °C um 3000 Ω. Die Temperatur des Trägergases zeigt über die gesamte Dauer des Experiments eine Temperaturänderung von ΔT = 1,2 °C. Prozessbedingungen während des Experiments: F_{gas} = 28 ml/min, θ_{med} =18 °C,21 °C,25 °C, $N_{\text{st}} = 400$ rpm, F_{air} = 1,4 L/min, d = 0,3 mm.

Um den Einfluss der Medientemperatur auf den unkompensierten Messwiderstand R_S mit dem Einfluss auf den Widerstand nach Kompensation der Trägergaseigenschaften zu vergleichen, wurde bei einer konstanten Konzentration von 0,5 Vol% die Medientemperatur θ_{med} stufenweise von 18 °C auf 21 °C und anschließend auf 25 °C erhöht. In Abb. 20 sind der Messwiderstand R_S, die Trägergaseigenschaften RH_{gas}, T_{gas} und p_{gas} über die Zeit aufgetragen. Es ist zu erkennen, dass der Einfluss der Medientemperatur auf den normierten Widerstand kleiner ist, als auf den unkompensierten Widerstand.

5.5 Bewertung der Software

Die bestehende Software wurde erfolgreich an die neuen Systemanforderungen angepasst und verfügt über die gleichen Funktionen, wie vor der Aktualisierung des Programms. Zusätzlich wurden alle geplanten Funktionen implementiert, die für die Verwendung der Methanolsonde in Bioprozessen essentiell sind. Die Methanolsondensoftware wurde demnach hinreichend für die Verwendung in Bioprozessen vorbereitet. Das Testen der Software hat gezeigt, dass ein Langzeitbetrieb ohne Störungen möglich ist. Es folgt eine tabellarische Auflistung aller Programmfunktionen und eine Erklärung der Funktion. Hierfür wird die Software in zwei Bereiche gegliedert: die Nutzeroberfläche und die Hintergrundprozesse. Die Nutzeroberfläche ist der Teil der Software, der dem Nutzer direkt zur Verfügung steht. Die Hintergrundprozesse sind die Funktionen, die im Hintergrund ablaufen und nicht direkt vom Nutzer gesteuert werden können. Die Nutzeroberfläche ist in Abb. (21) dargestellt

Stabilitätstest der Sondensoftware

Die Funktionalität, um beispielsweise die Medientemperatur oder andere Prozessvariablen von der Prozessleitsoftware abzurufen konnte nicht wiederhergestellt werden. Und wird über manuelle Eingabe der Werte durch den Benutzer gelöst. Die Stabilität der Kommunikation zwischen dem Prozessleitsystem und der Sondensoftware ist jedoch über fünf Tage problemlos und ohne Fehlermeldungen verlaufen.

Automatisierte Sondenqualifizierung

Nach einer Injektion ist ein deutliches Sinken des Sondenwiderstandes zu erwarten, weshalb die Verzögerungszeit als der Zeitpunkt definiert wurde, an dem die Steigung positiv wird. Die prozentuale Widerstandsänderung vor und nach Injektion dient als Maß für die Messempfindlichkeit und wurde als Kriterium bei der Auswahl der optimalen Messbedingungen berücksichtigt. Die auf diese Art und Weise ermittelten

Werte werden in Balkendiagrammen dargestellt, um den Vergleich unterschiedlicher Bedingungen zu erleichtern.

Funktionen der Sondensoftware

Hintergrundprozesse:

***OPC*:** Stellt Verbindung mit dem OPC Server der Prozessleitsoftware her, um Sensordaten zu übertragen.

***Serial*:** Ruft Sensordaten über die serielle Schnittstelle und gliedert den Datenstrom in seine einzelnen Bestandteile.

***sInterface*:** Umfasst alle vorgestellten Rechenoperationen und Kompensationsmöglichkeiten und ermöglicht die automatisierte Auslesung von Kalibrierungsrelevanten Sensordaten nach Angabe der Injektionszeitpunkte und injizierten Massen.

Grafische Nutzeroberfläche:

***Data*:** Darstellung aller momentanen Prozessvariablen.

***Settings*:** Einstellungen für Abfrageintervall, Speicherung von Nutzereinstellungen und Wahl des COM-Ports.

***Calibration*:** Kalibrier-Interface mit verschiedenen Möglichkeiten der Kalibrierung und Kompensation.

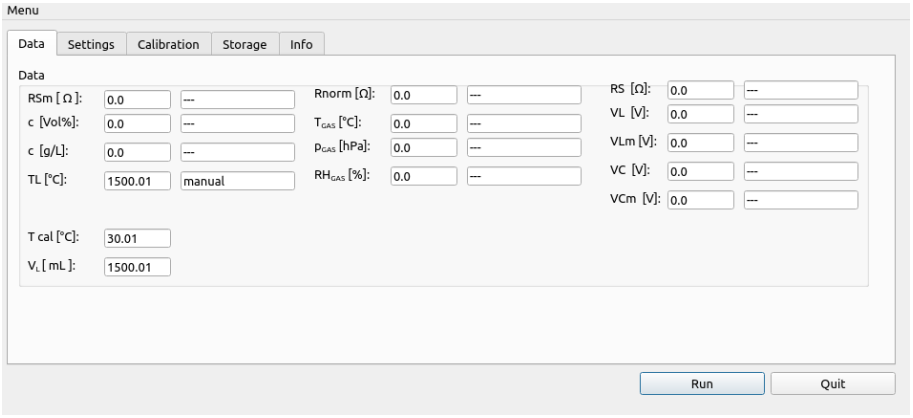
***Storage*:** Änderung der Speicherungsoptionen, wie Speicherort und Dateiname.

***Info*:** Information über die Software und den Urheber.

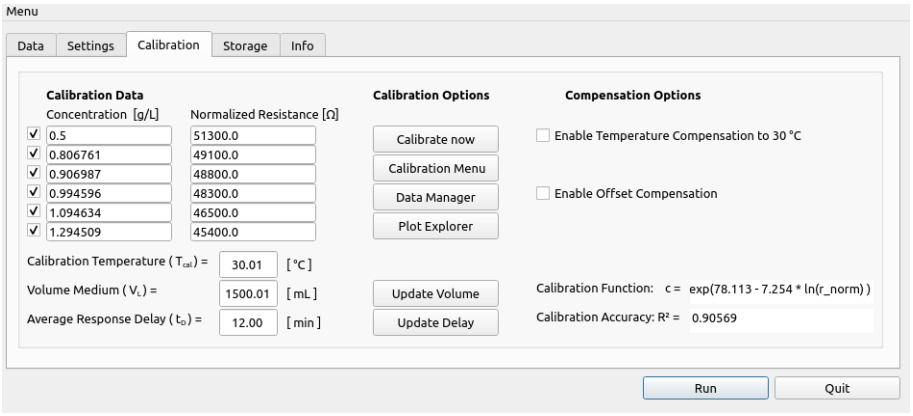
***DataCreator*:** Erstellung von Abbildungen aus Sensordaten für die Auswertung und Bewertung des Sondenverhaltens.

***DataExplorer*:** Darstellung der erstellten Plots in einem separaten Programmfenster.

Übersicht der Benutzerfunktionen der Methanolsondensoftware



(a) Prozessdatenübersicht



(b) Funktionen zur Kalibrierung und Kompensation.

Abbildung 21: **Übersicht der Benutzerfunktionen der Methanolsondensoftware** (21a): Ansicht aller Prozessdaten, die von der Sondensoftware erfasst werden. (21b): Kalibrierungsmenü mit allen Funktionen zur Kalibrierung und Kompensation.

6 Diskussion

Die Themengebiete, die im Rahmen dieser Diskussion behandelt werden, umfassen die Kalibrierungsmethode, die Funktionalität der Kalibrierung im Bioreaktor, potenzielle Ursachen für Messabweichungen, die Relevanz von den Messabweichungn in der Produktionsphase, den Einfluss der Verschleißkomponenten, Kompensationsalgorithmen sowie die Sensorsoftware. Die hier diskutierten Ergebnisse sind von Bedeutung, um die Zuverlässigkeit und Genauigkeit der Methanolmessungen in Bioreaktionsprozessen zu bewerten. Diese Erkenntnisse können dazu beitragen, das Verständnis der Methanolmessung zu erweitern oder Optimierungsmöglichkeiten zu identifizieren.

6.1 Dreistufiger Fermentationsprozess

In den Fermentationen aus den Vorversuchen konnten erste Aufschlüsse über den Messwertbereich der Methanolsonde, sowie praktische Erfahrungen mit der Methanolmessung in dreistufigen Bioprozessen gesammelt werden. Zum Zeitpunkt der Fermentationen diente der unkompensierte Messwiderstand R_S als qualitativer Indikator für die Anwesenheit von Methanol und wurde nicht für die Berechnung der Methanolkonzentration genutzt. Nach den Fermentationen hat sich der Zustand der Methanolsonde mehrfach geändert, weshalb der angezeigte Messwertbereich zwischen 52 $k\Omega$ und 20 $k\Omega$ in den nachfolgenden Kalibrierexperimenten nicht reproduziert werden konnte. Der aktuelle Zustand der Methanolsonde zeigt eine geringere Empfindlichkeit, weshalb die rückwirkende Berechnung mit aktuellen Kalibriergeraden nicht sinnvoll ist. Bezogen auf den heutigen Zustand der Methanolsonde, würde die Induktionskonzentration geschätzt werden auf einen Wert oberhalb des untersuchten Kalibrierbereiches, der zwischen 0,5 Vol% und 1,5 Vol% liegt. Die tatsächliche Konzentration lag bei etwa 0,2 Vol%. Die Diskrepanz liegt hauptsächlich an der veränderten Empfindlichkeit der Methanolsonde und zeigt, dass Kalibriergeraden nicht über einen längeren Zeitraum verwendet werden können.

6.2 Kalibrierungsmethode

Kalibrierqualität

Es wurde gezeigt, dass die verwendeten Methoden zu zufriedenstellenden Kalibriergeraden führen. Dies geschah anhand rückwirkender Berechnung der Methanolkonzentration aus den Sondenmesswerten. Um die Kalibrierqualität zu bewerten, wurde das Bestimmtheitsmaß mit der algorithmischen Auswertung der Sensordaten verglichen. Die Qualität der Kalibriergeraden über das Bestimmtheitsmaß abzuschätzen, führte jedoch nicht zu einheitlichen Ergebnissen. Es liefert nur eine Aussage darüber, wie genau das Kalibrierpolynom die Messwerte abbildet, nicht jedoch wie präzise die Kalibrierung durchgeführt wurde oder wie gut die vorgelegten Konzentrationen zu den Widerstandswerten passen. In einigen Fällen wurde beispielsweise beobachtet, dass der Messwiderstand nach einer kleinen Konzentrationsänderung (Δc = 0,1 Vol%) keine Änderung aufwies bzw. wieder anfing zu steigen. Die Protokollierung und Ablesung der Kalibrierungswertepaare konnte vor Entwicklung der Software nur manuell durch Auslesen des unkompensierten Messwiderstandswertes durchgeführt werden. Die Ablesung eines kalibrierungsrelevanten Widerstandes aus dem verrauschten Sensorsignal führt wie erwartet zu suboptimalen Ergebnissen. Als eine bessere Alternative wurde die algorithmische Berechnung der Verzögerungszeit und der Messempfindlichkeit vorgestellt. Für dessen Berechnung sind nur die Injektionszeitpunkte und die injizierten Methanolmengen notwendig. Dadurch wird der Ablese- oder Auswertungsfehler durch den Nutzer verringert und die aufgenommenen Werte eignen sich für die Aufnahme von Referenzwerten für zukünftige Kalibrierungen.

Kalibrierpolynom

Für die Berechnung der Methanolkonzentration wurden die Polynome ersten bis dritten Grades in Erwägung gezogen und miteinander verglichen. Es wurde gezeigt, dass die Polynome zweiter und dritter Ordnung außerhalb des jeweiligen Kalibrierbereiches ihre Steigung ändern können. In der Anwendung würde dies bedeuten, dass die berechnete Konzentration sinkt, wenn die tatsächliche Konzentration im Bioreaktor über den Kalibrierbereich hinaus steigt, was wiederum zu einer Aktivierung des Methanolfeeds führen würde und das Problem verstärkt. Im umgekehrten Szenario, würde ein Sinken Methanolkonzentration unterhalb des Kalibrierbereiches zu einer Deaktivierung des Methanolfeeds führen, was zu einer Unterbrechung der Induktion und Minderung der Ausbeute führen kann. Es wurde nicht beobachtet, dass Polynome zweiter und dritter Ordnung zu kleineren, simulierten Messabweichungen in der rückwirkenden Konzentrationsberechnung führen. Für die Sicherheit des Prozesses, ist es unerlässlich, dass die gemessene Methanolkonzentration von der reellen Konzentration innerhalb eines Toleranzbandes abweicht. Da die gemessene Methanolkonzentration eine Berechnung aus verschiedenen Sensorwerten darstellt und dieses Berechnungspolynom als Fehlerquelle ausgeschlossen werden kann, wenn lineare Regressionsmodelle gewählt werden, sind diese den höheren Polynomen vorzuziehen. Höhere Polynome bilden zwar die vorgelegten Kalibrierungswertepaare besser ab, aber führen in der Anwendung potentiell zu schwerwiegenden Fehlern.

Kalibrierpunkte

Es wurde gezeigt, dass sechs Kalibrierpunkte für die Kalibrierung ausreichen. Für die Untersuchung verschiedener Quereinflüsse und Messparameter wurde der gewählte Kalibrierbereich in kleinere Konzentrationsschritte unterteilt, was jedoch nicht die spätere Anwendung abbildet. Für die geplante Anwendung in beispielsweise einer Methanolgehaltsregelung während einer Produktionsphase ist es ausreichend, wenn die Methanolkonzentration konstant innerhalb eines Werteberei-

ches gehalten wird. Dafür ist es wichtig, mindestens den geplanten Kalibrierbereich bei der Wahl der vorgelegten Konzentrationen zu umfassen, da die Linearität außerhalb des Kalibrierbereiches für diese Arbeit (0,5 % bis 1,0 %) nicht untersucht wurde. Aus den Abbildungen (Poly Plots) geht hervor, dass die widerstandsabhängige Position der vorgelegten Kalibrierungswertepaare auf der Y-Achse für die Konzentration bei doppelt-logarithmischer Auftragung von einer Geradengleichung beschrieben wird. Die Steigung und der Achsenabschnitt einer Gerade lassen sich bereits mit zwei Messpunkten berechnen. Folglich sind zwei Messpunkte für die Kalibrierung bei geplantem Einsatz im Bioprozess theoretisch ausreichend. Der Konzentrationsbereich der Kalibrierung sollte möglichst angepasst sein, an die später geplante Methanolkonzentration in der Produktionsphase. Aufgrund der logarithmischen Natur des Kalibrierpolynoms, ist es sinnvoll eine untere Grenze für den Kalibrierbereich zu wählen, der oberhalb 0,1 Vol% Methanol liegt. Zhang et al. [14] beschreiben eine Induktionsmethanolkonzentration von 0,15 Vol%. Die Messempfindlichkeit der Sonde ist für kleine Konzentrationsbereiche am höchsten, was bei der Wahl der Konzentrationsschritte berücksichtigt werden sollte.

Trägergasstrom

Der Einfluss der Trägergasströmungsgeschwindigkeit wurde durch Variation der Einstellung des Schwebekörperdurchflussmessers und anschließende Aufnahme von Kalibrierkurven untersucht. Es ist zu erwarten, dass die Empfindlichkeit mit steigender Trägergasströmungsgeschwindigkeit abnimmt, da dadurch die Verweilzeit des pervaporierten Methanols über dem Sensor verringert wird. Es ist weiterhin auch anzunehmen, dass eine längere Verweilzeit zu einer höheren Empfindlichkeit bei der Messung führt, da mehr Moleküle von dem Sensorfilament oxidiert werden, was zu einem stärkeren Messsignal führt. Also bestand die Vermutung, dass eine optimale Trägergasströmungsgeschwindigkeit exisitiert, bei der sowohl die Verweilzeit am kleinsten ist, als auch die Empfindlichkeit am höchsten. Zwar wurde das exakte

Optimum nicht ermittelt, jedoch kann gesagt werden, dass der Trägergasstrom von 28 mL/min bei der algorithmischen Auswertung der Sondenparameter die besten Ergebnisse erzielt hat. Es muss jedoch angemerkt werden, dass der schützende Silikonschlauch, der um die Membran der Sonde positioniert ist, in der Zeit nach der Untersuchung des Trägergasstromoptimums ausgetauscht wurde und aktuell dicker ist (jetzt 0,5 mm, vorher 0,3 mm). Es sollte daher eine neue Evaluation der Trägergasströmungsgeschwindigkeitseinstellung durchgeführt werden, um auszuschließen, dass sich das empirisch-ermittelte Optimum verschoben hat.

Reproduzierbarkeit in der Anwendung

Die Kalibriergeraden, die mit der gewählten Kalibriermethode aufgezeichnet wurden, haben zu unterschiedlichen Messabweichungen in den simulierten Anwendungsbeispielen geführt. Konkret wurde untersucht, ob eine Kalibriergerade, die vor Autoklavieren der Methanolsonde aufgezeichnet wurde, zu der Berechnung der Methanolkonzentration in einer späteren Prozessphase geeignet ist. Für einen Bioprozess ist es notwendig steril zu arbeiten, weshalb es nicht möglich ist, die Kalibrierqualität nach der Autoklavierung zu überprüfen, bevor die erste Methanolzugabe stattfindet. Es wurde beobachtet, dass die Kalibrierung vor der Autoklavierung in der simulierten Anwendung zu einer ähnlichen Messabweichung führt, wie die Kalibrierung, die in einem Becherglas durchgeführt wurde. Die Abweichung der Kalibrierkoeffizienten vor und nach Kalibrierung war trotz identischer Kalibrierbedingungen jedoch überraschend groß. Hierbei muss jedoch angemerkt werden, dass der Silikonschlauch vor der Autoklavierung ausgetauscht wurde und das neue Material vorher nicht der thermischen Belastung einer Autoklavierung ausgesetzt war. Es gilt ferner zu bestätigen, dass die beobachteten Differenzen vor- und nach Autoklavierung mit der Anzahl der Autoklavierzyklen abnehmen und auszuschließen, dass sie die Reproduzierbarkeit der Methode einschränken. Für die Anwendung in einem Bioprozess bedeutet dies, dass die später aufgezeichnete Mess-

abweichung protokolliert werden sollte, nachdem die erste Methanolzugabe zur Induktion stattfindet. Die Differenz aus berechnetem und vorgelegten Konzentrationswert kann wie folgt für die Kompensation des Offsets genutzt werden wie in Gleichung 18 beschrieben.
Für die Anwendung der Methanolsonde unter simulierten Prozessbedingungen konnte jedoch gezeigt werden, dass sogar durchschnittliche Kalibrierkoeffizienten zu Ergebnissen führen, die für die Anforderungen während eines Bioprozesses ausreichen. Gekoppelt an die automatisierte Auswertung und Speicherung der Kalibrierdaten bildet sich somit die Möglichkeit die Qualität der durchschnittlichen Koeffizienten über die Dauer der Verwendung zu verfolgen und bessere Kompensationsmodelle auf Grundlage der Daten zu entwickeln. Die Vermutung besteht, dass die Sonde für die Messung im Prozess sogar ohne Kalibrierung mit den aktuellsten Durchschnittskoeffizienten betrieben werden könnte, solange die physikalischen Eigenschaften der Sonde (e.g. Silikonschlauch, Membran, etc.) nicht verändert werden.

Kalibrierungsmethode

Es konnte bestätigt werden, dass die konstruierte Methanolsonde zur Messung der Methanolkonzentration unter simulierten Prozessbedingungen geeignet ist. In einem Bioprozess mit methylotrophen Organismen wird Methanol zur Induktion der Expressionsgene verwendet, weshalb die Konzentration oberhalb eines Minimalwertes gehalten werden muss. Aktuell nutzt das, im Prozessleitsystem einprogrammierte Rezept zur Automatisierung des dreistufigen Fermentationsprozesses, eine Methanolgehaltsteuerung, in der zu Beginn der Produktionsphase eine Induktionskonzentration aus dem Volumen berechnet wird und anschließend zeitlich versetzt nachgefüttert wird. Mithilfe der neuen Software kann die im Reaktor gemessene Methanolkonzentration an das Prozessleitsystem übertragen werden. Dieses wiederum ist über die Prozesskontrolleinheit (DCU) in der Lage ein Signal für die Aktivierung einer Methanolfeedpumpe zu senden, was die Zugabe von Methanol in der Produktionsphase automatisiert. In der Praxis sollten

theoretisch keine cytotoxischen Konzentrationen im Bioreaktor entstehen, da die eingetragene Methanolmenge durch die Aktivierungsdauer der Pumpe begrenzt wird. Es ist nicht festgestellt worden, wie das Messverhalten der Sonde sich außerhalb des Kalibrierbereiches ändert, allerdings kann basierend auf den vorliegenden Ergebnissen angenommen werden, dass bei der Verwendung von linearen Regressionsmodellen auch Konzentrationen oberhalb des Kalibrierbereichs ausreichend genau gemessen werden können, um zu bestimmen dass die Konzentration oberhalb eines Maximalwertes liegt. Regelungstechnisch stellt dieses Szenario jedoch ein Problem dar, da es nicht möglich ist die Konzentration gezielt zu verringern. Bei Anwesenheit einer konstanten Zelldichte in der Produktionsphase, kann von einer konstanten Verbrauchsrate für Methanol ausgegangen werden, sodass keine Möglichkeit besteht die momentane Konzentration durch Anpassung einer Stellgröße zu senken. Bei der Wahl der Regelungsparameter ist daher zu berücksichtigen, dass ein Überschreiten der Zielkonzentration möglichst zu vermeiden ist.

Fehlerbetrachtung

Durch die algorithmische Auswertung der Sensordaten konnte ein Teil der Fehlerursachen eliminiert werden, allerdings bestehen noch weiterhin Fehlerquellen, die die Kalibrierqualität und anschließende Genauigkeit der Methanolmessung im Bioprozess beeinflussen. Da die Einstellung des Trägergasstromes manuell über ein dimensionslosen Schwebekörperdurchflussmesser aus eigenem Hause vorgenommen wird, ist trotz konstantem Trägergasvordruck nicht gewährleistet, dass der eingestellte Trägergasstrom bei Kalibrierung und anschließender Messung identisch ist. Die Position des Alkohol- und Kombisensors in der Methanolsonde sehen einen konstanten Trägergasstrom für den Transport von pervaporiertem Methanol in die Messkammer und anschließend in den umgebenden Raum vor. Dabei wird neben Methanol auch Wasser aus dem Bioreaktor ausgetragen, was das Volumen über die Dauer einer Kultivierung verringert. Dies kann dazu führen, dass der

kultivierte Organismus höheren Konzentrationen der Mediumsbestandteile ausgesetzt ist, als zuvor berechnet. Nachdem die Methanolsonde während einer Fermentation in Kontakt mit Zellbrühe kommt, ist auf eine gründliche Reinigung zu achten, da sich sonst ein steriltechnisches Risiko durch den Silikonschutzschlauch bildet. Wenn dieser Schlauch aufgrund von Verunreinigung oder Beschädigung ausgetauscht werden muss, ändern sich die Sondenparameter. Somit würden alle Methoden entfallen, bei denen historische Kalibrierungsdaten nötig sind (i.e. Kompensation der Medientemperatur). Bei der Aufnahme der Kalibrierkurve hat die Injektionsmethode einen Einfluss auf den zeitlichen Verlauf der Sensordaten gehabt. Hierbei wurde beobachtet, dass die unsterile Injektion von Methanol über ein Tauchrohr ohne Septum zu einem impulsartigen Anstieg des Sondenwiderstandes geführt hat, der sich anschließend normalisiert. Dieser Effekt entfällt, sobald ein Septum verwendet wird oder die Injektion über ein Loch im Reaktordeckel stattfindet, was möglicherweise daran liegt, dass durch die Begasung im Bioreaktor ein Druckgefälle zwischen dem Reaktorinnenraum, der Messkammer der Methanolsonde und dem umgebenden Raum besteht, welches sich über die Öffnung der Sterilkupplung des Tauchrohres ausgleicht. Die diskutierten Messabweichungen stellen für die Verwendung im Bioprozess allerdings keine Einschränkung dar.

6.3 Kompensationsalgorithmen

Kompensation Trägergaseigenschaften

Wei gezeigt werden konnte, eignet sich die Normierung des Sensorwiderstandes auf eine Trägergastemperatur von 20 °C und relative Feuchtigkeit von 65 % für die Kompensation der periodisch-wechselnden Trägergaseigenschaften. Ein direkter Vergleich mit dem unkompensierten Sensorwiderstand zeigt auf, dass eine Normierung des Sensorwiderstandes für die Anwendung in einem Bioprozess notwendig ist, da sonst keine reproduzierbaren Messungen möglich sind. Eine steigende Luftfeuchtigkeit würde im unkompensierten Szenario zu einem sinkenden Sensorwiderstand führen, was zu einer Überschätzung der Me-

thanolkonzentration führen würde. Während im leeren Bioreaktor ein Zusammenhang zwischen dem normierten Widerstand und den Tag-Nacht-Schwankungen über den untersuchten Zeitraum zu beobachten war, konnte dieser Effekt nicht mehr beobachtet werden, sobald eine Methanolkonzentration von 1.3 Vol% im Medium eingestellt wurde. Die Messempfindlichkeit des verwendeten Sensors nimmt mit zunehmender Konzentration von Methanol ab, was dazu führt, dass ebenfalls der Einfluss von der wechselnden Trägergasbeschaffenheit auf die Messung abnimmt. In einem Bioprozess ist der Widerstandswert des methanolfreien Prozessmediums keine relevante Kenngröße, daher wird angenommen, dass die wechselnden Eigenschaften des Trägergases hinreichend kompensiert sind.

Einfluss der Medientemperatur

Wie gezeigt werden konnte, ist der Einfluss der Medientemperatur auf Messungen unter simulierten Prozessbedingungen vernachlässigbar klein. Da in einem reellen Bioprozess die Temperatur eine essentielle Regelgröße ist und sich über die Dauer einer Fermentation nicht ändern sollte, bestand zuvor die Möglichkeit einen möglichen Einfluss der Medientemperatur zu minimieren, indem die Methanolsonde bereits bei der geplanten Prozesstemperatur kalibriert wird. Die Kalibrierung unter Prozessbedingungen ist ein zeitaufwändiger Prozess, da unter Anderem die Temperierung des Bioreaktors abgewartet werden muss und anschließend eine gründliche Reinigung folgen sollte, bevor beispielsweise die Mediumvorbereitung und Autoklavierung folgen können. Es wurde gezeigt, dass eine Kalibrierung im Becherglas bei Raumtemperatur mit unterschiedlichen Volumina in einem simulierten Anwendungsbeispiel zu ähnlichen, wenn nicht kleineren Messabweichungen führen kann, als eine Kalibrierung die unmittelbar vor der Autoklavierung durchgeführt wird. Es gilt weiterhin auszuschließen, ob der Einfluss der Medientemperatur nicht exisitiert oder das Messsystem für den Einfluss der Unterschiede nicht empfindlich genug ist. Die Annahme dass die Medientemperatur das Sondenmess-

verhalten beeinflusst, basiert auf der Tatsache, dass die Diffusionsrate der Gasmoleküle und die physikalischen Eigenschaften wie die Porengröße des Silikonschlauches sich durch die Medientemperatur verändern können. So steigt beispielsweise der Wasserdampfdruck mit der Temperatur um das Doppelte in dem untersuchten Temperaturbereich (20,647 hPa bei 18 °C vs. 42.470 hPa bei 30 °C). Der Einfluss der Temperatur auf die physikalischen Materialeigenschaften der Sondenkomponenten (i.e. Silikonschlauch, Membran) konnte nicht beobachtet werden, da der untersuchte Temperaturbereich 18 °C bis 30 °C zu klein ist. Während eine Querempfindlichkeit des unkompensierten Messwiderstandes in Abhängigkeit von der Temperatur festgestellt wurde, durch einen Einfluss auf die Trägergasfeuchtigkeit, konnte dieser jedoch durch die Normalisierung des Messwiderstandes auf die Trägergaseigenschaften von T_{gas} = 20 °C und RH_{gas} = 65 % bereits hinreichend verringert werden.

Kompensation der Medientemperatur

Es wurde der Ansatz geprüft, ob die lineare Interpolation von Koeffizienten einer Medientemperatur in die Koeffizienten für eine andere Temperatur übersetzt werden können. Während dieses Modell in der Theorie Koeffizienten produziert, die zu geringen Messabweichungen führen, wurden unter simulierten Prozessbedingungen, Fehler in derselben Größenordnung (kleiner als 0.5 Vol%) erreicht durch die Konzentrationsberechnung mit den durchschnittlichen Koeffizienten aus Kalibrierungen unter verschiedenen Prozessbedingungen. Dies ist jedoch aufgrund der Ähnlichkeit der verschiedenen Kalibrierkurven untereinander zu erwarten gewesen und spricht dafür, dass die Methanolsonde ein robustes Messverhalten bei den untersuchten Messbedingungen aufweist. Ferner gilt es jedoch weiterhin zu differenzieren, ob die Methanolsonde nicht empfindlich genug ist, um den Einfluss der Medientemperatur zu messen oder ob dieser Einfluss nicht existiert für Temperaturbereiche, die in Bioprozessen auftreten.

Implementierung in die Sondensoftware

Die Implementierung des Kompensationsalgorithmus war erfolgreich und zeigt die gewünschte Funktionalität. Um dies zu bewerkstelligen wurde eine automatische Speicherung aller Kalibrierungen in einer entsprechenden Datei in den Programmcode eingefügt. Durch das automatisierte Speichern wird über die Lebensdauer der Methanolsonde ein Datensatz generiert, was die Möglichkeit bietet eine Reiteration dieses Kompensationsproblems in Erwägung zu ziehen, sobald eine größere Anzahl an Kalibrierungen bei verschiedenen Temperaturen vorliegt.

Offset-Kompensation der Durchschnittskoeffizienten

Es wurde der Ansatz untersucht, inwiefern eine Offset-Kompensation der Durchschnittskoeffizienten programmiertechnisch umsetzbar ist und ob diese Methode für die Anwendung in einem Bioprozess geeignet ist. Die Schwierigkeit hierbei liegt in der logarithmischen Natur der Kalibrierpolynome und der resultierenden Tatsache, dass für das Kalibriermodell kein Nullpunkt definiert werden kann. Als Lösung des Problems wurden verschiedene Kalibrierkonzentrationen unterhalb 0.1 Vol% untersucht als simulierte Nullpunktkonzentration. Es zeigt sich, dass der Ansatz der Offset-Kompensation über diesen Ansatz nicht zu reproduzierbarer Verbesserung der Durchschnittskoeffizienten führt. Grundsätzlich ist es empfehlenswert vor jeder Verwendung der Sonde, eine Kalibrierung durchzuführen, weshalb die Verwendung von Durchschnittskoeffizienten nur für den Fall in Betracht gezogen werden soll, wenn die Quantifizierung des Methanolgehaltes im Medium nicht benötigt wird. Die bestehende Software besteht jedoch weiterhin, da die Offset-Kompensation auch nach Induktion mit Methanol durchgeführt werden kann. Es muss jedoch experimentell bestätigt werden, dass die vorliegende Offset-Kompensation zu reproduzierbarkleineren Messabweichungen unter reellen Prozessbedingungen führt.

6.4 Nicht untersuchte Einflussfaktoren

Silikonschutzschlauch

Eine nicht untersuchte Variable ist der Silikonschlauch, der zum Schutz um die Membran der Sonde gelegt wird. Diese hat sich im Verlauf der Experimente unvorhergesehenerweise geändert und hat einen deutlichen Einfluss auf die Sondenparameter. So wurde beispielsweise beobachtet, dass die Verzögerungszeit nach dem Wechsel zugenommen hat und die Empfindlichkeit geringer geworden ist. Da der Silikonschlauch eine physikalische Barriere zwischen der Messkammer und dem Kulturmedium darstellt, wirkt sie sich bei zunehmender Stärke negativ auf die Messeigenschaften aus. Dies liegt unter anderem daran, dass die Gasmoleküle eine längere Strecke überwinden müssen, um in die Messkammer zu gelangen.

Umgebungsdruck

Die Methanolsonde verfügt über einen Kombisensor, der neben der Temperatur und relativen Feuchtigkeit des Trägergases auch den Umgebungsdruck misst. Es wurde zwar beobachtet, dass der Umgebungsdruck sich während der Langzeitexperimente verändert hat, jedoch wurde der Einfluss auf den normierten Widerstand nicht quantitativ untersucht. Durch die Begasung des Bioreaktors liegt hier zwar theoretisch ein Überdruck vor, jedoch equilibriert sich dieser im vorliegenden System über die Abluftstrecke. Da die Messkammer der Methanolsonde, ebenso wie der Reaktionsraum des Bioreaktors ein offenes System darstellt, ist davon auszugehen, dass sich kein Druckgradient zwischen den beiden Systemen aufbauen sollte. Da der Umgebungsdruck sich über den Verlauf eines Tages nur geringfügig ändert, ist ein möglicher Einfluss auf das Messverhalten der Sonde während einer Kalibrierung nicht zu beobachten gewesen.

6.5 Sensorsoftware

Algorithmische Bewertung der Sondenparameter

Es wurde gezeigt, dass die Berechnung der Verzögerungszeit aus den Sensordaten bei Angabe der Injektionszeiten für die durchgeführten Experimente zuverlässig funktioniert und eine qualitative Bewertung des Sondenmessverhaltens unter verschiedenen Messbedingungen ermöglicht. Der Vorteil gegenüber einer, auf ein Toleranzband bezogenen, Totzeit (i.e. $t_{95\%}$) ist praktischer Natur. Da der Messwiderstand unter Bedingungen, wie beispielsweise bei einer Becherglaskalibrierung sehr verrauscht sein kann, würde eine Toleranzbandmethode zu einer sehr großen Totzeit führen, die nicht repräsentativ für das Sondenmessverhalten ist. Zudem würde eine Toleranzbandmethode zu errechneten Totzeiten führen, die den beobachteten Verdampfungseffekt des Methanols umfassen und somit zu einer Überschätzung der Totzeit führen. Für die Bewertung und den Vergleich verschiedener Messeinstellungen reicht die in der Software implementierte Berechnung der Verzögerungszeit aus. In einem Szenario, bei dem von unverauschten Messwiderständen ausgegangen werden kann, ist die Toleranzbandmethode jedoch eine valide Alternative zur Berechnung der Totzeit. Es ist ratsam in Erwägung zu ziehen, die normierten Messwiderstände direkt bei der Erfassung des Messwertes zu glätten (zum Beispiel durch einen PT1 Filter), um die Auswirkungen von Rauschen zu minimieren. Dies würde die Berechnung der Totzeit nach Toleranzbandmethode vereinfachen und die Reproduzierbarkeit der Messungen mit der Methanolsonde erhöhen. Dies würde vermutlich auch die unterschiedlich großen Verzögerungszeiten angleichen, die von dem Algorithmus erfasst werden (Abb. 8 und Abb. 11).

Bewertung der Sondensoftware

Das Ziel der Aktualisierung und Verbesserung der Methanolsondensoftware gilt als erfüllt, da alle Funktionen, die für die Implementierung einer Methanolgehaltregelung oder den Betrieb in einem Bio-

prozess nötig sind bereitgestellt sind und eine stabile Kommunikation zwischen den einzelnen Softwarekomponenten gewährleistet werden kann. In jeder simulierten Prozessanwendung wurden Messabweichungen unter 0.5 Vol% erreicht, was eine zuverlässige Messung der Methanolkonzentration ermöglicht. Es gilt jedoch weiterhin den Definitionsbereich für die Kalibrierformel auf Bereiche unterhalb 0.5 Vol% zu erweitern. Es konnte keine Lösung gefunden werden, um eine Übertragung der vom Prozessleitsystem verwalteten Prozessparameter an die Methanolsondensoftware zu ermöglichen. Dies liegt unter anderem daran, dass dafür eine administrative Veränderung der Konfiguration des MFCS-OPC Servers nötig wäre, für die spezielle Kenntnisse über die Konfiguration des Prozessleitsystems in einer Windows-Umgebung nötig sind. Eine andere Ursache für dieses Problem könnte das verwendete Python Modul sein, welches seit nicht mehr gepflegt wird und ursprünglich nicht für die Verwendung mit Python 3 entwickelt wurde. Es wurde als Notlösung für dieses Problem die Möglichkeit der manuellen Eingabe der Prozessparameter in die Methanolsondensoftware implementiert, um die Software dennoch in einem Prozess einsetzen zu können, bei dem beispielsweise die Medientemperatur nicht konstant ist.

7 Ausblick

Zusammenfassend lässt sich sagen, dass die Methanolsondensoftware, sowie die Methanolsonde nun in einem Zustand sind, der eine Verwendung in einem Bioprozess ermöglicht. Die Software ist stabil und alle getesteten Funktionen verhalten sich wie vorgesehen. Die Software wurde dabei so gestaltet, dass keine Startschwierigkeiten bei der initialen Bedienung auftreten sollten, aber bei Interesse viele Funktionen zum Experimentieren zur Verfügung stehen. Der weitere Ausbau der Software ist in jedem Falle sinnvoll, da beispielsweise nach der Aktualisierung kein Abrufen der Medientemperatur und des Füllstandes möglich war und diese Variablen für diverse Berechnungen in der Automatisierung des Bioprozesses benötigt werden. So wäre eine automatische Berechnung der Konzentration in g/L der erste Schritt die benutzerfreundlichkeit zu erhöhen.

Das Verhalten der Methanolsonde wurde aus verschiedenen Blickwinkeln hinsichtlich möglicher Störfaktoren beleuchtet und empirisch untersucht. Dabei wurde festgestellt, dass die Methanolsonde ein robustes und reproduzierbares Messverhalten besitzt, das sich gut in einen Bioprozess integrieren lässt. Der empfindlichste Teil des Sondenmessbereichs befindet sich im unteren Konzentrationsbereich, jedoch wurde bereits beobachtet, wie die Empfindlichkeit über den gewünschten Konzentrationsbereich bei einer Methanolgehaltregelung bereits abnimmt. Daher wäre es denkbar zu testen, wie gut dieses Messsystem über die Abluft des Prozesses funktioniert. Ein großer Vorteil wäre der ausbleibende thermische Verschleiß und damit verbundene Veränderungen der Sondenparameter. Im gleichen Zuge würde die Linearität des Messsystems im unteren Konzentrationsbereich besser aufgedeckt werden, was die Erweiterung des Definitionsbereichs bis evtl. 0 g/L Methanol ermöglichen könnte.

Literatur

[1] Mudassar Ahmad, Melanie Hirz, Harald Pichler, and Helmut Schwab. Protein expression in pichia pastoris: Recent achievements and perspectives for heterologous protein production. *Applied Microbiology and Biotechnology*, 98(12):5301–5317, 2014.

[2] Geoff P. Lin Cereghino, Joan Lin Cereghino, Christine Ilgen, and James M. Cregg. Production of recombinant proteins in fermenter cultures of the yeast pichia pastoris. *Current Opinion in Biotechnology*, 13(4):329–332, 2002.

[3] Gesine Cornelissen. *Integrierte Bioprozessentwicklung zur Herstellung pharmakologischer wirksamer Proteine mit Pichia pastoris*. PhD thesis, Hannover University, Düsseldorf, Germany, 2004. Also available at: `https://d-nb.info/973309466`.

[4] Bouke Wim de Jong, Verena Siewers, and Jens Nielsen. Systems biology of yeast: Enabling technology for development of cell factories for production of advanced biofuels. *Microbial Cell Factories*, 11(1), 2012.

[5] Arne Hagman, Torbjörn Säll, and Jure Piškur. Analysis of the yeast short-term crabtree effect and its origin. *The FEBS Journal*, 281(21):4805–4814, 2014.

[6] FIGARO ENGINEERING INC. Technical information for tgs2620 - technical information for volatile organic compound (voc) sensors. Data Sheet, 2022. Available at: `http://www.figaro.co.jp/en`.

[7] Christian Kaiser, Thorsten Peuker, T. Bauch, A. Ellert, and Reiner Luttmann. Pat - process analytical technology in cultivation processes with recombinant escherichia coli. In *Preprints of the 13th IFAC Symposium on System Identification*, pages 267–272, 2007.

[8] Mohsen Karbalaei, Seyed A. Rezaee, and Hadi Farsiani. Pichia pastoris: A highly successful expression system for optimal synthesis of heterologous proteins. *Journal of Cellular Physiology*, 235(9):5867–5881, 2020.

[9] M. Lampert. Qualifizierung einer Sonde zur Methanolmessung in der Flüssigphase eines Bioreaktors und Untersuchung ihrer Querempfindlichkeiten. HAW Hamburg, 2018.

[10] Anna Maráz. From yeast genetics to biotechnology. *Acta Microbiologica et Immunologica Hungarica*, 49(4):483–491, 2002.

[11] David A. Peña, Brigitte Gasser, Jürgen Zanghellini, Matthias G. Steiger, and Diethard Mattanovich. Metabolic engineering of pichia pastoris. *Metabolic Engineering*, 50:2–15, 2018.

[12] F. Schroeder and Goetz D. Etablierung und Charakterisierung einer Methanolsonde zur Steuerung der Produktionsphase einer Pichia pastoris Kultivierung. HAW Hamburg, 2020.

[13] O. Struck. Qualifizierung einer Methanolsonde. HAW Hamburg, 2017.

[14] W. Zhang, M. Inan, and M. M. Meagher. Fermentation strategies for recombinant protein expression in the methylotrophic yeast pichia pastoris. *Biotechnology and Bioprocess Engineering*, 5(4):275–287, 2000.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit in allen Teilen selbst und nur mit den angegebenen Quellen und Hilfsmitteln, einschließlich des World Wide Web und anderer elektronischer Quellen angefertigt habe. Alle Stellen der Arbeit, die ich anderen Werken dem Wortlaut oder dem Sinn nach entnommen habe, sind kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildlichen Darstellungen und dergleichen. Die Arbeit ist von mir in gleicher oder ähnlicher Form noch nicht eingereicht worden.

Hamburg, den 06.07.2023

Philipp Schmidt

Anhang

Anhang Programmcode

######### Author: Ullrich Scheffler #############
######### Maintenance and Improvements by Philipp Schmidt ###########

```
# -*- coding: utf-8 -*-
import os
import sys
import time
import serial
import OpenOPC
import xml.etree.ElementTree as ET
import pandas as pd
from PyQt5 import QtWidgets, QtCore     # necessary PyQT5 packages
from PyQt5.QtWidgets import QDialog, QVBoxLayout, QPushButton, QLabel, QScrollArea, QFileDialog,
QHBoxLayout,QMessageBox
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import Qt
# SensorInterface class for calibration and compensation methods
from SensorInterface import SensorInterface
# Created GUI elements
from TGS2620_UI import Ui_TGS2620_UI as Dlg
from SensorInterfaceGUI import SensorInterfaceGUI as siGUI
from PlottingInterfaceGUI import PlottingInterfaceWindow as piGUI
#from PlottingInterfaceGUI import PlottingInterfaceGUI as piGUI
#*****************************************************************************************************
# Globale Variablen
#*****************************************************************************************************
info_status = "---"
ohm = u"\u2126"
# com port settings
comPorts = [] # Liste der wählbaren COM Ports
for i in range(1,21):
   comPorts.append("COM" + str(i))
comPort = "COM8"
comPortOpenStatus = False

# serial port settings
ser = serial.Serial()
ser.baudrate = 9600
ser.bytesize = 8
ser.parity ="N"
ser.stopbits = 1
ser.timeout = 2
ser.xonxoff = False
ser.rtscts = False
ser.dsrdtr = False
requestMode = 1
showRawData = False
runState = False
requestInterval = 2 # "2" means all sensor data is being read, 1 means only some values are read

si = SensorInterface() # create an instance of the SensorInterface class
requestIntervals = ["1", "2", "5", "10", "15", "20" ,"30", "60", "120", "300", "600"]  # in seconds
dataStorage = False
storageFile = u"tgs2620_data.txt"
historicalDataFile = u"historical_calibrations.txt"
# Settings related to calibration and compensation


cal_status = 0 # fit accuracy of calibration curve, so 0 means uncalibrate
# set up global variables for calibration; an alternative using class variables would be more elegant though
cal_datensatz_benutzen = [0, 0, 0, 0, 0, 0]
cal_konzentrationen = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
cal_widerstaende = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

cal_temperature = 42.0 # value should be changed by user prior to calibration
cal_results = pd.DataFrame(columns=['fit_accuracy', 'th_med', 'a_0', 'a_1']) # default for linear regression
cal_coeffs = [0.0, 0.0] # default for linear regression, will be overwritten on initialization of the window
temperature = 30.0 # value is read from OPC server
```

```
temperatureCompensation = True # compensation means that the coefficients for measurement are interpolated
between the entered values at th_med and values that have been recorded previously
volume_med = 1500.0 # volume of the bioreactor in mL
autoTemperatureCompensation = False # autocompensation means that the value for th_med is expected to change
over time and is therefore read from the OPC server and used for compensation
time_delay = 0.0 # time delay in seconds between calibration injection and measurement
temperatureCompensationCoefficient=[]# TODO: add a compensation algorithm that calibrates the offset directly to
the normalized resistance values
useImportedCalibration = False # if True, the historical data is imported from selected file

xml_File_Name = u'tgs2620_config.xml'

OPC_enable = False

OPC_Server = u"BBI.MFCSSOPCS.1"
OPC_Item_1 = 'BIOSTAT A.MeOH_RS.Value'
OPC_Item_2 = 'BIOSTAT A.MeOH_c.Value'
OPC_Item_3 = 'BIOSTAT A.TL.Value'
OPC_Item_4 = 'BIOSTAT A.MeOH_Gas_T.Value'
OPC_Item_5 = 'BIOSTAT A.MeOH_Gas_p.Value'
OPC_Item_6 = 'BIOSTAT A.MeOh_Gas_RH.Value'

opc_init = 0
try:
    opc = OpenOPC.client()
    opc.connect(OPC_Server)
    opc_init = 1
except:
    opc_init = 0
    print("OPC server connect bei Systemstart nicht möglich")

logLevel = 1
logFileName = u'tgs2620_log.txt'
#*********************************************************************************************************
def logToFile(message, loggingLevel):
    logZeile = str(time.strftime("%d.%m.%Y %H:%M:%S")) + " " + message+ "\n"
    if loggingLevel >= logLevel:
        logFileObj = open(logFileName, "a")
        logFileSize = os.path.getsize(logFileName)
        if logFileSize == 0:
            logFileObj.write(u"TGS2620 Log-File" + u"\n")
        logFileObj.write(logZeile)
        #print logZeile[:-1]
        logFileObj.close()
    else:
        pass
    return

def openSerialPort():
    global comPortOpenStatus, comPort, ser
    try:
        if runState:  # Die serielle Schittstelle soll geöffnet werden
            logToFile("openSerialPort: bis jetzt war " + str(ser.port) + " der seriellen Schnittstelle zugewiesen",2)
            if not ser.isOpen():   # falls noch nicht geoeffnet
                ser.port = comPort # den Port zuweisen
                ser.open()         # und Port oeffnen
                logToFile("openSerialPort: Jetzt ist " + str(comPort) + u" geöffnet",2)
                comPortOpenStatus = True
                return True
            else:
                logToFile("openSerialPort:" + str(comPort) + u" NICHT geöffnet, da ein Port noch geöffnet ist",2)
                comPortOpenStatus = False
                return False
        else:
            logToFile("openSerialPort:", comPort, u" NICHT geoeffnet, da die Abfrageroutine läuft nicht. runState = " +
runState,2)
            comPortOpenStatus = False
            return False
    except IOError:
        logToFile("openSerialPort: IOError error: " + str(sys.exc_info()[0]),1)
        comPortOpenStatus = False
        return False
    except:
        logToFile("openSerialPort: Unexpected error: " + str(sys.exc_info()[0]),1)
        comPortOpenStatus = False
```

```
        return False

def closeSerialPort():
    # Die serielle Schittstelle soll geschlossen werden
    try:
        logToFile("closeSerialPort: bis jetzt war " + str(ser.port) + " der seriellen Schnittstelle zugewiesen",2)
        if not runState: # Also , wenn das Praogramm NICHT läuft dann ...
            if ser.isOpen(): # Also, wenn sie geoffnet ist dann ...
                ser.close()  # Den bisher zugewiesenen Port schliessen
                logToFile("closeSerialPort: Port geschlossen",2)
                comPortOpenStatus = False
                return True
            else:
                logToFile("closeSerialPort: " + str(ser.port) + u" NICHT geschlossen, da kein Port geöffnet war",2)
                comPortOpenStatus = False
                return False
        else:
           logToFile(u"closeSerialPort: Abfrageroutine läuft nicht. runState = " + str(runState),2)
    except IOError:
        logToFile("closeSerialPort: IOError error: " + str(sys.exc_info()[0]),1)
        comPortOpenStatus = False
        return False
    except:
        logToFile("closeSerialPort: Unexpected error: " + str(sys.exc_info()[0]),1)
        comPortOpenStatus = False
        return False

def opc_write(rs, c, gas_temperatur,gas_druck,gas_feuchtigkeit):
    global opc, opc_init
    #return True # Für Testzwecke
    try:
        opc[OPC_Item_1] = rs
        opc[OPC_Item_2] = c
        opc[OPC_Item_4] = gas_temperatur
        opc[OPC_Item_5] = gas_druck
        opc[OPC_Item_6] = gas_feuchtigkeit
        logToFile(u"opc_write:  Values written to OPC server",2)
        return True
    except ValueError:
        logToFile("opc_write: ValueError error: " + str(sys.exc_info()[0]),1)
        try:
            opc.close()
            opc_init = 0
            print("opc_write: ValueError: closed opc connection due to unexpected error")
        except Exception as e:
            print(str(e))
        return False
    except Exception as e:
        logToFile("opc_write: Unexpected error 1/2: " + str(sys.exc_info()[0]),1)
        logToFile("opc_write: Unexpected error 2/2: " + str(e),1)
        try:
            opc.close()
            opc_init = 0
            print("opc_write: Unexpected error: closed opc connection due to unexpected error")
            print(rs, c, gas_temperatur, gas_druck, gas_feuchtigkeit)
        except Exception as e:
            print(str(e))
        return False

def opc_read():
    global opc, opc_init
    '''
          Something is bugged and breaks the communication with the opc server from the MFCS after a while.
          The MFCS data storage protocol or the OpenOPC version might be the problem.
          It is hard to find a solution for this problem, because the openopc package is not maintained anymore for
python3
    '''

    try:
        '''
        status = "Calibration Test"
        temperature = 20.0
        wert = 0
        zeit = 0

        print ("opc_read : 0")
```

```
        opc = OpenOPC.client()
        print ("opc_read : 1")
        opc.connect(OPC_Server)
        print ("opc_read : 2")
        time.sleep(1.0)
        print ("opc_read : 3")
        # read OPC_Item_3 from OPC_Server
        wert, status, zeit = opc.read(OPC_Item_3)
        print ("opc_read : 4")
        logToFile("opc_read: Item: " + OPC_Item_3 + " ,Wert: " + str(wert) + " ,Status: " + str(status) + " ,Zeit: " +
str(zeit),2)
        print ("opc_read : 5")
        opc.close()
        print ("opc_read : 6")

        if status == "Good":
            logToFile("opc_read: Lesen erfolgreich! read-status:" + str(status),2)
            return float(wert), True
        else:
            logToFile("opc_read: Lesefehler! read-status:" + str(status),1)
            return temperature, False
        '''
        if True:
            return 100, False
    except ValueError:
        logToFile("opc_read: Value error: " + str(sys.exc_info()[0]),1)
        return temperature, False
    except Exception as e:
        print(str(e))
        logToFile("opc_read: Unexpected error 1/2: " + str(sys.exc_info()[0]),1)
        logToFile("opc_read: Unexpected error 2/2: " + str(e),1)
        return temperature, False

def xml_read(xmlFile):
    # XML Datei Lesen
    global comPort, requestInterval, requestMode, dataStorage, storageFile, OPC_enable, historicalDataFile,
time_delay
    global cal_datensatz_benutzen, cal_konzentrationen, cal_widerstaende, cal_temperature,
temperatureCompensation, temperature, autoTemperatureCompensation, volume_med
    try:
        ETtree = ET.parse(xmlFile)
        ETroot = ETtree.getroot()
        for parameter in ETroot.findall('settings'):
            comPort = parameter.find('comPort').text
            requestInterval = int(parameter.find('requestInterval').text)
            requestMode = int(parameter.find('requestMode').text)

            temp = parameter.find('dataStorage').text
            if temp == "1" or temp == "True":
                dataStorage = True
            else:
                dataStorage = False

            temp = parameter.find('opcEnable').text
            if temp == "1" or temp == "True":
                OPC_enable = True
            else:
                OPC_enable = False

            temp = parameter.find('compensationEnable').text
            if temp == "1" or temp == "True":
                temperatureCompensation = True
            else:
                temperatureCompensation = False

            temp = parameter.find('autoCompensationEnable').text
            if temp == "1" or temp == "True":
                autoTemperatureCompensation = True
            else:
                autoTemperatureCompensation = False

            storageFile = parameter.find('storageFile').text
            historicalDataFile = parameter.find('historicalDataFile').text
            cal_datensatz_benutzen[0] = int(parameter.find('cal_datensatz_benutzen_0').text)
            cal_datensatz_benutzen[1] = int(parameter.find('cal_datensatz_benutzen_1').text)
            cal_datensatz_benutzen[2] = int(parameter.find('cal_datensatz_benutzen_2').text)
```

```
            cal_datensatz_benutzen[3] = int(parameter.find('cal_datensatz_benutzen_3').text)
            cal_datensatz_benutzen[4] = int(parameter.find('cal_datensatz_benutzen_4').text)
            cal_datensatz_benutzen[5] = int(parameter.find('cal_datensatz_benutzen_5').text)

            cal_konzentrationen[0] = float(parameter.find('cal_konzentrationen_0').text)
            cal_konzentrationen[1] = float(parameter.find('cal_konzentrationen_1').text)
            cal_konzentrationen[2] = float(parameter.find('cal_konzentrationen_2').text)
            cal_konzentrationen[3] = float(parameter.find('cal_konzentrationen_3').text)
            cal_konzentrationen[4] = float(parameter.find('cal_konzentrationen_4').text)
            cal_konzentrationen[5] = float(parameter.find('cal_konzentrationen_5').text)

            cal_widerstaende[0] = float(parameter.find('cal_widerstaende_0').text)
            cal_widerstaende[1] = float(parameter.find('cal_widerstaende_1').text)
            cal_widerstaende[2] = float(parameter.find('cal_widerstaende_2').text)
            cal_widerstaende[3] = float(parameter.find('cal_widerstaende_3').text)
            cal_widerstaende[4] = float(parameter.find('cal_widerstaende_4').text)
            cal_widerstaende[5] = float(parameter.find('cal_widerstaende_5').text)

            cal_temperature = float(parameter.find('cal_temperature').text)
            #print "Read XML", cal_temperature
            temperature = float(parameter.find('temperature').text)
            volume = float(parameter.find('volume').text)
            if volume != 0:
                volume_med = float(volume)
            time_delay = float(parameter.find('delay').text)

        logToFile("xml_read: XML-Konfigurations-Datei " + xmlFile + " gelesen",2)
        xmlParameterShow()
        return True
    except:
        logToFile("xml_read: XML-Konfigurations-Datei " + xmlFile + " nicht gefunden oder fehlerhaft." +
str(sys.exc_info()[0]), 1)
        return False

def xml_write(xmlFile):
    global comPort, requestInterval, requestMode, dataStorage, storageFile, OPC_enable
    global cal_datensatz_benutzen, cal_konzentrationen, cal_widerstaende, cal_temperature,
temperatureCompensation, temperature, volume_med, time_delay
    # XML Datei Schreiben
    try:
        ETtree = ET.parse(xmlFile)
        ETroot = ETtree.getroot()
        for parameter in ETroot.findall('settings'):
            parameter.find('comPort').text = str(comPort)
            parameter.find('requestInterval').text = str(requestInterval)
            parameter.find('requestMode').text = str(requestMode)
            parameter.find('historicalDataFile').text = str(historicalDataFile)
            if dataStorage == True:
                parameter.find('dataStorage').text = "1"
            else:
                parameter.find('dataStorage').text = "0"

            parameter.find('storageFile').text = str(storageFile)

            if OPC_enable == True:
                parameter.find('opcEnable').text = "1"
            else:
                parameter.find('opcEnable').text = "0"

            if temperatureCompensation == True:
                parameter.find('compensationEnable').text = "1"

            else:
                parameter.find('compensationEnable').text = "0"

            if autoTemperatureCompensation == True:
                parameter.find('autoCompensationEnable').text = "1"
            else:
                parameter.find('autoCompensationEnable').text = "0"

            parameter.find('cal_datensatz_benutzen_0').text = str(cal_datensatz_benutzen[0])
            parameter.find('cal_datensatz_benutzen_1').text = str(cal_datensatz_benutzen[1])
            parameter.find('cal_datensatz_benutzen_2').text = str(cal_datensatz_benutzen[2])
            parameter.find('cal_datensatz_benutzen_3').text = str(cal_datensatz_benutzen[3])
            parameter.find('cal_datensatz_benutzen_4').text = str(cal_datensatz_benutzen[4])
            parameter.find('cal_datensatz_benutzen_5').text = str(cal_datensatz_benutzen[5])
```

```
            parameter.find('cal_konzentrationen_0').text = str(cal_konzentrationen[0])
            parameter.find('cal_konzentrationen_1').text = str(cal_konzentrationen[1])
            parameter.find('cal_konzentrationen_2').text = str(cal_konzentrationen[2])
            parameter.find('cal_konzentrationen_3').text = str(cal_konzentrationen[3])
            parameter.find('cal_konzentrationen_4').text = str(cal_konzentrationen[4])
            parameter.find('cal_konzentrationen_5').text = str(cal_konzentrationen[5])

            parameter.find('cal_widerstaende_0').text = str(cal_widerstaende[0])
            parameter.find('cal_widerstaende_1').text = str(cal_widerstaende[1])
            parameter.find('cal_widerstaende_2').text = str(cal_widerstaende[2])
            parameter.find('cal_widerstaende_3').text = str(cal_widerstaende[3])
            parameter.find('cal_widerstaende_4').text = str(cal_widerstaende[4])
            parameter.find('cal_widerstaende_5').text = str(cal_widerstaende[5])

            parameter.find('cal_temperature').text = str(cal_temperature)
            #print ("Write XML ", cal_temperature)
            parameter.find('temperature').text = str(temperature)
            parameter.find('volume').text = str(volume_med)
            parameter.find('delay').text = str(time_delay)
        stat=ETtree.write(xmlFile)
        logToFile("xml_write: XML Datei " + xmlFile + " geschrieben. Status =" + str(stat),2)
        return True
    except:
        logToFile("xml_write: XML Datei " + xmlFile + " nicht gefunden oder Fehlerhaft." + str(sys.exc_info()[0]),1)
        return False

def xmlParameterShow():
    logToFile("xmlParameterPrint: xml_File_Name:" + xml_File_Name,2)
    logToFile("xmlParameterPrint: comPort: " + str(comPort),2)
    logToFile("xmlParameterPrint: requestInterval: " + str(requestInterval),2)
    logToFile("xmlParameterPrint: requestMode:" + str(requestMode),2)
    logToFile("xmlParameterPrint: dataStorage:" + str(dataStorage),2)
    logToFile("xmlParameterPrint: storageFile:" + storageFile,2)
    logToFile("xmlParameterPrint: opcEnable:" + str(OPC_enable),2)
    logToFile("xmlParameterPrint: cal_datensatz_benutzen:" + str(cal_datensatz_benutzen),2)
    logToFile("xmlParameterPrint: cal_konzentrationen:" + str(cal_konzentrationen),2)
    logToFile("xmlParameterPrint: cal_widerstaende:" + str(cal_widerstaende),2)
    logToFile("xmlParameterPrint: cal_temperature:" + str(cal_temperature),2)
    logToFile("xmlParameterPrint: temperatureCompensation:" + str(temperatureCompensation),2)
    logToFile("xmlParameterPrint: autoTemperatureCompensation:" + str(autoTemperatureCompensation),2)
    logToFile("xmlParameterPrint: temperature:" + str(temperature),2)

def isnumeric(s):
    try:
        n=str(float(s))
        if n == "nan" or n=="inf" or n=="-inf" : return False
    except ValueError:
        try:
            complex(s) # for complex
        except ValueError:
            return False
    return True

def strToFloat(s):
    if len(s) > 0:
        return float(s)
    else:
        return 0.0

def readSerial(): # read serial data from sensor, calculate concentration, write to file
    global requestMode, runState, requestInterval, temperature, cal_temperature,
temperatureCompensationCoefficient, cal_coeffs
    global OPC_enable, info_status, opc_init, opc, si
    try:
        # Daten auslesen
        line = ""
        rs = 0.0
        vlRaw=0
        vcRaw=0
        rsMean=0.0
        vlRawMean=0.0
        vcRawMean=0.0
        temperature=0.0
        gas_druck=0.0
        gas_feuchtigkeit=0.0
```

```
    # TODO: remove deprecated requestMode 1
    if (runState and requestMode == 1 and comPortOpenStatus):
        logToFile("readSerial: requestMode = 1",2)
        logToFile("readSerial: Deprecated requestMode",2)

    # Read serial data and parse it
    if (runState and requestMode == 2 and comPortOpenStatus):
        logToFile("readSerial: requestMode = 2",2)
        ser.write(bytes([27]) + b"P1" + bytes([13, 10]))  # ESC P 1 CR LF
        line = ser.readline().decode()   # lese bis ein "CR" erkannt wird
        #print "Raw Line: " # debug
        #print line # debug

        # Parse data (do not touch)
        if (len(line.split())==27):
            if line.split()[0] == "RS:"          and line.split()[2] == "Ohm"  and isnumeric(line.split()[1]):
                rs       = float(line.split()[1])
                ok = 1
                dialog.le_Value_RS.setText("%8.2f" % (rs))
                dialog.le_TimeStamp_RS.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
            if line.split()[3] == "VL_Raw:"      and line.split()[5] == "rel"  and isnumeric(line.split()[4]):
                vlRaw     = int(line.split()[4])
                ok = 2
                dialog.le_Value_VL.setText("%8.2f" % (float(vlRaw)*5.0/1023))
                dialog.le_TimeStamp_VL.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
            if line.split()[6] == "VC_Raw:"      and line.split()[8] == "rel"  and isnumeric(line.split()[7]):
                vcRaw     = int(line.split()[7])
                ok = 3
                dialog.le_Value_VC.setText("%8.2f" % (float(vcRaw)*5.0/1023))
                dialog.le_TimeStamp_VC.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
            if line.split()[9] == "RS_m:"        and line.split()[11] == "Ohm" and isnumeric(line.split()[10]):
                rsMean    = float(line.split()[10])
                ok = 4
                dialog.le_Value_RSm.setText("%8.2f" % (rsMean))
                dialog.le_TimeStamp_RSm.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
            if line.split()[12] == "VL_Raw_m:"   and line.split()[14] == "rel" and isnumeric(line.split()[13]):
                vlRawMean = float(line.split()[13])
                ok = 5
                dialog.le_Value_VLm.setText("%8.2f" % (float(vlRawMean)*5.0/1023))
                dialog.le_TimeStamp_VLm.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
            if line.split()[15] == "VC_Raw_m:"   and line.split()[17] == "rel" and isnumeric(line.split()[16]):
                vcRawMean = float(line.split()[16])
                ok = 6
                dialog.le_Value_VCm.setText("%8.2f" % (float(vcRawMean)*5.0/1023))
                dialog.le_TimeStamp_VCm.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
            if line.split()[18] == "Temp:"   and line.split()[20] == "degC" and isnumeric(line.split()[19]):
                gas_temperatur = float(line.split()[19])
                ok = 7
                dialog.le_Value_temperatur.setText("%8.2f" % (float(gas_temperatur)))
                dialog.le_TimeStamp_temperatur.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
            if line.split()[21] == "Druck:"   and line.split()[23] == "hPa" and isnumeric(line.split()[22]):
                gas_druck = float(line.split()[22])
                ok = 8
                dialog.le_Value_druck.setText("%8.2f" % (float(gas_druck)))
                dialog.le_TimeStamp_druck.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
            if line.split()[24] == "RH:"   and line.split()[26] == "%" and isnumeric(line.split()[25]):
                gas_feuchtigkeit = float(line.split()[25])
                ok = 9
                dialog.le_Value_feuchtigkeit.setText("%8.2f" % (float(gas_feuchtigkeit)))
                dialog.le_TimeStamp_feuchtigkeit.setText(time.strftime("%d.%m.%Y %H:%M:%S"))

            # calculate r_norm for display # needs to be verified
            r_norm = si.normalize_resistance(rsMean, gas_feuchtigkeit, gas_temperatur)
            dialog.le_Value_Rnorm.setText("%8.2f" % (r_norm))
            dialog.le_TimeStamp_Rnorm.setText(time.strftime("%d.%m.%Y %H:%M:%S"))

            # check if calibration has been performed and calculate concentration from sensor readings with
SensorInterface Formula
            if cal_status > 0:
                c = si.get_concentration(rsMean, gas_feuchtigkeit, gas_temperatur, cal_coeffs)
                logToFile(u"readSerial(2): tempCompON:  c=" + str(c) + u" Vol% TL=" + str(temperature) + u" °C Tcal=" +
str(cal_temperature) + u" °C",2)
                #print "readSerial ", cal_temperature
                dialog.le_Value_c.setText("%8.6f" % (c))
                dialog.le_TimeStamp_c.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
```

```
            else:
                dialog.le_Value_c.setText("invalid")
                dialog.le_TimeStamp_c.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
                logToFile(u"readSerial(2): Konzentrations-Berechnung (Invalid): cal_status = " + str(cal_status),2)
            #OPC
            if OPC_enable == True:
                st = opc_write(rs, c,gas_temperatur,gas_druck,gas_feuchtigkeit)
                if st == False:
                    logToFile("readSerial(2): OPC write Error",1)
                # Skip reading for now, as it is not working due to MFCS server misconfiguration
                '''
                tl, st = opc_read()
                if st == False:
                    logToFile("readSerial(2): OPC read Error " + str(st) + " " + str(tl),1)
                else:
                    temperature = tl
                    dialog.le_Value_temp.setText("%5.2f" % (temperature))
                    dialog.le_TimeStamp_temp.setText(time.strftime("%d.%m.%Y %H:%M:%S"))
                '''
            else:
                temperature =float(dialog.le_Value_temp.text())
                dialog.le_Value_temp.setText("%5.2f" % (temperature))
                dialog.le_TimeStamp_temp.setText("manual")
            dialog.le_Value_temp_cal.setText("%5.2f" % (cal_temperature))
            info_header = "RS      RS_m     VC   VC_m VL   VL_m Stat c       Temp    Druck   RH    Date        Time"
            info_units  = "[Ohm]    [Ohm]    [V]  [V]  [V]  [V]  [-]  [Vol%]   [degC]  [hPa]   [%]   [dd.mm.yyyy] [HH:MM:SS]"
            info_data = "%8.2f %8.2f %4.2f %4.2f %4.2f %4.2f %1i    %8.6f %5.2f   %7.2f %5.2f " % (rs , rsMean,
float(vcRaw)*5.0/1023, float(vcRawMean)*5.0/1023, float(vlRaw)*5.0/1023, float(vlRawMean)*5.0/1023, ok, c,
gas_temperatur, gas_druck, gas_feuchtigkeit)
            info_data = info_data + time.strftime("%d.%m.%Y   %H:%M:%S")
            ok = 0
            # info2 = "Raw Data !!! " + " ['" + "', '".join(line.split()) + "']"  + " Elements: " + str(len(line.split()))
        else:
            info_data = "Invalid Data !!! " + " ['" + "', '".join(line.split()) + "']"  + " Elements: " + str(len(line.split()))
        if showRawData:
            logToFile("readSerial(2): " + line + " Length: " + str(len(line)),2)
            logToFile("readSerial(2): " + info_header,2)
            logToFile("readSerial(2): " + info_units,2)
            logToFile("readSerial(2): " + info_data,2)
            dialog.lbl_data.setText(info_header + "\n" + info_units + "\n" + info_data)

    # Save data to file
    if (runState and dataStorage):
        dataStorageObj = open(storageFile, "a")
        storageFileSize = os.path.getsize(storageFile)
        if storageFileSize == 0:
            dataStorageObj.write(info_header + "\n")
            dataStorageObj.write(info_units + "\n")
        dataStorageObj = open(storageFile, "a")
        dataStorageObj.write(info_data + "\n")
        dataStorageObj.close()
  except IOError:
    logToFile("readSerial: IOError error: " +  str(sys.exc_info()[0]),1)
  except Exception as e:
    logToFile("readSerial: Unexpected error:" + str(e),1)
  finally:
    if runState and comPortOpenStatus:
        QtCore.QTimer.singleShot(requestInterval*1000, readSerial)

# Kalibrierung
def calibration(cal_use, cal_c, cal_r, cal_temp, ergebnisse_anzeigen = False):        # PS 20230529
  try:
    global  cal_status, cal_results, cal_temperature, si
    cal_temperature = cal_temp
    si = si # overwrite current and create new instance of SensorInterface class for automatic calibration and
compensation
    si.update_th_med(cal_temperature) # update temperature for calibration
    cal_data_df = pd.DataFrame(columns=['c_cal', 'r_norm', 'cal_temp'])
    cal_results = pd.DataFrame(columns=['fit_accuracy', 'a_0', 'a_1', 'cal_temp'])
    for i in range(6):        # Add calibration data to cal_data_df if cal_use is True
        if cal_use[i] == 1:
            cal_data_df = pd.concat([cal_data_df, pd.DataFrame([{'c_cal': cal_c[i], 'r_norm': cal_r[i], 'cal_temp' :
si.th_med}])], ignore_index=True)
    if temperatureCompensation == True and cal_temp != 30.0:          # first perform normal calibration to get most
recent calibration coefficients at the current calibration temperature
        cal_results = si.fit_cal_data(cal_data_df)
```

```
            if not os.path.isfile(historicalDataFile):
                logToFile("calibration: historicalDataFile does not exist",1)
            else:
                coefficients_data = pd.read_csv(historicalDataFile, sep="\s+")
                num_of_coefficients = len(coefficients_data.columns) -2
                coefficient_dict = {'cal_temp': si.th_med}
                for i in range(num_of_coefficients):
                    coefficient_dict[f'a_{i}'] = cal_results[f'a_{i}'][0]
                coefficients_data = pd.concat([coefficients_data, pd.DataFrame([coefficient_dict])], ignore_index=True)
#print("Coeff data: ", coefficients_data) # show the whole data set used for linear interpolation of the coefficient at
target temperature
                comp_coefficients = si.get_temp_coeff(coefficients_data, 30) # get temperature compensated coefficients
for 30 degrees                # print(comp_coefficients) # debug
                cal_results = si.fit_cal_data(cal_data_df, comp_coefficients)               # calculate fit accuracy using
temperature compensated calibration coefficients
                # change cal_temp to 30
                cal_results['cal_temp'][0] = 30
                if si.getDebug() == True:
                    print('After Temperature compensation : \n', cal_results) # debug
                logToFile("calibration: Mit Kompensation : "+ str(comp_coefficients),2)
        else:
            cal_results = si.fit_cal_data(cal_data_df)
            logToFile("calibration: Ohne Kompensation: "+ str(cal_results),2)
        with open("calibration_results.txt", "a") as f:
            save_results = cal_results
            if os.stat("calibration_results.txt").st_size == 0:           # check if file is empty, if so write the header
                f.write(save_results.to_string(header=True))
                f.write("\n")
            else:
                f.write(save_results.to_string(header=False))
                f.write("\n")
            if si.getDebug() == True:
                    print("Calibration results saved to file: calibration_results.txt.")
        if True:
            cal_status = cal_results['fit_accuracy'][0]
        else:
            cal_status = 0
        if ergebnisse_anzeigen == True:
            logToFile("calibration: Koeffizienten : "+ str(cal_results),2)
            logToFile("calibration: Kalibration Status: " + str(cal_status),2)
        return cal_results
    except IOError:
        logToFile("calibration: IOError error: " +  str(sys.exc_info()[0]),1)
        return None
    except ValueError:
        logToFile("calibration: ValueError error: " +  str(sys.exc_info()[0]),1)
        return None
    except Exception as e:
        logToFile("calibration: Unexpected error:" + str(e),1)
        return None

def getFormula(cal_results):
    global cal_coeffs
    assert isinstance(cal_results, pd.DataFrame), "cal_results must be a DataFrame"
    degree = len(cal_results.columns) - 3    # get polynomial degree n from cal_results; columns are 'fit_accuracy',
'cal_temp', 'a_0', ... 'a_n'
    coeff = []    # list of coefficients a_0, ... a_n
    for i in range(degree + 1):
        coeff.append(cal_results['a_' + str(i)][0])
    coeff.reverse()    # inverse list of coefficients a_n, ... a_0
    coeff = [round(x, 3) for x in coeff]    # round coefficients to 3 decimal places
    formula = 'exp('    # formula string = exp( a_n * ln(r_norm)**n + ... + a_0)
    for i in range(degree + 1):
        if i == 0:        # if i = 0 then ln(r_norm)^0 = 1
                formula = formula + str(coeff[i])
        elif i == 1:        # if i = 1 then ln(r_norm)^1 = ln(r_norm)
            if coeff[i] < 0 :
                formula = formula + ' - ' + str(-coeff[i]) + ' * ln(r_norm) '
            else:
                formula = formula + ' + ' + str(coeff[i]) + ' * ln(r_norm) '
        else:
            if coeff[i] < 0:
                formula = formula + ' - ' + str(-coeff[i]) + ' * ln(r_norm)^' + str(i)
            else:
                formula = formula + ' + ' +  str(coeff[i]) + ' * ln(r_norm)^' + str(i)
    formula = formula + ')'
```

```
    cal_coeffs = coeff
    return formula

class ProgrammFenster(QtWidgets.QMainWindow, Dlg):
    def __init__(self):
        global comPort, info_status, opc, requestInterval, requestIntervals, storageFile, si
        global cal_coeffs, cal_status, cal_results, temperatureCompensation, historicalDataFile, volume_med
        super().__init__()
        comPort = comPorts[3]
        QtWidgets.QMainWindow.__init__(self)
        self.onDataCreator_opened  = False
        if si is None:
            logToFile("ProgrammFenster: si is None, using default values",1)
            self.si = SensorInterface()
            si = self.si
        else:
            self.si = si
        if True: ### GUI Elemente
            self.setupUi(self)
            self.setFixedSize(self.size())
            self.statusBar().showMessage(info_status)
            #self.statusBar().hide()
            self.cb_comPort.addItems(comPorts)
            self.cb_comPort.setCurrentIndex(10)
            self.cb_requestInterval.addItems(requestIntervals)
            self.cb_requestInterval.setCurrentIndex(1)
            self.lbl_data.hide()
            # Slots einrichten
            self.pb_run.clicked.connect(self.onOK)
            self.pb_quit.clicked.connect(self.onAbbrechen)
            self.cb_comPort.currentIndexChanged.connect(self.newComPort)
            self.cb_requestInterval.currentIndexChanged.connect(self.newRequestInterval)
            self.rb_Mode_1.clicked.connect(self.onMode1)
            self.rb_Mode_2.clicked.connect(self.onMode2)
            self.chk_showRawData.stateChanged.connect(self.onShowRawData)
            self.pb_selectFile.clicked.connect(self.showFileDialog)
            self.cb_dataStorage.stateChanged.connect(self.ondataStorage)
            self.pb_saveSettings.clicked.connect(self.onSaveSettings)
            self.pb_readSettings.clicked.connect(self.onReadSettings)
            self.le_storageFile.textChanged.connect(self.storageFileChanged)
            self.pb_cal.clicked.connect(self.onCalibration)
            self.cb_OPC.stateChanged.connect(self.onOPC)
            self.cb_temperature_compensation.stateChanged.connect(self.onTemperatureCompensation)
            self.cb_auto_temperature_compensation.stateChanged.connect(self.onAutoTemperatureCompensation)
            self.pb_cal_auto.clicked.connect(self.onAutoCalibration)
            self.cb_use_own_data.stateChanged.connect(self.onUseOwnData)
            self.pb_selectFile_2.clicked.connect(self.showFileDialog2)
            self.actionData_Plot_Explorer.triggered.connect(self.onDataExplorer)
            self.actionData_Plot_Creator.triggered.connect(self.onDataCreator)
            self.cb_offset_compensation.stateChanged.connect(self.onOffsetCompensation)
            self.pb_plot_creator.clicked.connect(self.onDataCreator)
            self.pb_plot_explorer.clicked.connect(self.onDataExplorer)
            self.pb_adjust_volume.clicked.connect(self.onUpdateVolume)
            self.pb_comp_offset.clicked.connect(self.onCompOffset)
            self.pb_adjust_delay.clicked.connect(self.onUpdateDelay)
            # Autostart GUI
            self.cb_use_own_data.hide()
            self.pb_selectFile_2.hide()
            self.lbl_offset_resistance.hide()
            self.le_lbl_comp_norm.hide()
            self.pb_comp_offset.hide()
            self.lbl_offset_resistance_sign.hide()
            # Autostart
            self.onReadSettings()
            self.onCalibration()
            self.onOK()

    def onUpdateDelay(self):
        global time_delay
        logToFile("onUpdateDelay wurde gedrückt",2)
        try:
            new_delay = float(self.le_lbl_responsedelay.text())
            self.si.update_delay(float(new_delay))
            if self.si.getDebug() == True:
                print(f"Update Volume: neue Verzögerung gesetzt auf {self.le_lbl_responsedelay.text()} Minuten")
            time_delay = new_delay
```

```
        except Exception as e:
            logToFile("onUpdateDelay error: " + str(e), 0)
            print("onUpdateDelay error: " + str(e))
    def onDataCreator(self):
        logToFile("onDataCreator wurde gedrückt",2)
        try:
            if self.onDataCreator_opened == False:
                self.plottingInterfaceGUI = piGUI(str(storageFile),self.si)
            self.plottingInterfaceGUI.show()
        except Exception as e:
            logToFile("onDataManager error: " + str(e), 0)
            print("onDataCreator error: " + str(e))

    def onDataExplorer(self):
        logToFile("onDataExplorer wurde gedrückt",2)
        try:
            if self.si.getDebug() == True:
                print("Opening Data Explorer Window")
            self.calTutorial = QDialog()           # Create an instance of the pop-up window (QDialog)
            self.calTutorial.setWindowTitle("Visualization of Methanol Probe Experiments")
            self.calTutorial.setFixedSize(800, 800)           # Set fixed size for the pop-up window
            layout = QVBoxLayout()           # Create a QVBoxLayout to hold the QScrollArea
            scroll = QScrollArea()           # Create a QScrollArea
            layout.addWidget(scroll)
            container = QtWidgets.QWidget()           # Create a widget container for the scroll area
            container_layout = QVBoxLayout()
            self.navigation_layout = QHBoxLayout()           # Create navigation buttons
            self.left_button = QPushButton("<")
            self.right_button = QPushButton(">")
            self.navigation_layout.addWidget(self.left_button)
            self.navigation_layout.addWidget(self.right_button)
            container_layout.addLayout(self.navigation_layout)
            self.image_label = QLabel()           # Create a QLabel to display the image
            container_layout.addWidget(self.image_label)
            # When no image is selected, display text saying "Select Folder with plot images" in the middle of the window
in big font
            self.image_label.setText("Select Folder with plot images")
            self.image_label.setAlignment(Qt.AlignCenter)
            self.image_label.setStyleSheet("font: 20pt")
            select_folder_button = QPushButton("Select Folder")           # Create a QPushButton to let the user select a
folder
            select_folder_button.clicked.connect(self.on_select_folder)
            container_layout.addWidget(select_folder_button)
            container.setLayout(container_layout)           # Assign the container layout to the container widget
            scroll.setWidget(container)           # Set the container widget as the widget for the QScrollArea
            scroll.setWidgetResizable(True)
            self.calTutorial.setLayout(layout)           # Set the QVBoxLayout as the layout for the QDialog
            self.image_files = []           # Initialize image list and index
            self.current_image_index = 0
            self.left_button.clicked.connect(self.show_previous_image)           # Connect navigation buttons
            self.right_button.clicked.connect(self.show_next_image)
            self.calTutorial.exec_()           # Show the pop-up window
        except Exception as e:
            print(f"Error in onDataExplorer: {e}")

    def on_select_folder(self):
        logToFile("on_select_folder wurde gedrückt",2)
        # Let the user choose a folder
        folder_path = QFileDialog.getExistingDirectory(self.calTutorial, "Select Folder")

        # Get the image files from the chosen folder
        if folder_path:

            self.image_label.setText("")           # remove the text saying "Select Folder with plot images" and the select
folder button
            self.image_files = [os.path.join(folder_path, f) for f in os.listdir(folder_path) if f.endswith('.png')]
            self.current_image_index = 0
            self.show_image(self.current_image_index)

    def show_image(self, index):
        if 0 <= index < len(self.image_files):
            pixmap = QPixmap(self.image_files[index])
            pixmap = pixmap.scaled(750, 750, aspectRatioMode=1)
            self.image_label.setPixmap(pixmap)

    def show_previous_image(self):
```

```
        self.current_image_index -= 1
        if self.current_image_index < 0:
            self.current_image_index = 0
        self.show_image(self.current_image_index)

    def show_next_image(self):
        self.current_image_index += 1
        if self.current_image_index >= len(self.image_files):
            self.current_image_index = len(self.image_files) - 1
        self.show_image(self.current_image_index)

    def onCompOffset(self):
        global si, cal_results, cal_status, cal_temperature, ohm
        logToFile(f"Compensate button has been pressed", 2)
        try:
            # read value from le_lbl_comp_norm
            c_offset = [0.1] # arbitrary low value for now
            r_offset = [float(self.le_lbl_comp_norm.text())]
            # use compensate linear from SensorInterface
            cal_results = si.compensate_linear(c_offset, r_offset, cal_results)
            cal_status = cal_results['fit_accuracy'][0]
            # display only 5 decimals for cal status if not a string already
            label_cal_status = cal_results['fit_accuracy'][0]
            self.le_lbl_cal_accuracy.setText(str(label_cal_status))
            cal_formula = getFormula(cal_results)
            self.le_lbl_cal_formula.setText(str(cal_formula))
            self.le_lbl_cal_temperature.setText(str(cal_temperature))
            if self.si.getDebug() == True:
                print(f"Calibration results with offset compensation for R_0: {r_offset[0]} {ohm} and c: {c_offset[0]}:",f"\n
{cal_results}")
        except Exception as e:
            logToFile(f"Error in onCompOffset: {e}", 2)
            print(f"Error in onCompOffset: {e}")

    def updateStatusBar(self):
        self.statusBar.showMessage(info_status, 1000)

    def onTemperatureCompensation(self, state): # checkbox gets checked
        global temperatureCompensation
        if state == QtCore.Qt.Checked:
            temperatureCompensation = True
            self.le_lbl_cal_temperature.show()
            self.cb_use_own_data.show()
            self.lbl_cal_temperature_sign.show()
            self.lbl_cal_temperature.show()
            self.cb_use_own_data.setChecked(False)
            self.cb_use_own_data.setEnabled(True)
            self.le_lbl_cal_temperature.setEnabled(True)
            logToFile("onTemperatureCompensation: temperatureCompensation gesetzt (" +
str(temperatureCompensation) + ")",2)
        else:
            temperatureCompensation = False
            self.pb_selectFile_2.hide()
            self.cb_use_own_data.hide()
            self.cb_use_own_data.setChecked(False)
            self.cb_use_own_data.setEnabled(False)

            logToFile("onTemperatureCompensation: temperatureCompensation NICHT gesetzt (" +
str(temperatureCompensation) + ")",2)

    def onOffsetCompensation(self, state): # checkbox gets checked
        global offsetCompensation
        if state == QtCore.Qt.Checked:
            offsetCompensation = True
            self.lbl_offset_resistance.show()
            self.le_lbl_comp_norm.show()
            self.lbl_offset_resistance_sign.show()
            self.pb_comp_offset.show()

            self.le_lbl_comp_norm.setEnabled(True)

            logToFile("onOffsetCompensation: offsetCompensation gesetzt (" + str(offsetCompensation) + ")",2)
        else:
            offsetCompensation = False
            self.lbl_offset_resistance.hide()
            self.le_lbl_comp_norm.hide()
```

```
        self.lbl_offset_resistance_sign.hide()
        self.pb_comp_offset.hide()
        logToFile("onOffsetCompensation: offsetCompensation NICHT gesetzt (" + str(offsetCompensation) + ")",2)

def onOK(self):
    global runState,opc, opc_init
    runState = not(runState)
    if runState:
        if openSerialPort():
            logToFile("onOK: Started",2)
            self.pb_run.setText("Stop")
            if opc_init == 0:
                try:
                    opc = OpenOPC.client()
                    opc.connect(OPC_Server)
                    opc_init = 1
                    logToFile(u"onOK:  OPC Verbindung mit " + OPC_Server + " hergestellt",2)
                except Exception as e:
                    logToFile(u"onOK:  OPC Verbindung mit " + OPC_Server + " konnte nicht hergestellt werden",2)
                    logToFile(u"onOK:  " + str(e),2)
                    opc_init = 0
            info_status = str(comPort + "      " + time.strftime("%d.%m.%Y %H:%M:%S"))
            self.statusBar().showMessage(info_status)
            readSerial()
    else:
        if closeSerialPort():
            logToFile("onOK: Stopped",2)
            self.statusBar().showMessage("...")
            self.pb_run.setText("Start")

def onAbbrechen(self):
    global runState, opc_init, opc
    closeSerialPort()
    if opc_init:
        opc.close()
        opc_init = 0
        logToFile(u"onAbbrechen:  OPC Verbindung mit " + OPC_Server + " geschlossen",2)
    logToFile("onAbbrechen: Ende",2)
    for widget in QtWidgets.QApplication.allWidgets():
        widget.close()
    self.close()

def newComPort(self, index):
    global comPort, runState
    comPort = comPorts[index]
    logToFile("newComPort: Der COM Port ist jetzt: " + str(comPort),2)
    if runState:
        logToFile("newComPort: onOK aufgerufen da runState = " + str(runState),2)
        self.onOK()
    else:
        logToFile("newComPort: serial Port " + str(comPort) + u" ausgewählt aber nicht geöffnet",2)

def onMode1(self):
    global requestMode
    requestMode = 1
    self.lbl_Tag_RS.hide()
    self.le_Value_RS.hide()
    self.le_TimeStamp_RS.hide()

    self.lbl_Tag_RSm.show()
    self.le_Value_RSm.show()
    self.le_TimeStamp_RSm.show()

    self.lbl_Tag_VL.hide()
    self.le_Value_VL.hide()
    self.le_TimeStamp_VL.hide()

    self.lbl_Tag_VLm.hide()
    self.le_Value_VLm.hide()
    self.le_TimeStamp_VLm.hide()

    self.lbl_Tag_VC.hide()
    self.le_Value_VC.hide()
    self.le_TimeStamp_VC.hide()

    self.lbl_Tag_VCm.hide()
```

```
        self.le_Value_VCm.hide()
        self.le_TimeStamp_VCm.hide()

        logToFile(u"onMode1: Request Mode 1 ausgewählt",2)

    def onMode2(self):
        global requestMode
        requestMode = 2
        self.lbl_Tag_RS.show()
        self.le_Value_RS.show()
        self.le_TimeStamp_RS.show()

        self.lbl_Tag_RSm.show()
        self.le_Value_RSm.show()
        self.le_TimeStamp_RSm.show()

        self.lbl_Tag_VL.show()
        self.le_Value_VL.show()
        self.le_TimeStamp_VL.show()

        self.lbl_Tag_VLm.show()
        self.le_Value_VLm.show()
        self.le_TimeStamp_VLm.show()

        self.lbl_Tag_VC.show()
        self.le_Value_VC.show()
        self.le_TimeStamp_VC.show()

        self.lbl_Tag_VCm.show()
        self.le_Value_VCm.show()
        self.le_TimeStamp_VCm.show()

        self.lbl_Tag_Rnorm.show()
        self.le_Value_Rnorm.show()
        self.le_TimeStamp_Rnorm.show()

        logToFile(u"onMode2: Request Mode 2 ausgewählt",2)

    def onShowRawData(self, state):
        global showRawData
        if state == QtCore.Qt.Checked:
            showRawData = True
            self.lbl_data.show()
        else:
            showRawData = False
            self.lbl_data.hide()

    def newRequestInterval(self, index):
        global requestInterval
        requestInterval = int(requestIntervals[index])
        logToFile("newRequestInterval: Das Abfrageinterval ist jetzt [sek]: %3i" % requestInterval, 2)

    def showFileDialog(self):
        global storageFile
        storageFile, _ = QtWidgets.QFileDialog.getSaveFileName(self, 'Save File', storageFile)
        self.le_storageFile.setText(storageFile)

    def onUseOwnData(self, state):
        global useImportedCalibration, historicalDataFile
        try:
            if state == QtCore.Qt.Checked:
                useImportedCalibration = True
                logToFile("onUseOwnData: useOwnData gesetzt",2)
                self.pb_selectFile_2.show()
                self.pb_selectFile_2.setEnabled(True)

            else:
                useImportedCalibration = False
                logToFile("onUseOwnData: useOwnData NICHT gesetzt",2)
                historicalDataFile = "historical_calibrations.txt"
                self.pb_selectFile_2.hide()
                self.pb_selectFile_2.setEnabled(False)
        except Exception as e:
            logToFile("onUseOwnData: Exception: " + str(e), 1)
            # open messagebox with error message
            msg = QtWidgets.QMessageBox()
```

```
            msg.setIcon(QtWidgets.QMessageBox.Critical)
            msg.setText("Error")
            msg.setInformativeText("Error: " + str(e))
            msg.setWindowTitle("Error")
            msg.exec_()
            return

    def showFileDialog2(self):
        global historicalDataFile
        historicalDataFile, _ = QtWidgets.QFileDialog.getOpenFileName(self, 'Open File', historicalDataFile)
        logToFile("showFileDialog2: historicalDataFile: " + str(historicalDataFile), 2)

    def ondataStorage(self, state):
        global dataStorage
        if state == QtCore.Qt.Checked:
            dataStorage = True
            logToFile("ondataStorage: dataStorage gesetzt",2)
        else:
            dataStorage = False
            logToFile("ondataStorage: dataStorage NICHT gesetzt",2)

    def storageFileChanged(self, string):
        global storageFile
        storageFile = self.le_storageFile.text()
        logToFile("storageFileChanged: New Storage File Name: " + str(storageFile), 2)

    def onSaveSettings(self):
        logToFile(u"onSaveSettings: Save Settings gedrückt",2)
        if self.si.getDebug() == True:
            print("onSaveSettings: Überschreibt die Einstellungen in der Datei: " + xml_File_Name + " mit den aktuellen
Einstellungen")
        self.onUpdate_UI()
        xml_write(xml_File_Name)

    def onReadSettings(self):
        logToFile(u"onReadSettings: Read Settings gedrückt",2)
        if self.si.getDebug() == True:
            print("onReadSettings: Übernimmt Einstellungen von : " + xml_File_Name)
        xml_read(xml_File_Name)
        self.onUpdate_UI()

    def onOPC(self, state):
        global OPC_enable, opc, opc_init
        try:
            if state == QtCore.Qt.Checked:
                OPC_enable = True
                logToFile("onOPC: OPC_enable gesetzt",2)
                if opc_init == 0:
                    opc = OpenOPC.client()
                    opc.connect(OPC_Server)
                    opc_init = 1
                    logToFile(u"onOPC:  OPC Verbindung mit " + OPC_Server + " hergestellt",2)
                else:
                    logToFile(u"onOPC:  OPC Verbindung mit " + OPC_Server + " besteht bereits.",2)
                self.le_Value_temp.setReadOnly(True)
            else:
                OPC_enable = False
                logToFile("onOPC: OPC_enable NICHT gesetzt",2)
                # if opc defined, close connection
                if opc_init == 1:
                    opc.close()

                opc_init = 0
                logToFile(u"onOPC:  OPC Verbindung mit " + OPC_Server + " geschlossen",2)
                self.le_Value_temp.setReadOnly(False)
        except Exception as e:
            logToFile("onOPC: Exception: " + str(e), 1)
            # open messagebox with error message
            msg = QtWidgets.QMessageBox()
            msg.setIcon(QtWidgets.QMessageBox.Critical)
            msg.setText("Error")
            msg.setInformativeText("Error: " + str(e))
            msg.setWindowTitle("Error")
            msg.exec_()
            return
```

```
def onCalibration(self):
    logToFile(u"onCalibration: Calibration gedrückt",2)
    try:
        global cal_status, cal_datensatz_benutzen, cal_konzentrationen, cal_widerstaende
        global cal_temperature
        global cal_results
        global si

        # Felder aus UI auslesen
        cal_datensatz_benutzen[0] = 1*self.cb_cal_use_1.isChecked()
        cal_datensatz_benutzen[1] = 1*self.cb_cal_use_2.isChecked()
        cal_datensatz_benutzen[2] = 1*self.cb_cal_use_3.isChecked()
        cal_datensatz_benutzen[3] = 1*self.cb_cal_use_4.isChecked()
        cal_datensatz_benutzen[4] = 1*self.cb_cal_use_5.isChecked()
        cal_datensatz_benutzen[5] = 1*self.cb_cal_use_6.isChecked()

        cal_konzentrationen[0] = strToFloat(self.le_cal_concentration_1.text())
        cal_konzentrationen[1] = strToFloat(self.le_cal_concentration_2.text())
        cal_konzentrationen[2] = strToFloat(self.le_cal_concentration_3.text())
        cal_konzentrationen[3] = strToFloat(self.le_cal_concentration_4.text())
        cal_konzentrationen[4] = strToFloat(self.le_cal_concentration_5.text())
        cal_konzentrationen[5] = strToFloat(self.le_cal_concentration_6.text())

        cal_widerstaende[0] = strToFloat(self.le_lbl_cal_resistance_1.text())
        cal_widerstaende[1] = strToFloat(self.le_lbl_cal_resistance_2.text())
        cal_widerstaende[2] = strToFloat(self.le_lbl_cal_resistance_3.text())
        cal_widerstaende[3] = strToFloat(self.le_lbl_cal_resistance_4.text())
        cal_widerstaende[4] = strToFloat(self.le_lbl_cal_resistance_5.text())
        cal_widerstaende[5] = strToFloat(self.le_lbl_cal_resistance_6.text())

        cal_temperature = strToFloat(self.le_lbl_cal_temperature.text())
        if self.si.getDebug() == True:
            print("cal_temperature: " + str(cal_temperature))
        cal_results = calibration(cal_datensatz_benutzen, cal_konzentrationen, cal_widerstaende, cal_temperature)
        self.cb_offset_compensation.setChecked(False)
        self.cb_offset_compensation.setEnabled(True)
        # Update labels
        cal_status = cal_results['fit_accuracy'][0]
        # display only 5 decimals for cal status if not a string already
        label_cal_status = cal_results['fit_accuracy'][0]
        if type(label_cal_status) is not str:
            label_cal_status = round(label_cal_status, 5)
        self.le_lbl_cal_accuracy.setText(str(label_cal_status))
        cal_formula = getFormula(cal_results)
        self.le_lbl_cal_formula.setText(str(cal_formula))
        self.le_lbl_cal_temperature.setText(str(cal_temperature))
        self.le_Value_temp_cal.setText(str(self.le_lbl_cal_temperature.text()))
        si = self.si
    except ValueError:
        logToFile(u"onCalibration: Calibration Fehlerhaft",2)
        self.le_lbl_cal_accuracy.setText("Error")
        self.le_lbl_cal_formula.setText("Error")
        self.le_lbl_cal_temperature.setText("Error")
    except Exception as e:
        logToFile(u"onCalibration: Calibration Fehlerhaft",2)
        self.le_lbl_cal_accuracy.setText("Error")
        self.le_lbl_cal_formula.setText("Error")
        self.le_lbl_cal_temperature.setText("Error")
        logToFile("onCalibration: " + str(e),2)

def onUpdateVolume(self):
    global volume_med, si
    new_volume =    strToFloat(self.le_lbl_volume.text())
    new_volume =    str(round(new_volume,2))
    logToFile(u"onUpdateVolume: Update Volume gedrückt, neues Volumen: "+new_volume,2)
    if self.si.getDebug() == True:
        print(f"onUpdateVolume: Update Volume gedrückt, neues Volumen: {new_volume}"    )
    self.si.update_volume(new_volume)
    self.le_lbl_volume.setText(new_volume)
    self.le_Value_volume.setText(new_volume)
    volume_med = new_volume
    si = self.si

def onAutoCalibration(self):
    logToFile(u"onAutoCalibrate: AutoCalibrate gedrückt",2)
    try:
```

```
            self.sensorInterface = siGUI()
            self.sensorInterface.show()
            self.sensorInterface.raise_()
            self.sensorInterface.activateWindow()
        except AttributeError:
            self.sensorInterface.show()

    def onAutoCalibrationClicked(self):
        logToFile(u"onAutoCalibrationClicked: Calibrate in SensorInterfaceGUI gedrückt",2)

    def onAutoTemperatureCompensation(self,state):
        global autoTemperatureCompensation
        if state == QtCore.Qt.Checked:
            autoTemperatureCompensation = True
            logToFile("onAutoTemperatureCompensation: autoTemperatureCompensation gesetzt (" +
str(autoTemperatureCompensation) + ")",2)
        else:
            autoTemperatureCompensation = False
            logToFile("onAutoTemperatureCompensation: autoTemperatureCompensation NICHT gesetzt (" +
str(autoTemperatureCompensation) + ")",2)

    def onUpdate_UI(self):
        global comPort, requestInterval, requestMode, dataStorage, storageFile
        global cal_datensatz_benutzen, cal_konzentrationen, cal_widerstaende, cal_temperature,
temperatureCompensation, temperature, autoTemperatureCompensation, si, volume_med, time_delay
        logToFile("onUpdate_UI: Update_UI augerufen",2)

        # comPort
        index = self.cb_comPort.findText(comPort, QtCore.Qt.MatchFixedString)
        if index >= 0:
            self.cb_comPort.setCurrentIndex(index)
            logToFile("onUpdate_UI: comPort - index: "+ str(index) + " comPort: " + str(comPort),2)
        else:
            logToFile("onUpdate_UI: comPort - fehler comPort: " + str(comPort),2)

        # requestInterval
        index = self.cb_requestInterval.findText(str(requestInterval), QtCore.Qt.MatchFixedString)
        if index >= 0:
            self.cb_requestInterval.setCurrentIndex(index)
            logToFile("onUpdate_UI: requestInterval - index: " + str(index) + " requestInterval: " + str(requestInterval),2)
        else:
            logToFile("onUpdate_UI: requestInterval - fehler requestInterval: " + str(requestInterval),2)

        # requestMode
        if requestMode == 1:
            logToFile("onUpdate_UI: requestMode 1: requestMode: " + str(requestMode),2)
            self.rb_Mode_1.setChecked(True)
            self.onMode1()

        if requestMode == 2:
            logToFile("onUpdate_UI: requestMode 2: requestMode: " + str(requestMode),2)
            self.rb_Mode_2.setChecked(True)
            self.onMode2()

        # dataStorage
        self.cb_dataStorage.setChecked(dataStorage)
        logToFile("onUpdate_UI: dataStorage: "+ str(dataStorage),2)

        # storageFile
        self.le_storageFile.setText(storageFile)

        # OPC_enable
        self.cb_OPC.setChecked(OPC_enable)
        logToFile("onUpdate_UI: OPC_enable: " + str(OPC_enable),2)

        # temperatureCompensation
        self.cb_temperature_compensation.setChecked(temperatureCompensation)
        logToFile("onUpdate_UI: Temperature Compensation: " + str(temperatureCompensation),2)

        # autoTemperatureCompensation
        self.cb_auto_temperature_compensation.setChecked(autoTemperatureCompensation)
        logToFile("onUpdate_UI: Auto Temperature Compensation: " + str(autoTemperatureCompensation),2)

        # cal_datensatz_benutzen
        self.cb_cal_use_1.setChecked(bool(cal_datensatz_benutzen[0]))
        self.cb_cal_use_2.setChecked(bool(cal_datensatz_benutzen[1]))
```

```
        self.cb_cal_use_3.setChecked(bool(cal_datensatz_benutzen[2]))
        self.cb_cal_use_4.setChecked(bool(cal_datensatz_benutzen[3]))
        self.cb_cal_use_5.setChecked(bool(cal_datensatz_benutzen[4]))
        self.cb_cal_use_6.setChecked(bool(cal_datensatz_benutzen[5]))

        self.le_cal_concentration_1.setText(str(cal_konzentrationen[0]))
        self.le_cal_concentration_2.setText(str(cal_konzentrationen[1]))
        self.le_cal_concentration_3.setText(str(cal_konzentrationen[2]))
        self.le_cal_concentration_4.setText(str(cal_konzentrationen[3]))
        self.le_cal_concentration_5.setText(str(cal_konzentrationen[4]))
        self.le_cal_concentration_6.setText(str(cal_konzentrationen[5]))

        self.le_lbl_cal_resistance_1.setText(str(cal_widerstaende[0]))
        self.le_lbl_cal_resistance_2.setText(str(cal_widerstaende[1]))
        self.le_lbl_cal_resistance_3.setText(str(cal_widerstaende[2]))
        self.le_lbl_cal_resistance_4.setText(str(cal_widerstaende[3]))
        self.le_lbl_cal_resistance_5.setText(str(cal_widerstaende[4]))
        self.le_lbl_cal_resistance_6.setText(str(cal_widerstaende[5]))

        self.le_lbl_cal_temperature.setText(str(cal_temperature))
        #print "Update UI ", cal_temperature
        self.le_Value_temp.setText("%5.2f" % (float(volume_med)))
        self.le_Value_volume.setText("%5.2f" % (float(volume_med)))
        self.le_TimeStamp_temp.setText("manual")
        self.le_lbl_volume.setText("%5.2f" % (float(volume_med)))
        self.le_lbl_responsedelay.setText("%5.2f" % (float(time_delay)))

app = QtWidgets.QApplication(sys.argv)
dialog = ProgrammFenster()
dialog.show()
sys.exit(app.exec_())
```

########## Programmcode SensorInterface.py ############

```
import pandas as pd
import numpy as np
from datetime import timedelta
import json

class SensorInterface:
    def __init__(self,  cal_curve = None, initial_volume=1000, th_med=30, degree=1, delay=12,
density_methanol=0.7815, debug = True): # Using the densitiy of methanol at 30 °C is 781.5 g/L or 0.7815 g/mL,
alternatively 0.791 g/mL at 20 °C
        # Experiment object for plotting
        self.th_med = th_med # °C
        self.density_methanol = density_methanol # g/mL
        self.degree = degree # Polynomial degree for calibration
        self.volume = initial_volume # mL
        self.initial_volume = initial_volume # mL
        self.data = pd.DataFrame(columns=['time', 'r_sensor', 'rel_hum', 'pressure', 'th_gas', 'th_med', 'r_norm',
'ah_gases']) # EMPTY DataFrame for storing the data
        self.delay = delay # Time delay between addition of methanol and measurement, depends on the carrier gas
flow rate and thickness of the silicone tubing around the membrane
        if cal_curve is None:
            avg_cal_curve = pd.DataFrame({'fit_accuracy': [0.9969420], 'cal_temp': [30], 'a_0': [-9.34], 'a_1': [99.19]})
# old average calibration curve
        self.cal_curve = avg_cal_curve # Calibration curve
        self.debug = debug # if True, print debug messages
        if self.debug:
            print(f"SensorInterface initialized with calibration curve:\n", f"{self.cal_curve}")       # has to be separated by
comma in order for the cal curve data to be displayed in new line strangely
    def update_degree (self, degree):
        self.degree = degree
        if self.debug:
            print(f"    Degree updated to {self.degree}")

    def update_volume (self, volume):
        self.volume = volume
        if self.debug:
            print(f"    Volume updated to {self.volume}")

    def update_th_med (self, th_med):
        self.th_med = th_med
```

```
        if self.debug:
            print(f"    th_med updated to {self.th_med}")

    def update_delay (self, delay):
        self.delay = timedelta(minutes=delay)
        if self.debug:
            print(f"    delay updated to {self.delay}")

    def update_data (self, data):
        self.data = data # dangerous because no checks are performed, only use if you know what you are doing
        if self.debug:
            print(f"    data updated to {self.data}")

    def update_cal_curve (self, cal_curve):
        self.cal_curve = cal_curve
        if self.debug:
            print(f"    cal_curve updated to {self.cal_curve}")
    def clean_txt(self, file_path):
        try:        # this function is used before importing txt file to delete any line that says "Invalid Data !!!  ["] Elements:
0", which results from faulty sensor readings or connection errors with the opc server
            line_counter = 0
            with open(file_path, 'r') as f:
                lines = f.readlines()
                f.close()
            with open(file_path, 'w') as f:
                for line in lines:
                    if line.strip("\n") != "Invalid Data !!!  ["] Elements: 0": # instead of checking for this exact string, it would be
better to check if the line is incorrectly formatted or contains "Invalid Data !!!" and delete it if it does
                        f.write(line)
                    else:
                        line_counter += 1 # this counter is used to print the number of deleted lines and can be used to check
if the cleaning function worked properly
                f.close()
            if self.debug and line_counter > 0:
                print(f"    Finished cleaning txt file. Deleted {line_counter} lines.")
        except Exception as e:
            if self.debug:
                print(f"    SensorInterface error while cleaning txt: {e}")
            return

    def import_txt(self, file_path):
        try:
            self.clean_txt(file_path)          # Clean the txt file, can be optimized by adding a check if the file is already
clean (e.g. by checking a prefix/suffix in the file name or changing the first line of the txt file to "cleaned")
            with open(file_path, 'r') as f:             # Open the file and read the data
                f.readline()                # Skip the first two rows (header)
                f.readline()
                times = []                # Parse the remaining rows and store the data in lists
                time_strings = []
                rs_sensors = []
                rel_hums = []
                pressures = []
                th_gases = []
                th_meds = []
                r_norms = []
                ah_gases = []
                for line in f:
                    values = line.split()
                    rs_sensors.append(float(values[1]))
                    rel_hums.append(float(values[10]))
                    pressures.append(float(values[9]))
                    th_gases.append(float(values[8]))
                    th_meds.append(self.th_med)
                    r_norm, ah_gas = self.normalize_resistance(float(values[1]), float(values[8]), float(values[10])),
self.get_AH(float(values[10]), float(values[8]))
                    r_norms.append(r_norm)
                    ah_gases.append(ah_gas)
                    time_strings.append(values[11] + " " + values[12])                    # convert DD.MM.YYYY HH:MM:SS to
YYYY-MM-DD HH:MM:SS to cause unnecessary problems and confusion for the user (as it turns out)
                    values[11] = values[11].split(".")
                    values[11] = "-".join([values[11][2], values[11][1], values[11][0]])
                    times.append(pd.Timestamp(values[11] + " " + values[12]))
            self.data = pd.DataFrame({'time': times,             # Create the DataFrame with the parsed data
                                      'time_string': time_strings, # this is only used for plotting and can be removed if not needed
                                    'r_sensor': rs_sensors,
                                    'rel_hum': rel_hums,
```

```
                        'pressure': pressures, # pressure is neither used in the calibration nor in the compensation function, but can be used for further analysis
                        'th_gas': th_gases,
                        'th_med': th_meds,
                        'r_norm': r_norms,
                        'ah_gas': ah_gases
                        })
            self.data = self.data.dropna()          # drop NaN values
            self.data = self.data.reset_index(drop=True)          # reset index
            if self.debug:
                print(f"    Finished importing txt file. Imported {len(self.data)} rows of data.")
            return self.data
        except Exception as e:
            if self.debug:
                print(f"SensorInterface error while importing txt: ", e)
            return None

    def convert_m_to_percent(self, m):
        return (m / (self.volume * self.density_methanol)) * 100 # example 0.4746 g / (1500 mL * 0.791 g/mL) * 100 = 0.04 vol%

    def convert_percent_to_m(self, c):
        return (c / 100) * self.volume * self.density_methanol # example 0.04 vol% * 1500 mL * 0.791 g/mL = 0.4746 g = 474.6 mg or as Volume 0.6 mL

    def get_cal_data(self, masses, addition_times):    # This function looks through the data and returns the average sensor value for the given time range (start_time, end_time) where start_time is the addition time plus the time delay a # change temperature and time delay before calling this function (if necessary)
        if self.debug:
            print(f"SensorInterface: Get_cal_data -> trying to get calibration data for {masses} g at {addition_times}\n Head \n{self.data.head(1)}\n Tail \n{self.data.tail(1)}")
        try:
            if type(addition_times) == str:          # convert strings in addition_times list to pd.Timestamps
                addition_times = pd.Timestamp(addition_times)
            temperature = self.th_med
            if type(self.delay) == str or type(self.delay) == int:          # check type of time delay and convert to timedelta if necessary
                time_delay = pd.Timedelta(minutes= self.delay)
            else:
                time_delay = self.delay
            self.volume = self.initial_volume
            cal_data = []
            cum_c = 0
            for mass, addition_time in zip(masses, addition_times):
                c_cal = self.convert_m_to_percent(mass)              # Convert mass to concentration
                cum_c += c_cal
                self.volume += mass / self.density_methanol              # Correct volume for mass addition
                start_time = pd.Timestamp(addition_time) + time_delay              # Calculate time range for averaging, 60 seconds is arbitrary, needs to be optimized
                end_time = start_time + pd.Timedelta(seconds=60) # instead of averaging noisy data, the code from the arduino can be modified to smooth the sensor readings directly
                mask = (self.data['time'] >= start_time) & (self.data['time'] <= end_time)              # Average sensor values in time range (removed currently)
                r_norm = np.mean(self.data.loc[mask, 'r_norm'])
                cal_data.append([cum_c, r_norm, temperature])              # Append data to cal_data list
            cal_data_df = pd.DataFrame(cal_data, columns=['c_cal', 'r_norm', 'cal_temp'])          # Convert cal_data list to DataFrame
            if self.debug:
                print(f"SensorInterface: Get_cal_data -> returning cal_data_df: {cal_data_df}")
            return cal_data_df
        except Exception as e:
            print(e)
            return None

    def get_AH(self, rel_hum, th_gas):  # calculate absolute humidity in g/m3
        S = 0.0259 * th_gas**2 + 0.0608 * th_gas + 5.7252
        AH = rel_hum * S / 100
        return AH

    def normalize_resistance(self, r_sensor, th_gas, rel_hum): # normalize resistance to 20°C and 65% relative humidity
        AH = self.get_AH(rel_hum, th_gas)       # Calculate absolute humidity
        r_norm = r_sensor + r_sensor * (1 - (2.7786 * AH**(-0.428)))       # Calculate normalized resistance
        return r_norm
```

```
    def fit_cal_data(self, cal_data_df, temp_coeffs=None):      # calculates calibration coefficients for calibration data or fits coefficients to calibration data and returns updated accuracy
        try:        # This function fits a polynomial function to the calibration data and returns the coefficients of the polynomial
            y = np.log(cal_data_df['c_cal'])            # Extract x and y values from calibration data DataFrame and use logarithmic scale
            x = np.log(cal_data_df['r_norm'])
            if temp_coeffs is None:
                coeffs = np.polyfit(x, y, self.degree)              # Fit polynomial function to calibration data
                multiplier = 1
                p = np.poly1d(coeffs)               # Create polynomial from the coefficients
            else:       # If coefficients are passed as input, the function uses the provided coefficients instead of fitting to calculate the fit accuracy for the coefficients to the calibration data
                coeffs = temp_coeffs                # Use provided coefficients instead of fitting
                multiplier = 0.1 # to make the fit accuracy comparable to the fit accuracy of the other sensors, needs to be updated in the GUI though
                p = np.poly1d(coeffs)  # coeffs should be in decreasing order of powers
            y_mean = np.mean(y)             # Calculate fit accuracy (R-squared value)
            ss_res = np.sum((y - p(x))**2)
            ss_tot = np.sum((y - y_mean)**2)
            r_squared = 1 - (ss_res / ss_tot)
            coeff_names = ['a_' + str(i) for i in range(len(coeffs))]                   # Create DataFrame with coefficients, calibration temperature, and fit accuracy
            coeff_vals = np.round(coeffs, 8)                                            # looks like this: a_0, a_1, ..., a_n and is accidentially in decreasing order of powers! Bug fixed by importing them in reversing order in fit_cal_data
            fit_accuracy = np.round(r_squared, 8)*multiplier
            coeff_data = np.concatenate([[fit_accuracy], [cal_data_df['cal_temp'].iloc[0]], coeff_vals])
            coeff_df = pd.DataFrame(coeff_data.reshape(1, -1), columns=['fit_accuracy'] + ['cal_temp'] + coeff_names) # looks like this: fit_accuracy, cal_temp, a_0, a_1, ..., a_n , to create empty use: coeff_df = pd.DataFrame(columns=['fit_accuracy'] + ['cal_temp'] + coeff_names)
            self.update_cal_curve(coeff_df)
            if self.debug:
                print(f"SensorInterface calibrated with cal_curve: ",f"\n{self.cal_curve}\nFor this calibration data: \n{cal_data_df}")
            return coeff_df
        except Exception as e:
            print(e)
            return None

    def linear_interpolation(self,x0, y0, x1, y1, x):
        # Check if the input values are numbers
        if not all(isinstance(val, (int, float)) for val in [x0, y0, x1, y1, x]):
            raise TypeError("All input values must be numbers")

        # Check for division by zero
        if x0 == x1:
            raise ValueError("x0 and x1 cannot be equal")

        # Calculate the slope (y1 - y0) / (x1 - x0)
        slope = (y1 - y0) / (x1 - x0)

        # Use the point-slope form of a linear equation to calculate y
        return y0 + slope * (x - x0)

    def get_temp_coeff(self, cal_data_df, measure_temp):
        try:
            prev_temp = self.getMediumTemp()
            self.update_th_med(measure_temp)
            polynomial_degree = len(cal_data_df.columns) - 2            # Extract degree of polynomial function from the number of coefficients
            cal_data_df = cal_data_df.sort_values('cal_temp')           # Sort calibration data DataFrame by temperature
            cal_data_df = cal_data_df.groupby('cal_temp').mean().reset_index()          # Double entries for one temperature all get averaged
            interpolated_coeffs = np.zeros(polynomial_degree)           # Initialize array for interpolated coefficients
            temperatures = cal_data_df['cal_temp'].values           # Extract temperature values and coefficients
            for i in range(polynomial_degree):          # Interpolate each coefficient at the measurement temperature
                coefficients = cal_data_df['a_' + str(i)].values
                for j in range(len(temperatures) - 1):              # Find the interval where the measurement temperature is
                    if temperatures[j] <= measure_temp <= temperatures[j + 1]:
                        break
                x0, y0 = temperatures[j], coefficients[j]               # Use linear interpolation for both interpolation and extrapolation
                x1, y1 = temperatures[j + 1], coefficients[j + 1]
                interpolated_coeffs[i] = self.linear_interpolation(x0, y0, x1, y1, measure_temp)
            coeff_names = ['a_' + str(i) for i in range(len(interpolated_coeffs))]
            coeff_df = pd.DataFrame(columns=['fit_accuracy'] + ['cal_temp'] + coeff_names)
```

```
            coeff_df['cal_temp'] = [prev_temp]
            coeff_df[coeff_names] = interpolated_coeffs
            self.update_cal_curve(coeff_df)
            if self.debug:
                print(f"Temperature compensation has been performed with these coefficients: \n{cal_data_df}")
            return interpolated_coeffs

        except Exception as e:
            print(e)
            return None

    def get_coeffs(self):   # reverses the coefficients so poly1d works with them
        coeffs = [self.cal_curve['a_' + str(i)].iloc[0] for i in range(self.degree)]
        return coeffs

    def get_concentration(self, r_sensor, rel_hum, th_gas, coefficients):
        try:
            # Create model from coefficients
            p = np.poly1d(coefficients)

            # Normalize sensor resistance
            r_norm = self.normalize_resistance(r_sensor, th_gas, rel_hum)

            # Calculate concentration from sensor resistance and temperature correction factor
            c_cal = np.exp(p(np.log(r_norm)))

            return c_cal
        except Exception as e:
            print(e)
            return None

    def get_concentration_rnorm(self, r_norm, coefficients):
        try:
            # Create model from coefficients
            p = np.poly1d(coefficients)

            # Calculate concentration from sensor resistance and temperature correction factor
            c_cal = np.exp(p(np.log(r_norm)))

            return c_cal
        except Exception as e:
            print(e)
            return None

    def compensate_linear(self, conc = 0.1, r_norm = 50000, avg_curve= None):      #this function is used to offset the
calibration curve to the new calibration point at a concentration of 0.1 vol% and a provided resistance value
        try:
            # Make a deep copy of avg_curve
            avg_curve_comp_cal = avg_curve.copy(deep=True)

            # Initialize a list to store the errors
            errors = []

            # Iterate over the additional calibration points
            for c, r in zip(conc, r_norm):
                # Calculate the predicted concentration using the original calibration curve
                a_0 = avg_curve_comp_cal.iat[0, avg_curve_comp_cal.columns.get_loc('a_0')]
                a_1 = avg_curve_comp_cal.iat[0, avg_curve_comp_cal.columns.get_loc('a_1')]
                c_pred = np.exp(a_0 * np.log(r) + a_1)

                # Calculate the error as the absolute difference between the actual and predicted concentrations
                error = abs(np.log(c) - np.log(c_pred))

                # Calculate the offset as the difference between the actual and predicted concentrations
                offset = np.log(c) - np.log(c_pred)

                # Adjust the intercept of the calibration curve by adding the offset
                avg_curve_comp_cal.iat[0, avg_curve_comp_cal.columns.get_loc('a_1')] += offset

                # Add the error to the list of errors
                errors.append(error)

            # Calculate the average error
            avg_error = np.mean(errors)

            # Adjust the fit accuracy by subtracting the average error from 100%
```

```
            avg_curve_comp_cal.iat[0, avg_curve_comp_cal.columns.get_loc('fit_accuracy')] = 100 - avg_error

            # Set the calibration temperature to 30
            avg_curve_comp_cal.iat[0, avg_curve_comp_cal.columns.get_loc('cal_temp')] = 30

            return avg_curve_comp_cal
        except Exception as e:
            print(e)
            return None

    def getMediumTemp(self):
        return float(self.th_med)
    def getDelay(self):
        return float(self.delay)
    def getCalCurve(self):
        return self.cal_curve
    def getVolume(self):
        return float(self.volume)
    def getDegree(self):
        return int(self.degree)
    def getMeOHdensity(self):
        return float(self.density_methanol)
    def getDebug(self):
        return bool(self.debug)
    def updateDebug(self, debug):
        self.debug = debug

    def dataToTXT (self):
        data = self.data
        # convert to dictionary
        #data = data.to_dict()
        # create txt file and write DataFrame to it line by line
        with open('data.txt', 'w') as f:
            f.write(data.to_string(header = True, index =True ))

        return 1

def main():
    si = SensorInterface()
    si.update_volume(1000)
    print(f"Table for conversion of percents to masses from 0.1 to 1.0 vol%: \n")
    percents = [0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1, 1.3]
    masses =  [si.convert_percent_to_m(percent) for percent in percents]
    for percent, mass in zip(percents, masses):
        print(f"{percent} vol% = {mass} g")
    #filepath = "/home/ratibor/Documents/Bachelorarbeit Biotech/Meth_Sensor/Meine GUI
Labor/data/data_merged.txt"
    #si = SensorInterface()
    #si.import_txt(filepath)
    #si.dataToTXT()

if __name__ == "__main__":
    main()



############ SensorInterface GUI  #################

################### Written By: Philipp Schmidt for Bachelor Thesis to analyse the influence of medium
temperature on calibration coefficients ###################   free to use for everyone   ###################
import sys
from PyQt5.QtWidgets import (QApplication, QWidget, QPushButton, QVBoxLayout, QHBoxLayout, QLabel,
QLineEdit,
                             QFileDialog, QListWidget, QFormLayout, QSpinBox, QDateTimeEdit, QPlainTextEdit, QTabWidget)
from PyQt5.QtWidgets import QMessageBox, QTreeWidget, QTreeWidgetItem, QCheckBox, QTableWidget,
QTableWidgetItem, QDoubleSpinBox
from PyQt5.QtCore import Qt, QDateTime
from SensorInterface import SensorInterface
import pandas as pd
from datetime import datetime, timedelta
import numpy as np
import os
```

```
class SensorInterfaceGUI(QWidget, SensorInterface):  # developed for the bachelor thesis of Philipp Schmidt to make handling of the sensor interface easier and more intuitive, 2023
                        # Instead of programming the GUI completely from scratch, QT Designer could have been used to create the GUI and then convert the .ui file to a .py file which makes it easier to add functionality to the GUI or make modifications
                        # The GUI is divided into 2 tabs: the calibration tab and the compensation tab. The calibration tab is used to calibrate with the sensor readings and the compensation tab is used to compensate the created calibration curves for different medium temperature values.
    def __init__(self, parent=None, si = None):

        super().__init__(parent)
        if si is None:
            self.si = SensorInterface()
        else:
            self.si = si
        self.setWindowTitle("Calibration Interface")
        self.setFixedWidth(550)
        self.txt_file_path = None       # Adding widgets to calibrationTab below here
        self.calibration_data = []
        self.degreeSpinBox = QSpinBox()
        self.degreeSpinBox.setRange(0, 10)
        self.degreeSpinBox.setValue(1)
        self.degreeSpinBox.setToolTip("Polynomial Degree should be lower than number of calibration data points")
        self.tempSpinBox = QDoubleSpinBox()
        self.tempSpinBox.setRange(0.0, 50.0)
        self.tempSpinBox.setValue(30)
        self.tempSpinBox.setToolTip("Medium temperature in degrees Celsius at time of calibration")
        self.delaySpinBox = QSpinBox()
        self.delaySpinBox.setRange(0, 99)
        self.delaySpinBox.setValue(0)
        self.delaySpinBox.setToolTip("Time delay in minutes between addition and measurement. At rotameter setting 80 (ca. 22ml/min), the delay is roughly 12 minutes")
        formLayout = QFormLayout()
        formLayout.addRow("Polynomial Degree:", self.degreeSpinBox)
        formLayout.addRow("Temperature:", self.tempSpinBox)
        formLayout.addRow("Time Delay:", self.delaySpinBox)
        self.calibrationButton = QPushButton("Load historical sensor data")
        self.calibrationButton.clicked.connect(self.start_calibration)
        self.calibrationButton.setToolTip("Select poly degree, medium temperature and time delay before loading calibration data. Data is gathered using the TGS2620.py script and saved to a txt file automatically.")
        self.massLineEdit = QLineEdit()
        self.massLineEdit.setPlaceholderText("Enter injected MeOH mass in grams(e.g. 4.0)")
        self.massLineEdit.setToolTip("Enter injected MeOH mass in grams for the given timestep (not cumulative)")
        self.timeEdit = QDateTimeEdit(QDateTime.currentDateTime())
        self.timeEdit.setDisplayFormat("yyyy-MM-dd HH:mm:ss")
        self.timeEdit.setToolTip("Select time of injection in format yyyy-MM-dd HH:mm:ss (from txt file)")
        self.addDataButton = QPushButton("Add Entry")
        self.addDataButton.clicked.connect(self.add_entry)
        self.addDataButton.setToolTip("Add entry to be used as a calibration point. Enter mass and time of injection and click Add Entry.")
        self.dataListWidget = QListWidget()
        self.deleteButton = QPushButton("Delete Selected")
        self.deleteButton.clicked.connect(self.deleteSelected)
        self.deleteButton.setToolTip("Delete highlighted entry from calibration data")
        # connect the delay spin box with the onDelayChanged function
        self.delaySpinBox.valueChanged.connect(self.onDelayChanged)
        # Historical data for testing
        # time delay = 12 , th_med = 30
        masses = [4.0, 2.4, 0.8, 0.7, 0.8, 1.6]
        times = ['2022-11-15 14:32:00', '2022-11-15 14:49:00', '2022-11-15 15:07:00', '2022-11-15 15:21:00', '2022-11-15 15:33:00', '2022-11-15 15:46:00']
        # time delay = 0, th_med = 21
        masses2 = [4.0, 2.4, 0.8, 0.7, 0.8, 1.6]
        times2 =['2023-01-23 16:39:00', '2023-01-23 16:50:00', '2023-01-23 17:00:00', '2023-01-23 17:11:00', '2023-01-23 17:21:00', '2023-01-23 17:33:00']
        # th_med = 18
        masses3 = [4.0, 2.4, 0.8, 0.7, 0.8, 1.6]
        times3 =  ['2023-01-23 14:35:00', '2023-01-23 14:54:00', '2023-01-23 15:06:00', '2023-01-23 15:28:00', '2023-01-23 15:38:00', '2023-01-23 15:48:00']
        # th_med = 25
        masses4 = [4.0, 2.4, 0.8, 0.7, 0.8, 1.6]
        times4 =  ['2023-01-23 12:21:00', '2023-01-23 12:33:00', '2023-01-23 12:48:00', '2023-01-23 13:01:00', '2023-01-23 13:16:00', '2023-01-23 13:31:00']
        # time_delay = 0, th_med = 26 *Kalibrierung im Becherglas 1 *fail*
        masses5 = [4.0, 2.4, 0.8, 0.7, 0.8, 1.6]
```

```
        times5 = ['2023-06-07 15:11:00','2023-07-06 15:27:00','2023-07-06 15:47:00','2023-07-06 16:04:00','2023-07-06
16:20:00','2023-07-06 16:38:00']
        # time_delay 0, th_med = 25.5 *Kalibrierung im Becherglas 2, teil der langen log datei vom 8.6.*
        masses6 = [4.02, 2.42, 0.79, 0.8, 0.82, 1.6]
        times6 = ['2023-06-08 15:59:00','2023-06-08 16:22:00','2023-06-08 16:40:00','2023-06-08 16:58:00','2023-06-08
17:17:00','2023-06-08 17:44:00']
        # time_delay 0, th_med= 30 *Kalibrierung im Reaktor vor Autoklavieren*
        masses7 = [4.04, 2.41, 0.78, 0.8, 0.83, 1.61]
        times7 = ['2023-06-09 11:02:00','2023-06-09 11:22:00','2023-06-09 11:42:00','2023-06-09 12:04:00','2023-06-09
12:25:00','2023-06-09 12:46:00']
        # time_delay 0, th_med= 30 *Kalibrierung im Reaktor nach Autoklavieren*
        masses8 = [4.01, 2.37, 0.78, 0.8, 0.83, 1.62]
        times8 = ['2023-06-12 13:57:00', '2023-06-12 14:20:00', '2023-06-12 14:45:00', '2023-06-12 15:07:00', '2023-06-
12 15:33:00', '2023-06-12 15:53:00']
        # prepopulate dataListWidget with historical data
        for mass, time in zip(masses, times):
            self.calibration_data.append((mass, time))
            self.dataListWidget.addItem(f"Mass: {mass},      Time: {time}")
        self.calibrateButton = QPushButton("Calibrate Now")
        self.calibrateButton.clicked.connect(self.calibrate)
        self.calibrateButton.setEnabled(False)
        self.calibrateButton.setToolTip("Select historical data txt file first. Create calibration curve from selected data
points and polynomial degree. Coefficients are saved to a txt file for later medium temperature compensation.")
        self.outputTextEdit = QPlainTextEdit()
        self.outputTextEdit.setStyleSheet("QPlainTextEdit {line-height: 10px;}")
        self.outputTextEdit.setMaximumHeight(50)
        # todo: dynamically adjust width of outputTextEdit to adapt to Polynomial Degree

        self.outputTextEdit.setReadOnly(True)

        calibrationTab = QWidget()
        temperatureTab = QWidget()

        calibrationLayout = QVBoxLayout()
        calibrationLayout.addLayout(formLayout)
        calibrationLayout.addWidget(self.calibrationButton)
        calibrationLayout.addWidget(self.massLineEdit)
        calibrationLayout.addWidget(self.timeEdit)
        calibrationLayout.addWidget(self.addDataButton)
        calibrationLayout.addWidget(self.deleteButton)
        calibrationLayout.addWidget(self.dataListWidget)
        calibrationLayout.addWidget(self.calibrateButton)
        calibrationLayout.addWidget(self.outputTextEdit)
        calibrationTab.setLayout(calibrationLayout)

        # Adding widgets to temperatureTab below here
        self.coefficients_file_path = None
        self.compensated_coeffs = None
        self.polynomial_degree = None

        self.coefficientsButton = QPushButton("Get Coefficients")
        self.coefficientsButton.clicked.connect(self.get_coefficients)
        self.coefficientsButton.setToolTip("Select coefficients file called 'calibration_results.txt'. All coefficients have to be
of the same polynomial degree as no checks have been implemented yet!.")

        self.coefficientsTreeWidget = QTreeWidget()
        self.coefficientsTreeWidget.setHeaderLabels(["Select", "Fit Accuracy", "Temperature"])

        self.compensationSpinBox = QSpinBox()
        self.compensationSpinBox.setRange(0, 99)
        self.compensationSpinBox.setValue(25)
        self.compensationSpinBox.setToolTip("Enter measurement medium temperature in degrees Celsius")

        self.compensateButton = QPushButton("Compensate Now")
        self.compensateButton.clicked.connect(self.compensate)
        self.compensateButton.setEnabled(False)
        self.compensateButton.setToolTip("Interpolate existing coefficients to compensate for medium temperature.
New Coefficients are displayed below and have to be validated before use.")

        self.coefficientsTextEdit = QPlainTextEdit()
        self.coefficientsTextEdit.setReadOnly(True)
        self.coefficientsTextEdit.setMaximumHeight(50)

        temperatureLayout = QVBoxLayout()
        temperatureLayout.addWidget(self.coefficientsButton)
        temperatureLayout.addWidget(self.coefficientsTreeWidget)
```

```
        temperatureLayout.addWidget(self.compensationSpinBox)
        temperatureLayout.addWidget(self.compensateButton)
        temperatureLayout.addWidget(self.coefficientsTextEdit)
        temperatureTab.setLayout(temperatureLayout)


        # Adding tabs to tabWidget
        tabWidget = QTabWidget()
        tabWidget.addTab(calibrationTab, "Calibration")
        tabWidget.addTab(temperatureTab, "Temperature Compensation")

        vBox = QVBoxLayout()
        vBox.addWidget(tabWidget)

        self.setLayout(vBox)
        print(f"Calibration Menu initialized")

# Methods for Calibration Tab

    def deleteSelected(self):
        current_item = self.dataListWidget.currentItem()
        if current_item is not None:
            row = self.dataListWidget.row(current_item)
            self.dataListWidget.takeItem(row)

    def start_calibration(self):
        try:
            temp = self.tempSpinBox.value()
            delay = self.delaySpinBox.value()
            degree = self.degreeSpinBox.value()

            options = QFileDialog.Options()
            options |= QFileDialog.ReadOnly
            self.txt_file_path, _ = QFileDialog.getOpenFileName(self,"Select txt file", "","Text Files (*.txt);;All Files (*)",
options=options)

            self.si.update_th_med(temp)
            self.si.update_degree(degree)
            self.si.update_delay(delay)
            self.si.import_txt(self.txt_file_path)
            self.calibrateButton.setEnabled(True)
        except Exception as e:
            QMessageBox.critical(self, "Error", str(e))

    def onDelayChanged(self, value):
        try:
            self.si.update_delay(value)
            self.si.import_txt(self.txt_file_path)
        except Exception as e:
            QMessageBox.critical(self, "Error", str(e))
    def add_entry(self):
        try:
            mass = float(self.massLineEdit.text())
            time = self.timeEdit.dateTime().toString('yyyy-MM-dd HH:mm:ss')
            self.calibration_data.append((mass, time))
            self.dataListWidget.addItem(f"Mass: {mass}, Time: {time}")
            self.massLineEdit.clear()
        except ValueError:
            QMessageBox.critical(self, "Error", "Mass must be a float number with decimal point.")
            self.massLineEdit.clear()
        except Exception as e:
            QMessageBox.critical(self, "Error", str(e))

    def calibrate(self):
        try:
            masses = [x[0] for x in self.calibration_data]
            times = [x[1] for x in self.calibration_data]
            cal_data = self.si.get_cal_data(masses, times)
            print("Calibration data from file: \n", cal_data) # better not remove this to make sure the data is correct each
time
            fit_data = self.si.fit_cal_data(cal_data)
            print("Calibration coefficients: \n", fit_data) # this part however can be removed if deemed unnecessary
            fit_data.index = [datetime.now().strftime('%Y-%m-%d %H:%M:%S')]
            self.outputTextEdit.setPlainText(str(fit_data))

            # # save the calibration results to a txt file called "calibration_results.txt"
```

```
            # check if file exists, if not create it and open in append mode
            with open("calibration_results.txt", "a") as f:
                # check if file is empty, if so write the header
                if os.stat("calibration_results.txt").st_size == 0:
                    f.write(fit_data.to_string(header=True))
                    f.write("\n")
                else:
                    f.write(fit_data.to_string(header=False))
                    f.write("\n")
                print("Calibration results saved to file.")

            # empty the calibration data list and the list widget
            #self.calibration_data = []
            #self.dataListWidget.clear()
            return fit_data
        except Exception as e:
            msg = QMessageBox()
            msg.setIcon(QMessageBox.Critical)
            msg.setText("Error while calibrating")
            msg.setInformativeText(str(e))
            msg.setWindowTitle("Error")
            msg.exec_()


# Methods for temperature compensation tab
    def get_coefficients(self):
        options = QFileDialog.Options()
        options |= QFileDialog.ReadOnly
        self.coefficients_file_path, _ = QFileDialog.getOpenFileName(self, "Select txt file", "", "Text Files (*.txt);;All Files
(*)", options=options)
        if self.coefficients_file_path:
            try:
                self.coefficients_data = pd.read_csv(self.coefficients_file_path, sep="\s+")
                self.polynomial_degree = max([int(col[2:]) for col in self.coefficients_data.columns if col.startswith('a_')])
                self.display_coefficients_data()
                self.compensateButton.setEnabled(True)
            except Exception as e:
                msg = QMessageBox()
                msg.setIcon(QMessageBox.Critical)
                msg.setText("Error getting coefficients")
                msg.setInformativeText(str(e))
                msg.setWindowTitle("Error")
                msg.exec_()

    def display_coefficients_data(self):
        try:
            self.coefficientsTreeWidget.clear()
            for i in range(self.coefficients_data.shape[0]):
                item = QTreeWidgetItem(self.coefficientsTreeWidget)
                checkbox = QCheckBox()
                self.coefficientsTreeWidget.setItemWidget(item, 0, checkbox)
                item.setText(1, str(self.coefficients_data.iloc[i]['fit_accuracy']))
                item.setText(2, str(self.coefficients_data.iloc[i]['cal_temp']))
        except Exception as e:
            msg = QMessageBox()
            msg.setIcon(QMessageBox.Critical)
            msg.setText("Error displaying coefficients data")
            msg.setInformativeText(str(e))
            msg.setWindowTitle("Error")
            msg.exec_()

    def compensate(self):
        try:
            checked_items = [self.coefficientsTreeWidget.indexOfTopLevelItem(item) for item in
self.coefficientsTreeWidget.findItems("", Qt.MatchContains | Qt.MatchRecursive, 0) if
self.coefficientsTreeWidget.itemWidget(item, 0).isChecked()]
            checked_data = self.coefficients_data.iloc[checked_items]
            checked_data = checked_data.reset_index(drop=True)
            print(checked_data)
            temp = self.compensationSpinBox.value()
            self.compensated_coeffs = self.si.get_temp_coeff(checked_data, temp)
            self.coefficientsTextEdit.setPlainText(str(self.compensated_coeffs))
            return self.compensated_coeffs
        except Exception as e:
            msg = QMessageBox()
            msg.setIcon(QMessageBox.Critical)
```

```
            msg.setText("Error compensating coefficients")
            msg.setInformativeText(str(e))
            msg.setWindowTitle("Error")
            msg.exec_()

    def closeEvent(self, event):
        pass
        #event.ignore()
        #self.hide()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    widget = SensorInterfaceGUI()
    widget.show()

    sys.exit(app.exec_())
```