Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Maher Achour

Entwicklung eines Lichtleiter-Sensors für die optische Ladezustandsbestimmung von Bleibatterien

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Maher Achour

Entwicklung eines Lichtleiter-Sensors für die optische Ladezustandsbestimmung von Bleibatterien

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. -Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr. -Ing. Jürgen Vollmer

Abgegeben am 5. Januar 2015

Maher Achour

Thema der Bachelorthesis

Entwicklung eines Lichtleiter-Sensors für die optische Ladezustandsbestimmung von Bleibatterien

Stichworte

Blei-Säure Batterie,Zellen, Lichtleiter, Ladezustand, Dichte, Brechungsindex, Elektrolyt, Transmissionsverluste, Sensorsonde, Mikrocontroller, optische Faser

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung und Optimierung eines optischen Sensors basierend auf Polymer-optischen Fasern. Anhand dieses Sensors wird die Dichte des Elektrolyten innerhalb einzelner Zellen einer Blei-Säure-Batterie gemessen und daraus der Ladezustand der Zellen ermittelt. Durch die Optimierung wird die mechanische Stabilität des Sensors verbessert, und der Einfluss von Störeffekten durch die Verwendung von Referenzmessungen, Kalibrierung und Fehlerkompensation minimiert.

Maher Achour

Title of the paper

Development of an optical fiber sensor for an optical state of charge determination from lead-acid batteries

Keywords

lead-acid-battery, cell, Polymeric optical fiber (POF), state of charge (SoC), density, electrolyt, refractive index, microcontroller

Abstract

This thesis deals with the development and optimization of an optical battery sensor based on polymeric optical fiber. This sensor measures the density of the electrolyte inside individual cells of a lead-acid-battery to determine the state of charge(SoC) of those cells. The mechanical stability of the sensor-construction is improved and the influence of interference effects are minimized through the use of reference measurements, calibration and error compensation.

Danksagung

An dieser Stelle möchte ich mich ganz besonders bei Herrn Prof. Dr. -Ing. Karl-Ragmar Riemschneider Projektleiter des Forschungsvorhabens BATSEN bedanken, der es mir im Rahmen dieses Forschungsprojektes die Möglichkeit gegeben hat, diese Arbeit zu schreiben und auch als Erstprüfer dieser Arbeit sich engagiert hat. Der Dank geht auch an den Zweitprüfer, Herrn Prof. Dr. -Ing. Jürgen Vollmer, der ebenso ein Projektleiter des BATSEN Projektes ist, und ein guter Adressat wenn es um fachliche fragen ging.

Ebenfalls gilt mein Dank Herr Dipl. -Ing. Valentin Roscher und Herr Dipl. -Ing. Günter Müller, die stets eine Hilfe waren, großes Engagement und vollen vielfältigen Einsatz im Laufe dieser Arbeit gezeigt haben.

Ein weiterer Dank geht an meinen Kommilitonen für die gute Zusammenarbeit und für die nutzbare Ratschläge, die mir während dieser Arbeit weitergeholfen haben.

Zum Schluss möchte ich mich insbesondere bei meinen Eltern und meiner Frau bedanken, die mir während meines Studiums eine große Unterstützung waren und viel Geduld gezeigt haben.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Die Bleibatterie, alte Erfolgsstory mit Fortsetzung

Die Erfindung der Bleiakkumulatoren geht auf das Jahr 1854 zurück. Basierend auf den von Wilhelm Ritter im Jahre 1802 durchgeführten Untersuchungen an galvanischen Elementen, ist es dem deutschen Mediziner und Physiker Wilhelm Josef Sinsteden gelungen, Blei als Elektrodenmaterial und Schwefelsäure als Elektrolyt miteinander zu verbinden. Daraus entstand die erste Blei-Säure-Batterie. Nur vier Jahre später baute der französische Forscher Planté den ersten brauchbaren Bleiakkumulator.

Durch die Forschungsarbeit zahlreicher Wissenschaftler wurde die Speicherfähigkeit der Bleibatterie im Laufe der Zeit stetig verbessert. Die von Fauré im Jahr 1881 entwickelte Pastierungsmethode[1], wird noch heute in der Herstellung der meisten aktuellen Gitterplattenzellen verwendet. Auch Die Planté-Technik oder die sogenannten Großoberflächen-Elektroden werden bis heute für Kraftwerksbatterien vereinzelt benutzt.

Neben den relativ neuen Batterietechnologien wie Nickel-Cadmium (NiCd), Nickel-Metallhydrid (NiMH) und Lithium-Ionen, spielt die Bleibatterie in dem Gebiet der wiederaufladbaren elektrischen Energiespeichern immer noch eine wichtige Rolle. Viele entscheidende Kriterien rechtfertigen diese Dominanz. Trotz der eher geringen Energiedichte, ermöglichen ihre Leistungsdichte, Lebensdauer bei geringer Last und insbesondere niedrige Herstellungskosten einen flexiblen Einsatz in vielen Anwendungsbereichen. Insbesondere werden Bleiakkumulatoren in der Automobilindustrie als Starterbatterien für Verbrennungsmotoren oder als Traktionsbatterie für den Elektroantrieb (Gabelstapler sowie günstige Elektrofahrzeuge), Notstromversorgung und Speicherung von Solar- und Windenergie oft eingesetzt.

[1] *die aktive Masse, die aus Bleiverbindungen besteht, wird separat gefertigt und später auf Bleigittern aufgetragen*

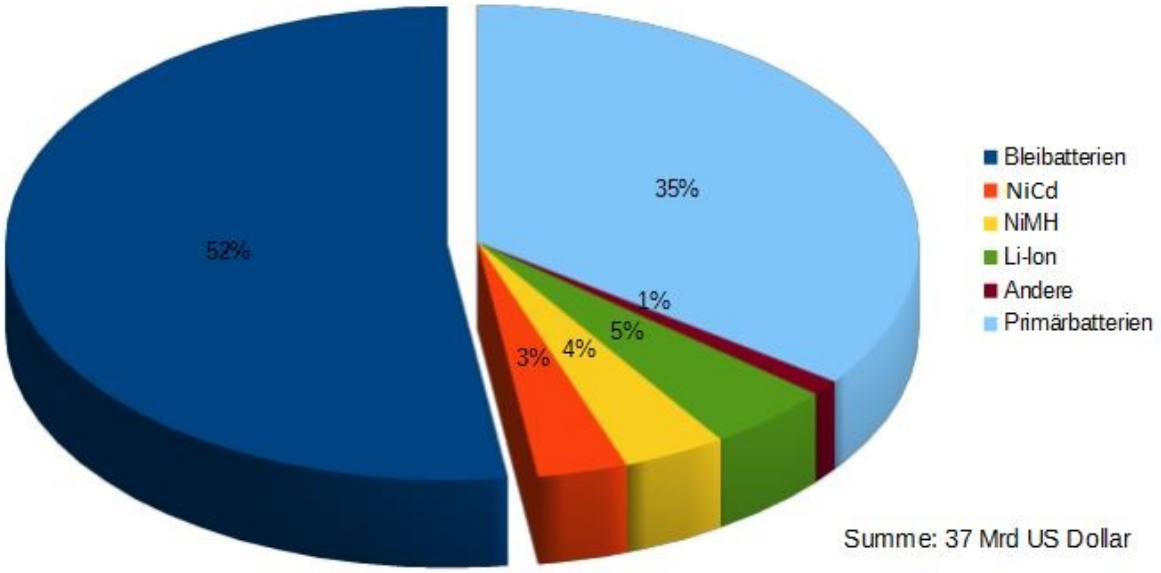


Abbildung 1.1.: Weltweite Marktaufteilung nach Batterietechnologien auf Kostenbasis für 2002 [1]

Die vorliegende Statistik spiegelt die führende wirtschaftliche Rolle der Bleibatterie gegenüber anderen herkömmlichen Akkumulatorentechnologien, die einfach keine ernst zunehmende Konkurrenz aufweisen können, aufgrund entweder deren hohen Herstellungskosten oder nicht genügend ausgereifter Technik bezüglich der Fertigung und Recyclingsprozesse. Primärbatterien, die ebenfalls einen bedeutenden Marktanteil haben, besitzen nicht die Wiederaufladbarkeit und eignen sich nur für den einmaligen Gebrauch. Damit sind sie als direkte Konkurrenz mit der Bleibatterie ausgeschlossen.

Eine weitere im Mai 2011 der ITRI[1] veröffentliche Studie zeigt eine positive Marktwachstumsphase ab dem Jahr 2009 und mit Ausblick auf das Jahr 2015 der Blei-Säure-Batterien, welche in der folgenden Abb. 1.2 dargestellt wird. Dieser Wachstum ist mit dem starken Aufschwung der Automobilbranche besonders im asiatischen Raum verbunden [2].

[1] *Industrial Technology Research Institute in Taiwan*

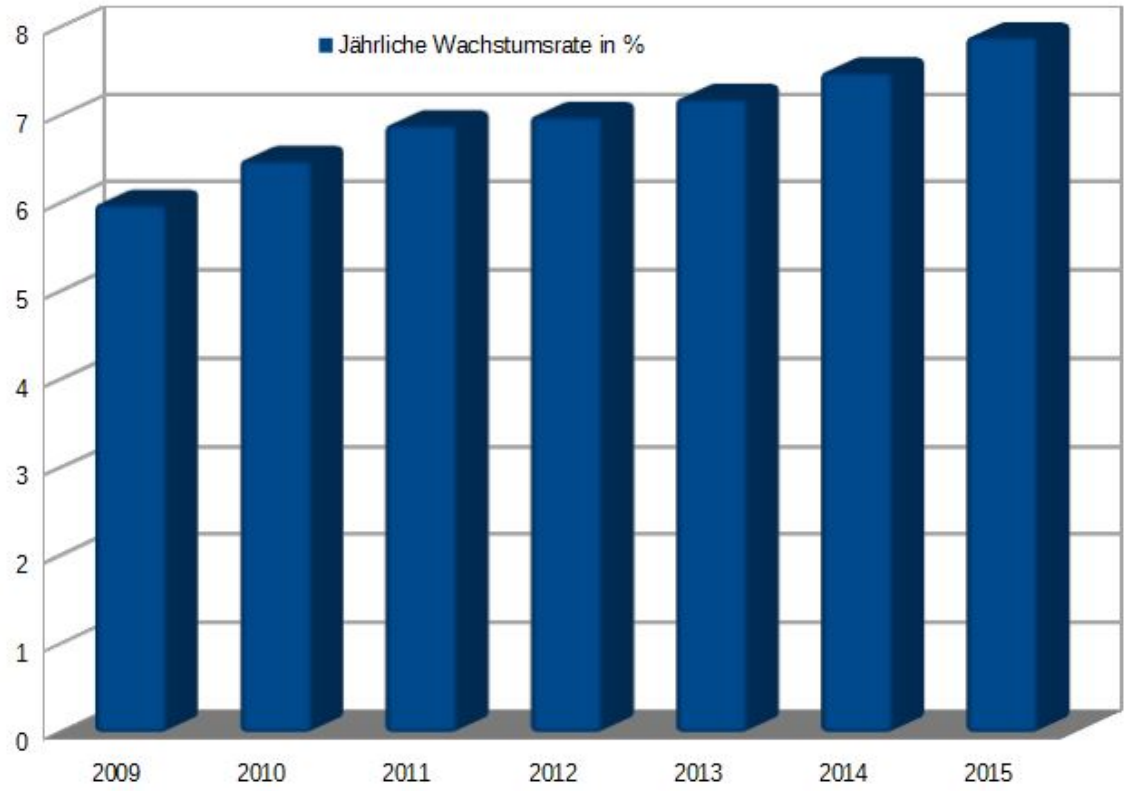


Abbildung 1.2.: Marktwachstumsrate der Bleibatterie (aus [2]).

Trotz den vielen neuen Erfindungen und innovativen Batterietechnologien, sind sich die meisten Experten einig, dass es zu einem bleifreien Welt der Energiespeicherung im Moment und auch in naher Zukunft nicht kommen wird. Doch es stellt sich die Frage, wie sich die Bleibatterie gegen diese effizienteren Technologien und harten Konkurrenz weiterhin durchsetzten kann, obwohl sie einige Schwachstellen aufweist.

1.2. Notwendigkeit von Überwachungs- und Batteriemangementsysteme für Bleibatterien

Trotz der intensiven Anwendung besitzt die Bleibatterie einige Defizite. Die niedrige spezifische Energiedichte von 30-40 Wh/kg und die beschränkte Lagerzeit sind nicht wirklich von großer Bedeutung im Gegensatz zu anderen entscheidenden Schwachstellen wie die be-

schränkte Lebensdauer und mangelnde Betriebszuverlässigkeit. Diese Eigenschaften machen sich hauptsächlich nach mehrfachen intensiven Zykliervorgänge[2] bemerkbar.

Besonders im Automotive-Bereich wird die Starterbatterie starken Anforderungen und hohen Belastungen ausgesetzt. Bei ständig zunehmender Bordelektronik und neuen Technologien, wie beispielsweise dem Start/Stopp-System, die heutzutage in den modernen Fahrzeugen fast zur Serienausstattung gehört, zusätzlich zum Startvorgang des Verbrennungsmotor, muss diese große Anzahl an elektrischen Verbrauchern dauernd, auch bei Motorstillstand, durch die Bleibatterie als alleinige Energiespeicherquelle im Fahrzeug mit Strom versorgt werden. Diese Aufgabe zu gewährleisten, wird mit zunehmender Alter der Batterie zu einer riesen Herausforderung.

Laut der Pannenstatistik des deutschen Automobilclubs ADAC im Jahr 2014, war die Starterbatterie mit 33% die Hauptpannenursache auf deutschen Straßen. Diese sind entweder auf eine tiefentladene[3] oder nicht funktionstüchtige Batterie zurückführbar [3]. Nach den Angaben des Automobilclubs, hat sich die Häufigkeit die durch die Batterie verursachten Pannen zwischen 1996 und 2010 um das Vierfache erhöht. Um die 10 Millionen Autobatterien werden in Deutschland jährlich aus dem Betrieb entnommen und durch neue ersetzt [4]. Die Mehrheit davon könnte durch das frühzeitige Erkennen des Alterungszustandes(SOH[4]) anhand Überwachungssysteme und mit dem Eingreifen der entsprechenden Wartungsmaßnahmen problemlos weiter benutzt werden.

Gerade bei den in modernen Fahrzeugen eingebauten sicherheitskritischen Systeme, wie ABS[5], ASR[6], ESP[7], muss eine kontinuierliche Stromversorgung gewährleistet werden und darf die Batterie an dieser Stellen nicht versagen [5].

Alternative Batterietechnologien sind aus technischen, und hauptsächlich aus wirtschaftlichen Gründen als Ersatz für die Bleibatterie beschränkt nicht einsetzbar. Wesentliche Sprünge in der Leistungsverbesserung und die Optimierung der Lebensdauer sind nicht zu erkennen. In diesem Zusammenhang ist das Interesse an Batteriemanagementsystemen enorm gestiegen. Ein Energiemanagementsystem hat grundlegend die Aufgabe, anhand der von der Batteriesensorik erfassten Daten über die gespeicherte Energie, die von dem Generator erzeugten Strom und die von der Fahrzeugelektrik beanspruchte elektrische Energie, durch gezielte Lade- und Entladephasen den Energiefluss zu optimieren und damit eine Balancierung zu erzielen.

---

[2] *Lade-Entladevorgänge*

[3] *Stromentnahme bei Unterschreiten der Entladeschlussspannung. Dieser Wert liegt bei herkömmlichen Bleibatterien bei 1,75 Volt pro Zelle*

[4] State Of Health

[5] *Antiblockiersystem, eng. Antilock Braking System, bei Fahrzeugbremsen*

[6] *Antriebsschlupfregelung, auch Traktionskontrolle*

[7] *Electronic Stability Programm, verhindert durch gezieltes Abbremsen einzelner Räder dem Ausbrechen des Fahrzeuges*

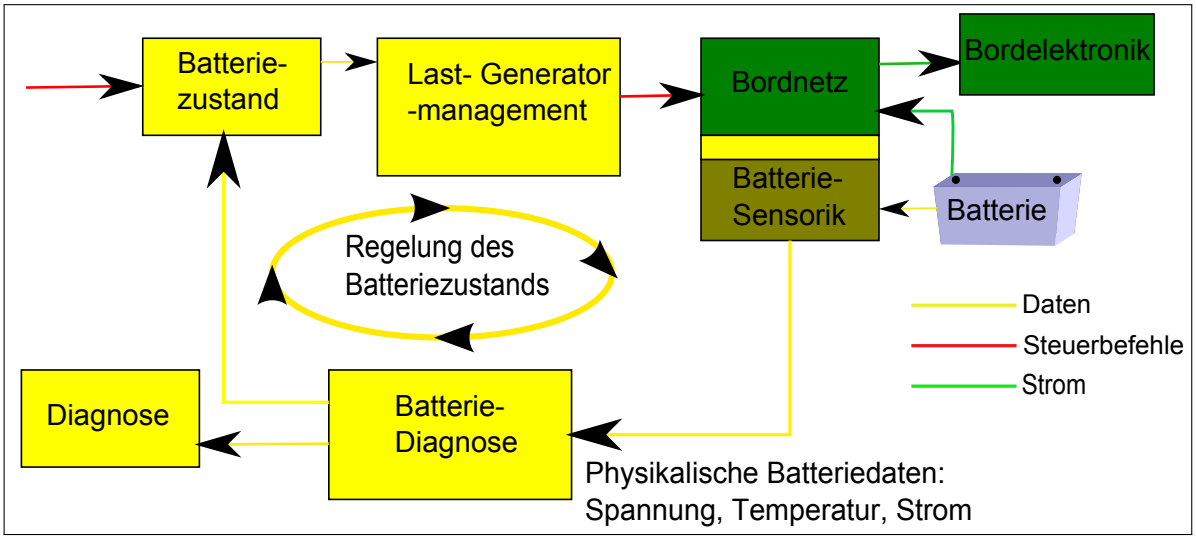


Abbildung 1.3.: Energiemanagementsystem in Kraftfahrzeugen, ein geschlossener Regelkreis [6]

In der Abb. 1.3 ist es deutlich zu erkennen, wie bedeutsam die Batteriesensorik in Energiemanagementsystemen ist. Es sind genaue Angaben über den Batteriezustand zu ermitteln, damit das System entsprechend reagieren kann, um die erwünschte Energiebilanz zu erreichen.

1.3. Ein effizientes Energiemanagementsystem verlangt präzise Batteriesensorik

Mit der gängigen Batteriesensorik erfasst man die physikalische Größen wie Spannung, Strom und Temperatur. Die gewonnenen Messdaten werden vom Steuergerät weiterverarbeitet und daraus werden Informationen über wichtige technische Merkmale der Batterie unter anderem der Ladezustand (SOC[8]) , Alterungszustand (SOH) und den Funktionszustand (SOF[9]) ermittelt.

Andererseits, wird bei der Mehrheit dieser Messverfahren die Bleibatterie wie eine Blackbox behandelt. Die Messungen der oben genannten Größen finden an den Polklemmen der Batterie statt. Anhand solcher Verfahren können keine präzise Erkenntnisse über den Zustand einzelner Zellen abgeleitet werden. Daran kann eine frühzeitige Erkennung von Fehlfunktionen, trotz der Überwachung, scheitern.

[8] *engl. State Of Charge*
[9] *engl. State of Function*

Da die Bleibatterie aus mehreren in Reihe geschalteten Zellen besteht, können diese aufgrund von Fabrikationstoleranzen und abweichenden Betriebsbedingungen, Anomalien in deren Verhalten und Zuständen aufweisen. Eine durch zum Beispiel Tiefentladung geschädigte Zelle, kann nach mehrfachem Zyklieren die Lebensdauer der gesamten Batterie negativ beeinträchtigen. Dementsprechend ist es sinnvoller, einzelne Zellen getrennt zu untersuchen und zu beobachten, um daraus genauere Informationen zu erhalten [5].

Insbesondere, eignet sich die zellenweise Überwachung sehr für Batteriemanagementsysteme, die eine Ladungsbalancierung durchführen. Die für den Balancierungsprozess notwendigen Kalibrierparameter, müssen je nach Ladezustand und Alterungszustand neu angepasst werden.

Batteriesensorik innerhalb des Projekts BATSEN

Im Rahmen des BATSEN Forschungsvorhabens (Drahtlose Zellensensoren für Fahrzeugbatterien) werden an der Hochschule für angewandte Wissenschaften Hamburg Sensoriksysteme entwickelt und untersucht, die eine zellenweise Überwachung von Fahrzeugbatterien (Starterbatterie, Traktionsbatterie[10]) ermöglichen. Es wurden innerhalb dieses Projektes drei Sensorklassen entwickelt, die stetig auf ihre Integration in verschiedenen Batterietechnologien (Bleibatterie, Li-Ion) geprüft und optimiert werden. Diese unterscheiden sich im Wesentlichen in der Übertragungsart, in der Interaktion mit der Basisstation (Batteriesteuergerät) und in der geometrischen Form bedingt je nach Einsatzgebiet.

Für die Überwachung an Bleibatteriezellen werden Sensoren der Klasse eins eingesetzt. Die angepasste Platinenform dieser Sensorklasse ermöglicht eine einfache Integration in den Zellen. Mit Hilfe dieser Sensoren werden Temperatur und Spannung erfasst. Da diese über kein Empfangsmodul verfügen, kann nur eine Datenübertragung im Uplink-Modus[11] gestattet werden. Aufgrund fehlender Synchronisation ist der Wechsel in den Sendebetrieb eine reine autonome Entscheidung des Sensors. Das Gefahr einer Datenkollision wird durch pseudozufällige Übertragungszeitpunkte und eine niedrige Senderate minimiert. Dabei muss die Senderate je nach auftretender Kollisionswahrscheinlichkeit eingestellt werden [7].

[10] *auch Antriebsbatterie oder Zyklenbatterien genannt, dient als Energiespeicher für Fahrzeuge mit reinem Elektroantrieb, häufig bei Elektro-Gabelstaplern eingesetzt.*

[11] *Die Kommunikation zwischen Sensor und Steuergerät ist nur in einer Richtung möglich, Die Sensoren können Daten an die Basisstation senden, jedoch keine Befehle empfangen.*

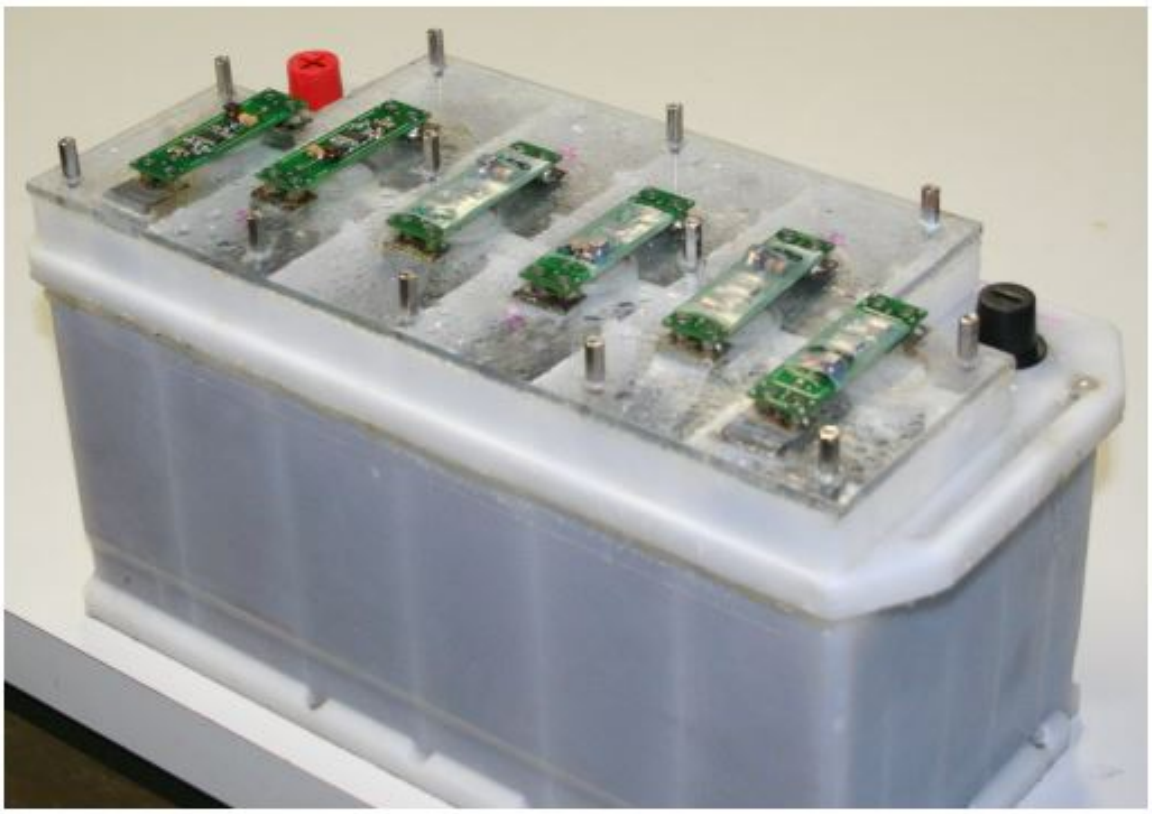


Abbildung 1.4.: Einsatz von Zellensensoren der Klasse eins in eine Starterbatterie. Die sechs Zellen werden getrennt überwacht. Zellspannung und Temperatur werden drahtlos an das Steuergerät übertragen und weiterverarbeitet [5]. Die Sensoren sind teilweise zum Schutz der Elektronik in Schrumpfschlauch gesichert.

Integration von Lichtwellenleiter-Sensorik in Blei-Säure-Batterien

Die eingesetzte Schwefelsäure als Elektrolyt bei Bleibatteriezellen nimmt aktiv an der chemischen Reaktion während des Lade-/Entladevorgangs teil. Je nach Ladezustand, ändert sich die Säurekonzentration (Säuredichte) und proportional dazu der Brechungsindex des Elektrolyten. Diese Eigenschaft bietet einen weiteren Anhaltspunkt, um noch genauere Rückschlüsse über den Ladezustand der Zellen zu ziehen.

Bisher wurden verschiedene analoge Messinstrumente wie das Aräometer[12] und das Refraktometer[13] benutzt, die eine optische Untersuchung oder Überwachung dieser chemischen Eingenschaft des Elektrolyten ermöglichen. Allerdings fehlt es bei diesen Messmethoden

[12] *Auch als Säureheber bekannt. Es ermittelt anhand eines schwimmenden Körpers relativ zum spezifischen Gewicht die Säurekonzentration des Elektrolyten.*

[13] *Ein optisches Messsystem, das durch den Brechungsindex die Säuredichte bestimmt.*

an Genauigkeit. Außerdem erweist sich die Anwendung solcher Systeme, besonders bei im Allgemeinen verschlossenen Starterbatterien, als sehr aufwendig.

Im Jahr 2010 wurde innerhalb eines Forschungsprojektes des Electronic Technology Department an der spanischen Universität Vigo ein Sensorsystem basierend auf Lichtwellenleitern für die optische Erfassung der Säuredichte innerhalb von Blei-Säure-Batterien entwickelt. Durch optische Fasern wird Licht geschickt. Die Fasern werden in verschiedenen Tiefen im Elektrolyten der Bleibatteriezelle eingetaucht. Das Ausbrechen des Lichtes aus den Fasern gilt als Transmissionsverlust, welcher stark von dem Brechungsindex des Ausbreitungsmediums abhängig ist. Der Brechungsindex wiederum ist linear abhängig von der Säurekonzentration. Auf diese Weise können Veränderungen von Transmissionsverluste als Änderung des Ladezustandes interpretiert werden [8].

Aufbauend auf den Informationen und Erkenntnisse aus dem Projekt der spanischen Universität, wurde innerhalb des BATSEN-Projektes im Rahmen einer Bachelorarbeit von Wahid Nasimzada [9] ein Sensorsystem mit einem ähnlichen Funktionsprinzip entwickelt. Ein aufsteckbares Lichtsensor-Modul wurde entwickelt und zusammen mit dem Zellensensor der Klasse eins, der ebenfalls für diesen Zweck erweitert wurde, betrieben. Für die Sensorsonde wurde POF[14] wegen der Biegbarkeit und der leichteren Verarbeitung im Gegensatz zur Glasfaser verwendet. Mittels dieser Sensorik wurde die Änderung der Säuredichte bei Bleiakkumulatorenzellen in Abhängigkeit des Ladezustandes festgestellt.

Die nach [9] entwickelte Sensorik weist einige Schwachstellen auf, die während dieser Arbeit untersucht und optimiert werden:

- Fremdlichtempfindlichkeit: während der Auswertung der Messdaten wurde der Einfluss der Umgebungslicht auf die optischen Messdaten beobachtet und bestätigt. Aus diesem Grund mussten die darauf folgenden Messungen unter Abdunklung des Messaufbaus durchgeführt [9] werden.
- Mechanische Stabilität der Sensorsonden: Da die Sensorsonden handgefertigt waren, können besonders an den Biegestellen der Lichtwellenleiter große Streuungen auftreten, was im schlimmsten Fall zur Unterschreitung des minimalen Biegeradius führen kann. Die mechanische Stabilität (im Betrieb in der Batterie) der Sonden muss ebenso optimiert werden.
- Anhand der Konstruktion der Sensorsonde können die optischen Leistungsverluste nur an Zwei Punkten gemessen werden. Da sich die Säuredichte in der vertikalen Richtung der Zellentiefe ändert, ist die Messung genauer, wenn man mehrere Messpunkte einbringt. So lassen sich integrative Messungen erzielen.

[14] *eng. Polymer Optical Fiber, Lichtwellenleiter aus Kunststoff*

- Temperaturabhängigkeit: Der Temperatureinfluss wurde für die optische-, und die Spannungsmessung vernachlässigt.
- Messstrategie: Die Messzyklen wurden manuell durchgefahren und betrieben. Hierbei wurden keine zeitlich äquidistanten Lade-/Entladephasen erzielt, so dass genauere zeitliche Beobachtungen der optischen Messungen gegenüber anderen Größen nicht möglich waren.

1.4. Motivation

Gegenstand dieser Arbeit ist die Entwicklung eines Lichtleiter-Sensors auf POF Basis für die optische Ladezustandsbestimmung von Bleibatteriezellen. Dabei soll die nach [9] entstandene Sensorik für die Dichtemessung optimiert und erweitert werden.

Zuerst wird eine neue Sensorsonde konzipiert und entwickelt. Die mechanische Konstruktion dieser neuen LWL-Sonde soll minimale Toleranzen und Fertigungsstreuung, dazu eine ausreichende Stabilität im Betriebszustand aufweisen. Des Weiteren soll anhand der neu entwickelten Sonde eine räumlich integrative Messung der Dichte des Elektrolyten in verschiedenen Höhenschichten gewährleistet werden.

Darüber hinaus sollen durch Temperatureinfluss und Fremdlicht verursachte Störeffekte und Messstreuung durch Kalibrierverfahren, Referenzmessungen und Fehlerkompensation beseitigt werden.

Weiterhin sollen Funktionserprobungen auf der Starterbatterie im Zyklierbetrieb durchgeführt werden. Eine sinnvolle Messstrategie soll das Zeitverhalten der optischen Messwerte gegenüber der anderen Messgrößen (Spannung, Strom, Temperatur) verdeutlichen.

Abschließend sollen die Messreihen ausgewertet und analysiert werden.

2. Grundlagen

2.1. Die Blei-Säure-Batterie

Um präzise Aussagen über Ladezustandsänderungen und Alterungszustände treffen zu können, sind fundierte Kenntnisse über das Funktionsprinzip, Merkmale und Effekte bei Blei-Säure-Batterien erforderlich. In diesem Unterkapitel, werden grundlegende und relevante Charakteristika von Bleibatterien dargestellt und untersucht.

2.1.1. Aufbau und Einordnung von Bleibatterien

Die Bleibatterie gehört zu den Sekundärbatteriesystemen[1] und ist ein elektrochemischer Energiespeicher. An einem elektrischen Verbraucher angeschlossen, dient sie als Gleichspannungsquelle. Die herkömmlichen Bleiakkumulatoren bestehen überwiegend aus mehrere in Reihe oder parallel geschalteten galvanischen[2] Zellen. Die Nennspannung einer einzelnen Zelle beträgt 2 Volt.

In der folgenden Abb. 2.1 wird der Aufbau einer klassischen Bleibatteriezelle dargestellt. Sie besteht grundsätzlich aus einer negativen und einer positiven Elektrodenplatte mit jeweils fein metallischem Blei (Pb) und Bleidioxid (PbO_2) als Aktivmaterial. Dazwischen ist ein Separator aus nicht leitendem mikroporösen Material, was einen Kurzschluss zwischen den beiden Elektroden verhindern soll. Als Elektrolyt dient ein Gemisch aus Schwefelsäure und Wasser. Das ganze System wird in einem aus säurebeständigen Metall- oder Plastikgehäuse eingebracht.

[1] *wiederaufladbare Akkumulatoren, die durch fast reversible elektrochemische Prozesse mehrfach entladen und geladen werden können, dagegen bezeichnet man Batterien, die diese Eigenschaft nicht besitzen als Primärbatterien.*

[2] *Ein galvanisches System besteht aus zwei Elektroden (Anode und Kathode) in einen Elektrolyten eingetaucht und wandelt chemische in elektrische Energie*

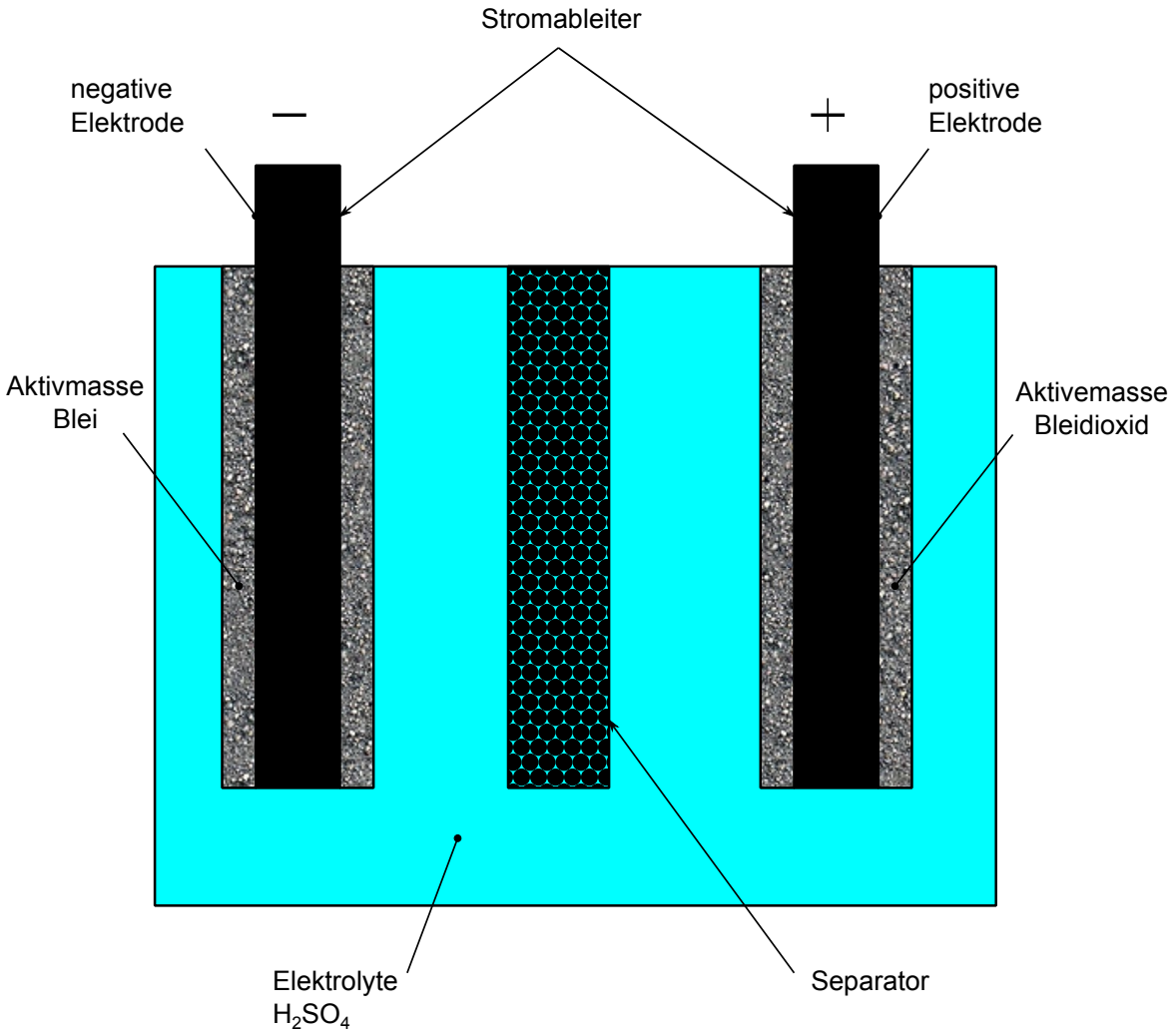


Abbildung 2.1.: Prinzipieller Aufbau einer Bleibatteriezelle (geschlossene Bauweise)[1]

Je nach Einsatzgebiet und Anforderung existieren unterschiedliche Bauformen, Größen und mehrere Ausführungen mit verschiedenen Nennkapazitäten[3]. Nach der Bauart der positiven Platte werden die Bleiakkumulatoren klassifiziert [10]:

- **Großoberflächenplatten-Zellen (Normbezeichnung Gro):**
  Die negative Elektrode besteht größtenteils aus Kastenplatten[4]. Sie benötigen gegenüber anderen Batteriebauarten ein höheres Volumen an Elektrolyt. Die Säuredichte

[3] *bezeichnet die Ladungsmenge, welche einer voll geladenen Batterie bei Nennbetriebsbedingungen entzogen werden kann und wird in Ah (Ampere Stunden) angegeben*

[4] *Der Masseträger besteht aus 2 Hartbleigitter. Diese werden einseitig durch gelochte Bleibleche abgedeckt und nach dem Füllen der negativen Masse zusammengenietet werden.*

beläuft sich auf $1{,}20\,\mathrm{g/cm^3}$ im geladenen Zustand. Sie haben eine hohe Zyklierbarkeit (bis zu 1000 Entladungen) und eine lange Lebensdauer bis zu 10 Jahren im Pufferbetrieb[5] und bis 15 Jahre im Bereitschaftsbetrieb[6]. Wegen des hohen Gewichts und der ausgedehnten Bauform werden Batterien dieser Bauweise überwiegend als ortsfeste Batterien eingesetzt.

- **Panzerplatten/Röhrchenplatten-Zellen (Normbezeichnung PzS, OPzS):**
  Als negative Elektrode, werden negative Gitterplatten verwendet. Im geladenen Zustand entspricht der Säuredichtewert bei manchen $1{,}24\,\mathrm{g/cm^3}$ und bei anderen $1{,}27\,\mathrm{g/cm^3}$. Bei lückenloser Wartung erreichen sie bis 1500 Zyklen und haben eine Lebensdauer von 8 bis 10 Jahren bei einer einmaligen täglichen Zyklierung. Als ortsfeste Batterie (OPzS) kommt sie häufiger zum Einsatz. Die Verwendung als Traktionsbatterie (Antriebsbatterie) für reine Elektrofahrzeuge, besonders in Gabelstaplern wird langsam durch andere Batterietechnologien (z. B. Gel-Batterie) ersetzt.

- **Gitterplatten-Zellen (Normbezeichnung Gi, GiS):**
  Beide Elektroden sind Gitterplatten. Gi- und GiS-Zellen unterscheiden sich hauptsächlich in der Art der Plattentrennung. Mittels einem an der positiven Platte dichtangrenzenden Scheider[7] (auch Taschen-Separator genannt), wird dessen Masseausfall abgefangen und damit eine erhebliche Verbesserung der Lebensdauer der Zelle erzielt. Jedoch haben Gitterplatten-Zellen gegenüber Panzerplatten- und Großoberflächen-Zellen eine niedrigere Lebensdauer und begrenzte Zyklierbarkeit. Die Säuredichte beträgt bei $20\,^\circ\mathrm{C}$ im geladenen Zustand $1{,}28\,\mathrm{g/cm^3} \pm 0{,}01$.

  Eine besonders verbreitete sonder Ausführung der Gi/GiS-Zellen sind Starterbatterien für Kraftzeuge und Krafträder. Durch das Herabsetzen der Plattendicke, der Plattenabstände und der Optimierung der Plattentrennungsart anhand mehrfachen mikroporösen Separatoren wird der Innenwiderstand der Zellen verringert. Diese Eigenschaft ist besonders effizient, um die beim Startvorgang benötigten relativ hohen Ströme von mehreren 100 Ampere bei minimalem Spannungsabfall innerhalb weniger Sekunden zu gewährleisten. Darüber hinaus, muss die Startfähigkeit nach dem DIN 72 311 bei einer Elektrolyttemperatur von $-18\,^\circ\mathrm{C}$ (Kaltstartfähigkeit) auch gewährleistet werden.

[5] *auch Dauerladung genannt. Die Batterie wird nur belastet bei Verbraucherspitzen und Ausfall der Gleichstromquelle, andernfalls wird sie dauerhaft von der Gleichstromquelle aufgeladen.*

[6] *auch als Bereitschaftsparallelbetrieb bezeichnet. Parallel mit dem Verbraucher und Stromquelle geschaltet, dient die Batterie ständig als Notstromreserve.*

[7] *Separator, Beispiel: Glaswolle*

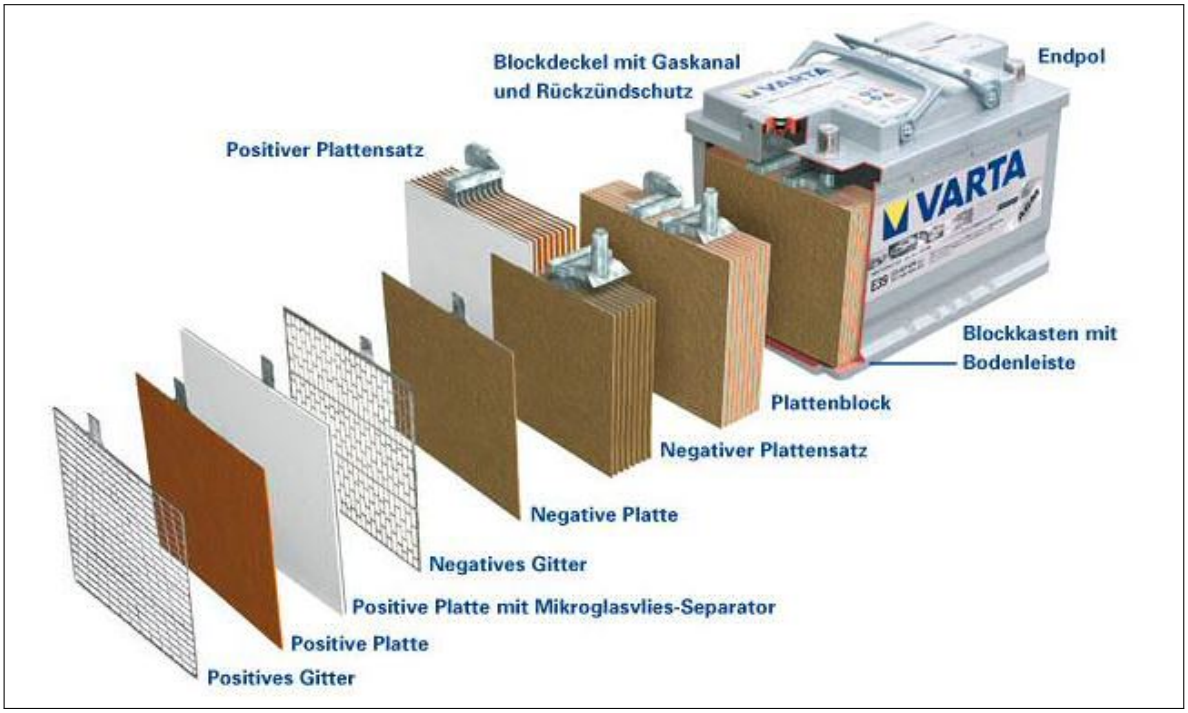


Abbildung 2.2.: Aufbau einer Blei-Starterbatterie mit AGM-Separator-Technologie [11]

Anhand der Abb. 2.2, ist der gitterförmige Aufbau der Elektrodenplatten deutlich zu erkennen. Positive und negative Elektroden werden als Plattensätze ineinander verschachtelt verbaut. Zusammen bilden sie einen Plattenblock (Zelle). Als Separator wird Mikroglasvlies verwendet (AGM-Separator[8]). Durch den Einsatz dieser Art der Plattentrennung, wird dem Abbröckeln des Aktiv-Materials an der positiven Platte, verursacht durch häufige Zyklierungen, vorgebeugt. Zusätzlich wird durch eine angemessene Porosität (mikroporöse Separatoren haben in der Regel eine gewellte Form und Porengröße von 0,2-1,0 μm [1]) des Separators eine bessere Ionenleitfähigkeit gewährleistet und gleichzeitig eine Dendritenbildung[9] verhindert.

2.1.2. Der Elektrolyt

Ist eine Ionen leitfähige Flüssigkeit (kann auch in fester Form verwendet werden z. B. in Gel-Bleibatterien). Im Einsatz in Naß-Bleibatterien, besteht sie aus verdünnter Schwefelsäure H_2SO_4 mit einer Gewichtskonzentration von $36{,}9\,\%$ und einer Säuredichte von $1{,}28\,\mathrm{g/cm^3}$

[8] *eng. Absorbent Glass Mat*

[9] *bezeichnet bei Tiefentladung das Wachstum eines Bleifadens durch große Poren des Separators, was zu einem Kurzschluss einer Zelle führen kann [12]*

bei einer voll geladenen Batterie. Diese Werte entsprechen dem Bereich der maximalen Leitfähigkeit der Säure (siehe Tab. 2.1). Bei einer vollständigen Entladung erreicht die Säuredichte einen Wert von $1{,}1\,\mathrm{g/cm^3}$. Beachtenswert ist, dass diese Eigenschaft nur bei einer Nenntemperatur von $20\,^{\circ}\mathrm{C}$ gilt. Die Schwefelsäure ist stark temperaturabhängig. Diese Abhängigkeit ist mit der Dichte verbunden bzw. dem Ladezustand der Batterie. Insbesondere sind die Gefriertemperaturen des Elektrolyten von großer Relevanz. In der Tabelle. 2.1 und in der Abb. 2.3 und 2.4 wird dieser Zusammenhang dargestellt. Da die Schwefelsäure am chemischen Prozess während des Lade-/Entladevorganges aktiv teilnimmt, ändert sich ihre Dichte (bzw. Brechungsindex) in Abhängigkeit des Ladezustandes. Des Weiteren spielt die Säuredichte eine Schlüsselrolle für das gesamte Verhalten der Bleibatterie, besonders für den Zustand der Ruhespannung. Diese Besonderheit wird in dieser Arbeit als Anhaltspunkt genutzt, um mittels optischer Untersuchungen der Dichteänderung vom Elektrolyten genauere Aussagen über den Ladezustand der Batteriezellen zu treffen.

Ladezustand [%]	Säuredichte [g/cm^3]	Batterie-Klemmspannung [V]	Gefriertemperatur [°C]
0	1,05	11,8	-7,7
25	1,12	11,9	-10,8
50	1,16	12,1	-17,9
75	1,21	12,36	-31,7
100	1,26	12,6	-56,5

Tabelle 2.1.: Abhängigkeit der Säuredichte vom Ladezustand und der Gefriertemperatur [13]

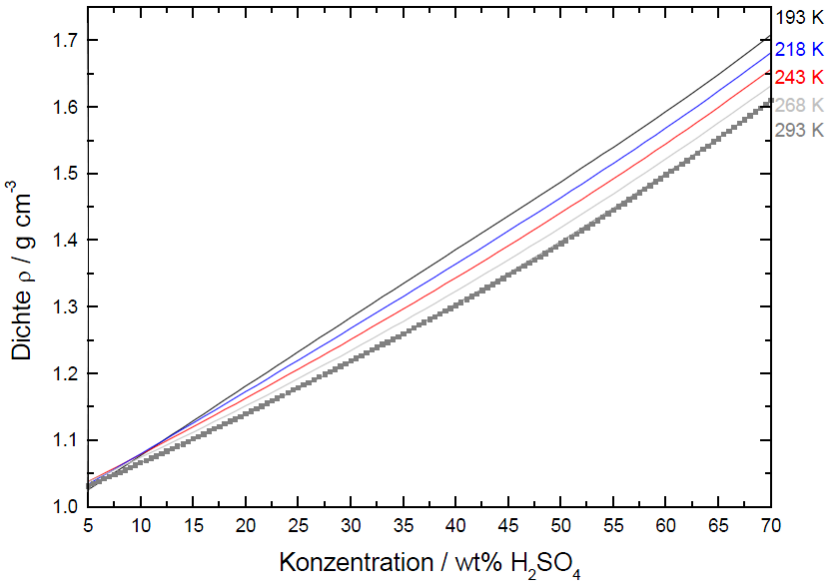


Abbildung 2.3.: Temperaturabhängigkeit der Säuredichte bei verschiedenen Konzentrationen (bei einer Wellenlänge $\lambda = 514{,}5\,\mathrm{nm}$) [14]

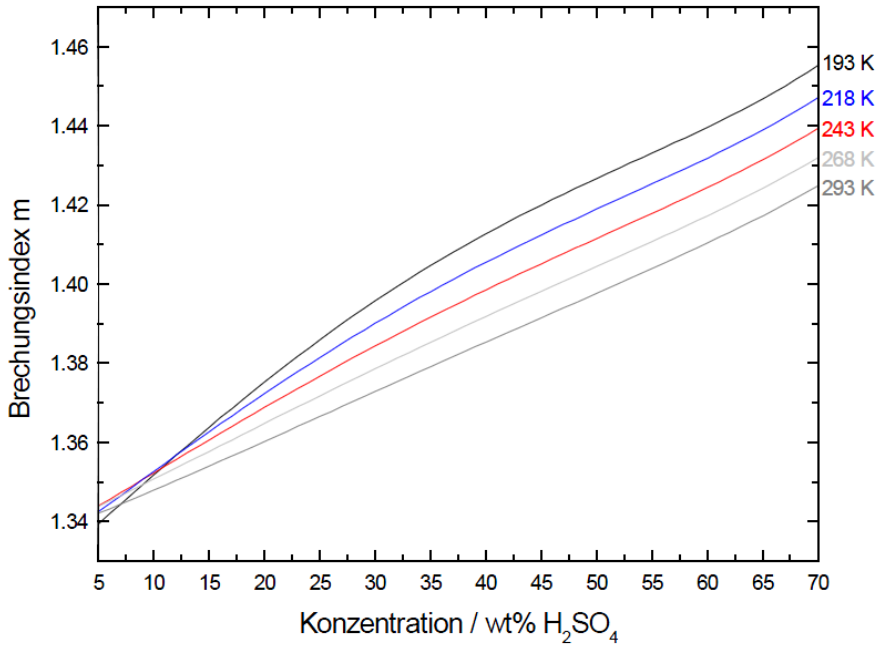


Abbildung 2.4.: Brechungsindex der Schwefelsäure in Abhängigkeit von der Temperatur und Konzentration (bei einer Wellenlänge $\lambda = 514{,}5\,\mathrm{nm}$) [14]

Die Abb. 2.3 zeigt, dass die Dichte mit zunehmender Konzentration fast linear steigt. Bei größeren Säurekonzentrationen sinkt die Dichte der Schwefelsäure beim Erhöhen der Temperatur. In verdünnter Form (Konzentration $< 15\,\%$) und in sehr tiefen Temperaturbereichen sinkt dagegen die Dichte mit abnehmender Temperatur. Ein ähnliches Verhalten der Brechungsindizes lässt sich auch anhand der Abb. 2.4 beobachten.

2.1.3. Chemische Reaktionen

Die chemischen Prozesse in einer Blei-Säure-Batterie unterteilen sich in Hauptreaktionen und Nebenreaktionen. In der Hauptreaktion finden zwei chemische Vorgänge statt. Beim Laden wird elektrisch zugeführte Energie in chemische Energie umgewandelt. Beim Entladen wird die gespeicherte chemische Energie in elektrische Energie umgesetzt. Nebenreaktion sind nicht erwünscht, jedoch unvermeidbar.

2.1.3.1. Hauptreaktionen

- **Chemische Reaktionen bei der Entladung:**
  Bei der Stromentnahme fließen Elektronen von der negativen Elektrode (Blei als Aktivmasse) zur positiven Elektrode (Bleidioxide als Aktivmasse). Mit der Beteiligung der Schwefelsäure entsteht durch Oxidation[10] am Minuspol Bleisulfat, gleichzeitig entsteht durch Reduktion[11] am Pluspol Bleisulfat. Die Reaktionsgleichungen sind wie folgt vereinfacht dargestellt:

$$\text{positive Elektrode} : \mathrm{Pb} + \mathrm{SO_4^{2-}} \rightarrow \mathrm{PbSO_4} + 2\mathrm{e^-} \tag{2.1}$$

$$\text{negative Elektrode} : \mathrm{PbO_2} + \mathrm{H_2SO_4} + 2\mathrm{H^+} + 2\mathrm{e^-} \rightarrow \mathrm{PbSO_4} + 2\mathrm{H_2O} \tag{2.2}$$

$$\text{Gesamtreaktion} : \mathrm{Pb} + \mathrm{PbO_2} + 2\mathrm{H_2SO_4} \rightarrow 2\mathrm{PbSO_4} + 2\mathrm{H_2O} \tag{2.3}$$

  Bei der Entladung, werden Sulfationen der Säure verbraucht und Wasserstoff gebildet (2.3). Durch diese Verdünnung sinkt die Säuredichte innerhalb der Batteriezelle. Mit steigender Entladung nimmt der ohmische Widerstand der Zelle zu. Dieses Verhalten

[10] *Eine chemische Reaktion, bei der ein Stoff (Atom, Ion oder Molekül) Elektronen abgibt [15]*

[11] *Eine chemische Reaktion, bei der ein Stoff (Atom, Ion oder Molekül) Elektronen aufnimmt. Zusammen mit der Oxidation bilden sie eine Redoxreaktion [15]*

ist durch die schlechte elektrische Leitfähigkeit des Bleisulfats begründet. Der Stromfluß von der positiven zur negativen Elektrode wird durch die beiden H^+-Ionen der aufgespaltenen Schwefelsäure Moleküle gewährleistet [16][10].

- **Chemische Reaktionen beim Laden:**
  Der Ladevorgang ist ein erzwungener Prozess, bei dem man Spannung an die Elektroden anlegt und somit die Stromrichtung und die bei der Entladung stattgefundenen elektrochemischen Vorgänge umkehrt. Beim Laden wird das Bleisulfat an der negativen Elektrode zum ursprünglichen metallischen Blei, respektive Bleidioxide an der positiven Elektrode zurückgewandelt. Folgende Gleichungen stellen den Umkehrprozess dar:

$$\text{positive Elektrode} : PbSO_4 + 2\,H_2O \rightarrow PbO_2 + H_2SO_4 + 2H^+ + 2e^- \tag{2.4}$$

$$\text{negative Elektrode} : PbSO_4 + 2e^- \rightarrow SO_4^{2-} + Pb \tag{2.5}$$

$$\text{Gesamtreaktion} : 2PbSO_4 + 2H_2O \rightarrow Pb + PbO_2 + 2H_2SO_4 \tag{2.6}$$

  Hierbei wird Wasserstoff verbraucht und Säure gebildet (2.6). Dies führt zu einem Anstieg der Säuredichte, der Strom fließt in umgekehrter Richtung als bei der Entladung [16][10].

- **Gesamte chemische Reaktion:**
  Die gesamte chemische Reaktion besteht aus der beiden Teil-Reaktionen, die beim Laden und Entladen stattfinden und kann zur einer reversiblen Reaktion zusammengefasst werden. Nach der klassischen Sulfattheorie von Gladstone und Tribe und anhand der Gleichung (2.7), wird der Umsetzungsprozeß sowohl qualitativ als auch quantitativ wiedergegeben [10].

$$Pb + 2\,H_2SO_4 + PbO_2 \overset{\overset{Entladen}{\rightleftharpoons}}{\underset{Laden}{}} 2\,PbSO_4 + 2\,H_2O \tag{2.7}$$

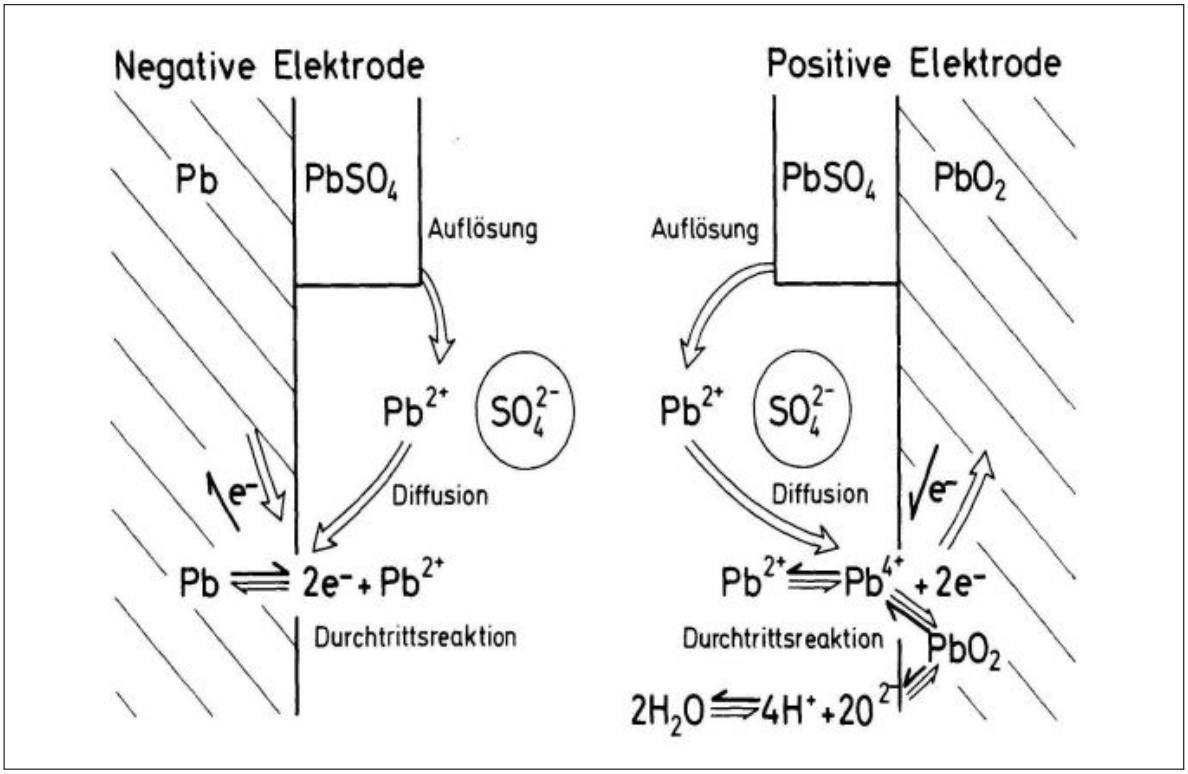


Abbildung 2.5.: Schematischer Ablauf der Ladereaktion bei einer Bleibatterie [17]

In der Abb. 2.5 werden die auftretenden chemischen Reaktionen innerhalb einer Bleibatteriezelle schematisch dargestellt. Die freigelösten Pb^{2+}-Ionen diffundieren an die Reaktionszonen und werden in der Durchtrittsreaktion zu Pb oder zu Pb^{4+} reduziert bzw. oxidiert. Die Diffusion[12] der Pb^{2+} bestimmt die Geschwindigkeit der Ladereaktion. Beachtenswert ist, wenn hohe Ladezustände erreicht werden, die Konzentration der Pb^{2+}-Ionen sinkt und durch nachgelagerten Durchtrittsreaktionen[13] fast vollständig verbraucht werden. Dieses Verhalten ist durch einen Diffusionsgrenzstrom $I_{Diff-grenz}$ bezeichnet. Bei Erreichen dieses Grenzstroms kann trotz Erhöhung der Ladespannung, eine zurückwandlung des Bleisulfats in aktives Material nicht mehr erfolgen.

Analog zur Ladereaktion, besteht die Entaldereaktion aus Durchtrittsreaktion und Diffusion. Beim Entladen spielt jedoch die Durchtrittsreaktion die Schlüsselrolle und ist damit der Taktgeber des chemischen Umsetzungsprozeß. Durch die Verbindung von

[12] *In einem abgeschlossenen System bewirkt Diffusion den Abbau von Konzentrationsunterschieden bis hin zur vollständigen Durchmischung [18].*

[13] *Der Elektrodenübergang zwischen Elektrodenmaterial und darauf absorbierten Stoffen, bestimmt das Elektrodenpotential bei kleinen Strömen. Reaktionen können vor oder nachgelagert sein [19].*

Pb^{2+}-Ionen mit den Sulfat-Ionen entsteht Bleisulfat, was aufgrund seiner schweren Löslichkeit sich in den Poren der Elektroden bevorzugt lagert. Aufgrund dieses Verhalten, lässt sich nur fast die Hälfte des Aktivmaterials entladen, der Rest wird für den Speichervorgang verwendet.

Die Zellspannung einer Bleibatterie liegt mit $2{,}1\,V$ oberhalb der Gleichgewichtsspannung von $1{,}23\,V$ der Elektrolysereaktion[14](H_2/H_2O). Die daraus entsehende Überspannungen versursacht durch Wasserstoff- und Sauerstoffzersetzung sind Ausschlaggebend für das Funktionieren der Bleibatterie [21].

2.1.3.2. Nebenreaktionen

Bedingt durch das hohe Potential der Bleibatteriezelle, verlaufen neben der Hauptreaktion parallele chemische Reaktionen die unvermeidbar sind. Ab einer Zellspannung von $1{,}229\,V$ finden folgende Reaktionen statt:

- Sauerstofferzeugung und Gitterkorrosion an der positiven Elektrode (2.8)
- Wasserstofferzeugung und Sauerstoffreduktion an der negativen Elektrode (2.9)

Diese Reaktionen werden durch folgende chemischen Gleichungen beschrieben:

$$\text{positive Elektrode} : H_2O \rightarrow \frac{1}{2}O_2 + 2H^+ + 2e^- \tag{2.8}$$

$$\text{negative Elektrode} : 2H^+ + 2e^- \rightarrow H_2 \tag{2.9}$$

$$\text{Gesamtreaktion} : H_2O \rightarrow H_2 + \frac{1}{2}O_2 \tag{2.10}$$

Die Gitterkorrosion verursacht eine langsame Zersetzung der positiven Gitterplatte und ist damit einer der wesentlichen Antriebe für den Alterungsprozess eines Bleiakkumulators. Der Ausmaß des Auftretens dieses Effektes hängt stark von der Art der Gitterliegierung und des Herstellungsprozesses ab. Die Sauerstoffreduktion findet nur statt, wenn die Sauerstoffmoleküle die positive Elektrode erreichen. Dieser chemische Vorgang spielt eine zentrale Rolle bei verschlossenen Bleibatterien. Bei dieser Aufbauart von Batterien wird die negative Elektrode überdimensioniert, so dass die positive Elektrode zuerst vollgeladen wird und die Sauerstoffreaktion (2.8) als erstes in Aktion tritt. Anhand der Sauerstoffreduktion (2.9) wird die negative

[14] *Aufspaltung einer chemischen Verbindung unter Einwirkung eines elektrischen Stromes [20]*

Elektrode geladen und damit eine Wasserstoffgasung vermieden. Der Sauerstoffkreislauf ist in diesem Fall mit Hilfe von Gaskanälen gewährleistet.

Anderseits sind die Einwirkungen der Sauerstoffreduktion bei Bleibatterien mit flüssigen Elektrolyten vernachlässigbar, da der Transport des Sauerstoffs innerhalb dieses Mediums sehr langsam verläuft [1].

2.1.4. Relevante Merkmale und Effekte in der Blei-Säure-Batterie

2.1.4.1. Ruhespannung

Die Ruhespannung ist die Spannung einer Zelle, die nach einem Lade-/Entladevorgang, gefolgt von einer Einschwingzeit sich einstellt. Sie hängt stark von der Säuredichte bzw. vom Ladezustand, Alterungszustand und der Temperatur ab. Gemäß Praxis lässt sich die Ruhespannung oder die Gleichgewichtsspannung wie folgt errechnen [10]:

$$\text{Ruhespannung} = \text{Säuredichte} + 0{,}84 \pm 0{,}01\,\text{V} \tag{2.11}$$

Es wird z. B. aus der Säuredichtewert $1{,}28\,\text{g/l}$ einer vollgeladenen Batteriezelle deren Ruhespannung wie folgt berechnet:

$$\text{Ruhespannung} = 1{,}28 + 0{,}84 = 2{,}12 \pm 0{,}01\,\text{V} \tag{2.12}$$

Aus dieser Gleichung 2.11 lässt sich die Säuredichte bestimmen:

$$\text{Säuredichte} = \text{Ruhespannung} - 0{,}84 \pm 0{,}01\,\text{g/l} \tag{2.13}$$

Jedoch, fehlt es mit den Formeln 2.11 und 2.13 berechneten Werte an Genauigkeit und gelten nur als Näherungen der tatsächlichen Werte. Die Begründung dafür ist, dass die Nebenbedingungen (Einschwingzeit der Spannung, Homogenität des Elektrolyten und Temperatur) für die Anwendung dieser Berechnung schwierig einzuhalten sind.

2.1.4.2. Ladezustand

Gemäß der DIN40729 Norm ist der Ladezustand wie folgt definiert:

"Der Ladezustand ist das Verhältnis einer aktuellen Elektrizitätsmenge zu einer zugeordneten n-stündigen Kapazität einer Batterie". [22]

Beachtenswert ist, dass es sich laut der Definition um die gespeicherte Energie und nicht um die entnehmbare Elektrizitätsmenge sich handelt. Der Ladezustand (State of Charge, SOC)

bezeichnet die noch entnehmbare Ladungsmenge im Verhältnis zur maximal verfügbaren Nennkapazität bei Vollladung unter Nenntemperatur und wird in Prozent angegeben. Dieses Verhältnis wird anhand der folgenden Gleichung (2.14) [22] verdeutlicht:

$$\mathrm{SOC} = \frac{\mathrm{Q^{ent}}}{\mathrm{Q^{verf}}} \tag{2.14}$$

Der Nenn-SOC-Wert einer voll geladenen Batterie würde $100\,\%$, und bei einer komplett entladenen $0\,\%$ entsprechen.

Analog zum SOC-Wert, stellt der DOD-Wert (Depth of Charge) den Entladezustand eines Akkus dar. Die beiden Werte stehen in einer direkten Beziehung und deren Summe muss immer $100\,\%$ ergeben.

Betreffend der Ladezustandsbestimmung ist der Ladefaktor von großer Bedeutung. Er kennzeichnet das Verhältnis von aufgenommener zu abgegebener Ladung (in Ah) innerhalb eines Zyklusses und beschreibt damit den Wirkungsgrad des Ladevorganges einer Batterie. Dabei muss der Lade-und Entladevorgang unter den gleichen Bedingungen verlaufen (Ladestrom, Nenntemperatur, Nennkapazität). Der Ladefaktor wird durch folgende Gleichung (2.15) definiert:

$$\mathrm{Ladefaktor} = \frac{Q_{auf}}{Q_{ab}} \tag{2.15}$$

Des Weiteren kann anhand des Ladefaktors die Ladezeit einer Batterie bestimmt werden, um eine bestimmte Ladung entnehmen zu können.

2.1.4.3. Alterungszustand

Der Alterungszustand oder der Gesundheitszustand SOH (State of Health) einer Batterie wird durch das Verhältnis von der aktuell verfügbaren Kapazität zur Nennkapazität bei Nenntemperatur begrenzt.

$$\mathrm{SOH} = \frac{\mathrm{C_{akt}}}{\mathrm{C_{Nenn}}} \tag{2.16}$$

Der SOH-Wert spielt ebenso eine bedeutungsvolle Rolle und ist Bezugspunkt für die Bestimmung des Ladezustandes. Der Alterungszustand und der Ladezustand sind von einander abhängig und beeinflussen sich gegenseitig.

2.1.4.4. Selbstentladung

Die Selbstentladung ist durch die freilaufende chemische Nebenreaktionen begründet (siehe Abschnitt 2.1.3.2). Es entsteht bei einer geladenen Batterie in Ruhezustand Kapazitätsverluste. Die Selbstentladungsrate ist sehr temperatuabhängig und beträgt bei einer Bleibatterie unter Lagertemperatur $20\,^{\circ}\mathrm{C}$ ca. $1\,\%$ bezogen auf die Nennkapazität pro Tag, und verringert sich im Laufe der Ruhezeit [10].

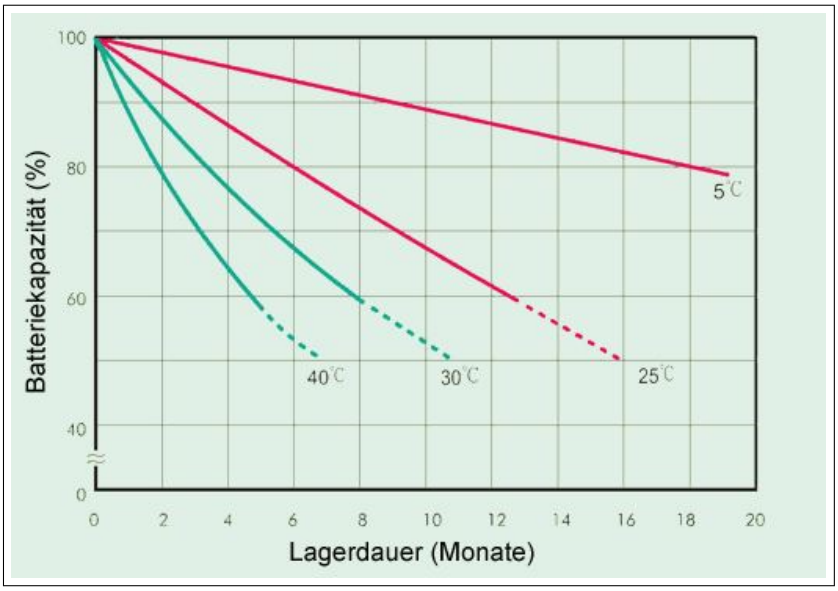


Abbildung 2.6.: Selbstentladung (Kapazitätsverlust) in Abhängigkeit von der Lagertemperatur bei Bleibatterien [23]

Anhand der Selbstentladungskurve in Abb. 2.6, lässt sich erkennen, dass die Selbstentladung bei niedrigen Temperaturen deutlich geringer ist als bei höheren. Es gibt eine Faustregel, die aussagt, dass einer $10\,^{\circ}\mathrm{C}$ Erhöhung der Lagertemperatur eine Verdopplung des Kapazitätsverlustes entspricht [23].

2.1.4.5. Säureschichtung

Die Säureschichtung ist die Bildung unterschiedlicher Säuredichten innerhalb einer Blei-Säure-Batterie. Diese Unterschiede sind abhängig von der Tiefe des Elektrolythaushaltes. Es entstehen während der Ladephasen zwischen den Platten höhere Konzentrationen der Schwefelsäure als im restlichen freiem Elektrolyt. Bedingt durch die Gravitationskräfte werden die konzentrierten Elektrolytanteile sich nach unten ablagern. Durch die Abschlammung wird die vollständige Teilnahme dieser Anteile an der chemischen Reaktion beim Entladen

behindert. Als Folge entsteht im unteren Bereich der Zelle eine höhere Säuredichte als im oberen Bereich. Dieses führt zu Ausgleichsströmen bzw. Lade-/Entlade-Überspannungen, die eine Erhöhung der gesamten Zellspannung verursachen.

Die Säureschichtung spielt bei mobilen Bleibatterien (z. B. Starterbatterie) keine bedeutungsvolle Rolle, da durch Bewegungen, die Säure ständig durchgemischt wird und das Auftreten dieses Effektes minimiert wird. Jedoch bei stationären Anwendung oder bei Laboruntersuchungen darf dieses Verhalten nicht vernachlässigt werden. Unter diesen Erkenntnissen wurde die optische Sensorsonde konzipiert, um Säurekonzentrationsunterschiede zu erfassen (die Funktionsweise wird im Kapitel 3 genauer erläutert).

2.1.4.6. Alterungseffekte

In der folgenden Tabelle werden Alterungseffekte bei einer geschlossenen Bleibatterie und deren Merkmale, Erscheinungsgründe und deren Folgen zusammengefasst dargestellt (übernommen und modifiziert aus [1]) :

Alterungseffekt	Merkmale	Ursachen	Folgen
Korrosion des positiven Ableiters	Siehe Kapitel 2.1.3 Unterkapitel Nebenreaktion	- hohe Ladespannung - hohe Temperatur - geringe Säuredichte - ungünstige Liegierungen - schlechte Gitterherstellung - Säureschichtung	- Erhöhung des ohmschen Widerstandes des Ableiters - schlechter Kontakt zwischen Aktivmasse und Ableiter - Ableiter löst sich im schlimmsten Fall auf
Ausfall der Aktivmasse	bedingt durch den mechanischen Stress der Aktivmasse	- mechnischer Stress beim Zyklieren - hohe Stromdichte beim Entladen - hohe Ladespannung die starke Gasung verursache Tiefentladung - Umpolung -Säureschichtung	- Ausfall der Aktivmasse - Abschlammung (Aktivmasse lagert sich am Batterieboden ab) - Kapazitätsverlust - Kurzschluss
Sulfatation	Bildung von relativ großen Bleisulfatkristalle an den Gitterplatten	- ständige Teilladungen - Tiefentladungen - Säureschichtung - längere Lagerzeiten	- Kapazitätsverlust - Innenwiderstand nimmt zu - Säuredichte nimmt ab
Verbleiung der negativen Aktivmasse	Die Spreizmittel werden aus der neg. Masse "herausgewaschen". Dadurch verdichtet sich das aktive Material und die zur Verfügung stehende Oberfläche verringert sich.	- Stillstand der Batterie im teilgeladenen Zustand - hohe Betriebstemperaturen - starkes Gasen beim Laden - wiederholte Teilladungen - Selbstentladung der negativen Platte	- Kapazitätsverluste
Gitterwachstum	Die Korrosionsprodukte nehmen ein größeres Volumen als die Ausgangsprodukte ein	Selbstentladung der negativen Platte	- Kurzschluss zwischen den Platten
Dendritenwachstum	Tritt bei Mangel an Kristallisationskeime auf	- hohe Stromdichte beim Entladen - Tiefentladung - Säureschichtung	- Kurzschluss zwischen den Platten - Abschlammung
Zerstörung des Separators	Durch die mechanische Belastung und die Korrosion des Separators erfolgt die Zerstörung des Separators	- Gitterwachstum - Dendritenwachstum - Oxidation des Separators	- Kurzschluss

Tabelle 2.2.: Alterungseffekte in der Bleibatterie [13]

2.2. Optik und Lichtwellenleiter

2.2.1. Grundlagen Optik

2.2.1.1. Licht als elektromagnetische Welle

In der Optik wird das Licht durch zwei Theorien erklärt. Die erste Theorie betrachtet das Licht als Bewegungsfluss von kleinen Teilchen, jedoch können anhand dieser Theorie wichtige optische Phänomene nicht erklärt werden. Laut der anderen Betrachtung, sind Lichtwellen als elektromagnetische Wellen eingeordnet, die mit einer Frequenz zwischen $0{,}3\,\mathrm{THz}$ und $3000\,\mathrm{THz}$ charakterisiert sind. Diese Wellen werden als optische Strahlung bezeichnet. Jedoch ist nur eine bestimmtes Frequenzbereich von $385\,\mathrm{THz}$ und $790\,\mathrm{THz}$ dieser Wellen für das menschliche Auge sichtbar und werden respektive als Rot und Violett wahrgenommen. Dieser sichtbares Bereich wird Licht genannt. In der folgenden Abb. 2.7 wird das Spektrum aller elektromagnetischen Wellen dargestellt.

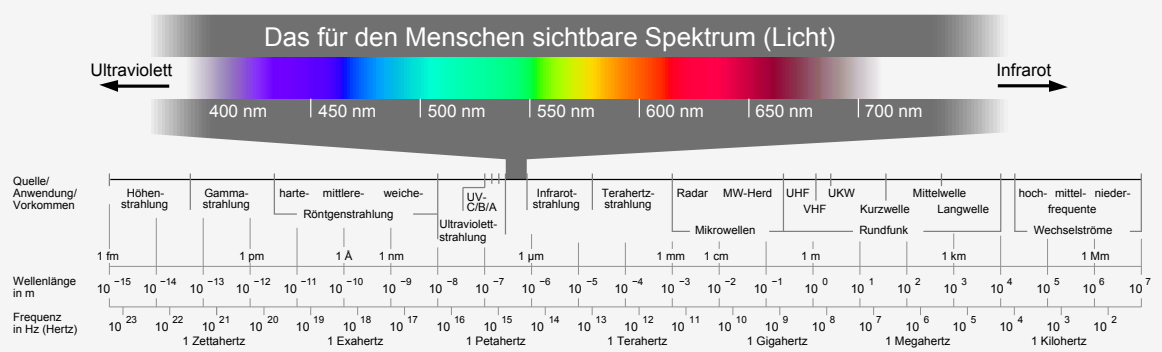


Abbildung 2.7.: Spektrum elektromagnetischer Wellen und Einordnung des sichtbares Bereichs [24]

Anhand der Abbildung 2.7 ist es zu erkennen, dass elektromagnetische Wellen auch durch ihre Wellenlänge λ gekennzeichnet sind. Diese Kennzeichnung beschreibt die Ausbreitung der Wellen im Vakuum und hat sich besonders für optische Wellen mehr etabliert als die frequenzabhängige Beschreibung. Die Wellenlänge bezogen auf die Ausbreitungsgeschwindigkeit im Vakuum wird in der folgenden Gleichung 2.17 definiert [13]:

$$\lambda = \frac{c_0}{f} \tag{2.17}$$

Wobei:

- λ: Wellenlänge in $[m]$
- c_0: Ausbreitungsgeschwindigkeit im Vakuum = Lichtgeschwindigkeit im Vakuum = $2{,}998.10^8\,\mathrm{m/s}$
- f: Frequenz der Welle in $[Hz]$

Laut dieser Definition 2.17 liegt der Wellenlängenbereich des sichtbaren Spektrums innerhalb $380\,\mathrm{nm}$ und $780\,\mathrm{nm}$.

Dabei ist die Frequenz für jede Welle festgelegt und stellt eine unabhängige Erhaltungsgröße dar. Eine Fortpflanzung einer elektromagnetischen Welle in verschiedenen Ausbreitungsmedien erzwingt keine Änderung in deren Frequenz.

Darüber hinaus, ist die Phasengeschwindigkeit (Ausbreitungsgeschwindigkeit) eine wichtige Eigenschaft der optischen Wellen. Diese Größe ist vom Material des Ausbreitungsmediums abhängig und lässt sich durch die Maxwell'schen Gleichung 2.18 definieren [13]:

$$\mathrm{v} = \frac{\mathrm{c_0}}{\mathrm{n}} \quad mit \quad n = \sqrt{\varepsilon_r . \mu_r} \tag{2.18}$$

wobei:

- v: Phasengeschwindigkeit in $[m/s]$
- c_0: Ausbreitungsgeschwindigkeit im Vakuum, siehe Gleichung 2.17
- n: Brechzahl oder Brechungsindex des Mediums
- ε_r: relative Dielektrizität, auch Permitivität genannt, bezeichnet die Durchlässigkeit des Materials für elektrische Felder [25]
- μ_r: relative Permeabilität, bezeichnet die magnetische Leitfähigkeit von Materialien [25]

Des Weiteren ist die relative Dielektrizitätszahl ε_r frequenzabhängig und damit auch die Phasengeschwindigkeit [13].

2.2.1.2. Reflexion und Brechung von Licht

Reflexion und Brechung sind grundlegende Phänomene in der Optik. Wenn ein Lichtstrahl an einer Grenzfläche zwei Ausbreitungsmedien mit unterschiedlichen Brechungsindizes trifft, wird ein Teil des Lichtstrahles reflektiert und ein anderer Teil in das zweite Medium gebrochen. Mit Hilfe der nächsten Abb. 2.8 werden diese beiden optischen Gesetze veranschaulicht.

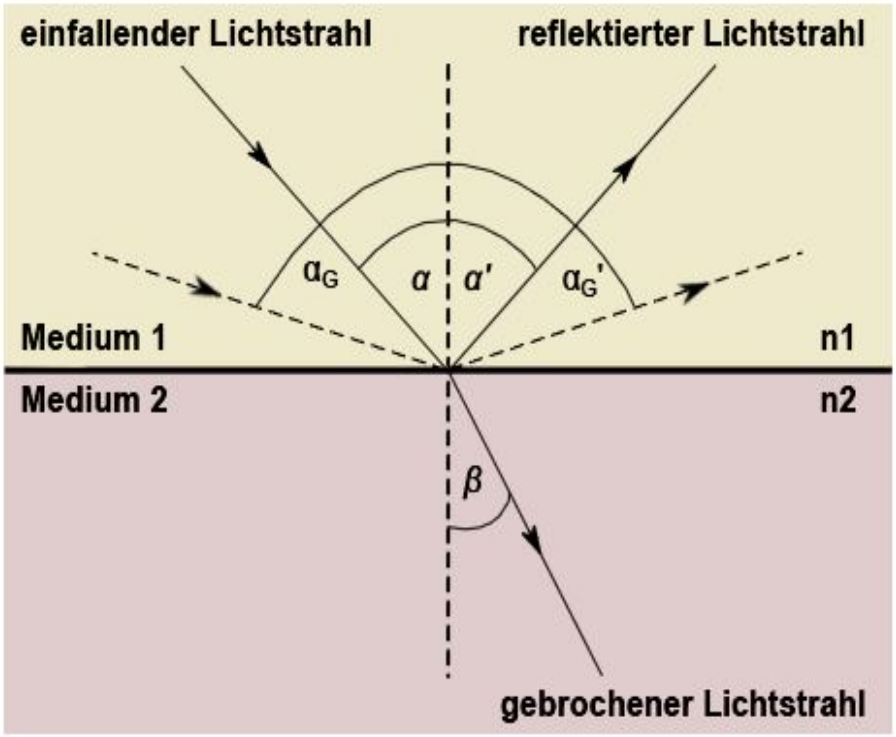


Abbildung 2.8.: Reflexion und Brechung von Licht [26]

Es gilt für die Reflexion allgemein und unabhängig von den Brechungsindizes der Medien n_1 und n_2 folgendes Gesetz 2.19:

$$Einfallswinkel\ \alpha = Ausfallswinkel\ \acute{\alpha} \tag{2.19}$$

Dagegen ist der Brechungswinkel abhängig von der Phasengeschwindigkeit des Lichtes und den Brechzahlen der Medien. Auch der Übergang von einem optisch dünneren Medium zu einem optisch dichteren und umgekehrt, spielt eine entscheidende Rolle. Dieses Verhältnis wird anhand folgende Gleichung dargestellt [26]:

$$\frac{sin\,\alpha}{sin\,\beta} = \frac{v_1}{v_2} = \frac{n_2}{n_1} \tag{2.20}$$

wobei

- α: Einfallswinkel des Lichtes, bezogen auf das Lot
- β: Brechungswinkel des Lichtes, bezogen auf das Lot
- v_1: Phasengeschwindigkeit des Lichtes im Medium 1
- v_2: Phasengeschwindigkeit des Lichtes im Medium 2

- n_1: Brechzahl oder Brechungsindex des Mediums 1
- n_2: Brechzahl oder Brechungsindex des Mediums 2

Ein Teil des Lichtes wird bei der Brechung auch reflektiert. Beim Übergang von einem Medium mit einer kleineren Brechzahl zu einem mit einer höhere Brechzahl, wird der Lichtstrahl zum Lot hin gebrochen. Anders herum wird der Lichtstrahl vom Lot weg gebrochen. Im Fall eines Überganges des Lichtes von einem optisch dichteren Medium zu einem optisch dünneren, wenn dabei ein bestimmter Einfallswinkel überschritten wird, tritt eine totale Reflexion des Lichtes an der Grenzoberfläche der beiden Medien ein. Dieser Winkel ist in der Optik als Grenzwinkel der Totalreflexion bezeichnet und wird durch folgende Beziehung 2.21 bestimmt [13]:

$$\sin \alpha_G = \sin \alpha = \frac{v_1}{v_2} = \frac{n_2}{n_1} < 1 \tag{2.21}$$

2.2.2. Grundlagen Lichtwellenleiter

Allgemein werden Lichtwellenleiter (Im folgenden LWL genannt) für die Übertragung von elektromagnetische Wellen verwendet. In Faserform werden sie hauptsächlich für die Übertragung von digitalen Signalen, Energie und als Sensoren eingesetzt.

Eine LWL-Faser besteht grundsätzlich aus zwei Schichten. Die innere Schicht wird als Faserkern (eng. Core) bezeichnet. Die äußere Schicht umschließt den Kern und wird Fasermantel (Cladding) genannt. Diese beiden Komponenten sind optisch hochtransparent und für die Lichtübertragung zuständig. Häufig werden Kern und Mantel, je nach Anforderungen, mit einer oder mehreren Kunststoffschichten (eng. coating) überzogen, diese dienen zum Schutz vor äußeren Einflüssen und Faserbruch [13].

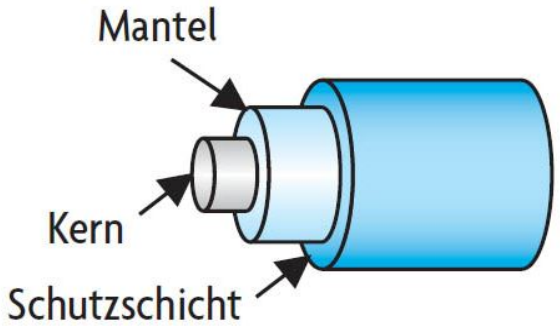


Abbildung 2.9.: Aufbau eines typischen Lichtwellenleiters [27]

Die Lichtübertragung innerhalb eines LWLs basiert auf dem Prinzip der Totalreflexion 2.21. Grundsatz dafür ist, dass die Brechzahl des Faserkerns größer als die des Fasermantels sein muß. Ist diese Bedingung erfüllt, wird das Licht an der Mantelgrenzschicht in den Kern zurück reflektiert und breitet sich dort weiter aus. Dabei muss in die Faser eingekoppeltes Licht nicht den Akzeptanzwinkel überschreiten. Liegt der Eintrittswinkel eines Lichtstrahles außerhalb der Akzeptanzbereich der Faser, wird die Lichtwelle am Cladding hinein gebrochen und tritt teilweise aus der Faser aus [13]. Anhand der folgenden Abb. 2.10 wird die Lichtausbreitung in einer LWL-Faser dargestellt:

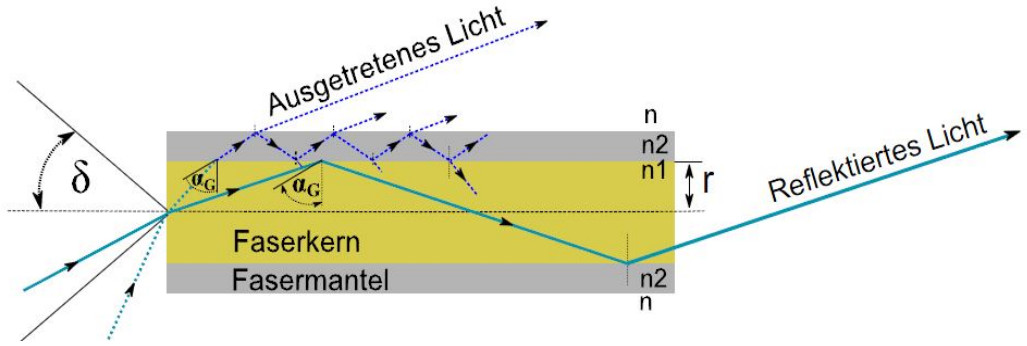


Abbildung 2.10.: Lichtausbreitung in einer LWL-Faser [26]

wobei:

- α_G: Grenzwinkel der Totalreflexion zwischen Mantel und Faserkern
- δ: Akzeptanzwinkel der Faser
- n: Brechungsindex des Mediums außerhalb der Faser
- n_1: Brechungsindex des Kernmaterials
- n_2: Brechungsindex des Mantelmaterials
- r: Radius des Faserkerns

Des Weiteren werden Lichtwellenleiter durch die Fähigkeit der Übertragung unterschiedlichen Wellenformen (Moden) und ihrer radialen Brechzahlprofiländerungen identifiziert. Gemäß dieser Charakteristika existieren folgende LWL-Fasertypen:

- **Multimodefasern (Mehrmoden-Fasern) mit Gradientenindex-Brechzahlprofil:**

  Bei Mehrmodenfasern mit Gradientenindex fällt der Brechungsindex des Faserkerns in Richtung Cladding monoton ab. Dagegen bleibt die Brechzahl des Mantels konstant. Solche Fasern besitzen einen relativ großen Durchmesser und erlauben damit

die Ausbreitung von verschiedenen Moden. Der Vorteil des Gradientenidexprofils liegt darin, die Laufzeitdifferenzen verschiedener Wellen zu unterdrücken. Dies geschieht durch die Manipulierung der Ausbreitungsgeschwindigkeiten der Moden mit Hilfe der kontinuierlichen Brechzahländerung des Kernes. Die Lichtstrahlen innerhalb der Faser breiten sich helixförmig aus. Eine kleine Signaldämpfung und geringe Impulsverbreiterung von Ausgangssignalen werden bei einer Bandbreite bis zu $1\,\mathrm{GHz.Km}$ durch diese Technik erzielt [28].

- **Multimodefasern (Mehrmoden-Fasern) mit Stufenindex-Brechzahlprofil:**

  Innerhalb dieser Fasern können auch mehrere Moden geführt werden. Im Gegensatz zu dem Gradientenindex-Brechzahlprofil, ist das Brechungsindex des Kernmaterials über die gesamte radiale Querschnittsfläche unverändert. Durch die stufenweise Änderung der Brechzahlen an der Kern-Cladding-Grenzschicht, erhalten die Moden einen zickzackförmig Ausbreitungsverlauf. Dadurch entstehen unterschiedliche Laufzeiten der gleichen Wellenlänge (Modendispertion), was eine beachtliche Impulsverbreiterung am Ausgangssignal verursacht. Weitere Nachteile dieser Art von LWLs sind starke Dämpfung und eine begrenzte Bandbreite von maximal $100\,\mathrm{MHz/Km}$ [28].

- **Singlemodefasern (Monomoden-Fasern):**

  Monomode-LWLs haben immer eine Stufenindex-Brechzahlprofil und sind durch einen sehr geringen Kerndurchmesser gekennzeichnet. Es kann nur eine einzige Wellenform in der Faser geführt werden. Innerhalb der Singlemodefasern lassen sich Signale mit einer sehr kleinen Dämpfung und großer Bandbreite übertragen. Bedingt durch die fast konstante Signallaufzeit der geführten Moden, tritt am Ausgangssignal keine Impulsverbreiterung auf. Sie werden häufig für Signalübertragungen über große Entfernungen eingesetzt [28].

Des Weiteren werden LWL-Fasern durch ihre Materialzusammensetzung gekennzeichnet. In der folgenden Tabelle 2.3 werden üblicher Fasertypen und deren Materialkombinationen dargestellt:

Namenskürzel	Kernmaterial	Mantelmaterial
Glasfaser	Quarzglas	Quarzglas
PCS-Faser (plastic cladded silica)	Quarzglas	Silikonharz
HCS-Faser (hard cladded silica)	Quarzglas	Hartpolymer
POF-Faser (polymer optical fiber)	Plexiglas(PMMA)	Fluoriniertes Polymer

Tabelle 2.3.: Bezeichnung und Materialzusammensetzung üblicher LWL-Fasern [13]

Für die Anwendung als optischer Sensor müssen die Fasern gebogen werden (dazu mehr Informationen im nächsten Abschnitt). Die POF-Faser weist die für diesen Zweck notwendige Flexibilität und ein stabiles Verhalten im gebogenen Zustand gegenüber Glasfasern und anderen LWL-Arten auf. Darüber hinaus, lässt sich diese Art von Kunststofffasern leichter verarbeiten (z.B. Entmantelung) im Gegensatz zur Glasfaser, die sehr bruchempfindlich ist. Ein weiteres wichtiges Kriterium der POF-Fasern ist die Säurebeständigkeit. Voruntersuchungen wurden in der Bachelorarbeit [9] durchgeführt und es fiel dabei die Entscheidung, aufgrund der genannten Kriterien, auch auf den Einsatz von POF-Fasern. Jedoch haben POF-Fasern große Signaldämpfung gegenüber Glasfasern. Es muss daher darauf geachtet werden, dass sie an sogenannten optischen Betriebsfenstern betrieben werden, wo der Dämpfungsgrad am geringsten ist. In der Abb. 2.11 wird der Dämpfungsverlauf und das Betriebsfenster einer POF-Faser verdeutlicht:

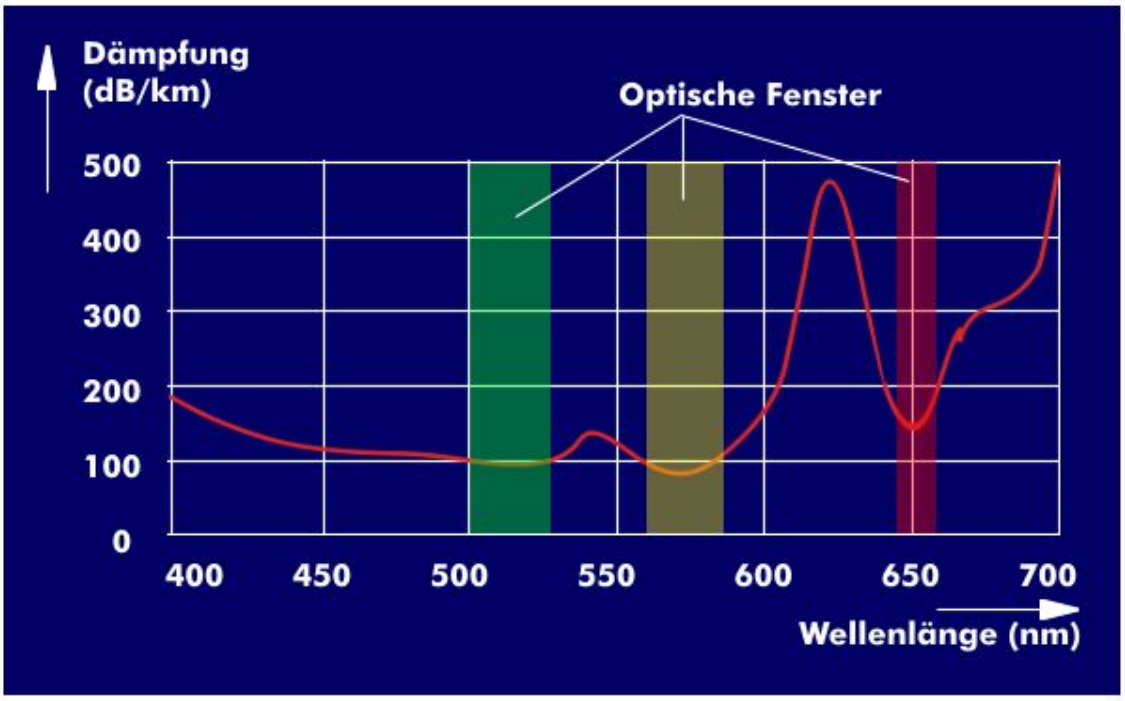


Abbildung 2.11.: Signaldämpfungsverlauf und optische Betriebsfenster im sichtbaren Wellenlängenbereich bei POF-Fasern [29]

Anhand der Abb. 2.11 ist zu erkennen, dass im sichtbaren Wellenlängenbereich drei Betriebsfenstern gibt (grün, gelb und rot markiert) wo keine erhebliche Signaldämpfung statt-

findet. Dämpfungen in POF-Faser sind durch Rayleighstreuung[15] und Resonanzabsorption von Kohlenwasserstoffionen-Schwingungen begründet.

Des Weiteren sind diese Erkenntnisse hilfreich für die Auswahl der LEDs (Wellenlänge der verwendeten LED muss den Betriebsfenstern entsprechen), die innerhalb dieser Arbeit entwickelte Sensorik mit der POF-Faser eingekoppelt werden.

2.2.3. Integration von Lichtwellenleiter für das optische Messverfahren

Biegung von Lichtwellenleiters:

Durch das Biegen von LWL-Fasern wird der Grenzwinkel zur Totalreflexion an der Grenzschicht zwischen Faserkern und Fasermantel unterschritten. Ein Lichtanteil wird im Claddingbereich gebrochen und breitet sich dort weiter aus. Ebenso kommt es beim Überschreiten eines bestimmten Biegeradius zum Lichtaustritt aus der Faser heraus. Die übertragene Lichtleistung im Kernbereich nimmt dadurch ab. Der Leistungsverlust ist dabei vom Brechungsindex des Mantelmaterials und des Mediums außerhalb der Faser im Biegebereich abhängig. 2.12 Anhand der Abb. wird dieses Effekt veranschaulicht:

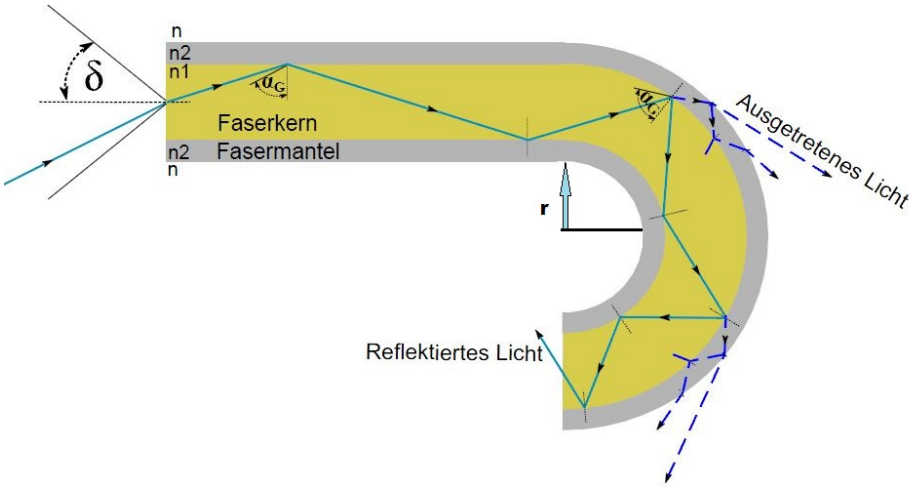


Abbildung 2.12.: Lichtausbreitung bei einer gebogenen LWL-Faser [26]

[15] *Die elastiche Streuung von Licht (elektromagnetische Wellen) an Molekülen, die geometrisch gegenüber der eingestrahlten Wellenlänge sehr klein sind [30].*

- α_G: Grenzwinkel der Totalreflexion zwischen Mantel und Faserkern
- δ: Akzeptanzwinkel der Faser
- n: Brechungsindex des Mediums außerhalb der Faser
- n_1: Brechungsindex des Kernmaterials
- n_2: Brechungsindex des Mantelmaterials
- r: Biegeradius der LWL-Faser

Einsatz von gebogenen LWLs zur Detektion der Brechzahländerung (bzw. der Dichteänderung) des Elektrolyten einer Blei-Säure-Batterie:

Je nach Ladezustand der Blei-Säure-Batteriezelle ändert sich die Dichte des Elektrolyten (Schwefelsäure). In dieser Arbeit und [9] werden in optischen Messverfahren diese Dichteänderungen am Elektrolyten erfasst, um Aussagen über den Ladezustand der Zelle zu treffen, unabhängig von der Batteriehistorie und den elektrischen Größen. Da die Dichte proportional zur Brechungszahl ist, werden die im vorherigen Abschnitt erwähnten Eigenschaften der Lichtausbreitung innerhalb einer gebogenen LWL-Faser zu Nutze gemacht. Zu diesem Zweck wurden POF-Fasern entmantelt (die Schutzschicht der Fasern wurde entfernt) um absichtlich den Lichtaustritt an den Biegepunkten zur verstärken und bei einem bestimmten Biegeradius gekrümmt. An einem Ende der Faser wird Licht durch eine LED (Sender) geschickt und am anderen Ende wird mit Hilfe einer Photodiode (Empfänger) die Lichtleistungsverluste ausgewertet. Mittels der nächsten Abb. 2.13 wird dieses Messverfahren verdeutlicht:

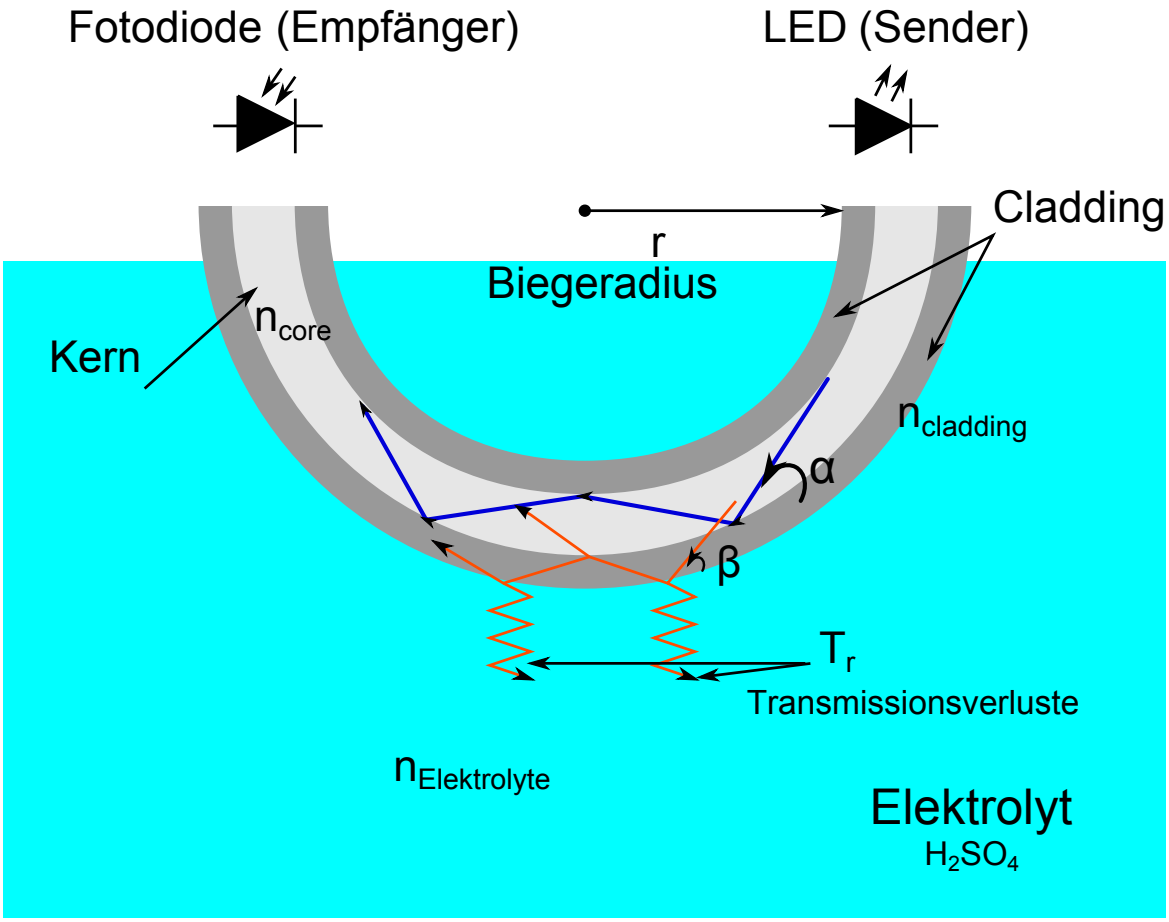


Abbildung 2.13.: optischer Sensor basiert auf gebogene LWL-Faser zur Ermittlung der Elektrolytendichte einer Bleibatteriezelle

wobei:

- β: Einfallswinkel auf die äußere Cladding/Elektrolyt-Grenzschicht des vom Kern ins Cladding gebrochenes Lichtstrahls
- α: Einfallswinkel des Lichtstrahles auf die Grenzschicht Kern/Cladding

Anhand dieser Sensorik sind folgende Effekte während des Zyklierens der Bleibatterie zu beobachten, zu erfassen und daraus die Ladezustände der Batteriezellen zu ermitteln:

- Entladephase:

  Säurekonzentration: $\searrow$ $\Rightarrow$ Säuredichte: $\searrow$ $\Rightarrow$ Säure-Brechungsindex: $\searrow$ $\Rightarrow$ Lichaustritt: $\searrow$ $\Rightarrow$ Transmissionsleistung: $\nearrow$

- Ladephase:

  Säurekonzentration: $\nearrow$ $\Rightarrow$ Säuredichte: $\nearrow$ $\Rightarrow$ Säure-Brechungsindex: $\nearrow$ $\Rightarrow$ Lichaustritt: $\nearrow$ $\Rightarrow$ Transmissionsleistung: $\searrow$

Wobei:

- $\searrow$: Abnahme
- $\nearrow$: Zunahme

3. Entwicklung der neuen LWL-Sensorsonde und des Messaufbaus

3.0.4. Entwicklung der LWL-Sensorsonde

3.0.4.1. Makro-Biegung (Makrokrümmung) der POF-Fasern

Für die Detektion der Dichteänderung des Elektrolyten, wurden in dieser Arbeit Multimode-Kunststoff-Fasern (PMMA) mit einem Stufenindexbrechzahlprofil und einem Durchmesser von $0{,}5\,\mathrm{mm}$ verwendet. Diese Fasern haben den Vorteil, sich leicht mehrfach verbiegen zu lassen und können ohne Schutzschicht beschafft werden, was die Entmantlungsarbeit erspart und das Risiko, während der Entmantelung die Fasern zu beschädigen, entfällt.

Die Fasern müssen gebogen werden, um das Austreten des Lichts außerhalb der Fasern in den Elektrolyten zu erzwingen. Es handelt sich hierbei um eine Makrokrümmung der Fasern. Dabei spielt der Biegeradius eine wichtige Rolle, um die erwünschten Signalverluste zu erzielen und gleichzeitig eine gewisse Signalqualität gewährleisten zu können. Da Große Transmissionsverluste an den Biegestellen können dazu führen, dass nur Rauschen empfangen werden kann. Anhand der Fresnel Koeffizientengleichung 3.1 und der Abb. 2.13 lassen sich die Transmissionsverluste in den Biegepunkten wie folgt definieren [8]:

$$T_r = \frac{4 sin\,\beta (sin^2\beta - sin^2\gamma)^{\frac{1}{2}}}{[sin\,\beta (sin^2\beta - sin^2\gamma)^{\frac{1}{2}}]^2} \tag{3.1}$$

wobei

$$\gamma = cos^{-1}(\frac{n_{elektrolyte}}{n_{Cladding}}) \tag{3.2}$$

- T_r: Transmissionsverluste
- β: Einfallswinkel auf die äußere Cladding/Elektrolyt-Grenzschicht des vom Kern ins Cladding gebrochenes Lichtstrahls
- $n_{elektrolyte}$: Brechungsindex des Elektrolyten

- $n_{Cladding}$: Brechungsindex des Claddingsmaterials

Zur Bestimmung des optimalen Biegeradius, wurden in der Arbeit [8] praktische Versuche durchgeführt. Dabei wurden POF-Fasern mit unterschiedlichen Biegeradien in mehrere Schwefelsäurelösungen mit verschiedenen Konzentrationgraden, die bestimmten Ladezuständen entsprechen, eingetaucht. Danach wurden durch die Faser Lichtimpulse geschickt und schließlich die Impulseantworten in Form von Spannung ermittelt und ausgewertet. Als Hauptbewertungskriterium gilt, dass die Differenz der Impulsantworten bei der geringsten und der höchsten Säurekonzentrationen möglichst groß ist, was einer gute Empfindlichkeit entsprechen würde.

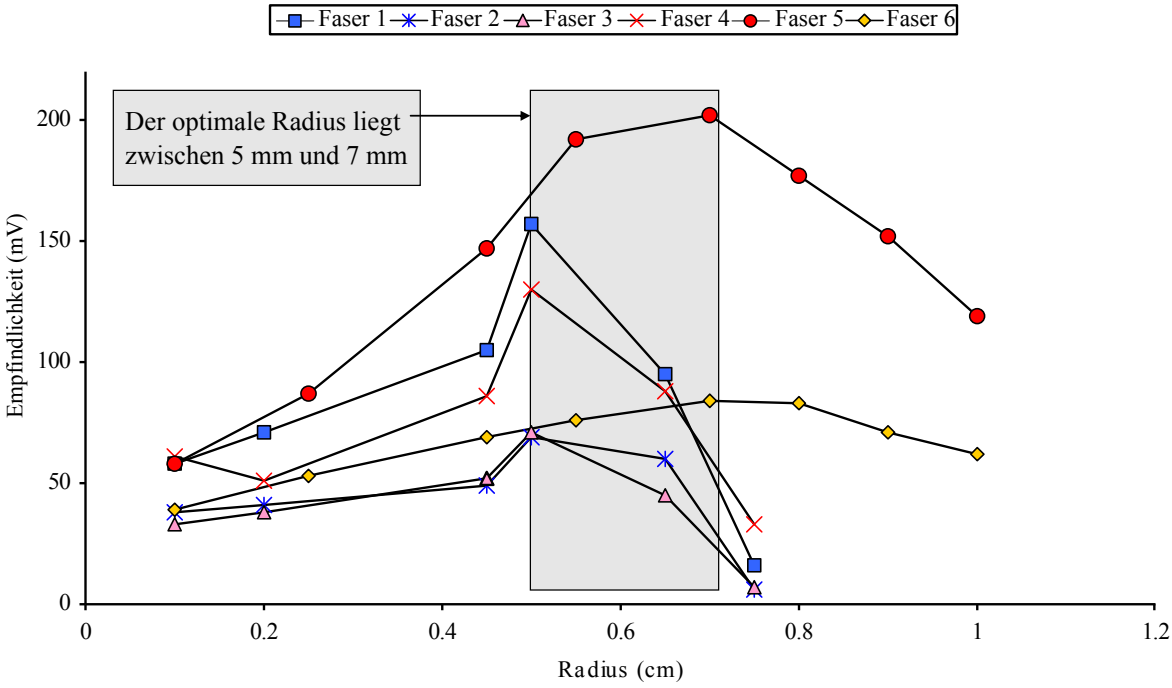


Abbildung 3.1.: Bestimmung des optimalen Biegeradius der POF-Fasern mit der größten Empfindlichkeit gegenüber der Konzentrationsänderung der Säure [9][8]

Anhand der Abb. 3.1 ist ersichtlich, dass der optimale Biegeradius, die den Kompromiss zwischen Signalverlust und Erhaltung der Signalqualität einhält, in einem Bereich zwischen 5 mm und 8 mm liegt.

3.0.4.2. Integrative Messung der Dichteänderung des Elektrolyten

Die für die optischen Untersuchungen verwendete Bleibatterie wird stationär betrieben. Dadurch entsteht eine Säureschichtung (siehe Abschnitt 2.1.4.5) innerhalb der Batteriezelle, was eine Konzentrationsinhomogenität bzw. unterschiedliche Brechungsindizes der Säure besonders in der vertikalen Richtung verursacht und damit zur Verfälschung der Messdaten führen kann. Aus diesem Grund werden durch mehrfache Biegestellen der POF-Fasern mehrere Messpunkte errichtet, die das Erfassen unterschiedlicher Brechungsindizes ermöglichen. Es entsteht eine integrative Messung über die gesamte Elektrolytenfläche und damit eine Kompensation der Konzentrationsunterschiede, was die Genauigkeit der erhaltenen Messdaten erhöht. Jedoch muss darauf geachtet werden, dass die Signalqualität erhalten bleibt, da die Leistung des Nutzsignales deutlich abnimmt, je mehr Biegepunkte vorhanden sind.

3.0.4.3. Mechanische Konstrunktion der Sensorsonde (Hinweis: Dieser Aufgabenteil wurde von Herrn Wahid Nasimzada im Rahmen des Batsen Projektes übernommen)

Für die Einbringung der gebogenen POF-Fasern in der Batteriezelle, wurde eine Sensorsonde (Sensorträger) entwickelt, die günstige und adaptive Möglichkeiten zur Integration in den Zellenanbau einer herkömmlichen Auto-Bleibatterie bietet. Für diesen Zweck, wurden mittels einer 3D-Konstruktionssoftware (free-CAD) 3D-Modelle erstellt und mit einem 3D-Drucker gefertigt. Mit Hilfe dieses Fertigungsverfahren können mehrere Sonden reproduziert werden, die kaum mechanische Streuungen aufweisen.

Eine besonders wichtige Voraussetzung für diese Anwendung, ist die chemische Neutralität des verwendeten 3D-Druck-Materials (Polylactide). Daraufhin wurden ausgedruckte Exemplare in Schwefelsäure eingetaucht. Es ergab keine erkennbaren Beschädigungen oder Verätzungsmerkmale an der Oberfläche der getesteten Gegenstände. Polylactide (auch Polymilchsäuren, oder PLA[1] genannt) sind thermoplastische Kunststoffe, die sich in ihren mechanischen und chemischen Eigenschaften sehr den Polyethylenterephthalat (PET) ähneln, die säurebeständig sind. Diese Erkenntnisse werden durch die Testergebnisse noch bestätigt.

Die entwickelten Sensorsonde haben die Grundform eines doppelgängige Gewindes und sollen die darin umgewickelten LWLs stabil halten. Im Laufe der Entwicklung sind viele Prototypen entstanden, die sich in der Anzahl der Windungen (Biegestellen) , Biegeradien und Formen unterscheiden. Diese wurden auf Signalqualität getestet und entsprechend weiter

[1] *engl. Polylactic acid*

optimiert. Dabei wurde darauf geachtet, dass die realisierten Biegestellen dem optimalen Radiusbereich entsprechen.

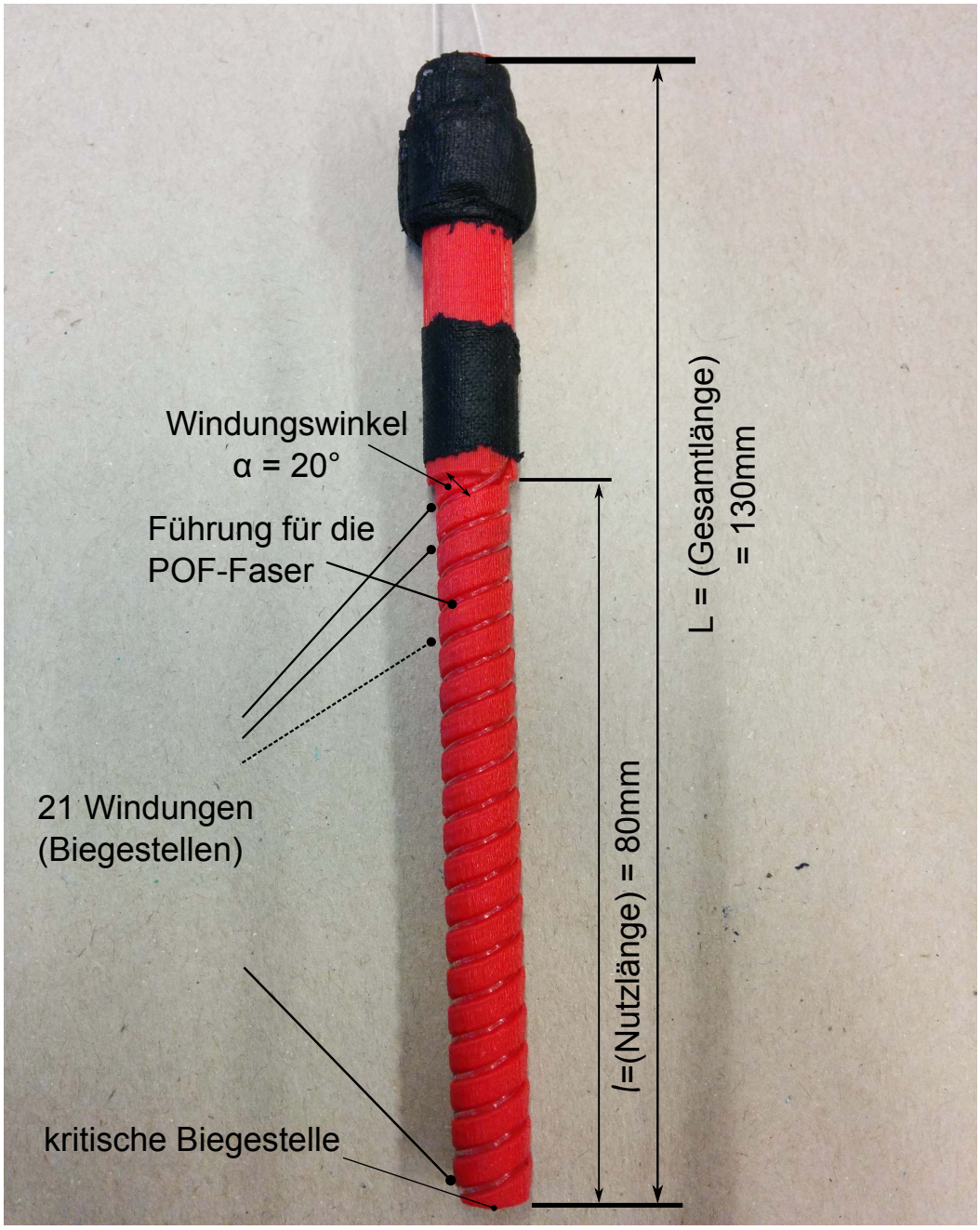


Abbildung 3.2.: Prototypen der Sensorsonde mit Konstruktionsmängeln, hergestellt mittels 3D-Druckers.

Der in der Abb. 3.2 dargestellte Prototyp, weist Konstruktionsmängeln auf. Die hohe Anzahl an Windungen über einer relativ geringen Nutzlänge, der flache Windungsverlauf durch den kleinen ausgewählten Windungswinkel (bzw. geringer Biegeradius) und die kantige Rückführung der Faser an der markierten kritischen Biegestelle führen zu großen Transmissionsverlusten. Dieses Verhalten wurde optisch während der Einkopplung der LWL mit einer Lichtquelle beobachtet und anhand der gemessenen Transmissionsverluste nachgewiesen.

Des Weiteren wurden Optimierungen durchgeführt, um bessere Ergebnisse bezüglich der mechanischen Stabilität und der Signalqualität zu erzielen. Die Anzahl der Windungen wurde reduziert, die Nutzlänge wurde erweitert und durch den relativ steilen Windungsverlauf wurde der Biegeradius vergrößert und damit die Transmissionsverluste so minimiert, dass die Detektion der Dichteänderung möglichst noch gewährleistet wird. Die Gesamtlänge der Sonde wurde durch ein PVC[2]-Rohr ergänzt um tief in die Zelle einzudringen und die LWLs und die Sonde stabil zu halten.

Der Biegeradius lässt sich anhand folgender Gleichung 3.3 berechnen:

$$r_{\text{Biege}} = \frac{r_{\text{Sonde}}}{cos\,\alpha} \tag{3.3}$$

wobei:

- r_{Biege}: Biegeradius der Faser
- r_{Sonde}: Radius der Sonde
- α: Windungswinkel

[2] *Polyvinylchlorid (säurebeständig)*

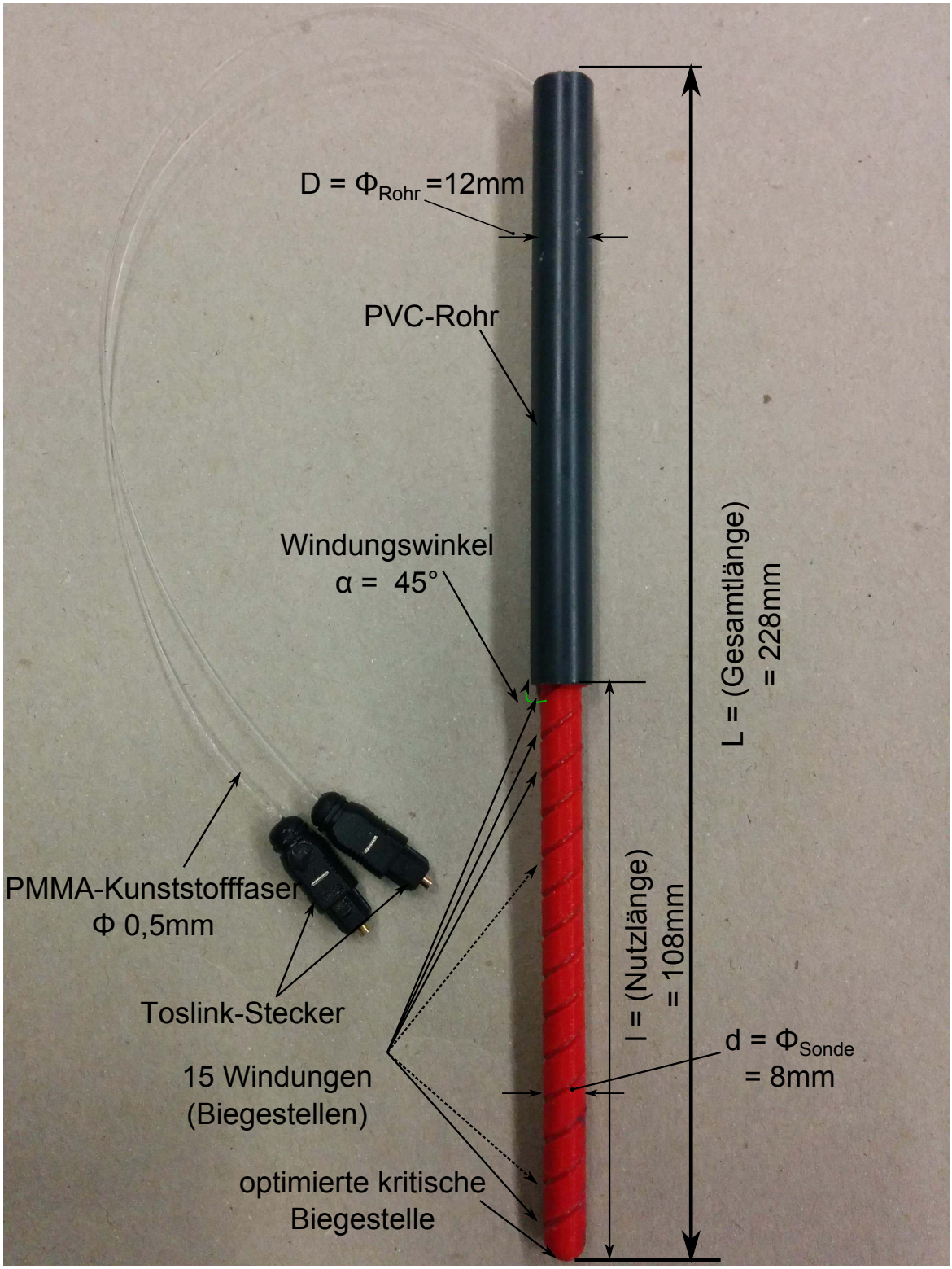


Abbildung 3.3.: optimierte Sensorsonde: Anzahl der Windungen auf 15 reduziert, Gesamtlänge und Nutzlänge ergänzt, Windungswinkel (bzw. Biegeradius) vergrößert und die krtitische Biegestelle optimiert.

Die aufgewickelten PMMA-Kunsstofffasern lassen sich bei dem klein ausgewählten Durchmesser (ϕ = 0,5 mm) leichter um die Sonde biegen und bewahren somit eine stabile Form. Für die Verbindung der Fasern mit der entwickelten Sensorik wurde zuerst das TOSLINK[3]-Anschlusssystem verwendet. Der Einbau der Faserenden mit den Steckern wurde manuell im Labor durchgeführt. Ein fertiges Fasersystem, was die Anforderungen für diese Anwendung entsprechen würde, war nicht im Markt erhältlich. Dieser Verbindungsprozess wurde ebenso optimiert. Da der Durchmesser der TOSLINK-Stecker ($1\,mm$) größer als der von den Fasern ($0{,}5\ mm$) war und eine genaue zentrierte Positionierung der Faser bei den ersten Versuchen nicht möglich. Diese nicht optimale Verbindung hat erhebliche unerwünschte Transmissionsverluste verursacht. Aus diesem Grund wurden die Faserenden mit einem Stück Isolierschlauch umhüllt. Dadurch wurde eine stabile und zentrierte Einbringung der Fasern in die Stecker erreicht.

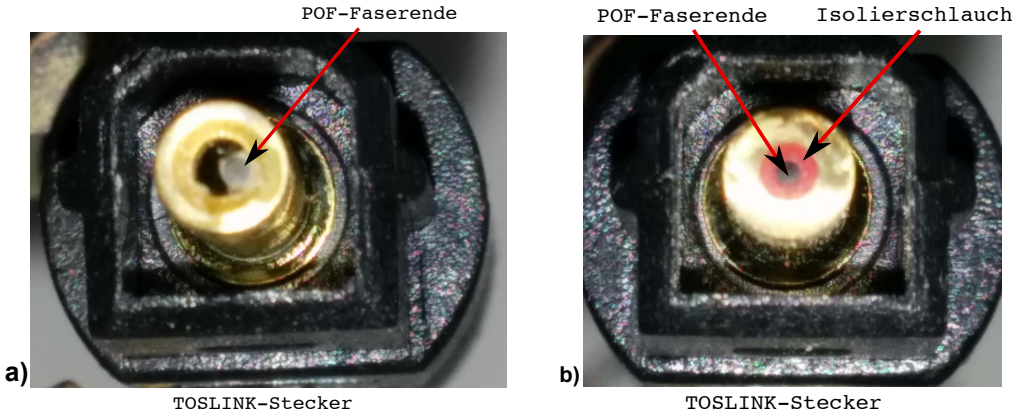


Abbildung 3.4.: Verbindung der POF-Fasern mit den TOSLINK-Steckern. **Links: a)** Version 1, die POF-Fasern sind nicht zentriert in den Stecker eingebaut. Beim Einkoppeln der Fasern mit der LED wird zuerst ein großer Anteil des Lichts nicht übertragen. Zusätzlich wird der optimale Ausbreitungsverlauf der aufgenommenen Lichtstrahlen durch die schiefe Lage der LWLs beeinsflusst (überschreitung der Grenzwinkel für die totale Reflexion) und somit entstehen unerwünschte Transmissionsverluste. Außerdem können die Lichtstrahlen aufgrund des schiefen Einbaus vor dem empfindlichen Bereich des Empfängers nicht vollständig aufgenommen ausgewertet werden. **Rechts: b)** optimierte Version 2 mit eingepasster Faser/Stecker-Verbindung. Die Fasern wurden mit Hilfe eines Isolierschlauch zentriert und in den Stecker eingebaut. Die in der ersten Version entstandenen Störeffekte konnten damit beseitigt werden.

[3] *ist ein standarisiertes LWL- Anschlusssystem, entwickelt von Toshiba. Wird grundsätzlich in der Übertragung von optischen Signalen in der Audiotechnik benutzt.*

Die POF-Fasern wurden mit Hilfe eines Zweikomponentenklebers mit dem TOSLINK-Stecker verbunden. Es wurde bei einigen Exemplaren der hergestellten Sonden festgestellt, dass der benutzte Kleber die Fasern chemisch angreift. An den Stellen, wo der Klebstoff massiv verwendet wurde, wurden beachtliche Transmissionsverluste optisch beobachtet (Licht ist aus der Faser herausgetreten). Die beschädigte Fasern wurden ausgetauscht und ein neuer Klebstoff, der chemisch neutral ist, wurde für die Verbindung der Stecker eingesetzt.

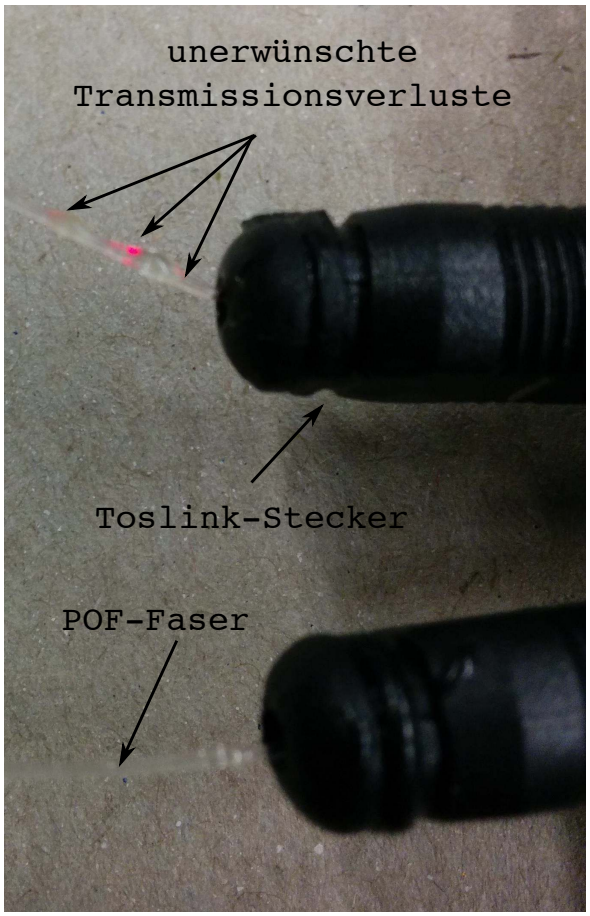


Abbildung 3.5.: Einfluss des Klebstoffs auf die POF-Fasern: Die Faser werden durch das übermäßige Auftragen des Klebstoff verätzt. Die optischen Eigenschaften (Brechungsindizes) werden dadurch verändert. Licht bricht aus dem Faser heraus.

Zusammenfassung und Darstellung der bisher entwickelten Sensorsonden:

Anschließend, werden die im Rahmen dieser Arbeit und die nach [9] entwickelten Sensorsonden zwecks der optischen Ladezustandsbestimmung von Bleibatterien dargestellt. Die relevanten Merkmale, die mechanischen Eigenschaften und die technische Daten werden gegenübergestellt.

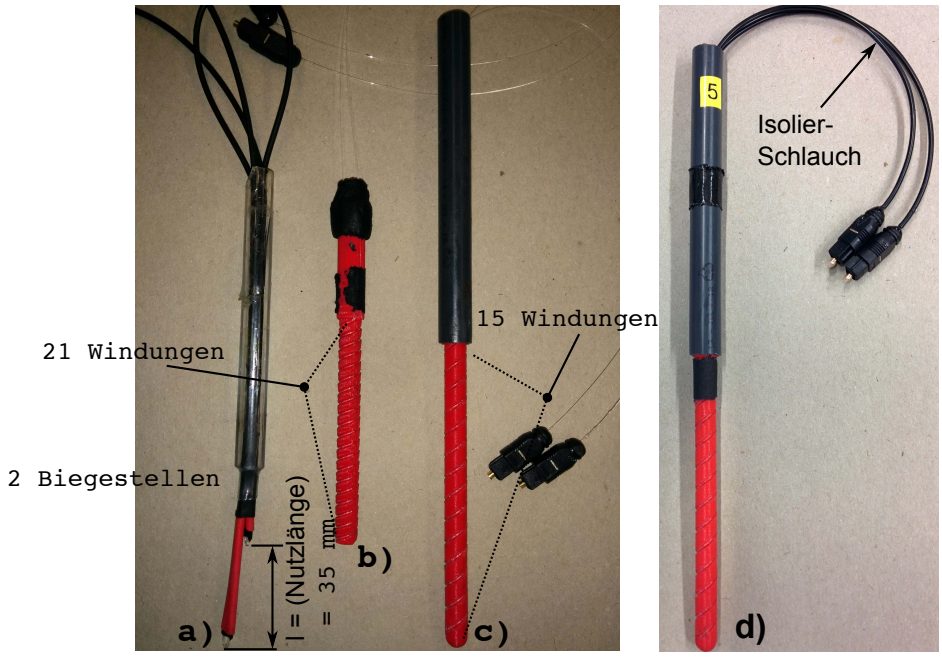


Abbildung 3.6.: Gegenüberstellung der entwickelten Sensorsonden: **a)** Die in der Arbeit von Wahid Nasimzada entwickelte Sensorsonde: Diese Sonde wurde manuell hergestellt. Zwei Messpunkte wurden durch zwei LWL-Biegungen im Abstand von 35 mm eingebaut. **b)** Der erste Prototyp, hergestellt mittels 3D-Druckers mit Kontruktionsmängeln: hohe Anzahl der Windungen, kleine Biegerradien. **c)** optimierte Version des erstens Prototypen : Die Anzahl der Windungen wurde reduziert, der Biegeradius wurde vergrößert und die Gesamt- und Nutzlänge wurden erweitert. **d)** ist die in dieser Arbeit entwickelte Endversion der Sensorsonden. Im Vergleich zu Version **c)** wurden die POF-Fasern mit einem Isolierschlauch ummantelt. Dieser dient als Schutz der Fasern gegen Verätzung durch Kontakt mit dem Klebstoff (siehe Abb. 3.5) und verhindert gleichzeitig die Bildung von unerwünschten Krümmungen (siehe Abb. 4.10)

In der folgenden Tabelle 3.1 werden die technischen Charakteristika der entwickelten Sensorsonden dargestellt.

Sensorsonde	Gesamtlänge [mm]	Nutzlänge [mm]	Windungs-winkel [grad]	Biegeradius [mm]	Anzahl Windungen (Biege-stellen)
a)	167	35	-	2	2
b)	130	80	20	4,26	21
c)	228	108	45	5,66	15

Tabelle 3.1.: Überblick der technischen Daten der hergestellten Sensorsonden im Rahmen des BATSEN-Projektes. Die Berechnung der Biegeradien für die Sonden a) und b) erfolgt mittels der Anwendung der Formel 3.3

Anhand der Abb. 3.6 und der Tabelle. 3.1 sind die Unterschiede zwischen den hergestellten Sensorsonden und die durchgeführten Optimierungen zu erkennen. Die in dieser Arbeit optimierte und eingesetzte Sonde weist durch den 3D-Fertigungsprozess keine mechanischen Streuungen auf. Ein stabiler Einbau in der Batteriezelle wird ebenso erreicht. Die Anpassung der Nutzlänge, Anzahl der Windungen und insbesondere der Biegeradien ist ausschlaggebend für eine genauere Erfassung der Dichteänderung des Elektrolyten.

3.0.5. Aufbereitung der Bleibatterie

Zwecks der Untersuchung und Erprobung der Sensoren, wurde eine herkömmliche $12\,\mathrm{V}$ Auto-Starterbatterie der Marke Panther mit einer Nennkapazität von $100\,\mathrm{Ah}$ bereitgestellt. Diese wurde im trockenen Zustand beschafft, um die Weiterverarbeitung zu ermöglichen. Zusätzlich war die Bauform der Batterie ein wichtiges Auswahlkriterium, da gewisse Modifizierung an den einzelnen Batteriezellen günstig gestattet werden können.

Die Kontaktierung der entwickelten Sensorik mit den Elektroden der Batteriezellen wurde mit Hilfe von Bleizungen realisiert. Zuerst wurde durch aufgebohrte Löcher auf der Oberseite der Batterie im Niveau der Plattenträger Raum zum Einbringen der Bleizungen geschafft. Diese wurden auf den Plattenträgern entsprechend der Elektrodeneinordnung der jeweiligen Zellen angelötet. Dabei wurde zuerst ein kleines Stück Blei auf den Trägern vorgelötet, um die beim Lötvorgang wärmeaufnehmende Kontaktfläche zu minimieren. Anschließend wurden die verlöteten Bleizungen an die Oberfläche Herausgeführt. Die Bohrungen wurden mittels

präparierter Kunststoffverschlüsse verschlossen und mit einem Silikonkleber abgedichtet, um das Entweichen der Säure außerhalb der Batterie zu verhindern.

Für das Hineinführen der Sensorsonden in den Batteriezellen wurden weitere Löcher aufgebohrt. Diese wurden an den Ecken zwischen Zelltrennwand und äußerer Batteriewand so positioniert, dass genügend Abstand von den Zellplatten gewährleistet wird (siehe Abb. 3.8). Diese Maßnahme dient zur Minimierung der Störeffekte, verursacht durch die Gasblasenbildung, die besonders in der Nähe der Zellplatten während des Ladevorgangs auftreten. Eine Beeinträchtigung der Messdaten aufgrund dieses Phänomens wurde in der Arbeit von Wahid Nasimzada [9] beobachtet.

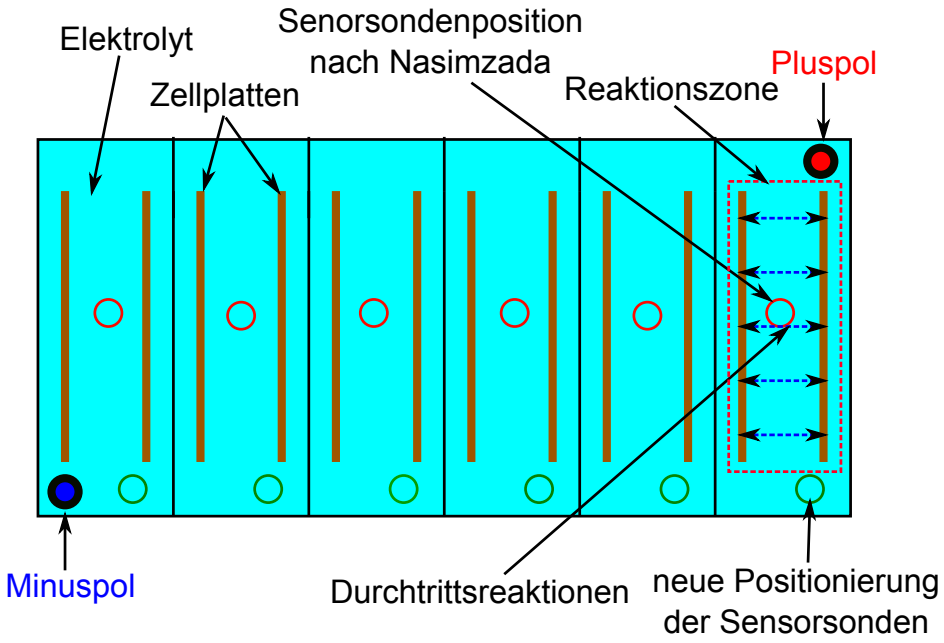


Abbildung 3.7.: Schematische Darstellung der Sensorsonden-Positionierung in der präparierten Bleibatterie zur Minimierung der Störeffekte während der optischen Messung, verursacht durch Blasenbildung: **rot** markiert sind die Stellen, wo die alten Sensorsonden nach [9] in die Batteriezellen integriert wurden. Die Sensorsonden wurden zwischen den Zellplatten innerhalb der Reaktionszone positioniert. Störeinflüsse in der optischen Untersuchung, ausgelöst durch Blasenbildung im Verlauf der Durchtrittsreaktionen, konnten festgestellt werden. **Grün** markiert sind die neuen Positionierungen der Sensorsonden. Diese wurden am Rand des Batteriegehäuse außerhalb der Reaktionszone integriert, so dass die Blasenbildung die optische Messung nicht beeinflusst.

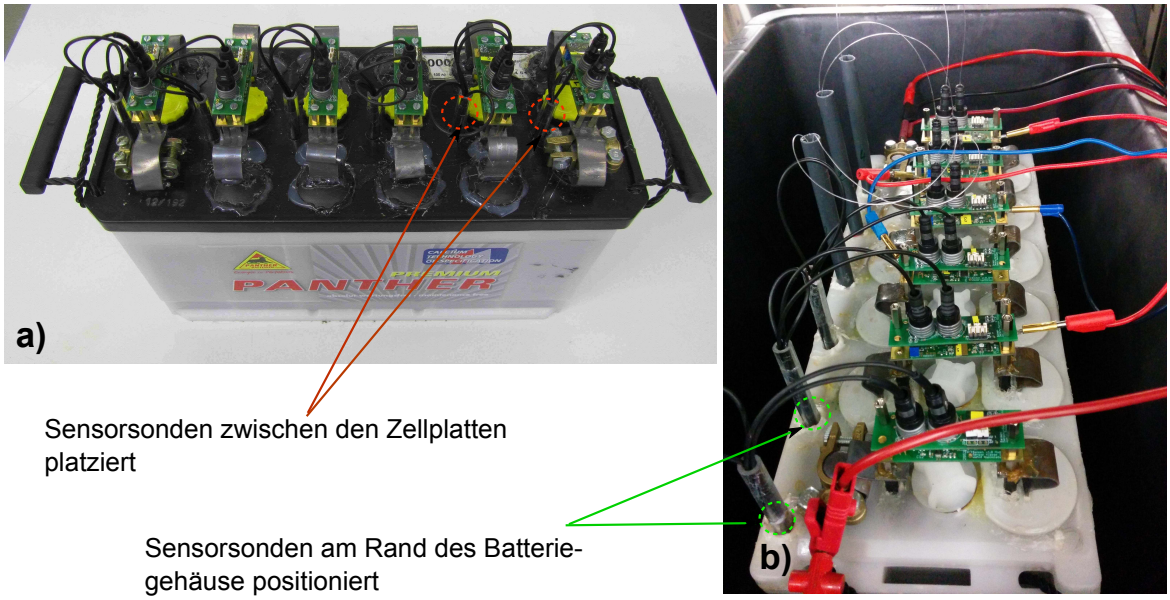


Abbildung 3.8.: Verschiebung der Sensorsonden-Positionierung in der präparierten Bleibatterie zur Minimierung der Störeffekte während der optischen Messung, verursacht durch Blasenbildung: **a)** aufbereitete Batterie nach [9]: die Sensorsonden wurden dicht an den Zellplatten montiert. Störeinflüsse in der optischen Untersuchung ausgelöst durch Blasenbildung konnten festgestellt werden. **b)** Die in dieser Arbeit präparierte Bleibatterie, die Sensorsonden wurden am Rand des Batteriegehäuse integriert, so dass die optische Messung durch die Blasenbildung nicht verfälscht werden kann und gleichzeitig die Erfassung der Dichteänderung des Elektrolyten gewährleistet werden kann.

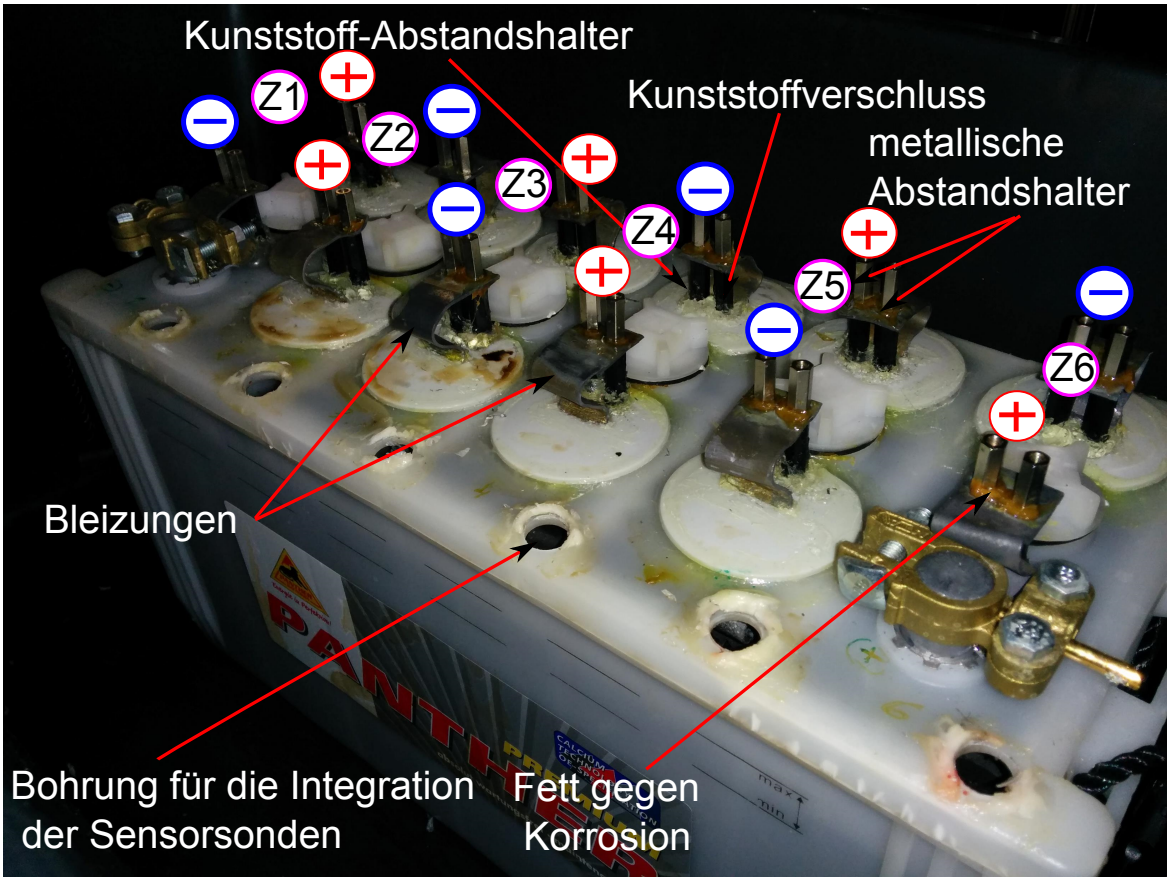


Abbildung 3.9.: Modifizierte Starterbatterie für den Einsatz der Sensorik. Die Kontaktierung zu den einzelnen Zellenelektroden wurde mit Hilfe der Bleizungen durchgeführt. Diese eignen sich gut für diesen Zweck aufgrund der leichten Verformbarkeit, der chemischen Neutralität mit der Säure und der elektrischen Leitfähigkeit. Die Kunsstoff-Abstandshalter dienen zur Stabilisierung der Bleizungen. Für die Befestigung der Zellensensoren wurde eine Vorrichtung mittels metallischer Abstandshalter an den Bleizungen festgeschraubt und deren Kontaktfläche mit Korrosionsschutz-Fett eingeschmiert, um die Leitfähigkeit zu verbessern. Der Bohrungsdurchmesser für die Sensorsonden wurde so angepasst, dass eine mechanische Stabilität erzielt werden kann.

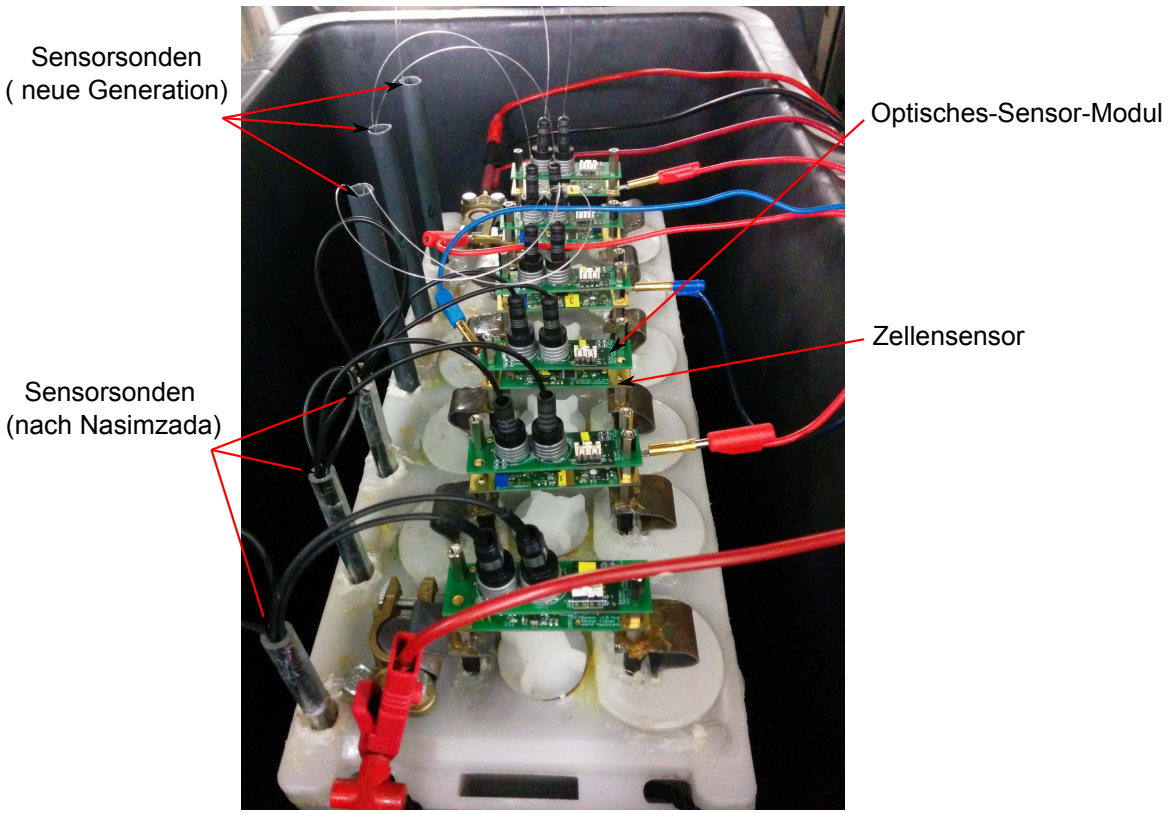


Abbildung 3.10.: Präparierte Batterie mit aufmontierter Sensorik im Zyklierbetrieb. Die Sensorsonden sind in den realisierten Bohrungen in jeder Zelle hineingeführt. Die Faserenden werden mittels eines TOSLINK-Steckeranschlusses mit dem optischen Modul verbunden.

In der Abb. 3.10 ist eine modifizierte Auto-Starterbatterie mit der aufmontierten Sensorik im Zyklierbetrieb dargestellt. Es wurden zwecks einer Vergleichsmessung zwei Sensorsonden-Versionen eingesetzt. Die neue Generation ist die mittels eines 3D-Druckers hergestellt und optimiert worden. Die Faserenden der Sensorsonden werden respektive senderseitig (LED) und mit dem optischen Empfänger des Optischen-Sensor-Moduls per TOSLINK-Anschlusssystem verbunden. Der Zellensensor wird bei der Montage zuerst auf der mit den Bleizungen angefertigten Vorrichtung verschraubt. Das optische Modul wird anschließend modular auf den Zellensensor aufgesteckt und befestigt.

4. Minimierung von Störeinflüssen

In diesem Kapitel werden die Störeinflüsse, welche die Genauigkeit der Messungen anhand der Sensoren beeinträchtigen, systematisch identifiziert und anschließend analysiert. Diese werden mittels angepassten Messverfahren und durch die Anwendung von Kalibrierverfahren unterdrückt.

4.1. Temperatureinfluss der Sende-LED

Die relative Lichtstärke der Sende-LED, die im optimierten Dichte-Sensor-Modul eingebaut wurde, ist temperaturabhängig. Diese Abhängigkeit wird in der folgenden Abb. 4.1, die aus dem Datenblatt entnommen wurde, dargestellt.

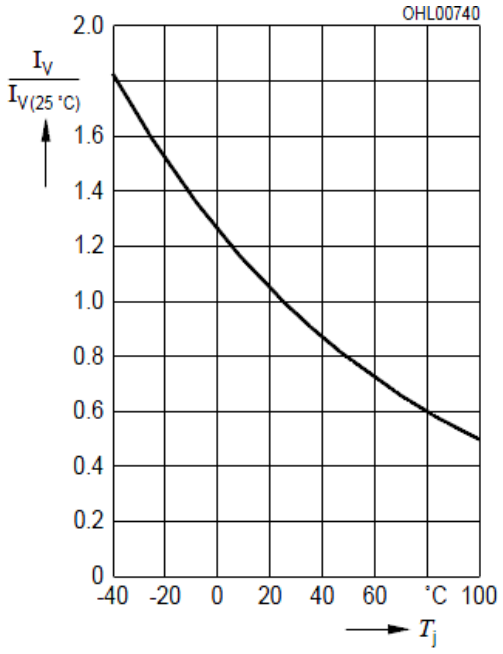


Abbildung 4.1.: Relative Lichtstärke der Sende-LED (Top-LED LSYT67B von OSRAM) in Abhängigkeit von der Temperatur [31].

Anhand der dargestellten Kennlinie in Abb. 4.1 ist ersichtlich, dass die Lichtintensität bei steigender Betriebstemperatur abklingt. Eine Wärmeentwicklung innerhalb des LED-Gehäuses, verursacht durch lange Einschaltzeiten der LEDs, kann demzufolge die optischen Messergebnisse stark verfälschen. Um das Auftreten dieses Verhaltens zu unterbinden, wurde während der Messung für lange Abkühlzeiten gesorgt. Diese wurde zuerst anhand des gepulsten Betriebes mit gleichgewichteter Einschalt-/ und Pausendauer ($t_{an} = t_{aus} = 500ms$) je Messung für eine LED realisiert. Des Weiteren wurde durch den Wechselmessbetrieb zwischen den beiden LEDs (Siehe 5.3) zusätzliche Abkühlzeit (3 s) pro Messdurchlauf geschaffen.

Für einen Messbetrieb der Sensorik im Temperaturschrank bei extrem hohen und tiefen eingestellten Temperaturen muss die Lichtintensitätsänderung dementsprechend bei der Auswertung der optischen Messwerte berücksichtigt werden. Diese wurde im Rahmen dieser Arbeit nicht betrachtet, da die Messungen an der Bleibatterie bei Raumtemperatur durchgeführt wurden.

4.2. Fremdlichtunterdrückung

Das in dieser Arbeit angewandte optisches Messverfahren für die Bestimmung des Ladezustandes einer Bleibatterie, beruht auf der Erfassung der Transmissionsverluste, die sich proportional zu der Elektrolytendichte (bzw. Ladezustand) ändern. Da das optische Messsystem sehr empfindlich auf Änderung der Lichtverhältnisse reagiert, werden fremde Lichtquellen in der Umgebung des Messaufbaus wahrgenommen und mit dem Nutzsignal überlagert. Diese Überlagerung führen zu unerwünschten Verfälschungen der Messergebnisse, da eine Veränderung des Umgebungslichts als eine Dichteänderung der Säure interpretiert werden könnte.
Dieses Verhalten wurde in der Arbeit von Wahid Nasimzada in den ersten Erprobungen beobachtet. Dabei wurde beim Übergang vom Nacht- in den Tages-Messbetrieb eine atypische Erhöhung des Ausgangssignals, die nicht auf eine Dichteänderung zurückführbar ist, festgestellt. Eine Veränderung des Ausgangssignals, entsprechend dem Ein-/Ausschalten der Laborbeleuchtung wurde ebenso beobachtet. Als Gegenmaßnahme wurde der komplette Messaufbau für weitere Erprobungen mit Hilfe eines Kartons abgeschirmt. So konnte verhindert werden, dass Messergebnisse durch das Umgebungslicht beeinflusst werden. Jedoch entspricht diese improvisierte Lösung nicht den realen Betriebsbedingungen der Sensoren und muss aus diesen Grund dringend optimiert werden.
Prinzipiell ist die Fremdlichtunterdrückung bei den optischen Messeinrichtungen, wo optoelektronische Komponenten eingesetzt werden (wie z. B. in Lichtschranken, spektrale Untersuchungen von chemischen und physikalischen Eigenschaften von Materien, Bildsensorik...) von ausschlaggebender Bedeutung. Dabei werden unterschiedliche Verfahren und Techniken für das Abtrennen der Störsignale (verursacht durch Fremdlichtquellen) verwendet.

Überwiegend werden für diesen Zweck optische Farbfilter und Modulation mit Bandpassfilter eingesetzt.

4.2.1. Differenzmessung

Entsprechend der vorhandenen Hardware, dem Softwareumfang und den zu erfüllenden Anforderungen und Randbedingungen (Umgebungslichtverhältnisse im Labor), eignete sich das Verfahren der Differenzmessung zur Beseitigung von Fremdlichteinflüssen, die das Ausgangssignal verfälschen.
Dieses Verfahrens besteht darin, zwei Messungen durchzuführen und anhand einer Subtraktion, die dem Nutzsignal überlagerten Störanteile (Fremdlichtanteile) zu eliminieren. Die Herangehensweise besteht aus drei Hauptschritten, die im Folgenden beschrieben werden:

1. Erste Messung: Hierbei wird zuerst eine Messung mit eingeschalteter LED durchgeführt. Die Messdauer beträgt $t_{AN} = 500\,\text{ms}$. Das Ausgangssignal beinhaltet das Nutzsignal (LED-Lichtanteile) und Fremdlichtanteile.
2. Zweite Messung: Die LED wird ausgeschaltet und im Laufe einer Zeit von $t_{AUS} = 500\,\text{ms}$ gemessen. Das erfasste Ausgangswert entspricht lediglich dem Umgebungslicht.
3. Abschließend wird der bei der zweiten Messung erhaltene Ausgangswert von dem Messergebnis der ersten Messung subtrahiert. Als Resultat entsteht nur das reine Nutzsignal, des schließlich codiert und an die Basisstation übermittelt wird.

Die erfolgreiche Umsetzung dieses Verfahrens ist für niederfrequente Störsignale gewährleistet. Die Grenzfrequenz für die möglich erfassten Störsignale (Umgebungslichtänderung) lässt sich anhand der folgenden Gleichungen definieren:

$$T = t_{AN} + t_{AUS} = 500\,ms + 500\,ms = 1\,s \tag{4.1}$$

$$f = \frac{1}{T} = \frac{1}{1\,s} = 1\,Hz \tag{4.2}$$

Wendet man hier das Abtasttheorem an, so ergibt sich für die Grenzfrequenz:

$$f_g = 0{,}5\,Hz \tag{4.3}$$

Anhand der Gleichung 4.3 ist ersichtlich, dass durch die Differenzmessung nur optische Störsignale bis zu einer Grenzfrequenz von 0,5 Hz hinreichend ausgefiltert werden. Gleichlicht und sporadische Änderung der Raumbeleuchtung lassen sich hiermit erfolgreich unterdrücken.

4.2.1.1. Vergleichsmessung

Für die Erprobung des neuen Verfahrens der Differenzmessung wurden auf einem Testaufbau sechs Sensoren montiert. Unterschiedliche Softwareversionen (alte Software nach Nasimzada, bzw. neue Software mit dem implementierten Verfahren für die Fremdlichtunterdrückung) wurden zwecks einer Vergleichsmessung auf die Sensoren programmiert.
Auf den Sensoren S 1, S 2 und S 3 läuft die optimierte Softwareversion. Der Rest der Sensoren S 5, S 6 und S 7 wird mit der alten Softwareversion nach Nasimzada betrieben. Für die Spannungsversorgung sorgt eine Spannungsquelle von Rhode und Schwarz, die auf 2 V eingestellt wurde.

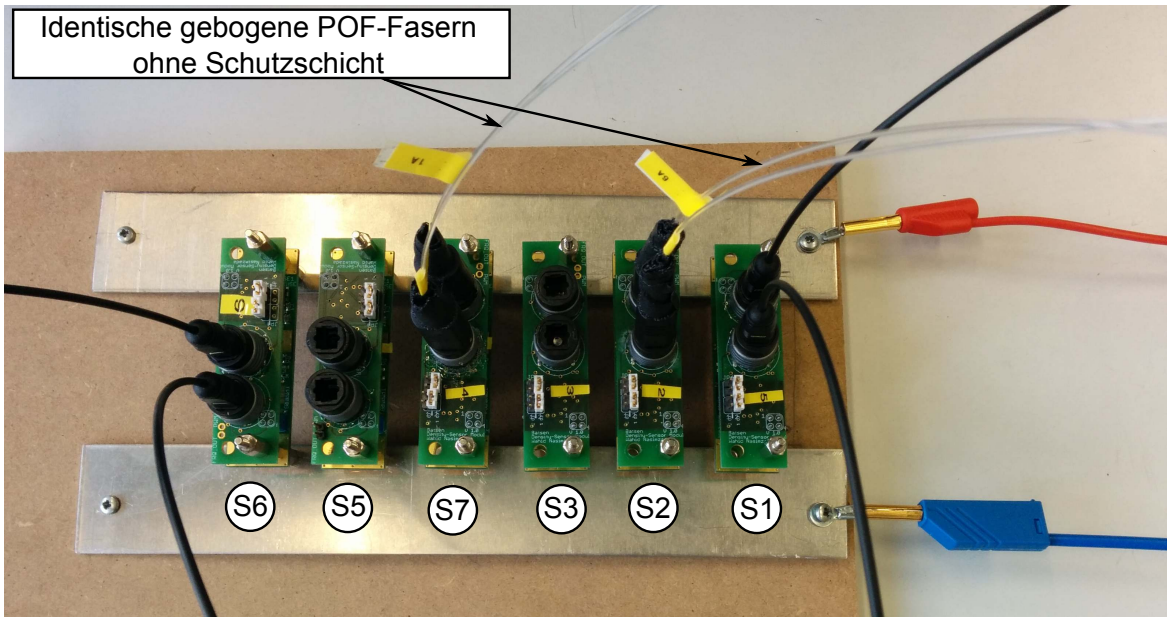


Abbildung 4.2.: Testaufbau für die praktische Untersuchung des implementierten Verfahrens der Differenzmessung zwecks Fremdlichtunterdrückung. Die Sensoren S 2 und S 7 wurden mit zwei identischen gebogenen POF-Fasern (ohne Schutzschicht) eingekoppelt, S 3 und S 5 wurden ohne Faserverbindung betrieben und S 1 und S 6 wurden mit unbearbeiteten Fasern verbunden.

Während der Messung wurden die Sensoren in unterschiedlichen Zeitabständen mit einer externen Lichtquelle eingestrahlt. Die aufgenommenen Ausgangssignale mittels der Matlab-Auswertesoftware werden in der folgenden Abb. 4.3 dargestellt.

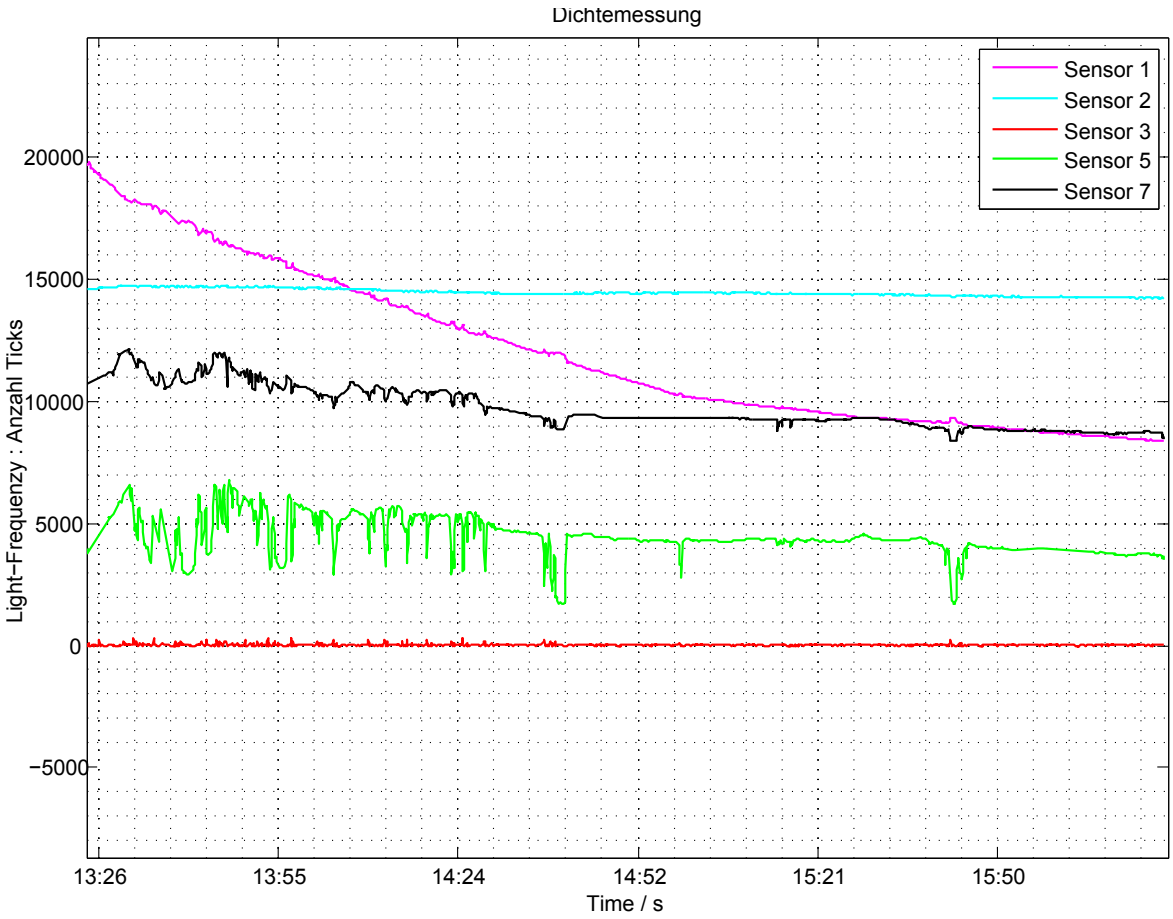


Abbildung 4.3.: Auswertung der praktischen Untersuchung des implementierten Verfahrens der Differenzmessung. Am **Sensor 2** (Hellblau, neue Software und Einkopplung mit der gebogenen POF-Faser) werden die optischen Störsignale ausgefiltert. Darüber hinaus verhält sich **Sensor 3** (Rot, neue Software und ohne LWL-Verbindung) wie erwartet. Das Ausgangssignal ist nahezu gleich Null, da der Sensor ständig nur das Umgebungslicht erfasst und es ausfiltert. Bei **Sensor 5 und 7** (Grün , Schwarz und mit nicht optimierter Softwareversion) ist der Einfluss der Fremdlichtquelle deutlich zu erkennen. Für **Sensor 6** sind leider keine Aufnahmen des Ausgangssignals entstanden, da die Sensoradresse von der Empfangsstation nicht erkannt werden konnte. Bei **Sensor 1** (Pink) hat sich der TOSLINK-Stecker während der Messung von der Buchse an der Empfangsseite langsam abgelöst. Dies erklärt den fallenden Verlauf dessen Ausgangssignals. Aus den eben erwähnten Gründen können keine Aussagen bezüglich der Fremdlichtunterdrückung für die beiden Sensoren (S 1 und S 6) getroffen werden

Anhand der Abb. 4.3 ist es deutlich zu erkennen, dass durch das Differenzmessungsverfahren die Fremdlichtanteile erfolgreich ausgefiltert werden. Die Vergleichsmessung gegenüber der nicht-optimierten Software bestätigt die Effizienz des Verfahrens für niederfrequente Störsignale. Weitere Untersuchungen für hochfrequente modulierte Fremdlichtquellen wurde in dieser Messung nicht weiter verfolgt. Die Begründungen dafür werden im folgenden Abschnitt 4.2.2 dargestellt.

4.2.2. Modulation als alternatives Verfahren

Eine geeignete alternative Lösung für die Unterdrückung des Fremdlichtes wäre die Modulation. Bei diesem Verfahren wird grundsätzlich die Lichtquelle (z. B. LED) mit einer bestimmten Frequenz (Trägerfrequenz) moduliert oder gepulst. Mittels einer abgestimmten Bandpass-Filtervorrichtung werden Frequenzen oberhalb und unterhalb der Trägerfrequenz an der Empfängerseite herausgefiltert. Dementsprechend wird das Nutzsignal von Störungen, verursacht durch das Umgebungslicht, nahezu vollständig befreit.
Eine typische Anwendung dieses Verfahrens bei einer optischen Signalübertragung ist die IR[1]-Fernbedienung. Dabei wird die IR-LED gewöhnlich mit einer Frequenz zwischen $30\,\mathrm{kHz}$ und $60\,\mathrm{kHz}$ moduliert. Auf der Empfängerseite wird mittels eines optischen IR-Farbfilters Lichtquellen anderer Wellenlängen herausgefiltert und durch das Bandbassfilter das Nutzsignal von den Gleichlichtanteilen und Störanteilen aus modulierten Umgebungslichtquellen (z. B. Glühlampe mit $50\,\mathrm{Hz}$) befreit.
Eine andere effiziente, aber aufwändige Variante für die Unterdrückung von optischen Störsignalen ist das Lock-in-Verfahren (Lock-in-Verstärker). Die Anwendung dieses Verfahrens findet auf der Empfängerseite statt. Anhand der nächsten Abb. 4.4 wird das Funktionsprinzip dieses Verstärkers erörtert.

[1] *Infrarot*

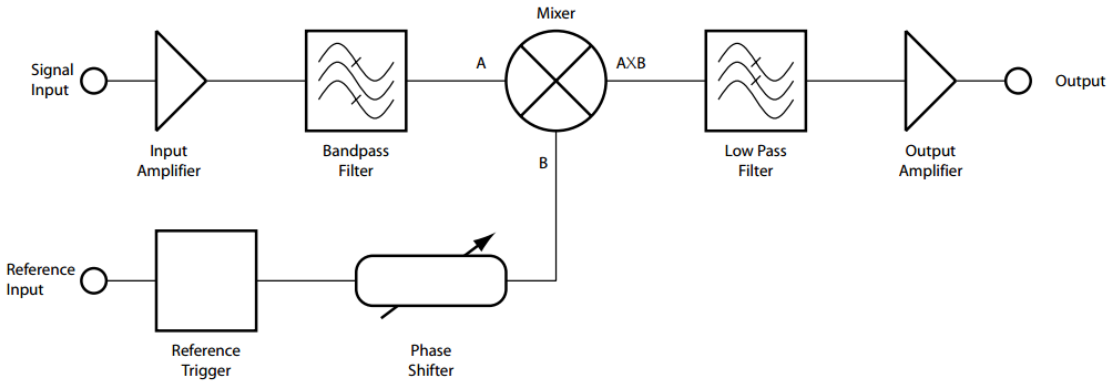


Abbildung 4.4.: Blockschaltbild eines Lock-in-Verstärkers [32]

Anhand der Abb. 4.4, wird grundlegend bei diesem Verfahren das modulierte, verrauschte Nutzsignal verstärkt und mittels eines Bandpassfilters von Rauschanteilen oberhalb und unterhalb der Trägerfrequenz befreit. Das durch einer Regelschleife und den Phasenschieber synchronisierte Referenzsignal wird in einem Mischer mit dem verstärkten Nutzsignal multipliziert. Anschließend wird das Mischsignal durch das Tiefpassfilter über mehrere Perioden der Modulationsfrequenz integriert. Hiermit werden Anteile des Rauschens und Störsignale, die nicht mit der Modulationsfrequenz synchronisiert sind, ausgefiltert [32].
Diese Verfahren, zwecks der Unterdrückung des Umgebungslichtes, wurde in dieser Arbeit nicht weiter verfolgt. Grund hierfür ist der Hardware-Aufwand und die Softwarekomplexität für die Umsetzung solcher Verfahren, die mit einem hohen Rechenaufwand verbunden ist, was der verwendete Mikrocontroller MSP430G2553IPW20 nicht leisten kann.

Anhand der folgenden Tabelle 4.1 werden die erwähnten Verfahren für die Unterdrückung des Fremdlichtes mit der, in dieser Arbeit eingesetzten Lösung, gegenübergestellt.

Verfahren	Hardware-Aufwand	Software-Aufwand	Signalqualiät
Differenz-Messung	+ Keine Hardwareänderung nötig	+ Niedriger Rechenaufwand (keine Multiplikation)	- Gilt nur für sehr langsame Störsignale (Umgebungslicht-änderungen) bis zu 0,5 Hz
Modulation (IR-Signal)	- IR-LED (senderseitig) - IR-Farbfilter (empfängerseitig) - Analoges Bandpassfilter (empfängerseitig)	- Die analoge Filterschaltung kann mit digitalen Filtern realisiert werden, jedoch mit Rechenaufwand verbunden	+ Störsignale niedriger und höher als die Modulationsfrequenz werden komplett ausgefiltert.
Modulation mit Lock-in Verstärker	- Verstärkerschaltug (empfängerseitig)	- komplexe Software-Realisierung	+ Rauschsignale werden vollständig ausgefiltert + Störsignale, die dieselbe Frequenz wie die Trägerfrequenz haben, werden auch durch die Phasenverschiebung und das Tiefpassfilter ausgefiltert.

Tabelle 4.1.: Gegenüberstellung der möglichen Verfahren für die Umgebungslichtunterdrückung mit Vor- und Nachteilen. Die Vorteile sind mit (+) und die Nachteile mit (-) gekennzeichnet

Anhand der Tabelle 4.1 ist ersichtlich, dass die alternativen Lösungen für die Umgebungslichtunterdrückung mit viel Hard- und Softwareaufwand gekoppelt sind. Anderseits kann mittels dieser Verfahren eine effiziente Unterdrückung von hochfrequenten Störsignalen im Gegensatz zu der verwendeten Lösung erzielt werden.
Die in dieser Arbeit eingesetzte Vorgehensweise mit der Differenzmessung ist einfach umsetzbar. Die Softwareimplementierung lässt sich ohne hohen Rechenaufwand umsetzen. Eine Hardwareänderung ist nicht erforderlich. Durch dieses Verfahren können auftretende Änderungen der Umgebungslichtverhältnisse im Versuchslabor während des Messbetriebs, wie z. B. das Ein-/ Ausschalten der Raumbeleuchtung und der Übergang vom Nacht- ins Tagesbetrieb, erfasst und hinreichend unterdrückt werden. Ein Nachteil dieses Verfahrens ist, dass hochfrequente optische Störsignale (hoch gepulste Lichtquellen in unmittelbaren Nähe des Messaufbaus) nicht vollständig ausgefiltert werden können.

4.3. Kalibrierung

In diesem Kapitel wird die durchgeführte Kalibrierung der Zellensensoren bezüglich der Temperatur- und Spannungsmessung erläutert. Die Messwerterfassung der Temperatur und Spannung erfolgt durch den internen ADC des Mikrocontrollers im Zellensensor. Die Messung ist jedoch mit Abweichungen behaftet. Diese Messabweichungen sind durch folgende Effekte begründet [33]:

- Systematische Messabweichungen bedingt durch Produktionstoleranzen
- Temperaturabhängigkeit der elektrischen Komponenten des Zellensensors
- DNL[2] und INL[3] der Übertragungsfunktion des ADCs
- Quantisierungsunsicherheit des 10-Bit-ADCs ($\pm$ 2 LSB)

Diese Messabweichung werden anhand eines geeigneten Kalibrierverfahrens ermittelt und kompensiert, um eine genauere Auswertbarkeit der Messdaten zu erzielen.

Da die Messdaten durch das Steuergerät manipuliert werden [34] (die Messdaten der Temperatur- und Spannungsmessung werden durch nicht mehr gültige Kalibrationsparametern verfälscht), wurde die Implementierung des Kalibrierverfahrens am PC anhand eines MATALB-Skripts und nicht in der Zellensensorsoftware realisiert. Dadurch wird die Verfälschung der Messdaten durch das Steuergerät in der Kalibrierung berücksichtigt. Zusätzlich kann eine höhere Genauigkeit erzielt werden. Gegenüber der 64-Bit-CPU des PCs werden auf dem Mikrocontroller mit der 16-Bit-CPU mehr Rechenschritte und damit auch mehr Rechenzeit benötigt, um die gleiche Genauigkeit zu erreichen. Darüber hinaus kann mit der direkten Anwendung von Fließkomma-Arithmetik am PC eine zusätzliche Genauigkeit erfolgen, die eine aufwändige Implementierung in Software auf dem Mikrocontroller erfordern würde.

4.3.1. Kalibrierverfahren

Für die Kalibrierung der Temperatur- und Spannungsmessung wird ein lineares Ausgleichsverfahren verwendet. Dabei wird die Methode nach dem Gaußschen Prinzip der kleinsten Quadrate zur Kurvenanpassung der Messdaten eingesetzt. Diese lineare Ausgleichsgerade lässt sich anhand der folgenden Funktionsgleichung 4.4 beschreiben:

$$y = a \cdot x + b \tag{4.4}$$

[2] *Differentielle Nichtlinearität*
[3] *Integrale Nichtlinearität*

wobei:

- a: Steigung der Regressionsgeraden
- b: y-Achsenabschnitt oder Offset genannt
- x: Eingangsgröße (Sensordaten)
- y: Ausgangsgröße

Die Regressionskoeffizienten a und b werden wie folgt mathematisch bestimmt:

$$a = \frac{n \cdot \sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i) \cdot (\sum_{i=1}^{n} y_i)}{\Delta} \tag{4.5}$$

$$b = \frac{(\sum_{i=1}^{n} x_i^2) \cdot (\sum_{i=1}^{n} y_i) - (\sum_{i=1}^{n} x_i) \cdot (\sum_{i=1}^{n} x_i y_i)}{\Delta} \tag{4.6}$$

$$\Delta = n \cdot \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2 \tag{4.7}$$

Während der Kalibrierung werden für y die gemessenen Werte des Kalibriernormals eingesetzt und die Parameter berechnet. Für die Korrektur der Messwerte werden die Koeffizienten a und b auf die Sensordaten x angewendet, y stellt die korrigierten Messwerte dar.

4.3.2. Kalibriermessung

Die Kalibriermessung der Messgrößen Temperatur und Spannung, erfasst durch die Zellensensoren, wurde im Temperaturschrank durchgeführt. Die neu bestückten Zellensensoren (mit geänderter Hardware- und Softwareversion) wurden parallel geschaltet und im Temperaturschrank betrieben. Als Referenzspannungsquelle wurde das Labornetzgerät NGM 35/1 von Rhode und Schwarz verwendet. Für die Bestimmung der Spannung wurde das Fluke-Multimeter, welches eine ausreichend hohe Genauigkeit aufweist, als Kalibriernormal eingesetzt. Es wurde mithilfe des Temperaturschranks mehrere Temperaturstufen eingestellt. Bei jeder Temperaturstufe wurden mehrere Spannungen durchfahren. Der zur Kalibrierung verwendete Messplan sieht acht Temperaturstufen von $-20\,^{\circ}\mathrm{C}$ bis $50\,^{\circ}\mathrm{C}$ in $10\,^{\circ}\mathrm{C}$ Schritten vor. An dem Temperaturschrank wird nur die Lufttemperatur angezeigt, nicht jedoch die der Sensoren. Aus diesem Grund wurde ein externes Thermometer auf der Unterlage der

Sensoren angebracht, um zu überprüfen, ob die Sensoren die eingestellte Lufttemperatur erreicht haben. Für jeden Temperaturschritt wird die Spannung im Bereich von $1{,}7\,\mathrm{V}$ bis $2{,}4\,\mathrm{V}$ in $50\,\mathrm{mV}$ Spannungsschritten gemessen. Dieser Spannungsbereich entspricht dem an der Bleibatterie erwarteten Spannungsbereich.

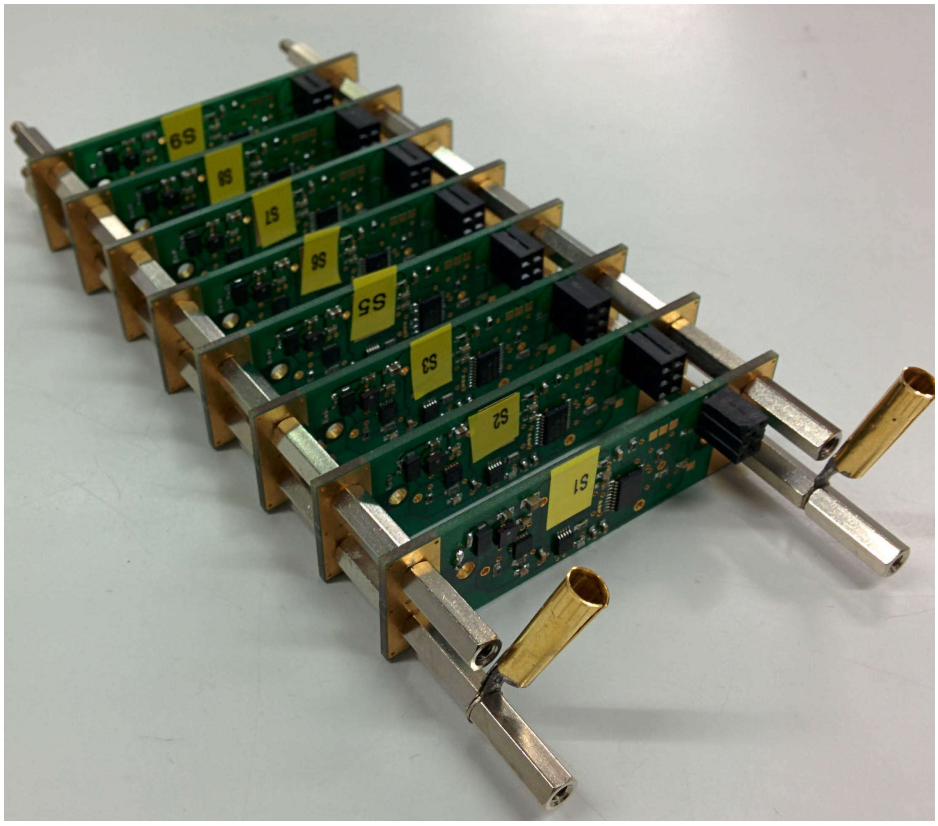


Abbildung 4.5.: Parallelschaltung der Zellensensoren für die Kalibriermessung im Temperaturschrank

4.3.3. Auswertung der Kalibriermessung

Mittels des Matlabskriptes (siehe B.2.2) werden die Kalibrierkoeffizienten berechnet. Tabelle 4.2 zeigt die mit dem Spannungsnormal bestimmten Spannungswerte.

Tabelle 4.2.: Spannungsmesswerte des Spannungsnormals (alle Werte in mV)

Spannung (ideal)	**Temperatur**							
	−20 °C	−10 °C	0 °C	10 °C	20 °C	30 °C	40 °C	50 °C
1700	1703	1700	1700	1701	1701	1701	1701	1701
1750	1753	1753	1749	1749	1751	1752	1751	1754
1800	1802	1803	1797	1797	1802	1800	1803	1799
1850	1850	1847	1853	1849	1851	1851	1846	1852
1900	1902	1904	1902	1899	1900	1905	1903	1901
1950	1952	1952	1950	1949	1950	1950	1950	1952
2000	2003	2001	2002	2000	1999	2002	2000	1999
2050	2050	2050	2049	2049	2052	2051	2052	2048
2100	2102	2101	2099	2099	2101	2100	2100	2101
2150	2150	2150	2153	2149	2150	2151	2151	2152
2200	2202	2199	2203	2204	2199	2201	2203	2200
2250	2255	2249	2247	2251	2252	2251	2252	2253
2300	2302	2303	2303	2308	2303	2301	2301	2301
2350	2352	2349	2349	2350	2351	2349	2351	2351
2400	2402	2402	2402	2397	2398	2398	2399	2398

Die Abb. 4.6 zeigt die gemessen Spannungswerte des Sensors 5 bei den unterschiedlichen Temperaturen und die dazugehörigen Ausgleichsgeraden. Die lineare Abhängigkeit ist, wie erwartet, vorhanden.

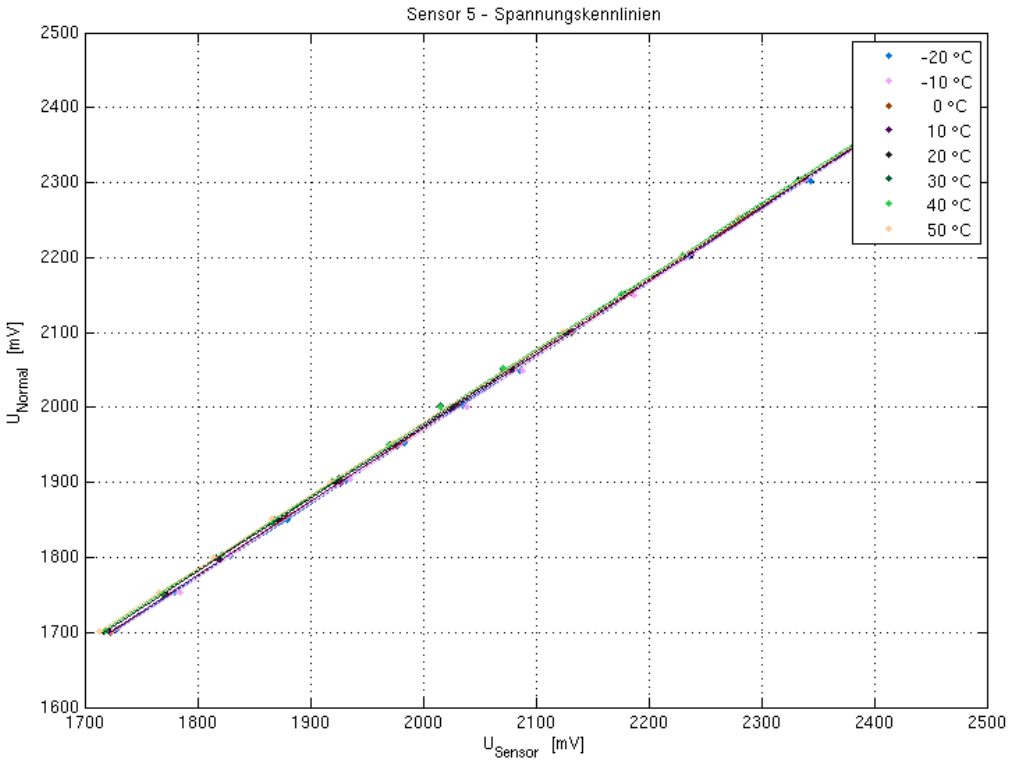


Abbildung 4.6.: Spannungskennlinien Sensor 5

Die gemessenene Temperaturwerte von Sensor 5 sind in der Abb. 4.7 dargestellt. Hier ist ebenfalls der erwartete lineare Zusammenhang erkennbar.

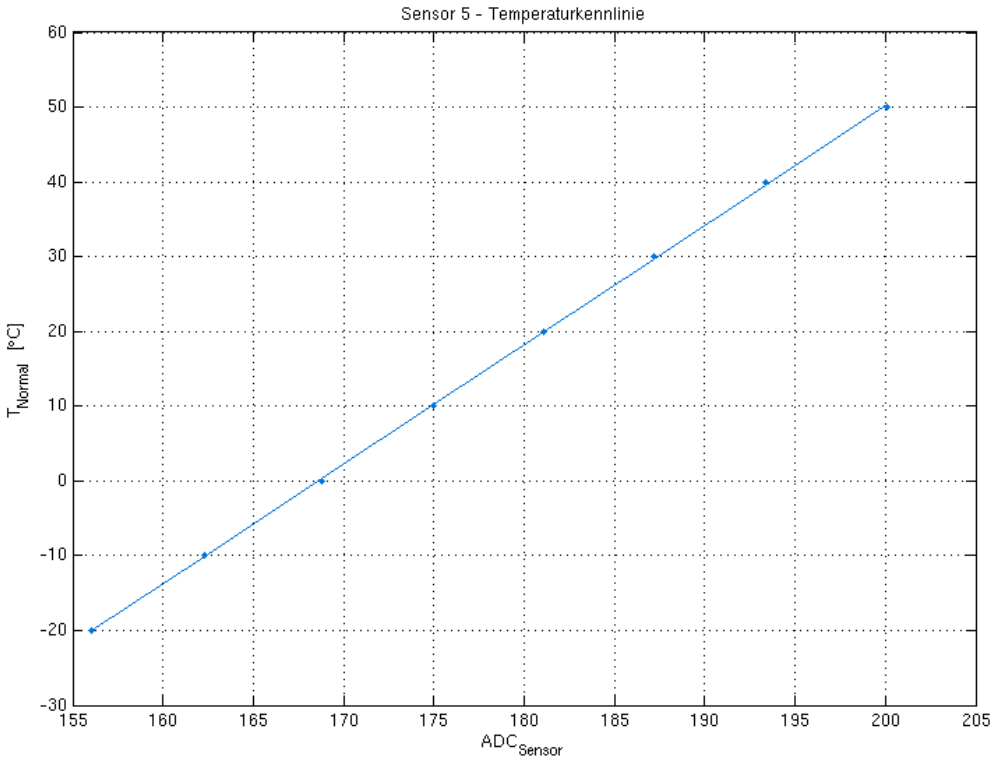


Abbildung 4.7.: Temperaturkennlinie Sensor 5

Die Spannungs- und Temperaturkennlinien sowie die dazugehörigen grafischen Darstellungen der Residuen aller untersuchten Zellensensoren sind im Anhang D einsehbar. Die Tabelle 4.3 zeigt die berechneten Regressionsparameter für die Temperaturkennlinien. Die Regressionskoeffizienten für die Spannungskennlinien sind in der Tabelle 4.3 dargestellt.

Tabelle 4.3.: Koeffizienten der Temperaturkennlinien

Sensor	Steigung [°C]	Achsenabschnitt [°C]
2	1.63179	-274.349
3	1.67785	-286.108
5	1.60204	-270.110
6	1.61932	-267.373
7	1.68682	-290.713
9	1.62014	-270.260

Tabelle 4.4.: Koeffizienten der Spannungskennlinien

	Temperatur	Sensor 2	3	5	6	7	9
Steigung	−20 °C	1.00459	1.00866	0.98287	0.97179	1.01688	0.99412
	−10 °C	1.00490	1.01057	0.98213	0.97663	1.01451	1.00187
	0 °C	0.97976	1.00646	0.98325	0.98234	1.02010	1.00087
	10 °C	0.99523	1.00588	0.98008	0.97505	1.01872	0.99696
	20 °C	0.99370	0.99537	0.97528	0.96856	1.01027	0.98904
	30 °C	0.98764	0.98988	0.97215	0.96531	1.00472	0.98524
	40 °C	0.99173	1.01628	0.97675	0.96895	1.00868	0.98730
	50 °C	0.98577	0.98756	0.96843	0.96127	0.99834	0.97918
Achsenabschnitt	−20 °C	-11.38757	6.79089	3.70272	27.46262	10.66583	11.25508
	−10 °C	-13.10905	2.02724	4.70387	15.85366	13.90642	-4.79746
	0 °C	31.68395	10.57727	5.90409	6.74568	3.89594	-0.88748
	10 °C	6.12466	9.29728	12.26474	19.36401	4.52960	4.88326
	20 °C	6.21931	29.51296	25.59601	33.19553	20.37406	20.94361
	30 °C	20.12882	41.79416	34.24985	41.93176	33.04284	30.91715
	40 °C	10.43120	-9.39134	25.68422	33.20471	23.59850	24.74150
	50 °C	22.28151	45.91047	42.45294	50.14177	44.81030	43.35145

Für jede Temperatur existiert eine Spannungskennlinie. Nachfolgend soll die Abhängigkeit der Regressionskoeffizienten von der Temperatur untersucht werden. Für die Steigung ist bei allen untersuchten Sensoren ein linearer Zusammenhang erkennbar. Die Abb. 4.8 zeigt dieses Verhalten exemplarisch für Sensor 5. Für den Achsenabschnitt ist nicht bei jedem Sensor eine Linearität zu beobachten. Die Abb. 4.9 zeigt die Koeffizienten des Achsenabschnittes für Sensor 9 gegenüber der Temperatur. Die grafische Auswertung der Koeffizien-

ten für alle kalibrierten Sensoren sowie die dazugehörigen Residuen sind aus dem Anhang D entnehmbar.

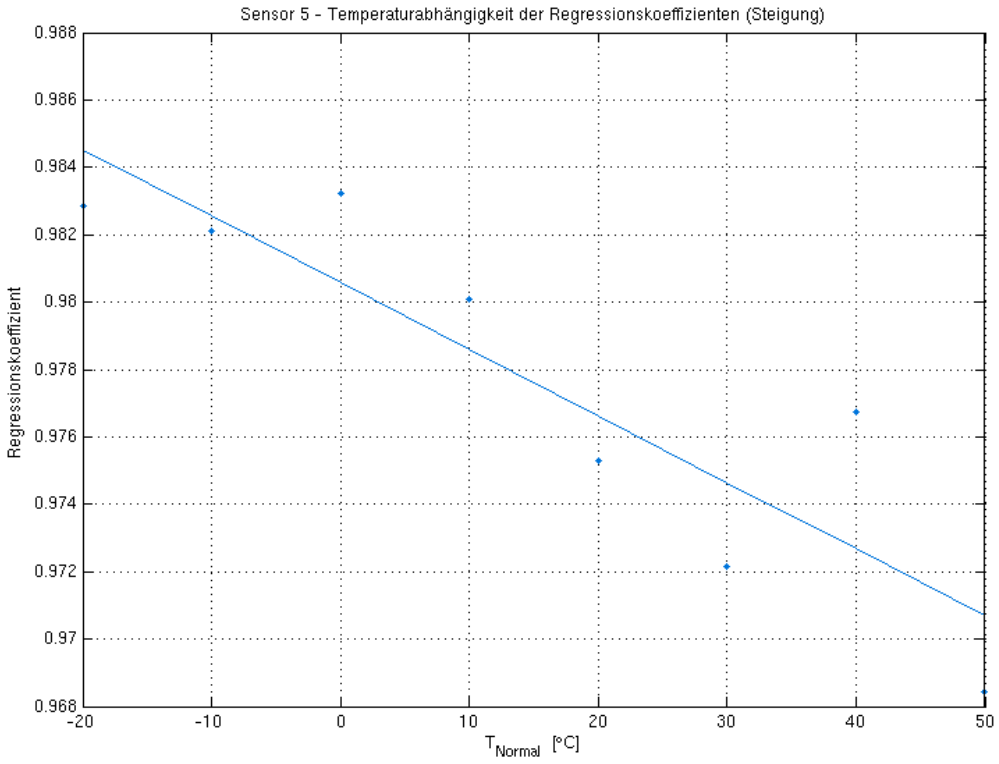


Abbildung 4.8.: Temperaturabhängigkeit der Spannungskennlinie (Steigung) Sensor 5

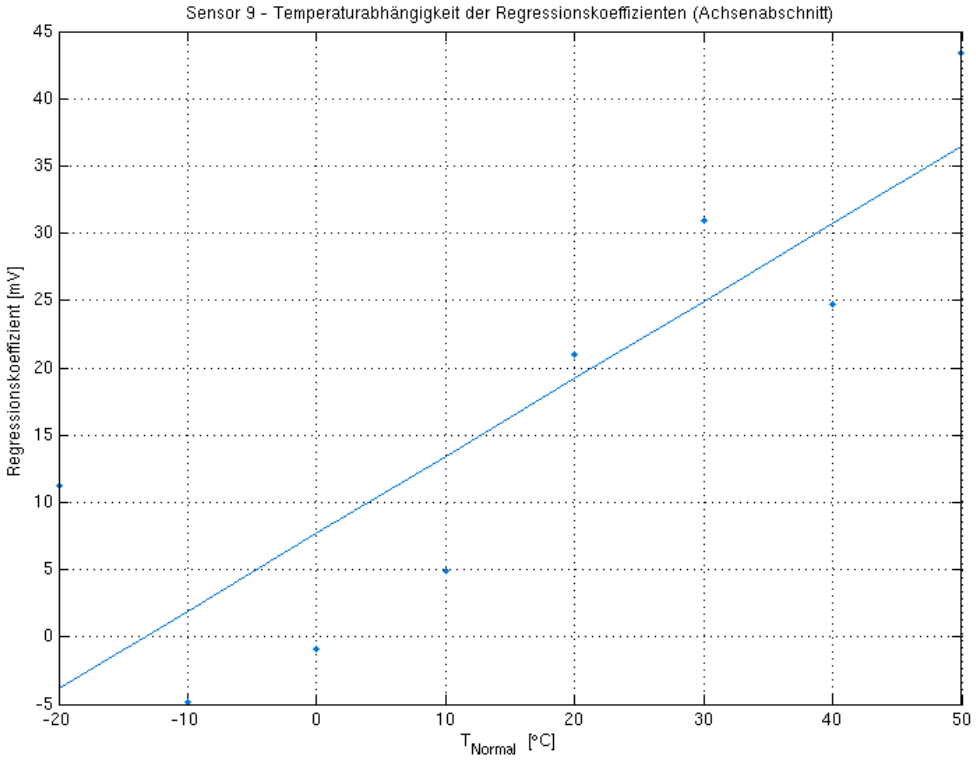


Abbildung 4.9.: Temperaturabhängigkeit der Spannungskennlinie (Achsenabschnitt) Sensor 9

Für die Steigung ist eine Linearität bei allen Sensoren vorhanden. Somit kann die Regressionsgerade berechnet werden. Anhand dieser lässt sich der Koeffizient für die Steigung für jeden Temperaturwert interpolieren. Die Koeffizienten dieser Regressionsgeraden sind in der Tabelle 4.5 aufgelistet.

Tabelle 4.5.: Koeffizienten der Regressionsgerade (Steigung)

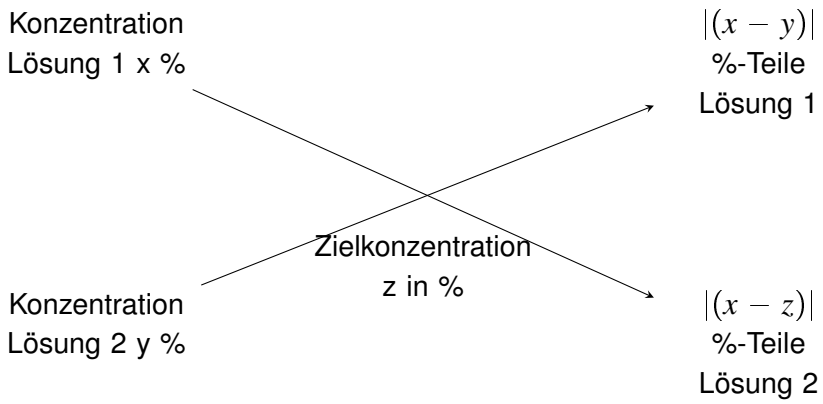
Sensor	Steigung	Achsenabschnitt
2	-0.000208882	0.996048
3	-0.000213684	1.00579
5	-0.000197699	0.980583
6	-0.000201919	0.974267
7	-0.000254189	1.01534
9	-0.000276463	0.995968

Aufgrund des fehlenden linearen Zusammenhangs, kann beim Koeffizienten des Achsenabschnitts dieses Verfahren nicht angewendet werden. Bei der Auswertung sollte der beim Kalibrierverfahren ermittelte Koeffizient, welcher am nächsten zum korrigierenden Messwert liegt, verwendet werden.

4.3.4. Kalibrierung der optischen Messung

Zwecks einer genaueren Auswertung der optischen Messwerte durch geeignete Kalibrierverfahren, wurde das Verhalten der entwickelten Sensorsonden bei verschiedenen Schwefelsäurekonzentrationen außerhalb der Bleibatterie untersucht.

Angefangen mit einer 100-prozentigen Wasserlösung bzw. einer $0\,\%$-H_2SO_4-Lösung wurde in regelmäßigen Abständen bei Raumtemperatur die Konzentration der Säure erhöht. Dabei wurde für jeden Messvorgang eine $120\,\mathrm{ml}$ Lösung mit der gewünschten Konzentration aus Wasser und verdünnter Schwefelsäure ($37{,}5\,\%$) präpariert. Diese wurde anschließend gleichmäßig auf die Reagenzgläsern verteilt, so dass die Sensorsonden in deren Nutzlänge vollständig in der präparierten Lösung eingetaucht sind (siehe Abb. 4.10).
Für die Berechnung der benötigten Wasser- und Säureverhältnisse, um die Zielkonzentration zu erreichen, wurde das Mischungskreuz verwendet:



In der folgenden Tabelle 4.6 sind mittels das Mischungskreuz berechneten Wasser- und Säureanteile für alle Konzentrationsschritte dargestellt.

Messungs-nummer	**Zielkonzentration H_2SO_4 [%]**	**Wasservolumen [ml]**	**Säurevolumen [ml]**
1	0	120	0
2	2	113,6	6,4
3	6	100,8	19,2
4	10	88	32
5	14	75,2	44,8
6	18	62,4	57,6
7	22	49,6	70,4
8	26	36,8	83,2
9	30	24	96
10	34	11,2	108,8
11	37,5	0	120

Tabelle 4.6.: Bestimmung des Wasser- und Säurevolumens für einer 120 ml Lösung mit den realisierten Zielkonzentrationen

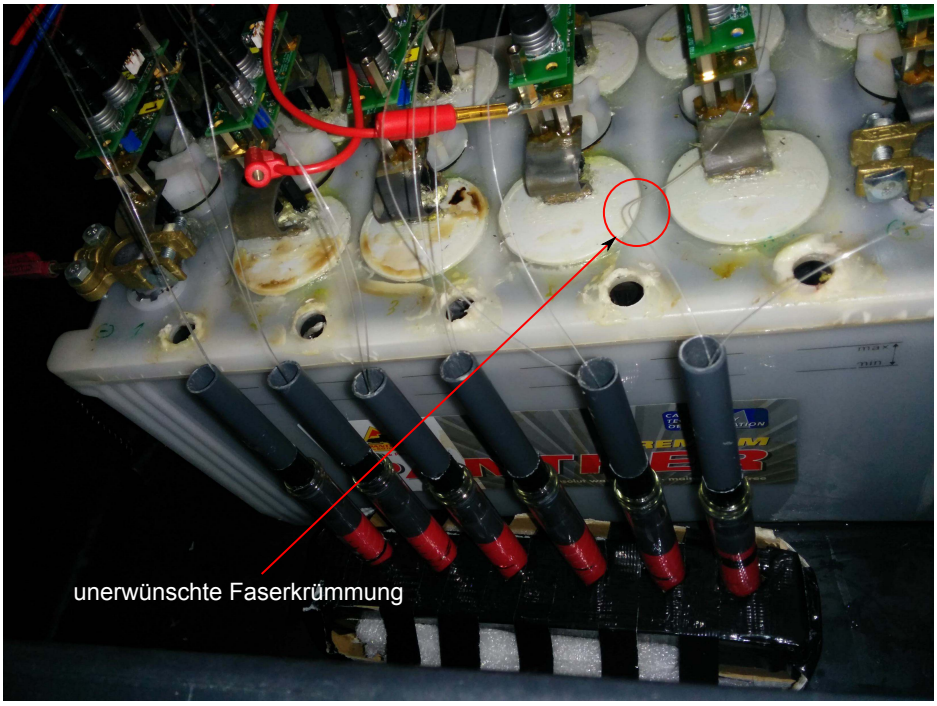


Abbildung 4.10.: Messaufbau: Kalibrierung der optischen Messung. Die Sensorsonden wurden vollständig mit deren Nutzlänge in Reagenzgläsern eingetaucht.

Für jede Konzentrationsstufe wurde die Transmissionsleistung mit den Sensoren erfasst und mittels der MATLAB-Auswertesoftware ausgewertet. Aufgrund mangelnder Anzahl von funktionsfähigen Sensoren wurde diese Untersuchung nur mit 5 Sensoren durchgeführt. Die Ergebnisse werden in den folgenden Abbildungen dargestellt.

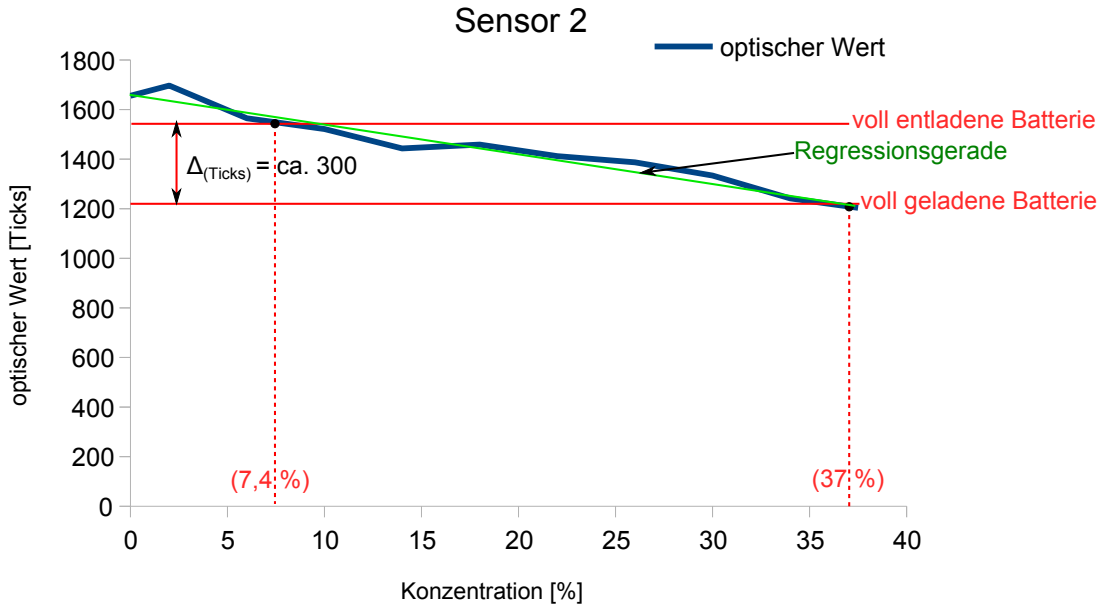


Abbildung 4.11.: Transmissionsleistung in Abhängigkeit der Konzentration für Sensor 2. Wie erwartet nimmt die Transmissionsleistung bei Erhöhung der Konzentration der Schwefelsäure ab. Anhand der dargestellten Regressionsgeraden (grün) ist zu erkennen, dass der Zusammenhang nahezu linear ist. Die Änderung der Transmissionsleistung im spezifischen Konzentrationsbereich entsprechend dem voll entladenen-/ geladenen-Zustand der Bleibatterie (rot markiert) sind mit Δ_{Ticks} gekennzeichnet. Wobei die Flankenänderungen des Ausgangssingals aus dem Licht-Frequenz-Umsetzer werden innerhalb der Messdauer (500 ms) mithilfe des Timers vom Mikrocontroller gezählt und in Ticks ausgewertet (siehe 6.1.3). Diese betragen in diesem Fall ca. 300 Ticks (ca. 20% bezogen auf den oberen Messwert).

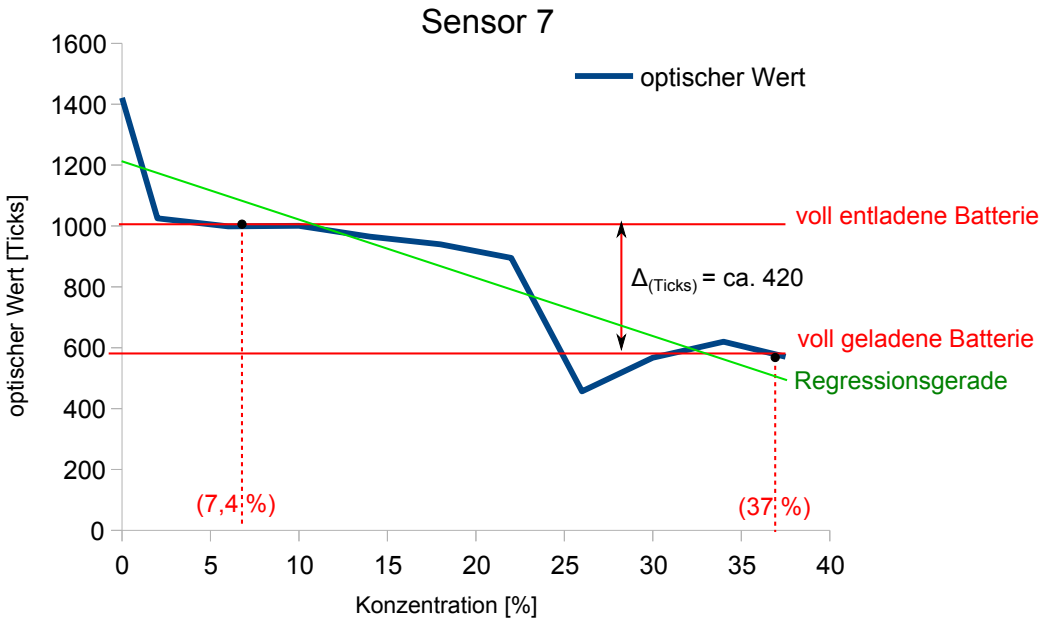


Abbildung 4.12.: Transmissionsleistung in Abhängigkeit der Konzentration für Sensor 7. Die Transmissionsleistung sinkt auch hier mit zunehmender Konzentration. Der Verlauf lässt sich linearisieren, jedoch sind die Abeichungen zur Regressionsgerade deutlich größer als bei Sensor 2. Die Steigung und der Offset sind je nach Konzentrationsbereich sehr unterschiedlich und von wenigen Ausreißern geprägt (insbesondere im Konzentrationsbereich zwischen 25 % und 37 % steigt die Transmissionsleistung von der Regel abweichend an). Die Transmissionsänderung im spezifischen Konzentrationsbereich (rot markiert) entspricht ca 42 % der oberen Messwert.

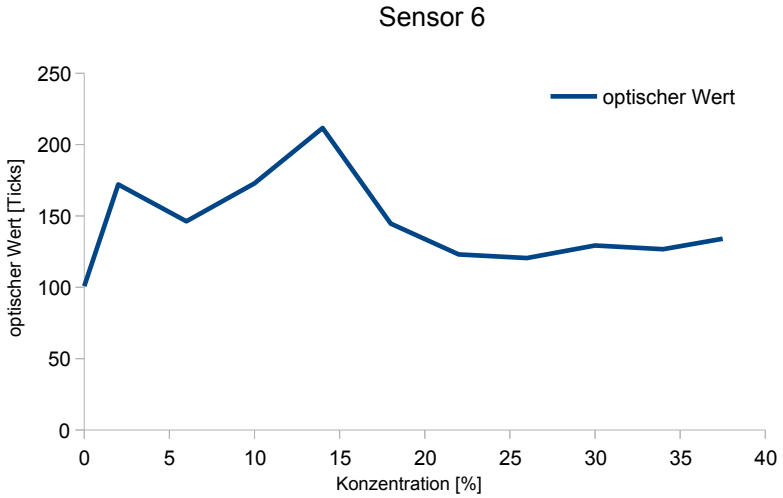


Abbildung 4.13.: Transmissionsleistung in Abhängigkeit der Konzentration für Sensor 6. Anhand der Darstellung ist der erwartete Verlauf (Transmissionsverluste mit zunehmender Konzentration) nicht zu erkennen. Einen nachvollziehbaren Zusammenhang der Transmissionsleistung mit der Konzentrationsänderung (Trendlinie) konnte in diesem Fall nicht festgestellt werden. Dieses Verhalten kann durch die nicht ausreichende mechanische Stabilität der Steckverbindung (siehe 3.4) und Fasern erklärt werden, da die Messungen mit nicht optimierter Hardware durchgeführt wurde. Für jeden Konzentrationsschritt müssen die Sensorsonden aus den Reagenzgläsern herausgezogen werden und dadurch wird die mechanische Ausgangslage für jeden Messvorgang beeinflusst. Darüber hinaus ist ersichtlich, dass die gemessene Transmissionsleistung deutlich geringer als die von Sensor 7 (Transmissionsleistung des Sensors 6 ist kleiner als die Transmissionsänderung des Sensors 7). Diese ist ebenfalls auf die mangelnde mechanische Stabilität des Stecksystems und der ungleichmäßigen Verformung der Fasern (siehe 4.10) zurückzuführen.

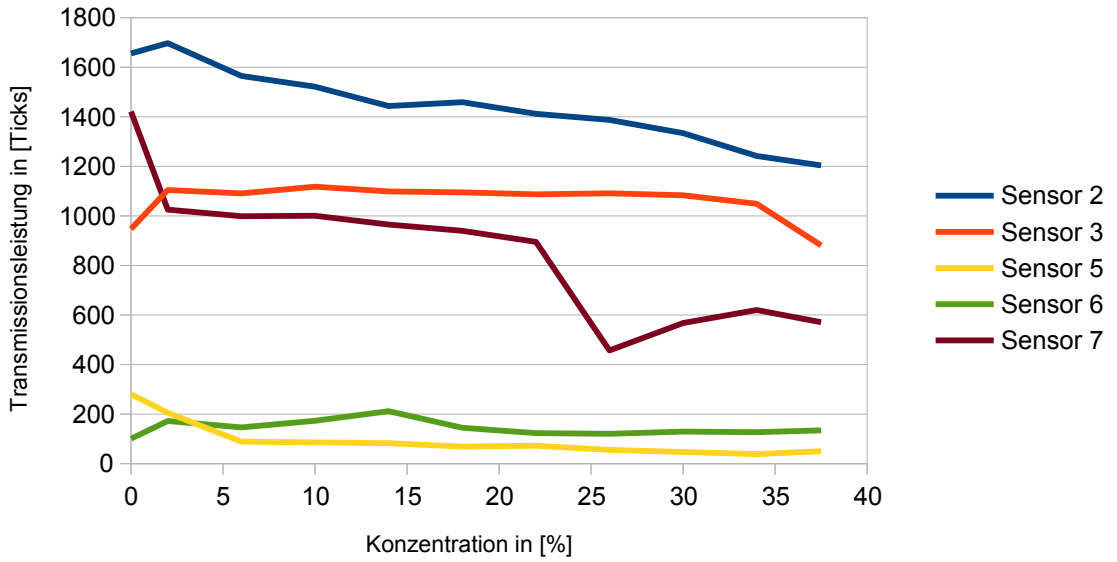


Abbildung 4.14.: Gesamtdarstellung der Transmissionsleistung in Abhängigkeit der Konzentration für alle untersuchten Sensoren. Hier sind unterschiedliche Verhalten der Sensoren deutlich erkennbar. Bei **Sensor 2** und **7** ist die Abhängigkeit der Transmissionsleistung mit der Konzentration annähernd linear. Der typische erwartete Verlauf ist neben den wenigen Ausreißern ebenfalls ersichtlich. Dagegen sind die Messergebnisse der **Sensoren 3, 5 und 6** unbrauchbar und spiegeln keinen sinnvollen Zusammenhang der Transmissionsleistung mit der Konzentrationsänderung wider. Die Transmissionsleistung bei **Sensor 5 und 6** ist wesentlich kleiner im Vergleich mit dem Rest der Sensoren. Dieses Verhalten ist durch die nicht ausreichende mechanische Stabilität der bearbeiteten TOSLINK-Stecksysteme und Fasern begründet.

Die erhaltenen Messergebnisse für die Mehrheit der untersuchten Sensoren (Sensor 3, 5 und 6) sind aufgrund der mechanischen Instabilität untauglich für eine Anwendung eines Kalibrierungsverfahrens. Daraufhin wurde zuerst auf eine Untersuchung des Verhaltens der Sensoren bei Verdünnung der Konzentration sowie auf die Temperaturabhängigkeitsuntersuchung verzichtet (sehr zeitaufwändig). Eine erneute Untersuchung mit der optimierten Hardware konnte aus Zeitgründen nicht mehr in dieser Arbeit bearbeitet werden. Jedoch lässt sich aus den Ergebnissen von Sensor 7 und 2 ein linearer Zusammenhang der Transmissionsleistung mit der Konzentrationsänderung der Schwefelsäure erkennen, die durch ein Regressionsverfahren kalibriert werden könnten.

5. Hardwareänderung

5.1. Hintergrund der Hardwareoptimierung

Grund für die Hardwareoptimierung ist die Untersuchung der Transmissionsverluste bzw. der Signaldämpfung in Abhängigkeit unterschiedlicher Wellenlängen des Sende-LEDs. Ein zusätzlicher Zusammenhang neben der Brechzahlabhängigkeit könnte von großer Vorteil sein, um Referenzmessungen und Kalibrierverfahren bei diesem optischen Messverfahren anwenden zu können.

Ein optisches Messkonzept wo die Wellenlängenabhängigkeit eine relevante Rolle spielt, ist die Evaneszenzfeld-Sensorik. Dabei werden Lichtwellenleiter teilweise entmantelt (das Cladding entfernt) und dort durch das zu messende Medium ersetzt (z. B. Flüssigkeit). Je nach Zustand, ändern sich die Absorptionseingenschaften des Mediums für bestimmte Wellenlängen. Stimmt die Wellenlänge des gesendeten Lichtes mit der dazu korrespondierten Absorptionswellenlänge des untersuchten Materials überein, werden Lichtanteile absorbiert und somit eine Signaldämpfung der Lichtintensität am Ende der Faser erfasst. Wellenlängen, bei denen keine Lichtabsorption stattfindet, werden als Referenzgrößen verwendet.

Ein weiteres optisches Messverfahren, das auf den Absorptions- und Reflexionseigenschaften unterschiedlicher Wellenlängen beruht, ist die Pulsoxymetrie. Es handelt sich um ein photometrisches Messsystem in der Medizintechnik zur Bestimmung der Sauerstoffsättigung im arteriellen Blut und der Herzfrequenz. Bei der Oxymetrie, wird die physikalische Eigenschaft, dass Oxihämoglobin (HbO_2, mit Sauerstoff angereichertes Hämoglobin) und Desoxihämoglobin (Hb, desoxiginiertes Hämoglobin) Licht in verschiedener Wellenlängen in ungleichmäßigem Ausmaß absorbieren, zu Nutze gemacht, um Rückschlüsse auf den Sauerstoffgehalt im Blut zu ermöglichen. Dabei wird das Gewebe mit Licht in zwei Wellenlängen (660 nm als rotes Licht wahrgenommen, und 940 nm im infraroten Lichtbereich) alternierend durchleuchtet. Das transmittierte Licht wird mittels einer gegenüberliegenden Photodiode erfasst. Bei der Messung mit der roten LED wird das Absorptionsmaximum des Desoxihämoglobins erreicht und die Differenz zwischen den Absorptionsspektren des HbO_2 und Hb am größten ist. Die Messung im infraroten Lichtbereiches dient als Referenzmessung, da bei dieser Wellenlänge, die Lichtabsorption des Oxihämoglobins am höchsten ist (siehe Abb. 5.2). Nach

dem Lambert-Beer-Gesetz und bezogen auf die Vergleichswerte der Messung im infraroten Lichtbereiches wird der Sauerstoffgehalt im Blut prozentual bestimmt.

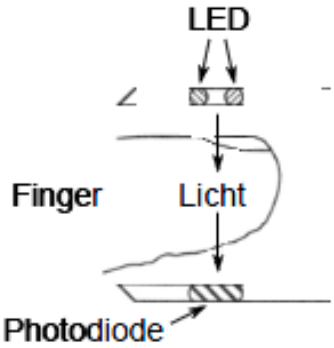


Abbildung 5.1.: Pulsoxymetrie-Messung, basierend auf den Absorptionseigenschaften unterschiedlicher Lichtwellenlängen: Alternierend mit der roten und infraroten LED wird das Gewebe durchleuchtet. Anhand der Photodiode werden die durch Lichtabsorption verursachten Transmissionsverluste detektiert [26].

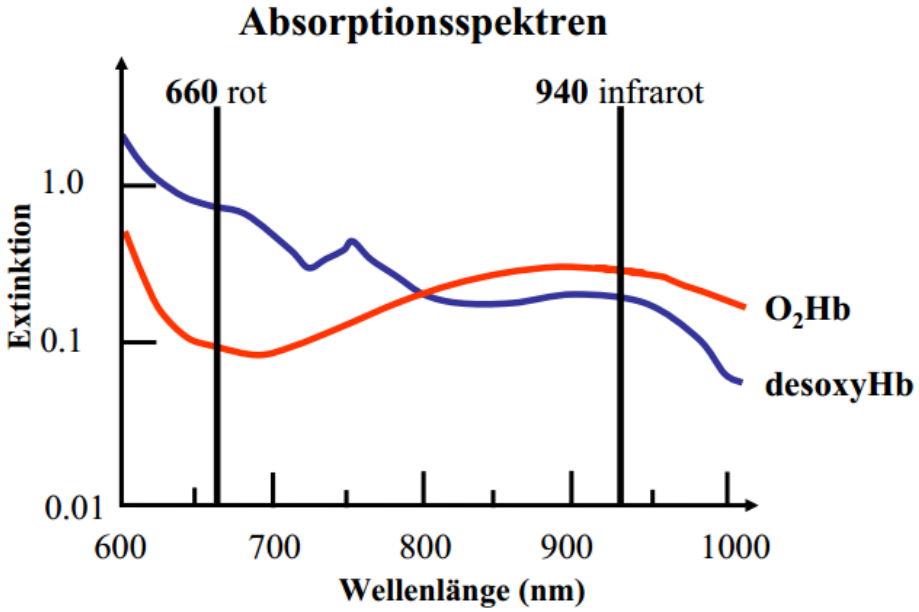


Abbildung 5.2.: Absorptionsspektren von Oxihämoglobin (HbO_2) und Desoxihämoglobin (Hb): Bei einer Wellenlänge von 660 nm lassen sich die größten Transmissionsverluste bezüglich einer Änderung der Sauerstoffsättigung erfassen. Bei 940 nm wird das Absorptionsmaximum von Oxihämoglobin erreicht. Hierbei dienen die erfassten Werte als Referenzen [35].

Neben dieser Messmethodik, die auf den optischen Absorptionseigenschaften verschiedener Wellenlängen basiert, existiert noch ein alternatives Messverfahren für die Oxymetrie, das auf die Refelexionseigenschaften (siehe Abb. 5.4) ebenfalls verschiedener Wellenlängen beruht. Hierbei werden ebenso zwei Lichtquellen im roten und infraroten Lichtbereich eingesetzt. Mittels der Photodiode werden die reflektierten Lichtstrahlen detektiert. Bei der Wellenlänge 660 nm wird auch hier die Hauptmessung durchgeführt, da die Reflexionsspektren des Oxihämoglobins und Desoxihämoglobins in diesem Wellenlängenbereich die größte Differenz aufweisen. Bei einer Wellenlänge von 890 nm überschneiden sich beide Reflexionsspektren. An diesem Punkt hat der Sauerstoffgehalt keinen Einfluss auf die Lichtreflexionseigenschaften. Aus diesem Grund erfolgt die Messung im infraroten Lichtbereich als Referenzmessung.

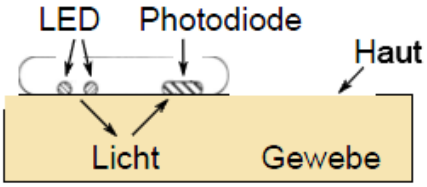


Abbildung 5.3.: Pulsoxymetrie Messung basierend auf die Reflexionseigenschaften unterschiedlicher Lichtwellenlängen: hierbei wird mit der roten LED gemessen. Die Messung mit der infraroten LED dient als Referenzmessung. Mittels der Photodiode wird die Intensität der reflektierten Lichtstrahlen erfasst [26].

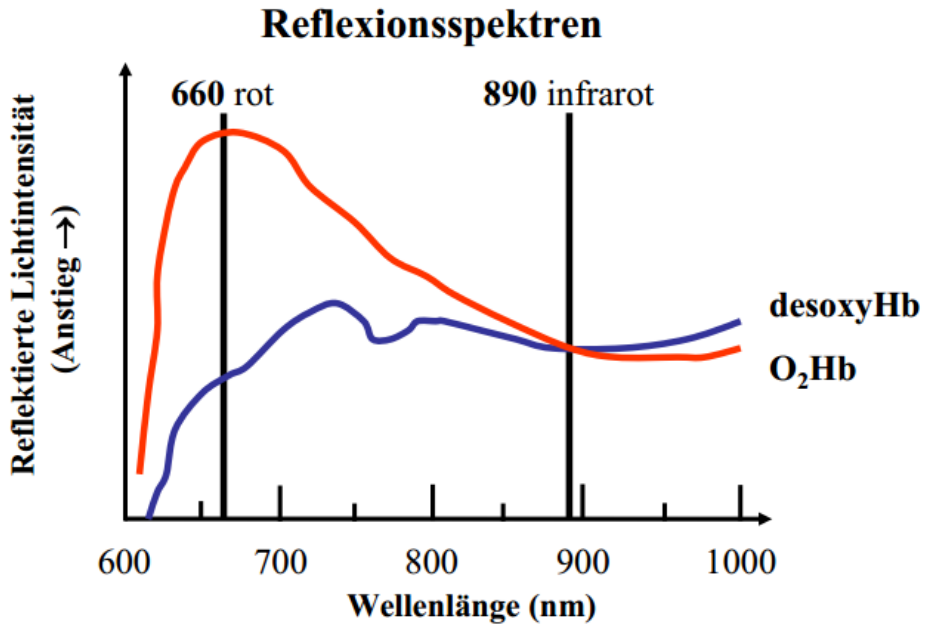


Abbildung 5.4.: Reflexionsspektren von Oxihämoglobin (HbO_2) und Desoxihämoglobin (Hb): Bei einer Wellenlänge von 660 nm, findet die größte Reflexionsänderung bezüglich einer Änderung der Sauerstoffsättigung statt. Bei 890 nm wird die Reflexion des Lichtes nicht vom Sauerstoffgehalt beeinflusst. Aus diesem Grund könne die erfassten Werte als Referenzwerte benutzt werden [35].

Ein ähnliches Messprinzip wie bei der Absorptionsoxymetrie, wurde bei optischen Sensoren für die Ladezustandsbestimmung (SOC) von Bleibatteriezellen verwendet. Dabei handelt es sich um einem Absorptionssensor, der, ähnlich wie das in dieser Arbeit entwickelte Messsystem, aus optischen Fasern und optoelektronischen Komponenten (LED, Photodiode) aufgebaut ist. Jedoch werden hier die Transmissionsverluste in Bezug auf die spektrale Absorptionsänderung des Elektrolyten bei unterschiedlichen Konzentrationen (bzw. Ladezustände) für eine bestimmte Wellenlänge erfasst. Grundsätzlich wird bei diesem Messverfahren zuerst die Absorptionseigenschaften des Elektrolyten für verschiedene Wellenlängen untersucht. Dabei wird die optimale Wellenlänge nach bestimmten Kriterien und Merkmale (nachvollziehbare Signalpeaks, die bestimmte Änderungen der Elektrolytenkonzentration entsprechen) ausgewählt und als Betriebswellenlänge für die Messung verwendet.

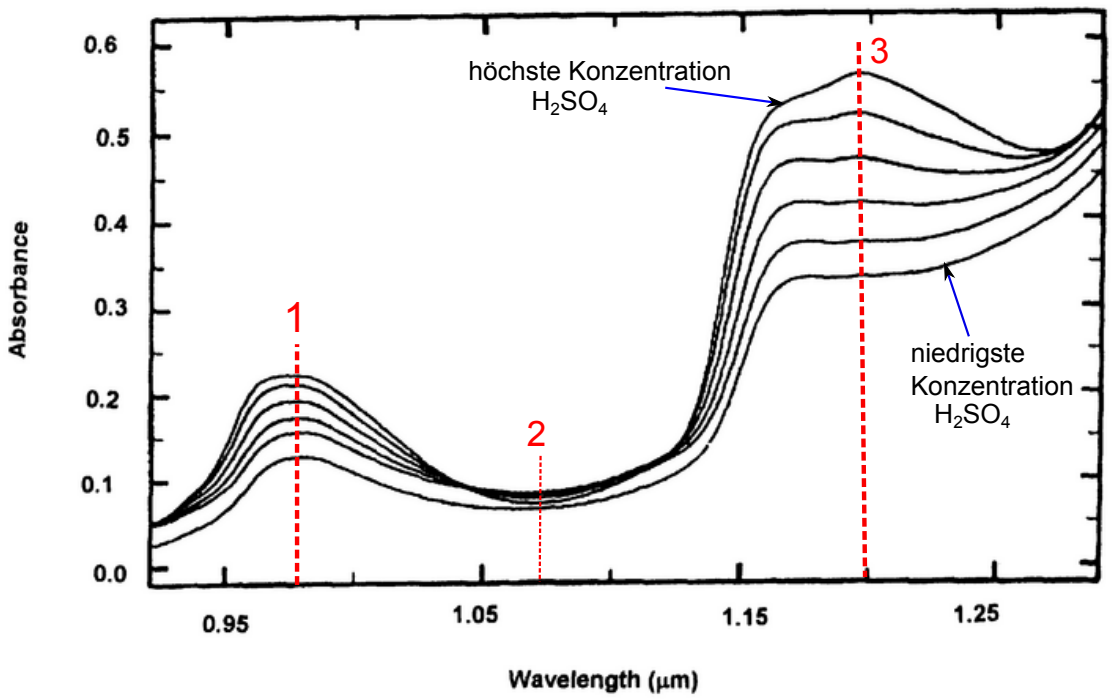


Abbildung 5.5.: Absorptionsspektren von unterschiedlichen Schwefelsäurekonzentrationen bei verschiedenen Wellenlängen bei einer Lichtwellenleiterlänge von 10 mm: Die Absorbance (auch als Absorptivität bezeichnet) ist als Multiplikation von einer Konstanten mit dem Absorptionskoeffizienten und der Brechzahl des Mediums (Elektrolyt) definiert. Die Unterschiede der Absorbance für verschiedene Konzentrationen ist im Wellenlängenbereichen (**1**, **2** und **3**) deutlich erkennbar. Übernommen und modifiziert nach [36]

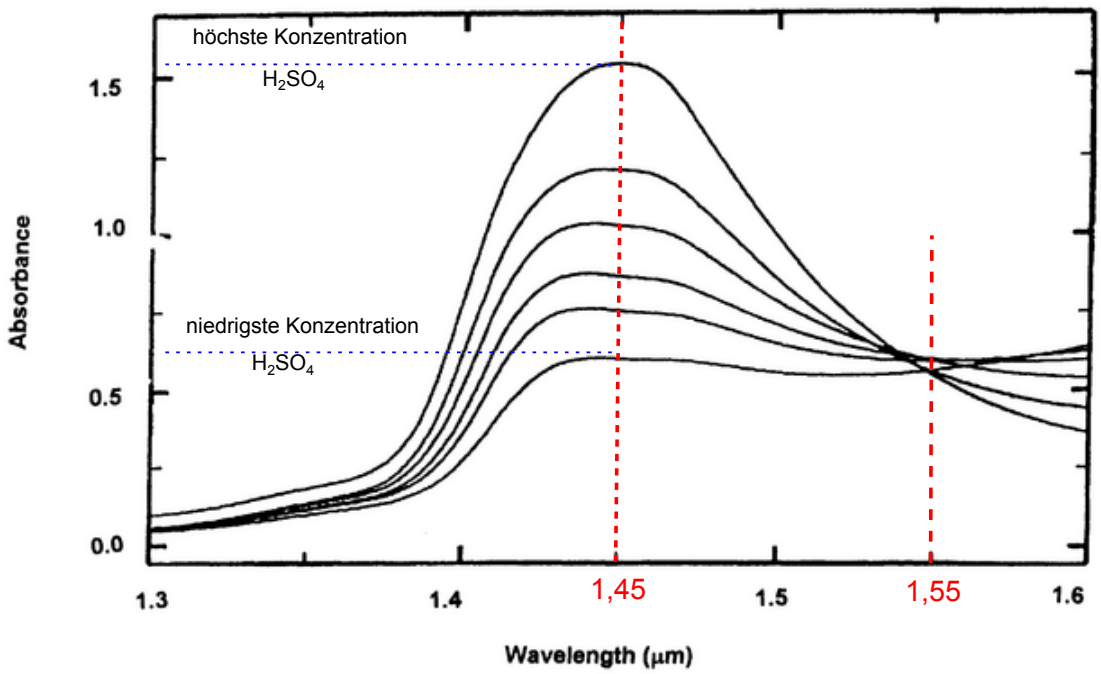


Abbildung 5.6.: Absorptionsspektren von unterschiedlichen Schwefelsäurekonzentrationen bei einer optimalen Wellenlängen (Betriebswellenlänge) mit einer Lichtwellenleiterlänge von 1 mm: Die Differenz der Absorbance von der niedrigsten und der höchsten Säurekonzentration ist bei einer Wellenlänge von $1450\,\mathrm{nm}$ am größten. Bei einer Wellenlänge von $1550\,\mathrm{nm}$ wird die Absorptivität nicht von der Konzentration der Schwefelsäure beeinflusst und kann damit als Referenzwellenlänge verwendet werden. Übernommen und modifiziert nach [36]

Anhand der analysierten Absorptionspektren für die Betriebswellenlänge wird anschließend der Ladezustand in Relation mit der Absorbance (Absorptivität) ermittelt (siehe Abb. 5.7).

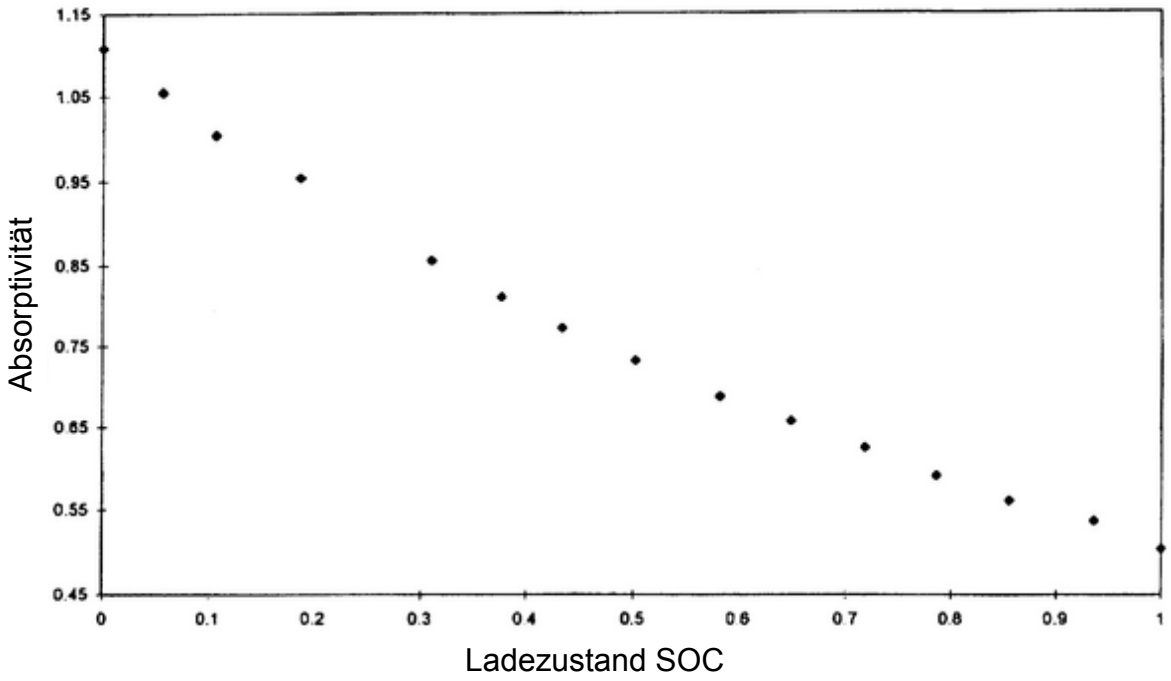


Abbildung 5.7.: Absorptivität in Abhängigkeit vom Ladezustand (State of charge) bei einer Betriebswellenlänge von $1450\,\mathrm{nm}$: In dieser Abbildung ist der Zusammenhang der Absorptivität mit dem Ladezustand einer Bleibatterie dargestellt. die 0 und 1 auf der X-Achse enstprechen respektive eine entladene und eine vollgeladene Batterie. Der Zusammenhang ist nahezu linear [36]

Durch die Hardwareoptimierung, die ein alternierendes Umschalten zweier Lichtquellen mit unterschiedlichen Wellenlängen und die simultane Erfassung der Transmissionsverluste beider Signale ermöglicht, können, neben der optischen Untersuchung der Brechungsindexänderung des Elektrolyten in Abhängigkeit des Ladezustands, die Absorptionseigenschaften in unterschiedlichen Wellenlängenbereichen auch in Bezug einer Ladezustandsänderung untersucht werden. Mittels der gewonnenen Informationen lassen sich Kalibrierungsverfahren bzw. Referenzmessungen durchführen. Die Tauglichkeit ist zu überprüfen.

5.2. Vorhandene Hardware

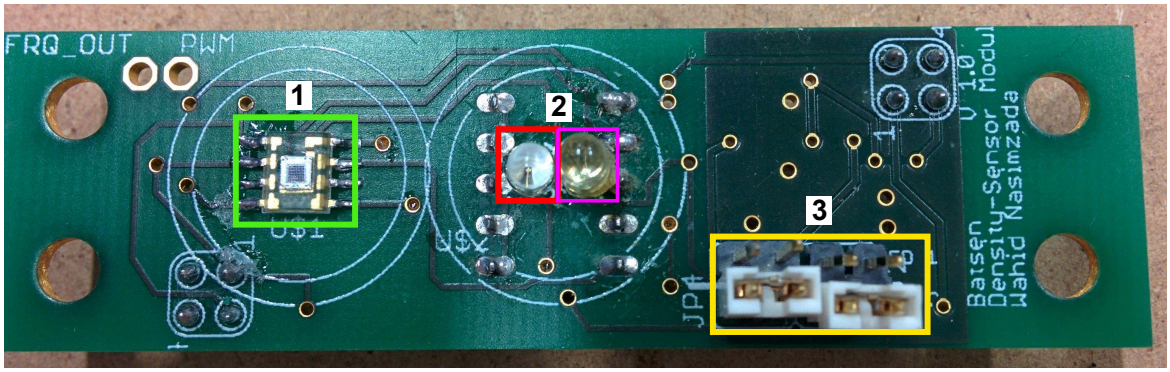
5.2.1. Zellensensor

Der in der Arbeit von Wahid Nasimzada eingesetzte Zellensensor für das optische Messverfahren der Säuredichte bei Bleibattreien, ist eine erweiterte Version des aktuellen Zellensensors der Klasse 1 drahtloser Sensoren, die im Rahmen des BATSEN-Projektes entwickelt worden. Dieser Sensor wird mit dem 16 bit RISC CPU MSP430G2553IPW20 Ultra-Low Power Microcontroller von Texas Instruments betrieben und besitzt einen internen 10-Bit A/D-Converter und 16-Bit Register [37]. Für die Versorgungsspannung des Sensors sorgt der Gleichspannungsregler TPS60201 [38], ebenfalls von TI hergestellt. Dieser liefert ab einer Eingangsspannung von $1{,}6\,\mathrm{V}$ eine $3{,}3\,\mathrm{V}$ stabile Ausgangsspannung. Mit Hilfe des Transmitters SI4012 [39] und auf der Platine realisierten Antenne wurde die drahtlose Kommunikationsschnittstelle realisiert, damit können Daten drahtlos an die Empfangsstation gesendet werden.
Der Zellensensor (folgend ZS abgekürzt [9]) wurde zwecks der Ansteuerung des neuentwickelten Dichte-Sensor-Moduls (folgend DSM abgekürzt [9]) erweitert. Die Verbindung zwischen den ZS und den DSM wurde mittels einer aufsteckbaren physikalischen Schnittstelle ($2x2$-Pin Buchse am ZS und dementsprechend $2x2$-Pinleiste am DSM) ermöglicht. Dabei wurden zwei Pins für die Spannungsversorgung des DSM, ein Pin für die Ansteuerung des LED-Treibers am DSM sowie eine Datenleitung zwischen der Photodiode am Optischen-Sensor-Modul und dem Mikrocontroller verwendet.
Des Weiteren wurde eine Schutzschaltung vor Überspannungen und Verpolung mittels einer Zener-Diode und zwei Sicherungen realisiert.

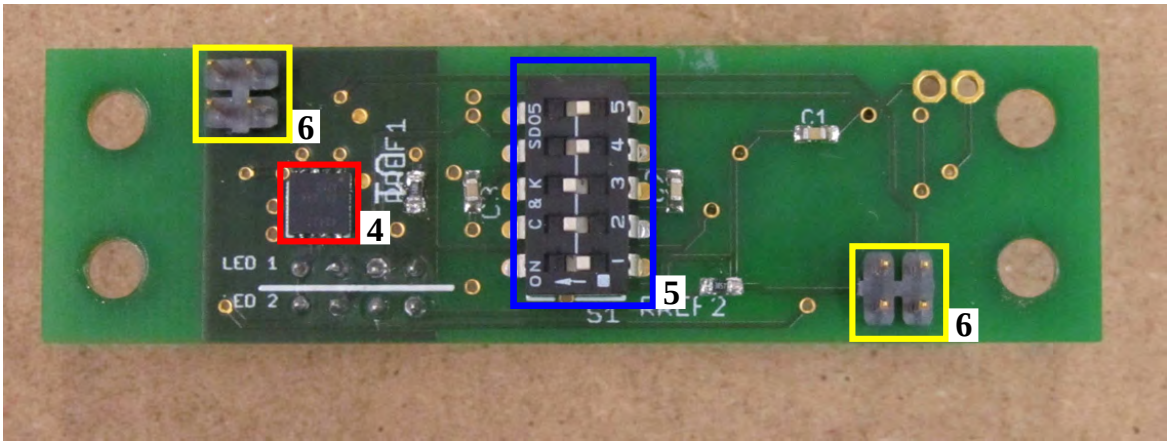
5.2.2. Optisches-Sensor-Modul

Für die Durchführung der optischen Messung zwecks der Ermittlung der Säuredichte innerhalb der Bleibatteriezelle, wurde während der Bachelorarbeit das von Wahid Nasimzada [9] entwickelte Dichte-Sensor-Modul vollständig entworfen. Als optischer Sender dienen eine rote LED mit einer Wellenlänge von $640\,\mathrm{nm}$ sowie eine infrarote LED mit $890\,\mathrm{nm}$. Für die Ansteuerung der beiden LEDs sorgt der LED-Treiber TL4242-Q1 von TI [40]. Dieser liefert am Ausgang einen konstanten Referenzstrom von $177\,\mathrm{mV}$ und wird durch ein vom Mikrocontroller am ZS generierten High-Pegel-Signal an- bzw. ausgeschaltet. Die Begrenzung der Durchlassströme der jeweiligen LEDs wird mittels Vorwiderständen realisiert.
Als optischer Empfänger wurde ein Light-to-Frequency TSL230RD Converter (IC mit integrierter Si-Photodiode) der Firma TAOS eingesetzt [41]. Diese wandelt die erfasste Lichtintensität in Frequenz um und gibt sie am Ausgang mit einer absoluten Toleranz von $\pm 20\,\%$

aus. Diese Information wird anschließend über die realisierten Datenleitung zum Microcontroller weitergeleitet und dort weiterverarbeitet. Der Lichtconverter wurde mittels eines DIP-Schalters auf einen Lichtempfindlichkeitsbereich mit voller Auflösung des Eingangssignals eingestellt, um möglichst maximale und unverfälschte Darstellungen der Messwerte zu erhalten [9].
In der folgenden Abb. 5.8 wird das nach Nasimzada entwickelte Dichte-Sensor-Modul (DSM) dargestellt.



(a) Vorderseite des DSM [9]



(b) Rückseite des DSM [9]

Abbildung 5.8.: Dichte-Sensor-Modul (DSM): Light-to-Frequency-Converter (1), Rote LED als Sender (2, Rot markiert), Infrarote LED als Sender (2, Lila markiert), Jumper für das Umschalten zwischen der roten und der infraroten LED (3), LED-Treiber (4), DIP-Switch für die Konfiguration des Converters (5) und DSM Schnittstelle zum ZS (6) (Übernommen und modifiziert aus [9]).

5.3. Optimierung des Optischen-Sensor-Moduls

Zur Untersuchung der Wellenlängenabhängigkeit bei der Durchführung der optischen Säuredichtemessung, musste das alternierende Schalten der beiden LEDs ermöglicht werden um daraus Referenzierungs- bzw Kaliebrierungsmöglichkeiten des Ausgangssignals zu erschaffen. Dies war mit der vorhandenen DSM-Hardware nicht möglich. Dabei erfolgte der Umschaltbetrieb zwischen den zwei verbauten LEDs lediglich durch manuelles Umschalten der verwendeten Jumper.
Im Rahmen dieser Arbeit wurde ein Platinen-Redesign für das Optische-Sensor-Modul (DSM) entworfen und realisiert. Dabei wurde die Jumper-Lösung durch den Einsatz eines zusätzlichen LED-Treibers ersetzt. Hierfür wurde der TL4242-Q1 von TI ebenso eingesetzt. Für die Ansteuerung des zweiten Treibers wurde die physikalische Schnittstelle des DSM-Moduls um einen Pin erweitert und mit dem Microcontroller am ZS vernetzt. Ein gezieltes Umschaltverhalten zwischen den beiden LEDs kann dadurch ausgeführt werden.

Des Weiteren wurde eine zweifarbige Top-LED des Serientyps LSYT67B [31] von OSRAM als optischer Sender verwendet anstatt der roten und infraroten LED. Zwei unterschiedliche Wellenlängen mit jeweils $633\,\mathrm{nm}$ (super-Rot) und $587\,\mathrm{nm}$ (Gelb) können durch die neuen LED emittiert werden. Eine weitere Besonderheit dieser LED ist die hohe Lichtintensität, die einen Wert bis zu $450\,\mathrm{mcd}$[1] für super-Rot, bzw. $710\,\mathrm{mcd}$ für Gelb erreichen können.

Die für die Begrenzung des nötigen Durchlassstroms der LED verbauten Referenzwiderstände wurden neu dimensioniert, damit ein maximaler Vorwärtsstrom von $30\,\mathrm{mA}$ (gilt für beide LED-Farben) [31] zwischen dem Referenzausgang des LED-Treibers und der LED fließen darf. Laut den Angaben im Datenblatt liefert der LED-Treiber eine konstante Referenzspannung von $177\,\mathrm{mV}$ [40]. Der Referenzwiderstandswert lässt sich anhand folgender Gleichungen 5.1 berechnen:

$$R_{Ref} = \frac{V_{Ref}}{I_V} = \frac{177\,\mathrm{mV}}{30\,\mathrm{mA}} = 5{,}9\,\Omega \tag{5.1}$$

wobei:

- V_{Ref}: Referenzspannung am LED-Treiber
- I_V: Maximaler Vorwärtsstrom der Top-LED
- R_{Ref}: Referenzwiderstand

Ein mechanischer Vorteil bietet ebenso die neue Top-LED gegenüber der alten Hardware-Lösung. Durch das kompakte Gehäuse können verschiedene Buchsen-Technologien

[1] Millicandela, 1 Candela ist die Grundeinheit der Lichtintensität im internationalen Einheitssystem

(TOSLINK-, SMA[2]-Steckverbindung) zentrierter und stabiler verbaut werden. Im Gegensatz zum vorhandenen DSM (Abb. 5.8(a)), ist Durch den sehr geringen Abstand zwischen den beiden LEDs ist eine nahezu vollständige und gleichmäßige Lichtaufnahme über der eingekoppelten POF-Fasern gewährleistet.

Als optischer Empfänger wurde der Light-to-Frequency TSL230RD Umsetzer der Firma TAOS übernommen und weiterverwendet. Jedoch wurde auf den DIP-Schalter, der für die Realisierung der gewünschten Konfiguration der Photodiode benutzt wurde, verzichtet und durch direkte Verbindungen der betroffenen Eingänge dementsprechend ersetzt. Diese optimale Einstellung für einen vollen Lichtempfindlichkeitsbereich und eine volle Auflösung des Ausgangssignals der Photodiode wurde wie folgt realisiert.
Nach dem Datenblatt [41] sind Konfiguriermöglichkeiten definiert und durch kombinierte Schaltzustände der Eingänge einstellbar.

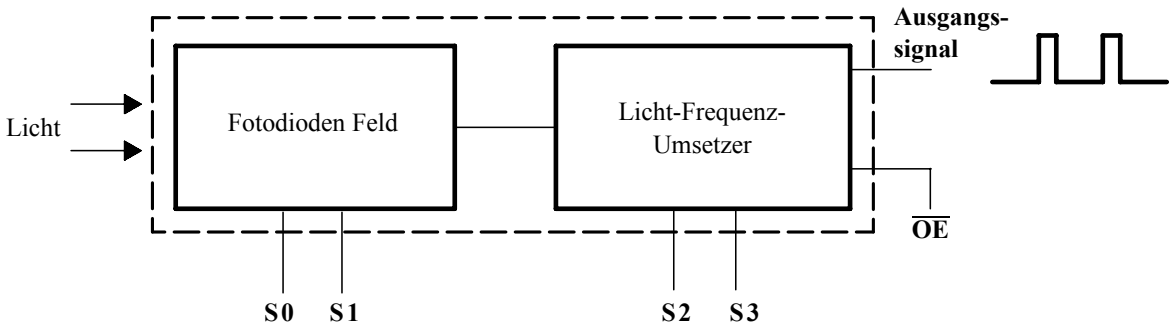


Abbildung 5.9.: Funktions-Blockdiagramm des Licht-Frequenz-Umsetzers TSL230RD [9] [41]

Für eine volle Auslastung des lichtsensitiven Feldes gilt folgende Schaltzustandskombination:

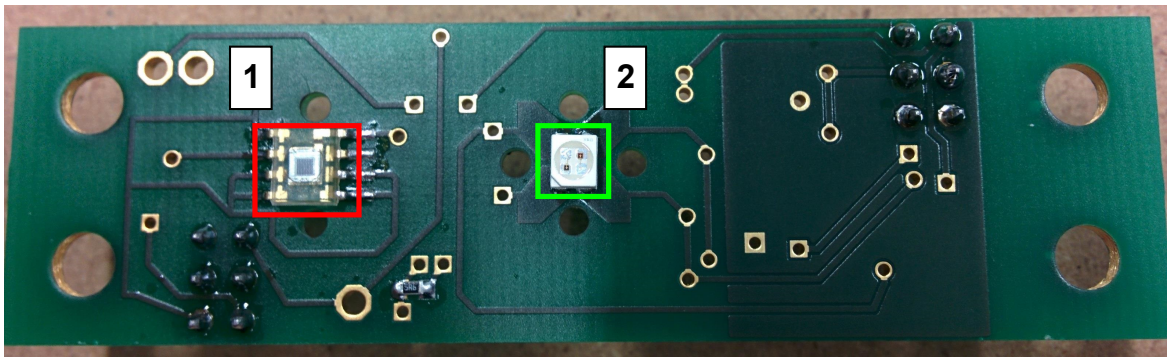
- **S0 = High; S1 = High:** Die Eingänge S0 und S1 wurden mit der Versorgungsspannung $3{,}3\,\mathrm{V}$ verbunden und somit auf einem High-Pegel gelegt.

Für die volle Auflösung des Ausgangssignals gilt folgende Einstellung:

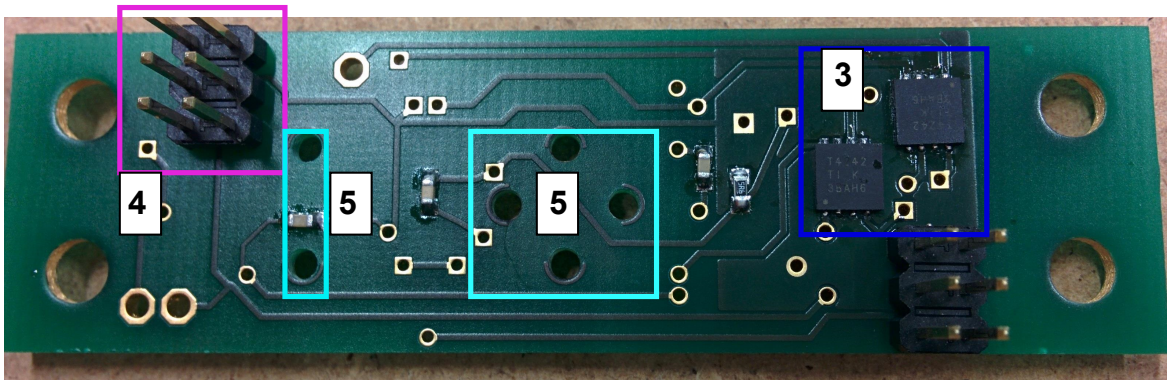
- **S2 = LOW; S3 = LOW:** Die Eingänge S2 und S3 wurden mit Masse verbunden und somit auf einem Low-Pegel gelegt.

[2] *SUB-Miniatur-A, werden grundsätzlich für die Verbindung von dünneren Lichtwellenleiter insbesondere Glasfaserkabeln verwendet*

Das Ausgangssignal wurde ebenso mit dem Microcontroller verbunden. Timergesteuert werden die Flankenänderungen der erfassten Frequenzen detektiert und in Form von Ticks ausgewertet.



(a) Vorderseite des optimierten DSM



(b) Rückseite des optimierten DSM

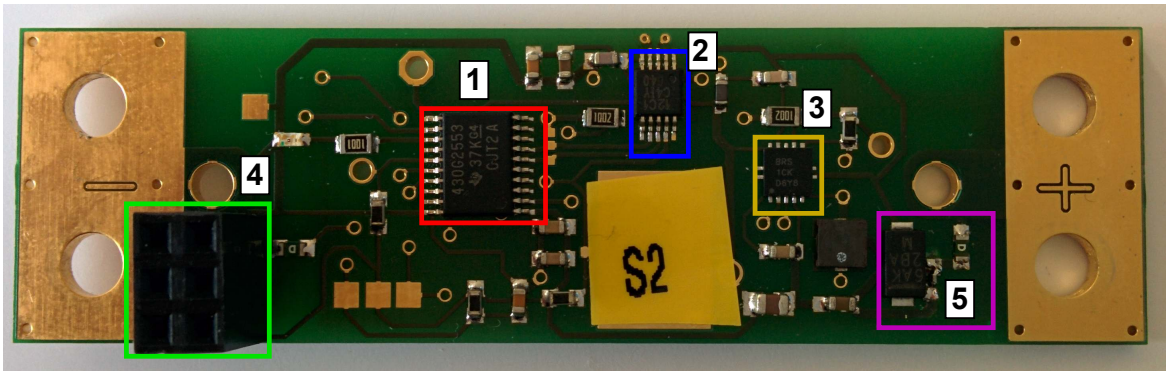
Abbildung 5.10.: Optimierter Dichte-Sensor-Modul ohne eingebaute Buchsen für die Steckverbindung: **[1]** Light-to-Frequency-Converter, **[2]** zweifarbige (super-Rot, Gelb) Top-LED, **[3]** beide LED-Treiber zum Umschaltbetrieb der LEDs, **[4]** DSM-Schnittstelle wurde um einen Pin erweitert für die Ansteuerung des zweiten LED-Treibers, **[5]** vorgebohrte Löcher für SMA-Steckverbindung

Angaben zum Schaltplan und dem Platinen-Layout des optimierten Dichte-Sensor-Moduls befinden sich im Anhang (siehe C.2).

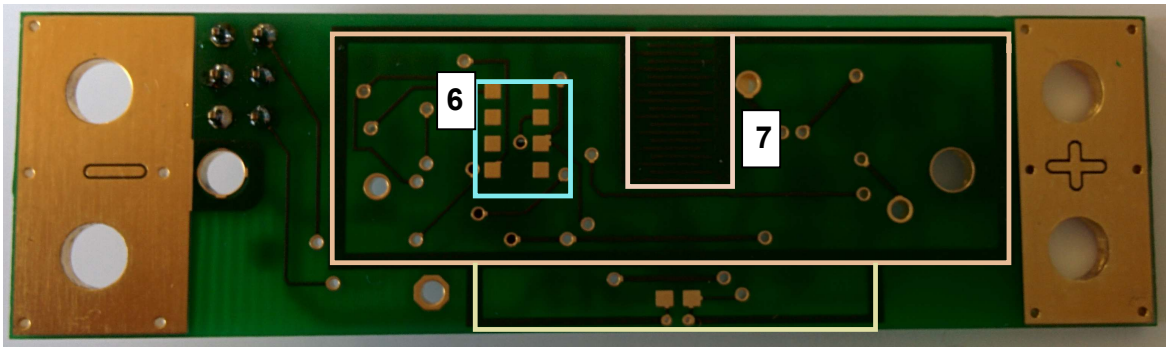
5.4. Erweiterung des Zellensensors

Entsprechend der durchgeführten Optimierungen auf dem DSM, wurde der ZS ebenfalls erweitert. Für die Ansteuerung des zweiten LED-Treibers wurde ein Ausgangspin vom Mikrocontoller durch die Erweiterung der physikalischen Schnittstelle des Zellensensors mit dem Eingang des LED-Treibers verbunden.

Des Weiteren wurde der Verpolungsfehler [9], der nachträglich auf der Platine korrigiert wurde, durch verbesserte Masseverbindung im Schaltplan behoben und somit ist eine fehlerfreie Platine entstanden.
Fernerhin wurden alle zuvor erwähnten Bauteile und elektrischen Komponenten des nach Nasimzada modifizierten Zellensensors der Klasse 1 übernommen (siehe C.1).



(a) Top Layer der Zellensensorplatine



(b) Bottom Layer der Zellensensorplatine

Abbildung 5.11.: Erweiterter Zellensensor (ZS 2): **[1]** Mikrocontroller MSP430, **[2]** Transmitter, **[3]** DC-Regler für die 3,3 V Spannungsversorgung, **[4]** Schnittstelle zur Verbindung des DSM (nur 5 Pins werden benutzt), **[5]** Schutzschaltung für Verpolung und Überspannung, **[6]** Programmier-Pads, **[7]** Antenne

Der Schaltplan und das dazugehörige Platinenlayout zum optimierten Zellensensor sind im Anhang beigefügt (siehe C.1).

6. Softwareänderung

6.1. Zellensensor-Software

Die in dieser Arbeit verwendete Software für die Zellensensoren beruht auf die nach [9] erweiterte Softwareversion der Klasse 1 Zellensensoren. Die Grundfunktionen für die Temperaturmessung und Spannungsmessung sind unverändert implementiert worden. Gemäß der durchgeführten Hardwareänderungen auf den Zellensensor sowie auf das Dichte-Sensor-Modul, wurden die Funktionen für die optische Messung und Datenübertragung angepasst. Des Weiteren wurde das entwickelte Verfahren für die Umgebungslichtunterdrückung (Differenzmessung) in der Software implementiert. Die Grundstruktur des Programmablaufs ist mit den durchgeführten Veränderungen aufrechterhalten und wird in der Abb. 6.1 dargestellt.

Nach der Initialisierung der Grundfunktionen und die Peripherie des Mikrocontrollers, wird zuerst mit Hilfe eines Schieberegisters eine zufällige Wartezeit aus der Sensoradresse generiert. Da die Zellensensoren nicht synchron betrieben werden können, werden anhand dieser pseudozufälligen Variablen unterschiedliche Sendezeitpunkten für die betriebenen Sensoren realisiert. Durch dieses Verfahren wird eine Kollision von gesendeten Datenpakete mehrere Sensoren, die im selben Zeitpunkt in Betrieb genommen wurden, vermieden. Dieses Verfahren wurde bereits bei früheren Abschlussarbeiten [42], [34] und [9] eingesetzt. Es wird anschließend im ersten Programmdurchlauf nach einer zusätzlichen Wartezeit von 500 ms die Transmissionsleistungsmessung mit der roten Sende-LED, die Differenzmessung sowie die Spannungs- und Temperaturmessung ausgeführt. Nach einer weiteren 500 ms langen Wartezeit wird der gesamte Messblock wiederholt. Die Messerwerte aus den beiden Messblöcken werden gemittelt, aufbereitet und abschließend an den Transmitter gesendet. Beim zweiten Programmdurchlauf werden die erwähnten Programmschritte mit der gelben LED abgearbeitet.

Die Kalibrierung der erfassten Temperatur- und Spannungswerte wurde in dieser Arbeit nicht mehr innerhalb der Sensorsoftware implementiert. Diese wurde in der Auswertesoftware realisiert, um einer möglichen Verfälschung der Messdaten, verursacht durch die Basisstation zu vermeiden und damit mehr Genauigkeit zu gewährleisten (siehe 4.3).

In der folgenden Abb. 6.1 wird der ursprüngliche Programmablauf der Zellensensorsoftware dargestellt.

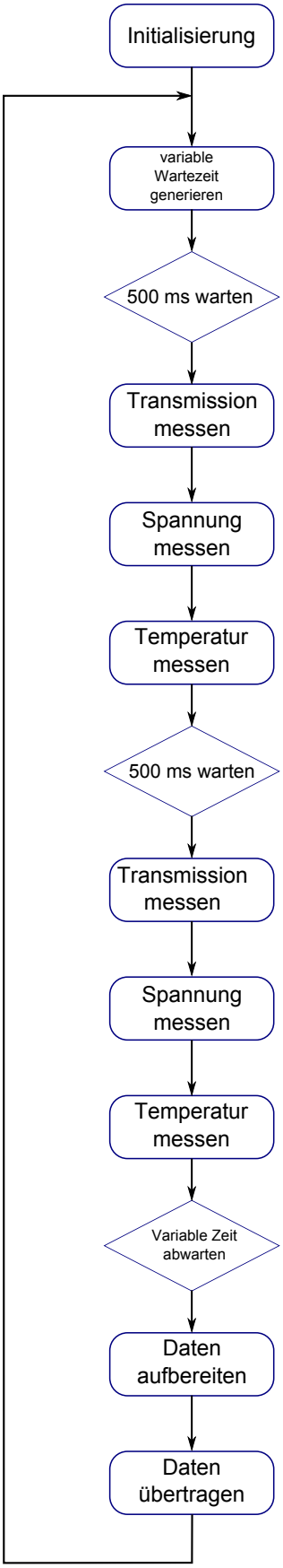


Abbildung 6.1.: Programmablaufplan der Sensorsoftware nach Nasimzada [9]

Die Abb. 6.2 zeigt den in dieser Arbeit veränderten Programmablauf der Sensorsoftware. Dadurch wird die Differenzmessung und der Wechselbetrieb mit den zwei LEDs unterschiedlicher Wellenlänge realisiert.

Abbildung 6.2.: Struktogramm des Hauptprogrammablaufs

Die Prozeduren Messblock und Messung werden in den Abb.6.3 und Abb.6.4 beschrieben.

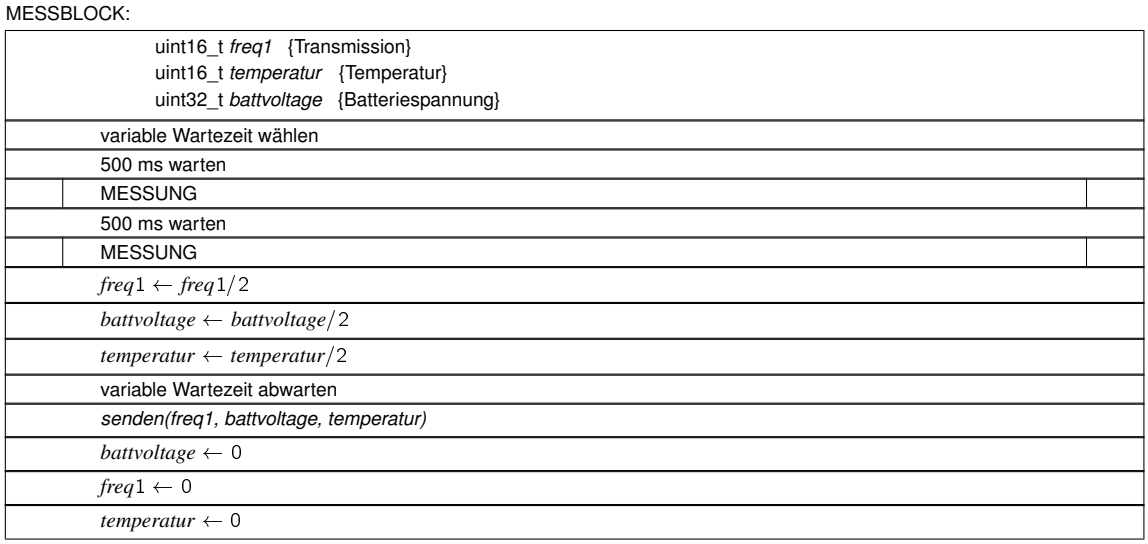


Abbildung 6.3.: Struktogramm der Prozedur Messblock

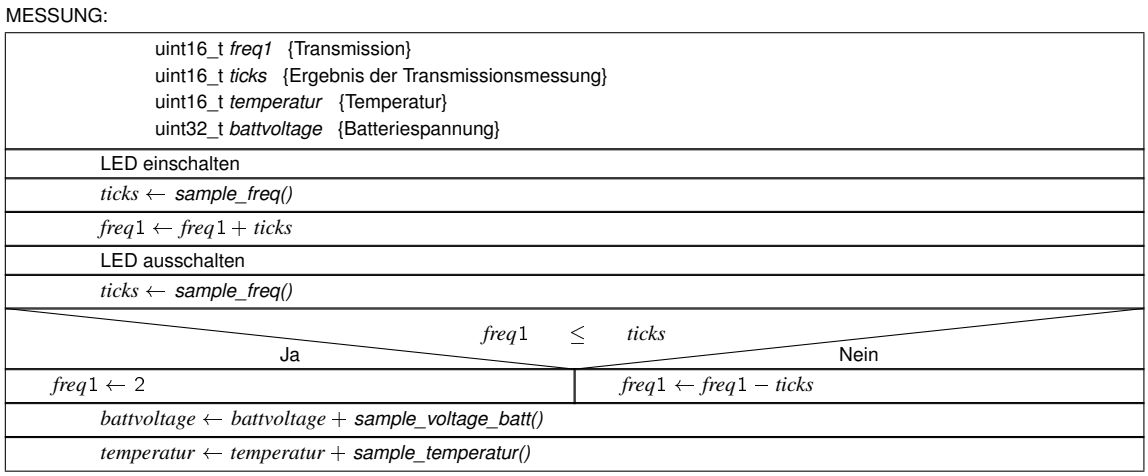


Abbildung 6.4.: Struktogramm der Prozedur Messung

6.1.1. Temperaturmessung

Die Messwerterfassung der Temperatur geschieht durch den internen Temperatursensor des Mikrocontrollers. Anhand eines NTC[1]-Widerstands wird der Spannungsabfall an diesem Widerstand bezogen auf eine interne Referenzspannung von 1,5 V als Maß für temperaturänderung verwendet. Diese Temperaturabhängige Spannung lässt sich anhand folgender Formel berechnen [37]:

$$U_{Sensor} = TC_{Sensor}.T[^{\circ}C] + U_{Sensor}(T = 0^{\circ}C)[mV] \tag{6.1}$$

wobei:

- U_{Sensor}: Temperaturabhängiger Spannungsabfall am NTC-Widerstand
- TC_{Sensor}: Empfindlichkeit des NTC-Widerstands 3,5 mV/°C
- T: Temperatur
- U_{Sensor}: Offset bei 0°C

Anschließend wird dieser Spannungswert mit dem internen 10-Bit-ADC[2] in einem digitalen Wert umgesetzt und an die Basisstation übertragen. Die Umwandlung dieses Wertes in die Temperatureinheit Celsius und die Linearisierung mittels eines Kalibrierungsverfahrens wurde im MATLAB-Auswerteprogramm implementiert.

[1] *eng. Negative Temperature Coefficient*
[2] *eng. Analog-to-Digital-Converter: Ein elektronisches Bauteil, wandelt analoge Eingangssignale in digitale Daten (Signale)*

6.1.2. Spannungsmesung

Die Spannungsmessung erfolgt durch den internen ADC des Mikrocontrollers. Als Referenzspannung dient eine positive interne Spannung von 2,5 V. Die minimale Auflösung der Spannungsmessung ist durch folgende Gleichung bestimmt.

$$U_{LSB}{}^{3} = \frac{U_{Ref}}{2^{10}} = \frac{2{,}5\,V}{1024} = 2{,}44\,mV \tag{6.2}$$

Die Messung der Eingangsspannung (Zellspannung) erfolgt durch die Spanngsteiler-Vorschaltung. Diese hat den Vorteil, die Messung von Eingangsspnnugen bis zu 5 V zu ermöglichen, jedoch gleichzeitig den Nachteil der Halbierung der Auflösung des ADCs auf 4,88 mV.

Der Messvorgang der Spannung wird im Programmablauf mit jeder LED-Wellenlänge zweimal durchgeführt. Anschließend wird der Mittelwert aus beiden Messungen gebildet und abschließend an das Steuergerät übertragen. Die Umrechnung der ADC-Werte in Spannungswerte geschieht an der Basisstation und wird mit der MATLAB-Auswertesoftware kalibriert [9].

6.1.3. Optische Messung

Die Messwerterfassung des von dem Light-to-Frequency-Converter gelieferten Ausgangssignals geschieht durch den Timerbaustein des Mikrocontrollers. Die Messdauer wird durch Timer-Interrupts gesteuert und auf 500 ms eingestellt. Anhand des Compare-Verfahrens werden über die Messdauer die steigenden Flanken des Ausgangssignals gezählt und in Ticks ausgegeben (siehe [9]).

Für die Implementierung der Differenzmessung zwecks Unterdrückung des Umgebungslichts (siehe 4.2.1) wurde in dieser Arbeit eine zusätzliche Messung bei ausgeschalteter LED realisiert. Hierbei wird der erfasste Messwert von dem ersten Wert (mit eingeschalteter LED) subtrahiert. Diese Messprozedur wird ebenfalls wie die Spannungs- und Temperaturmessung zweimal durchgeführt und abschließend das gemittelte Ergebnis an das Steuergerät gesendet. Es wird bei jedem Programmdurchlauf abwechselnd mit beiden LED-Wellenlängen gemessen.

[3] *eng. Least Significant Bit, bezeichnet die Bitwertigkeit in einer Binärzahl. Die letzte Bitposition hat den niedrigsten Stellenwert.*

6.1.4. Datenübertragung

Die Messwerte werden zuerst nach der Manchestercodierung codiert und anschließend in Form von Datenpaketen in das Übertragungsframe der Livedaten (siehe Abb 6.5) geladen.

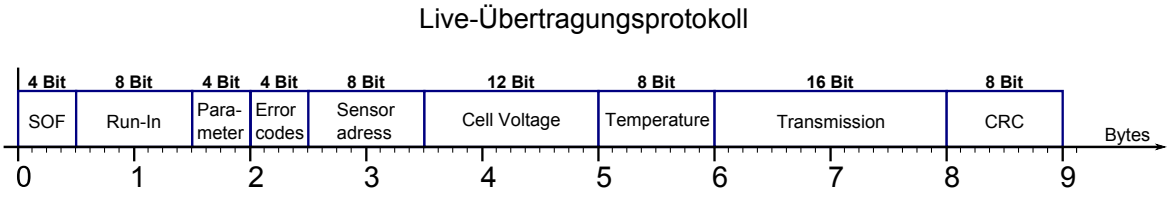


Abbildung 6.5.: Übersicht des Live-Übertragungsprotokolls [9] [34]

Eine detaillierte Beschreibung dieses Protokolls ist in der Arbeit von Michael Meinzer [34] und Nasimzada [9] vorhanden.

Für das Abgrenzen der Messdaten bei unterschiedlichen Wellenlängen wurde in dieser Arbeit die Sensoradresse in dem Übertragungsframe so manipuliert, dass die jeweiligen Daten mit unterschiedlichen Sensoradressen gesendet werden und dadurch die Auswertung ermöglicht wird. Die Messdaten mit der ersten LED-Farbe werden mit der tatsächlichen Sensoradresse übermittelt. Für die Übertragung der Messergebnisse mit der zweiten LED-Wellenlänge wird die Sensoradresse mit einem Offset versehen. Dieser kann je nach Gesamtanzahl der betriebenen Sensoren angepasst werden (siehe B.1).

Eine Erweiterung des Übertragungsprotokolls müsste empfängerseitig in der Software des Steuergeräts angepasst werden. Diese alternative Lösung wurde in dieser Arbeit aufgrund der Unübersichtlichkeit des Codes (mehrere Revisionen) nicht weiter verfolgt.

6.2. MATLAB-Auswertesoftware

Anhand der MATLAB-Auswertesoftware werden zuerst die Datenpakete im Laufe der Messung aus dem Steuergerät per seriellerschnittstelle des PCs ausgelesen und in Dateien gesichert. Nach dem Stoppen des Aufzeichnungsvorgangs werden die Daten kalibriert und abschließend grafisch dargestellt.

Die nach Nasimzada [9] entwickelte Softwareversion wurde in dieser Arbeit entsprechend der durchgeführten Hard- und Softwareänderungen der Zellsensoren optimiert und um Kalibrierungsmodule der Temperatur- und Spannungsmessung erweitert. Im Laufe der Voruntersuchungen mit dem MATLAB-Programm nach Nasimzada trat mehrfach bei der Auswertung

von Langzeitmessungen ein Absturz des MATLAB-Programms auf. Aus diesem Grund wurde die Software in zwei Teilprogramme aufgegliedert.

Das erste Teilprogramm (StgReaderX, siehe B.2.1) ist für die Aufnahme und Sicherung der Messdaten zuständig. Dieses wurde so angepasst, dass die Zuordnung der Datenpakete mit unterschiedlichen Sensoradressen (siehe 6.1.4) eines Zellensensors gewährleistet wird. Nachdem die Messaufnahme mit der auf der grafischen Bedienoberfläche realisierten Stopp-Taste beendet wird, werden die Daten, die im Laufe der Messung in kleineren Dateien gespeichert sind, in einer einziger großen Datei gespeichert.

Anschließend wird der zweite Teilprogramm (Auswertung, siehe B.2.3) ausgeführt. Die Datei mit den gespeicherten Messdaten muss zuerst ausgewählt werden. Demzufolge muss die Datei mit den vorher bestimmten Regressionssparameter mittels der Kalibrierungssoftware (Kalibration, siehe B.2.2) für die Temperatur- und Spannungskalibrierung geladen und der Offset der Sensoradressen, der mit einem Defaultwert initialisiert wurde, eingegeben werden. Danach erfolgt die grafische Ausgabe der kalibrierten Messdaten.

7. Funktionserprobung und Auswertung

7.1. Planung und Durchführung von Messreihen im Zyklierbetrieb

Die bereits kalibrierten Sensoren wurden auf der Batterie montiert. Ebenso wurden die entwickelten Sensorsonden in die realisierten Bohrungen der Batteriezellen hineingeführt und die POF-Faserenden mit dem DSM gekoppelt. Nach einem bestimmten zeitgesteuerten Zyklierplan wurden mehrere Messreihen bei Raumtemperatur durchgeführt.

Der Zyklierplan beinhaltet folgende Phasen:

- 6-stündige Ladephase
- 6-stündige Pause, damit sich die Ruhespannung einpendelt und sich auf einen stabilen Wert einstellt.
- 6-stündige Entladephase

Für einen langen Messbetrieb wurden diese Zyklierschritte wiederholt ausgeführt. Anhand dieses äquidistanten zeitgesteuerten Zyklierplans können genauere Beobachtungen und Analysen des Zeitverhaltens der erfassten optischen Messsignale gegenüber den anderen Messgrößen (Spannung, Temperatur und Strom) erzielt werden.

Für die Realisierung des entwickelten Messplans, wurde das im Rahmen anderer Bachelorarbeiten entwickelte Zykliersystem verwendet. Innerhalb einer Konfigurationsdatei wird der gewünschte Zyklierplan implementiert und die sicherheitskritischen Parameter wie z. B. Spannungs- und Stromgrenzen eingestellt, um einerseits die Batterie schonend zu betreiben (Schutz gegen Tiefentladung und Überladung), anderseits bei Langmessungen unerwarteten Komplikationen vorzubeugen. Darüber hinaus werden die Spannungs-, Temperatur- und Stromverläufe der Batterie während der Messung zusätzlich vom Zykliersystem aufgezeichnet.

Das Laden der Batterie wurde mittels eines Konstanters durchgeführt. Dieser wurde auf eine $14{,}4\,\mathrm{V}$ Ladeschlussspannung eingestellt. Höhere Ladespannungen bzw. hohe Ladeströme können Gitterkorrosion verursachen und beeinträchtigen die Lebensdauer der Batterie. Zusätzlich wurde eine Zellspannungsgrenze von $2{,}39\,\mathrm{V}$ eingestellt. Wird diese überschritten,

kommt es zu einem vorzeitigen Abbruch des Zyklierens.
Für das Entladen wurde eine elektronische Last verwendet und der Entladestrom auf einen konstanten Wert von 5 A eingestellt. Die Entladeschlussspannung wurde auf 1,8 V festgelegt, um Tiefentladung der Batteriezellen zu vermeiden. Die Last und der Konstanter sind an die Batterie angeschlossen und werden vom Zykliersystem durch interne Relais angesteuert.

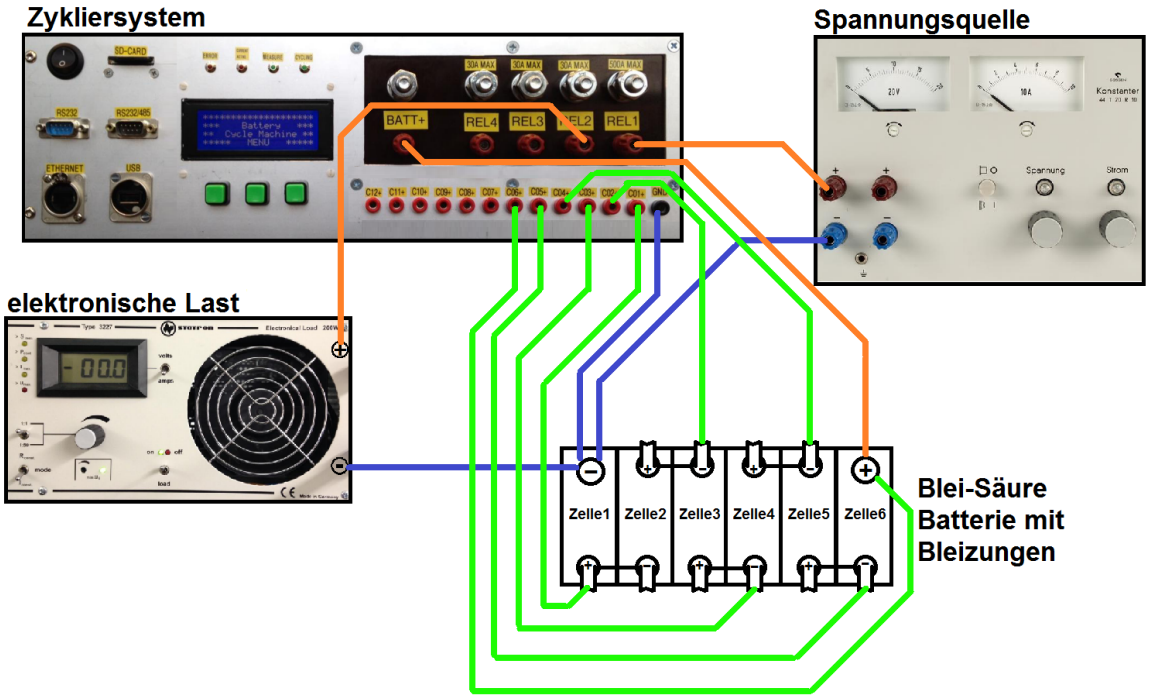


Abbildung 7.1.: Schematische Darstellung des Messaufbaus im Zyklierbetrieb: die 6 Batteriezellen werden mit dem dazugehörigen Kanälen mit dem Zykliersystem verbunden, dadurch kann die Spannung für jede Zelle gemessen werden. Die Spannungsquelle (Konstanter) und die elektronische Last sind am Relais 1 und 2 angeschlossen und werden entsprechend den Zyklusschritten angesteuert. Alle Masseverbindungen sind am Minuspol der Batterie angebracht. Es wurde festgestellt, dass eine Verlegung des gemeinsamen Massepunktes am Zykliersystem, die Spannungsmessung verfälschen würde. Die Strommessung erfolgt über die äußeren Batterieklemmen [43].

Anhand folgender Abb. 7.2 wird ein Beispiel eines eingesetzten Zyklierplans in Form von Spannungs- und Stromverlauf der Bleibatteriezellen dargestellt. Die Implementierung dieses Zyklierplans in die Konfigurationsdatei des Zykliersystems ist im Anhang (siehe B.3) beigefügt.

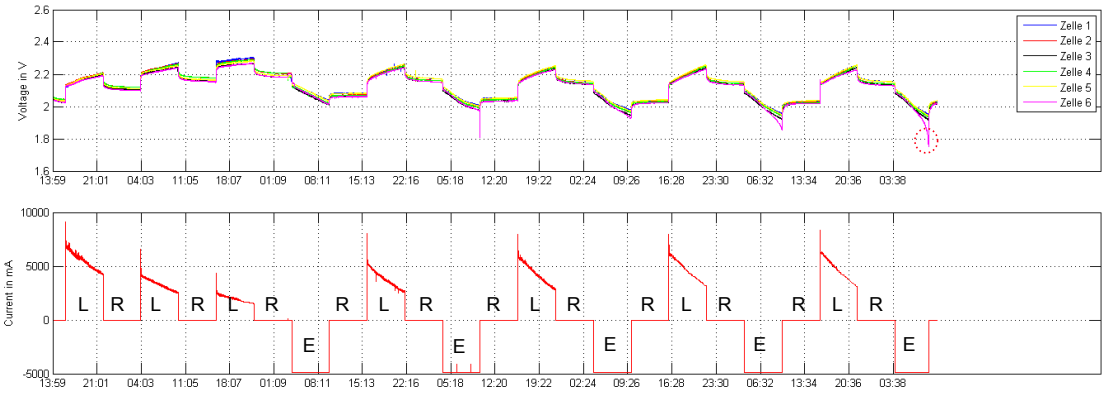


Abbildung 7.2.: Darstellung eines Zyklierplans anhand der Spannungs- und Stromverläufe der Bleibatterie. Oben sind die Spannungsverläufe der Batteriezellen und unten ist der Stromverlauf über die äußeren Batterieklemmen dargestellt. Mit **L**, **R** und **E** sind respektive die Lade-, Ruhe- und Entladephasen gekennzeichnet (diese Kennzeichnung gilt für folgenden Abbildungen). In diesem Zyklierplan wurden zuerst 3 aufeinander folgenden Ladephasen (mit Ruhephasen dazwischen) realisiert, da die Batterie am Anfang der Messung tief entladen war. An der rot markierten Stelle am Spannungsverlauf, wurde die Zyklierung abgebrochen. Die sicherheitskritische eingestellte Spannungsgrenze (1,8 V) wurde bei Zelle 6 unterschritten (Spannung bricht ein). Der Stromverlauf ist dadurch nicht beeinflusst worden, da die Zellen in Reihe geschaltet sind. Der Ladestrom ist von dem Ladezustand der Batterie abhängig.

Bei der Erprobung wurde das Zyklieren der Batterie oftmals vorzeitig abgebrochen. Die folgenden Ursachen für dieses Fehlverhalten wurden anhand der Fehlermeldungen des Zykliersystems identifiziert und analysiert:

1. Watchdog-Fehler im Software des Zykliersystems. Die Messungen über die gesamten Kanäle konnte nicht in dem vorgegebenen einsekündigen Zeitraum vollständig abgearbeitet werden.

2. Einschwingen der Spannung bzw. des Stroms im Einschaltmoment des Relais (Umschalten nach der Ruhephase in den Entlade-/ Ladebetrieb) außerhalb der eingestellten sicherheitskritischen Spannungsgrenzen. Dieser Fehler trat sporadisch auf.

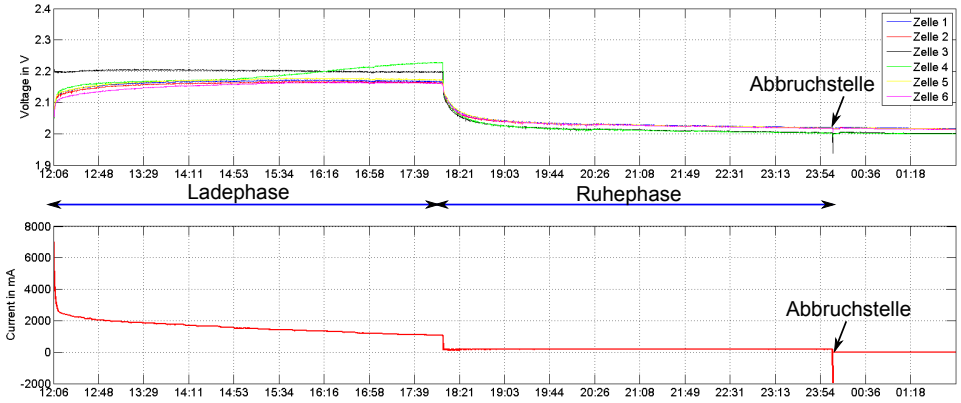


Abbildung 7.3.: Abbruch des Zyklierens im Einschaltmoment des Relais beim Umschalten des Zyklierbetriebs. Der Strom (bzw. Spannung) schwingt beim Umschalten von der Ruhephase in den nächsten Entladebetrieb stark ein und überschreitet dabei den eingestellten kritischen Wert. Die Zyklierung bricht sofort ab, jedoch bleibt die Messung erhalten, (übernommen und modifziert aus [43]).

3. Vorzeitiges Über- bzw Unterschreiten der eingestellten Lade- bzw Entladeschlussspannungen beim Lade- und Entladevorgang aufgrund einer stark gealterten Zelle.

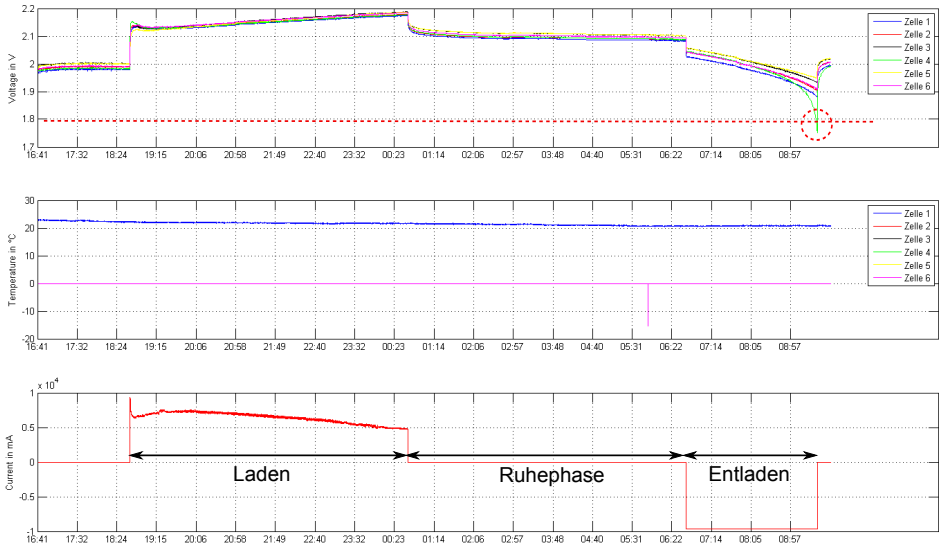


Abbildung 7.4.: Unterschreiten der eingestellten Entladeschlussspannung während der Entladephase. Anhand der Abbildung ist ersichtlich, dass die aufgezeichnete Spannung der Batteriezelle 4 (grün) schneller als bei den anderen Zellen die eingestellte untere Spannungsgrenze von $1{,}8\ V$ an der markierten Stelle erreicht. Die vorgesehene 6-stündige Entladephase wird dadurch vorzeitig abgebrochen. Dieses Verhalten ist durch die Alterung (SOH) der Zelle 4 begründet.

4. Frühzeitiges Unterschreiten der zulässigen eingestellten Entladeschlussspannung, da beim Entladen mehr Strom aus der Batterie entnommen als geladen wird. Dies tritt nach mehreren Zyklierungen auf.

Für weitere Messungen wurden diese erwähnten Fehler durch entsprechende Maßnahmen vermieden. Den ersten Fehler betreffend, wurde die Messdauer auf 3 Sekunden erhöht, da bei der Zyklierung der Bleibatterie eine sekündliche Auflösung am Zykliersystem nicht erforderlich ist (Spannung wird ebenfalls anhand der Zellensensoren gemessen). Die dritte Fehlerquelle ist durch die Balancierung der Zellen vor dem Zyklieren behandelt worden. In Bezug auf die unter Punkt 4 erwähnte Fehlerursache wurde eine Anpassung des Zyklierplans (z. B. mehrere aufeinander folgende Ladephasen) vorgenommen, die für ein Gleichgewicht der

Stromentnahme und Stromzufuhr sorgt. Für das unter Punkt 2 beschriebene Problem gibt es derzeit keine Lösung.

7.2. Auswertung

Im Folgenden werden die ausgewerteten Messdaten aus dem Zyklierbetrieb der Batterie untersucht. Insbesondere werden die Ergebnisse der optischen Messung auf deren zeitliches Verhalten und Abhängigkeiten gegenüber der Spannungs-, Strommessung und Ladung genauer analysiert.

7.2.1. Transmissionsänderung im Zyklierbetrieb

In der folgenden Abb. 7.5 werden die erfassten Transmissionsänderung gegenüber der Zellspannung dargestellt. Anhand des Zykliersystems wurde die präparierte Bleibatterie ca. drei Tage lang zykliert. Die Batterie wurde mit 14,4 V geladen und mit einem konstanten 5 A Strom entladen. Die Zyklusschritte wurden auf 6 Stunden Dauer eingestellt. Dieser Versuch wurde bei Raumtemperatur durchgeführt.

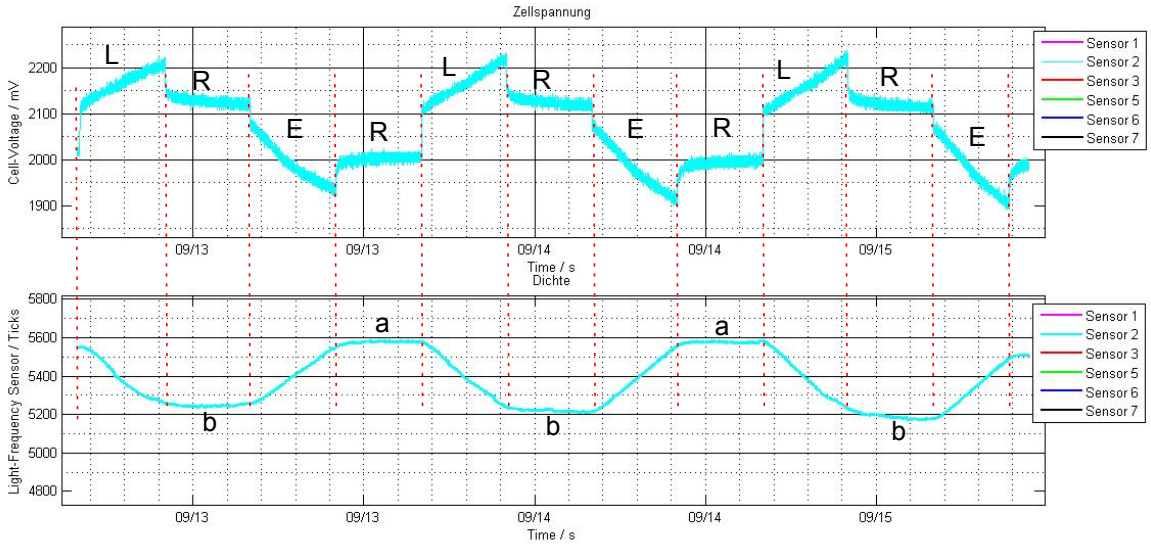


Abbildung 7.5.: Transmissionsänderung gegenüber der Zellspannung im Zyklierbetrieb der Zelle 2 (Sensor 2). Es ist zu erkennen, dass die Transissionsleistung bei einer Zellspannungsänderung beeinflusst wird. Während der Ladephasen (**L**) sinkt die Transmissionsleistung. Beim Entladevorgang (**E**) steigt dagegen die Transmission. Dieses Verhalten entspricht den theoretischen Erwartungen. Beim Entladen der Bleibatterie werden durch den Oxidationsprozess die Sulfationen der Säure verbraucht und Wasserstoffmoleküle gebildet. Es tritt eine Verdünnung der Schwefelsäure ein, so dass die Säurekonzentration bzw. die Dichte abnimmt und proportional dazu der Brechungsindex. Es tritt weniger Licht aus den Fasern heraus und damit steigt die Transmissionsleistung. Beim Ladevorgang werden durch den chemischen Reduktionsprozess die Schwefelsäuremoleküle wieder freigesetzt. Die Säuredichte steigt und dementsprechend der Brechungsindex. Es entstehen mehr Transmissionsverluste an den Biegestellen der Sensorsonden, so dass die Transmisssionsleistung deutlich fällt. Des Weiteren ist anhand der Abbildung erkennbar, dass die Spannungs- bzw die Transmissionsverläufe in den Ruhephasen (Ruhespannungsbereich) nach den Ladevorgängen (**a**) und den Entladevorgängen (**b**) sich bezüglich der Einschwingzeit unterschiedlich verhalten. In den Abschnitten (**a**) stellen sich die Verläufe schnell auf einen konstanten Wert ein. Es bildet sich ein plateauförmiges Verlauf (Sättigung). Dieses Verhalten ist durch Abschlammung (siehe 2.1.3.1 und 2.1.4.5) der Aktivmasse begründet. Praxisgemäß werden nur etwa 30 bis 40% des vorhandenen Aktivmaterials während des Entladevorgangs für den chemischen Prozess genutzt. Die Einschwingzeit ist nach einem Ladevorgang dagegen deutlich länger. Dieser Trägheit ist durch die Säureschichtung begründet, da diese Zeit für den Konzentrationsausgleich des Elektrolyten innerhalb der Batteriezelle benötigt wird, um das Gleichgewicht zu erreichen. Dieses Verhalten ist reproduzierbar.

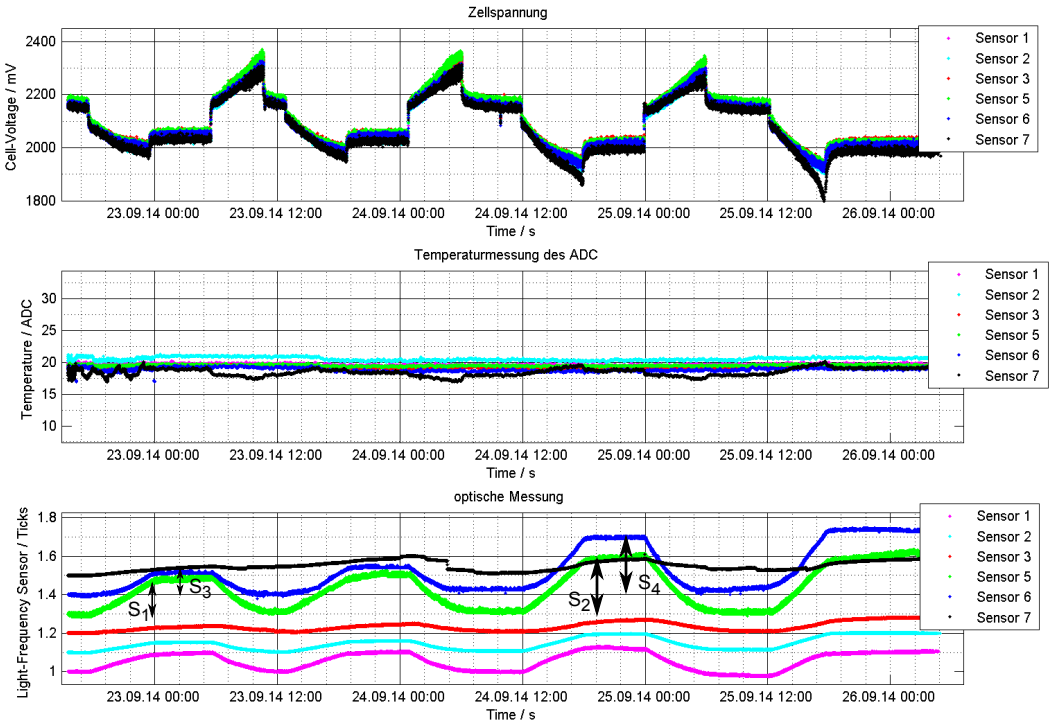


Abbildung 7.6.: Signalhub der Transmissionsleistung in Abhängigkeit von dem Ladegrad der Batteriezellen. Es ist erkennbar, dass der Signalhub der Transmission je nach Ladezustand sich ändert. Mit S_1 und S_2 ist der Signalhub der Transmissionsleistung des Sensors 5 (Grün) bei jeweils 10% und 15% Entladung (bezogen auf die Anfangsruhespannung) gekennzeichnet. Dieser entspricht respektive ca 20% und 30% (bezogen auf den Anfangswert der Transmissionsleistung). Ein ähnliches Verhalten ist ebenfalls bei den Rest der Sensoren (z. B. die Änderung von S_3 und S_4 bei Sensor 6 in Blau dargestellt) beobachtbar. Sensor 7 (schwarze Kennlinie) weist ein undefiniertes Verhalten auf.

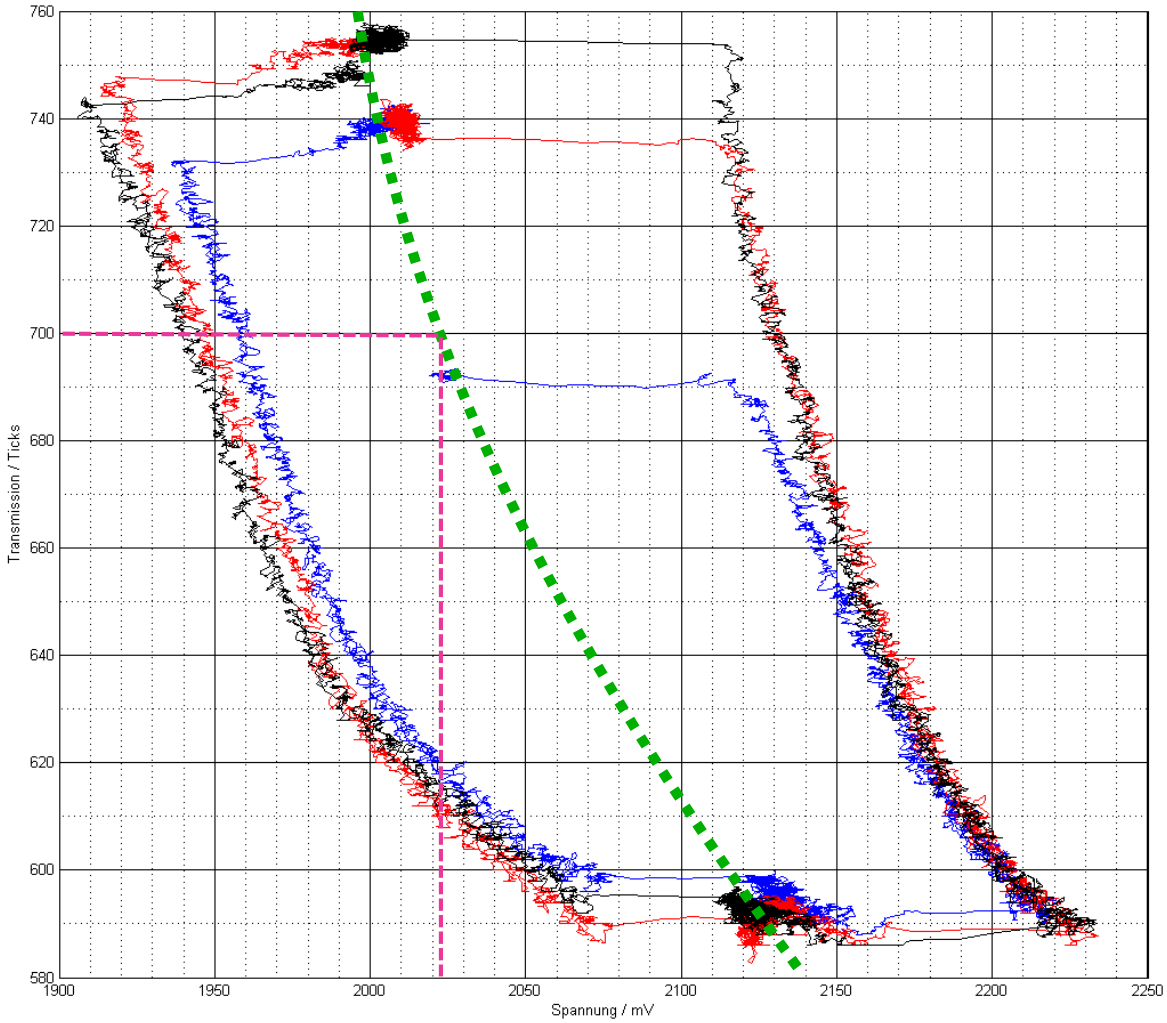


Abbildung 7.7.: Transmissionsleistung in Abhängigkeit von der Zellspannung. Der Verlauf der Ruhespannungen ist in Grün dargestellt. Anhand dieser Darstellung kann ein beliebiger Wert der optischen Messung dem dazugehörigen Ruhespannungswert zugewiesen werden. Ein Beispiel dieses Verfahrens ist in der Abbildung in Pink dargestellt. Dadurch kann der Ladezustand der Batteriezelle anhand der optischen Messung bestimmt werden.

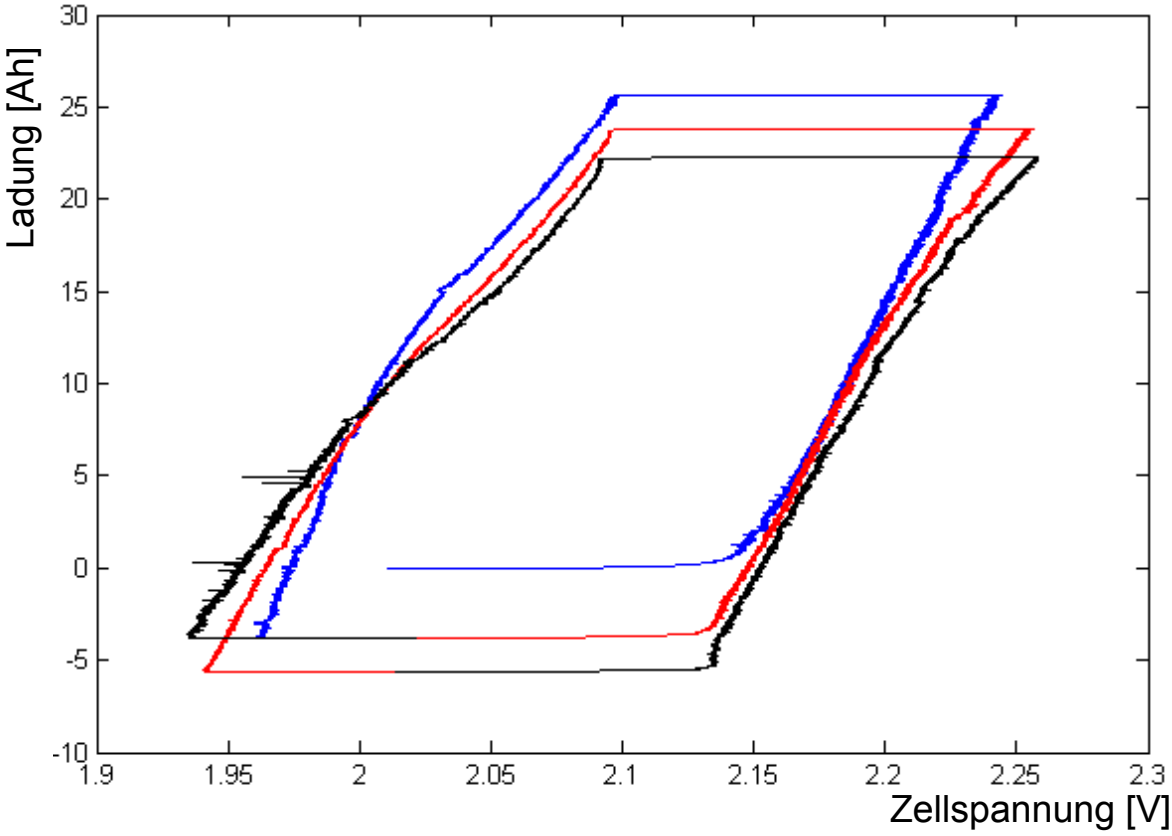


Abbildung 7.8.: Ladung in Abhängigkeit von der Spannung. Der Nachteil dieser Darstellung und insgesamt der Strommessung ist, dass die Ströme bei Ruhespannungen gleich Null sind und somit kann dort der Zusammenhang mit dem Ladezustand nicht nachvollzogen werden.

7.2.2. Analyse des Zeitverhaltens der optischen Messung

Im Folgenden werden die Signalverläufe der Transmissinsleistung auf ihr Zeitverhalten bezüglich einer Spannungsänderung im Zyklierbetrieb untersucht. Folgende Abbildungen stellen Abschnitte der optischen und Spannungsmessung des Sensors 6 dar, die beim Versuch am 12.09.14 durchgeführt wurden. Die Signale der Transmissionsleistung (optische Messung) sind auf deren Anfangswert normiert worden und zwecks einer besseren Auswertbarkeit mit einem Offset versehen worden.

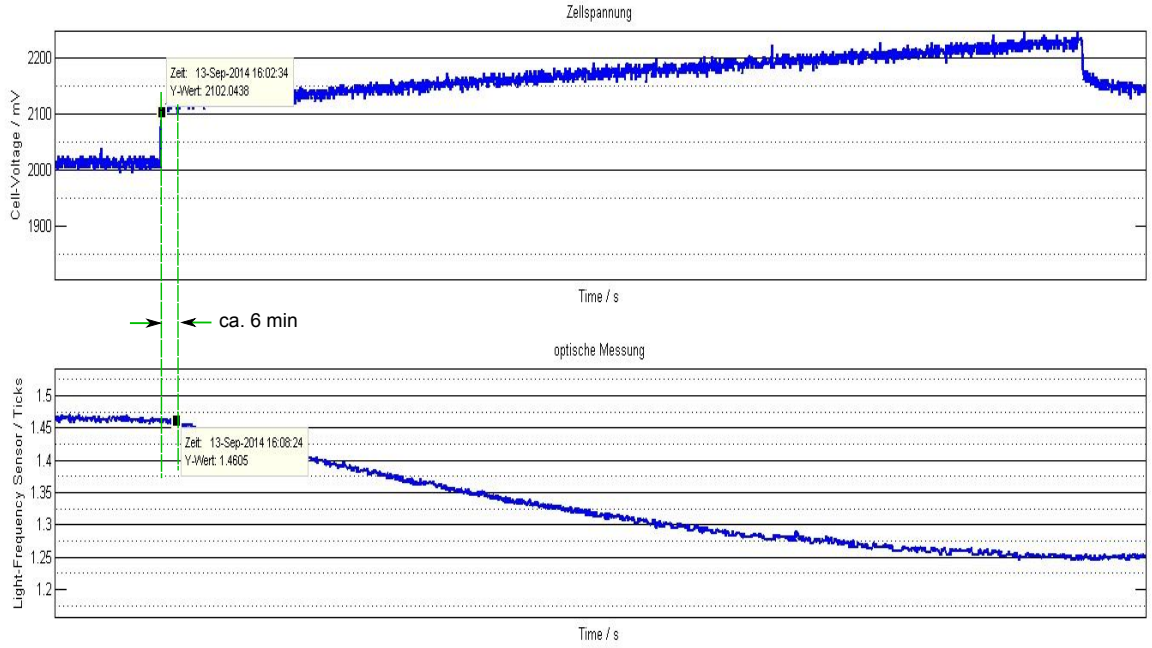


Abbildung 7.9.: Signalantwort der Transmissionsleistung auf Spannungsänderung am Anfang des Ladevorgangs. Die Verzögerung der Transmissionsänderung beträgt ca. 6 min. Diese geringe Verzögerung kann durch die Position der Sensorsonden, die sich außerhalb der Reaktionszone sich befindet (siehe 3.0.5), erklärt werden. Die Schwefelsäuremoleküle werden zuerst in den Poren der Platten gebildet und anschließend findet ein Dichteausgleich mit der dünneren Säure zwischen den Platten statt.

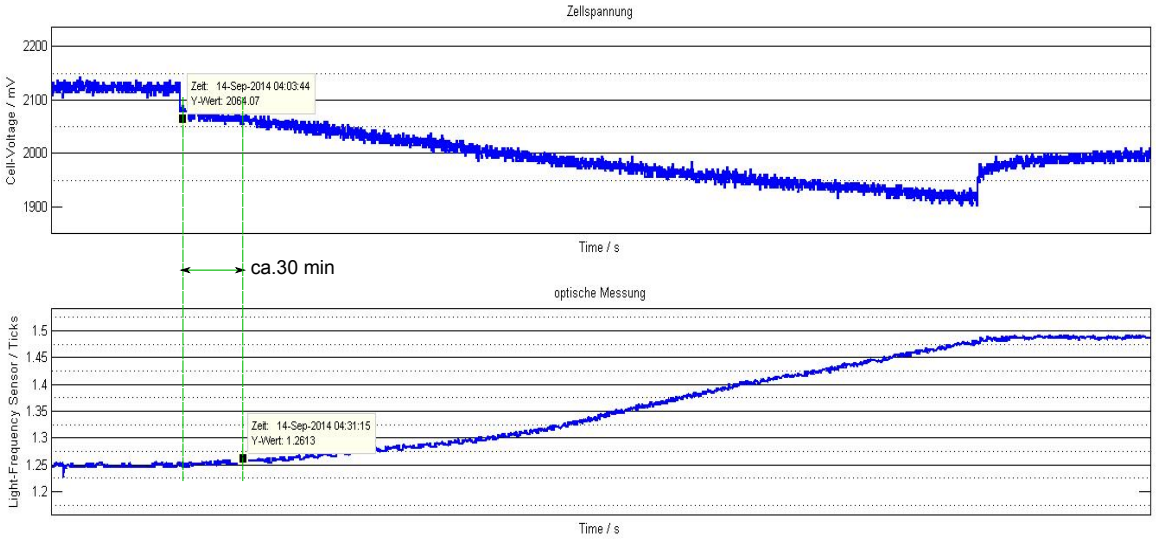


Abbildung 7.10.: Signalantwort der Transmissionsleistung auf Spannungsänderung am Anfang des Entladevorgangs. Eine Änderung der Transmissionleistung nach dem Start des Entladevorgangs ist erst nach ca. 30 Minuten erkennbar. Die Trägheit der Dichteänderung des Elektrolyten am Anfang der Entladung ist durch die chemische Reaktion bedingt. Für die Bildung des Bleisulfats durch die Oxidation werden Blei-Ionen benötigt. Diese sind schwer löslich und lagern sich an den Poren der Elektroden ab. Ferner wird durch die Bildung der Bleisulfatmoleküle an der postiven Platte, die ein großes Volumen haben, die Porosität der Elektroden reduziert und führt damit zu einer Behinderung der Transportvorgängen von Blei und Bleidioxid. Durch die Gitterkorrosion und Abschlammung wird der chemische Zersetzungsprozess zusätzlich verlangsamt (siehe 2.1.3).

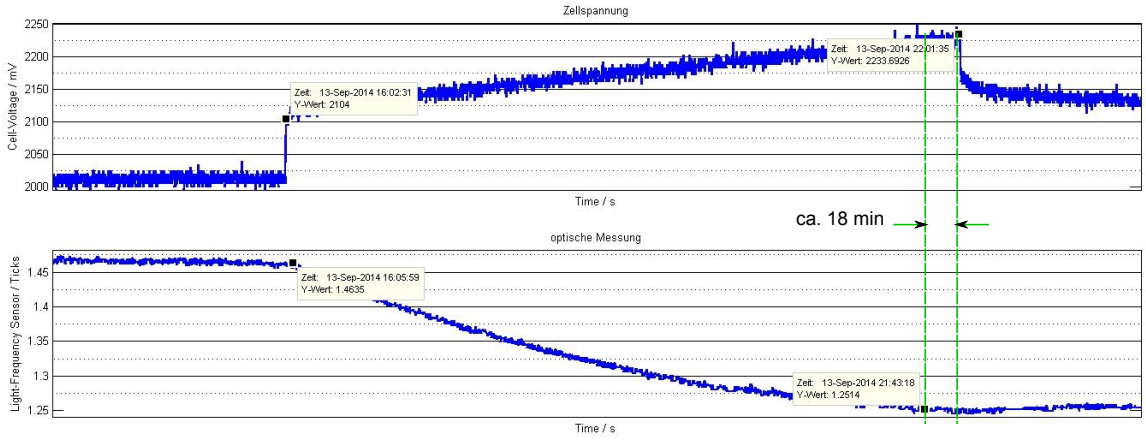


Abbildung 7.11.: Verhalten der Transmissionsleistung am Ende des Ladevorgangs. Es ist zu beobachten, dass am Ende der Ladephase (ca. 18 min vor dem Abschalten) die Transmissionsleistung nicht mehr weiter sinkt trotz Erhöhung der Spannung. Beim Erreichen von hohen Ladezuständen werden die Pb^{2+}-Ionen fast vollständig verbraucht. Der Diffusionsgrenzstrom wird erreicht und die Zurückwandlung des Bleisulfats in Aktivmaterial kann, trotz Erhöhung der Ladespannung, nicht mehr erfolgen.

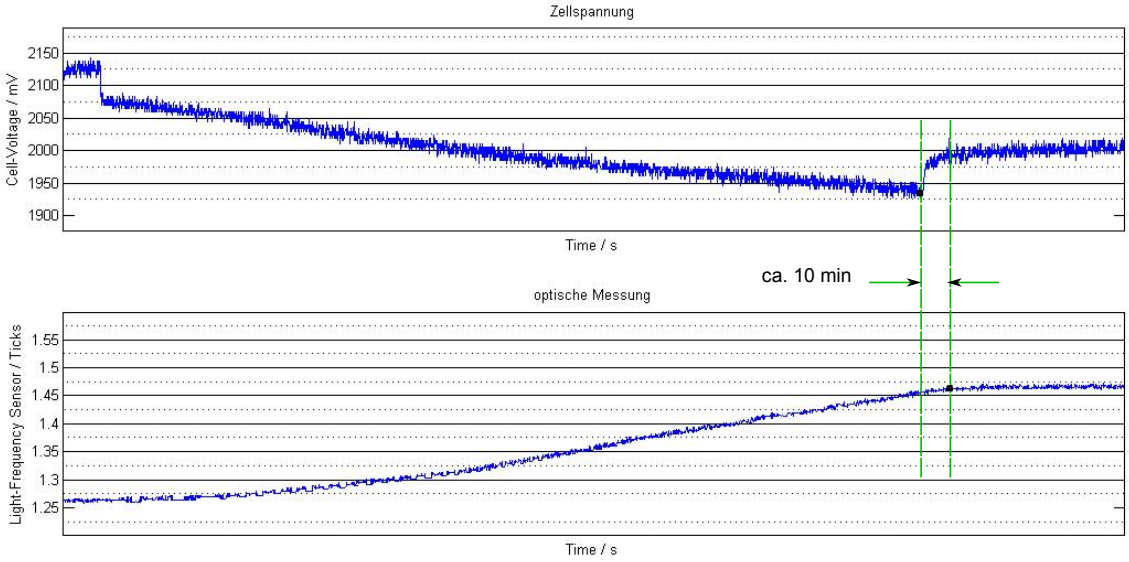


Abbildung 7.12.: Verhalten der Transmissionsleistung am Ende des Entladevorgangs. Es ist erkennbar, dass nach der Beendigung der Entladephase die Transmissionsleistung um weitere ca. 10 Minuten steigt bis ein stabiler Wert erreicht wurde. Diese Verzögerung ist durch den Dichteausgleichprozess in der Zelle begründet.

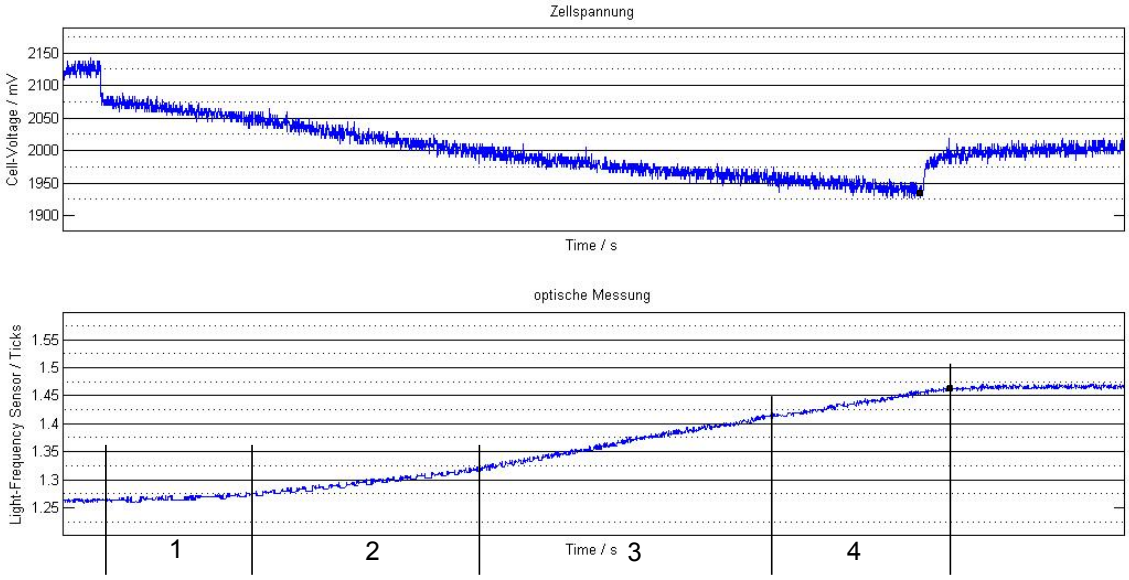


Abbildung 7.13.: Signalverlauf der Transmissionsleistung während das Entladevorgang. Der Verlauf der Transmissionsänderung in Abhägigkeit der Spannung beim Entladen kann in vier Abschnitte unterteilt werden. Diese sind durch unterschiedliche Steigungen charakterisiert. Im ersten Abschnitt (**1**) ist der Verlauf deutlich flach und die Transmission steigt nur minimal an. Im zweiten Abschnitt (**2**) ist ein Anstieg der Transmission erkennbar. Im dritten Abschnitt (**3**) ist der Verlauf nahezu linear und fällt in den letzten Abschnitt (**4**) gegen Ende der Entladephase wieder ab. Dieses Verhalten hängt stark von den chemischen Prozessen, dem Entladestrom und dem SOH der Batteriezelle ab.

7.2.3. Auswertung der Langzeitmessungen im Zyklierbetrieb

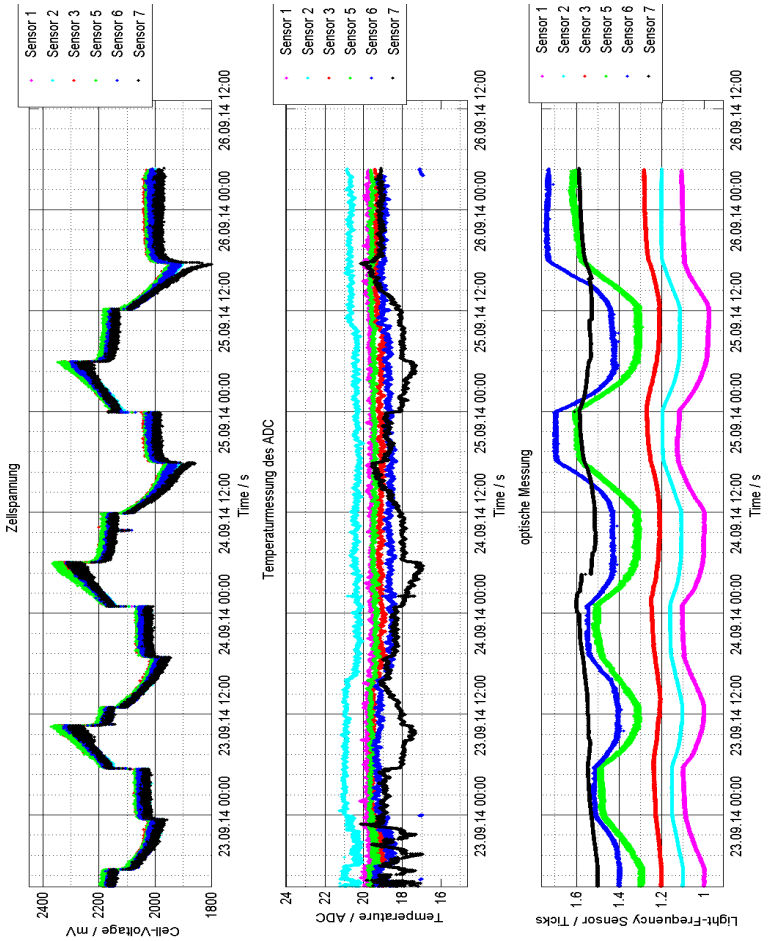


Abbildung 7.14.: Zykliermessung 1: Die Abbildung zeigt die Messung vom 22.09 bis zum 26.09.2014. Die Zyklierung wurde abgebrochen, da die kritische eingestellte Spannung von 1,8 V bei Sensor 6 unterschritten wurde. Die Zelle weist einen Kapazitätsverlust auf. Die Verläufe der Optischen Messsignalen zeigen eine Abhängigkeit zwischen Signalhub und Ladegrad der Zelle. Sensor 6 weist Störungen auf die durch den Stark alternden Zustand der Zelle begründet werden können.

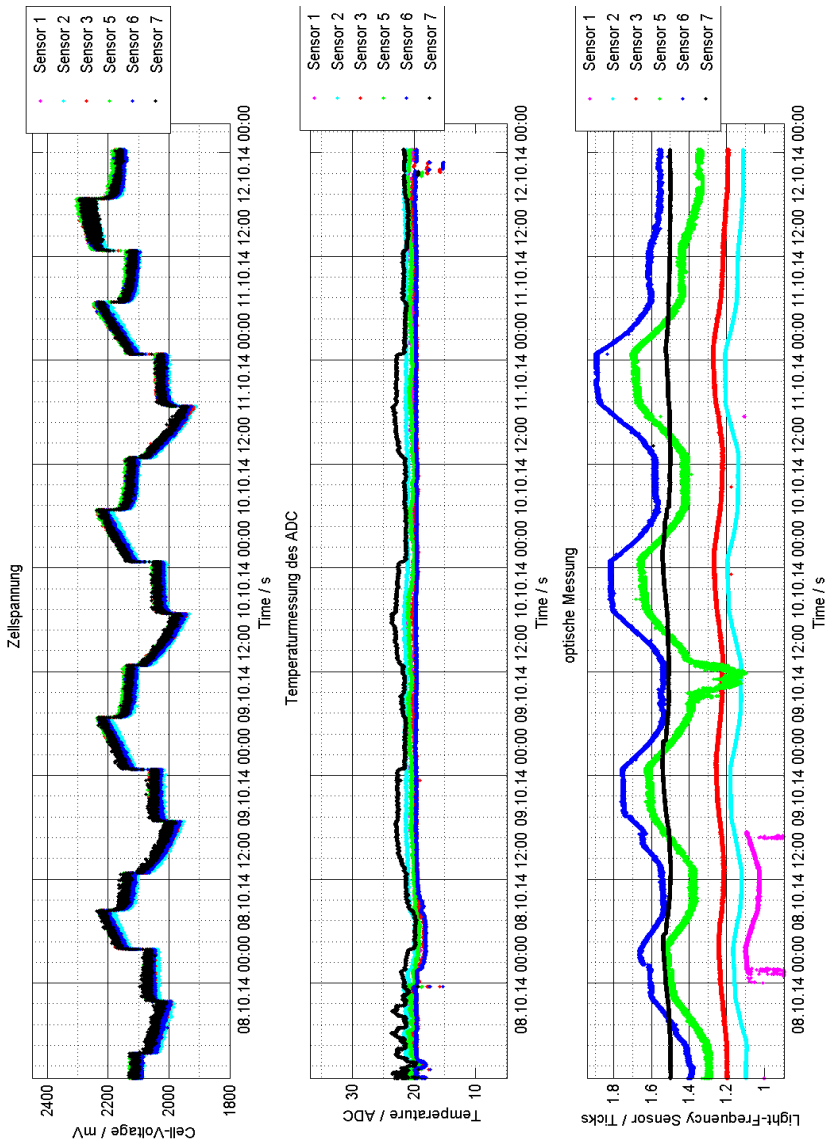


Abbildung 7.15.: Messung vom 07.10 bis 12.10.2014. Es sind Störungen der optischen Signale bei Sensor 5 (Blau), 5 (Grün) und 1 (Pink) erkennbar. Die Gasblasenbildung könnte in diesem Fall nicht die Ursache für diese Störungen sein, da dieser Effekt schließlich beim Laden auftritt. Eine Streuung der Signalstärke ist ebenfalls ersichtlich. Diese ist hauptsächlich durch mechanische Ausführung der Steckverbindungssysteme begründet. Darüber hinaus ist die Abhängigkeit des Signalhubs von dem Ladegrad reproduzierbar.

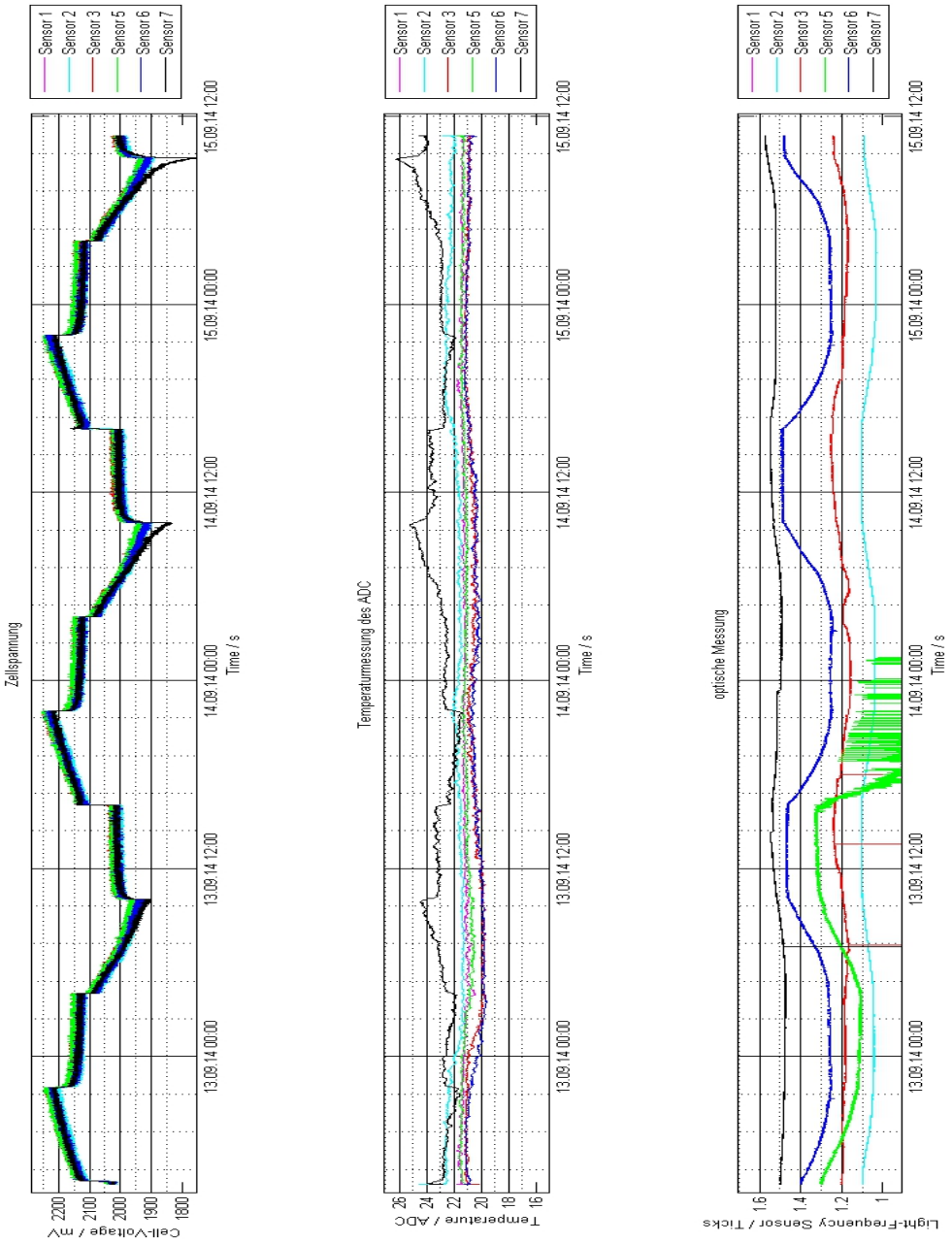


Abbildung 7.16.: Messung vom 12.09 bis 15.09.2014. Die Übertragung des optischen Signals vom Sensor 5 (Grün) ist nach starken Rauschen unterbrochen wurde. Diese kann durch eine Fehlfunktion der Sende-LED oder des Licht-Frequenz-Umsetsers erklärt werden. Das Umgebungslicht kann als Störeffekt der optischen Messung des Sensors 3 (Rot) ausgeschlossen werden, da sonst ebenfalls die benachbarten Zellen beeinflusst wären.

7.2.4. Auswertung der Messung mit geänderter Hardware

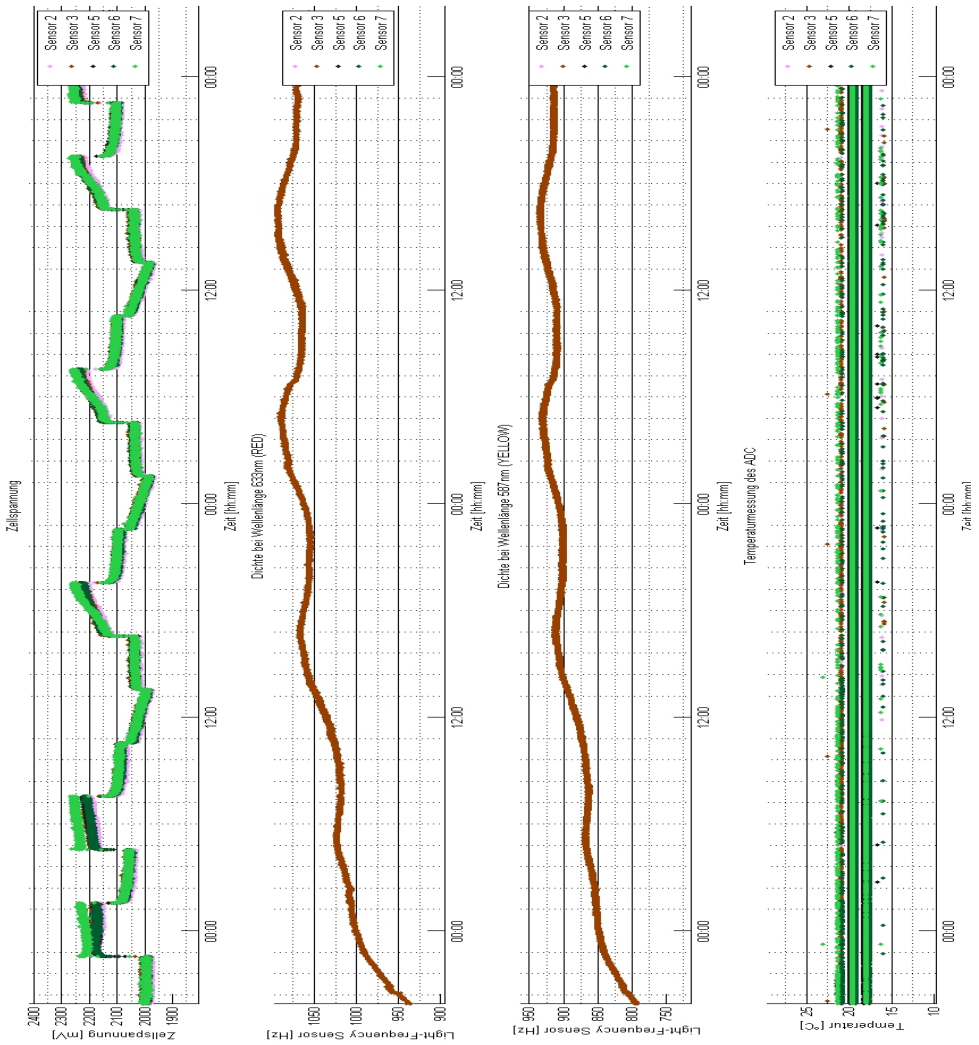


Abbildung 7.17.: Messung im Zyklierbetrieb mit geänderter Hardware (2 LEDs unterschiedlicherwellenlängen). In Braun sind die Verläufe der optischen Messung für Sensor 3 mit den Wellenlängen 633 nm und 587 nm dargestellt. Eine Änderung in Abhängigkeit von der Spannung lässt sich bei beiden Verläufen beobachten. Jedoch ein unterschiedliches Verhalten ist nicht erkennbar. Dieses könnte dadurch erklärt werden, dass die untersuchten Wellenlängen zu nah beieinander liegen. Untersuchungen auf Zusammenhänge mit anderen Wellenlängenbereiche, wie z. B den Infrarot Bereich, konnten in dieser Arbeit aus Zeitgründen nicht mehr durchgeführt werden.

8. Fazit

Abschließend sollen in diesem Kapitel die in dieser Arbeit durchgeführten Aufgaben zusammengefasst und die Erzielten Ergebnisse bewertet werden. Zuletzt werden Ansätze für weitere Optimierungsmaßnahmen dargelegt.

8.1. Zusammenfassung und Bewertung erzielter Ergebnisse

Aufbauend auf die Erkenntnisse und Untersuchungen aus der Arbeit von Nasimzada [9] wurden im Rahmen dieser Bachelorarbeit folgende Aufgaben zur Optimierung, Unterdrückung von Störeinflüssen und Erweiterung des bestehenden optischen Sensors für die Ladezustandsbestimmung von Bleibatterien durchgeführt.

- Zuerst wurde eine neue Sensorsonde zum Einbringen der POF-Faser entwickelt. Die mechanische Konstruktion des 3D-Modells wurde von Herrn Nasimzada im Rahmen des BATSEN-Projekts realisert und anschließend mittels der 3D-Drucktechnologie gefertigt. Dadurch ist eine gute Reproduzierbarkeit der Fertigung mit minimalen Streuungen und eine mechanische Stabilität gewährleistet. Im Laufe der Entwicklung sind unterschiedliche Prototypen entstanden, die ständig untersucht und weiter optimiert wurden. Durch die Erhöhung der Anzahl der Biegestellen und der Nutzlänge der Sensorsonde kann der Einfluss der Säureschichtung in der Zelle dank der differentiellen Messungen in verschiedenen Höhen kompensiert werden.
- Für die Messerprobungen der entwickelten Sensoren wurde eine Starterbatterie aufbereitet. Dabei wurde darauf geachtet, dass die Positionierung der Sensorsonden möglichst außerhalb der Reaktionszone in der Batteriezelle liegt, um Störeffekte auf die optische Messung durch Gasblasenbildung zu vermeiden. Das Auftreten solche Effekte konnte in der Auswertung der Messdaten nicht beobachtet werden.
- Für die Unterdrückung der Offsetverfälschung des optischen Messsignals, verursacht durch das Umgebungslicht, wurde ein Messverfahren welches auf einer Differenzmessung beruht, entwickelt. Dadurch können Fremdlichtanteile, die bei Änderung von Lichtverhältnissen in der Umgebung auftreten, ausgefiltert werden. Die Umsetzung

dieses Verfahrens ist für niederfrequente Lichtstörsignale bis zu einer Grenzfrequenz von $0{,}5Hz$ gewährleistet und wurde unter Versuchslaborbedingungen und anhand einer Vergleichsmessung gegenüber Sensoren mit dem alten Softwarestand erfolgreich getestet. Nachteil dieses Verfahren ist, dass hochfrequente Störsignale nicht vollständig unterdrückt werden können. Dies kann jedoch mit Hilfe von Modulationsverfahren erzielt werden. Allerdings wäre die Realisierung solcher Verfahren mit hohem Rechenaufwand, komplexer Softwareimplementierung und Hardwareänderung verbunden.

- Durch den gepulsten Einschaltbetrieb und die realisierten langen Abkühlzeiten der Sende-LEDs, wurde der Temperatureinfluss auf die Lichtintensität minimiert. Im Betrieb bei hohen bzw. tiefen Temperaturen muss die Temperaturabhängigkeit der Lichtstärke der LEDs berücksichtigt werden. Dieses wurde in dieser Arbeit nicht betrachtet, da die Messungen schließlich bei Raumtemperatur durchgeführten wurden.

- Zwecks Untersuchung von Kaliebrierungsmöglichkeiten der optischen Messung wurde das Verhalten der Sensoren bei unterschiedlichen Säurekonzentrationen untersucht. Aus den Ergebnissen dieser Kalibriermessung konnte ein linearer Zusammenhang der Transmissionsleistung mit der Konzentrationsänderung nur bei zwei von sechs untersuchten Sensoren beobachtet werden. Dieses Ergebnis ist durch die mechanische Instabilität des bearbeiten TOSLINK-Steckverbindungssystems zwischen Faser und DSM begründet. Eine erneute Untersuchung sowie eine Umsetzung eines Kalibrierverfahrens wurde im Rahmen dieser Arbeit aus Zeitgründen nicht mehr umgesetzt.

- Aufgrund der Untersuchung der Transmissionsleistungsänderung in Abhängigkeit unterschiedlicher Wellenlängen der Sende-LED, wurde der Dichte-Sensor-Modul mit einem zusätzlichen LED-Treiber und einer zweifarbigen LED ausgestattet. Dadurch wurde ein alternierendes Umschalten zwischen beiden LEDs sowie die Auswertung der Ausgangssignale ermöglicht. Dementsprechend wurde der Zellensensor ebenfalls erweitert.

- Gemäß der Hardwareänderung wurde zuerst die Software des Zellensensors angepasst. Zusätzlich wurde die Differenzmessung implementiert. Die Kalibrierung der Messgrößen Temperatur und Spannung wurde modular in der MATLAB-Software realisiert. Dadurch wird die Verfälschung der Messdaten, verursacht durch die in dem Steuergerät nicht mehr aktuellen Kalibrierparameter, in der Kalibrierung berücksichtigt.

- Für die Kalibrierung der Temperatur, Spannungsmessung sowie die Korrektur der Temperaturabhängigkeit der Spannung auf den Zellensensoren wurde ein Kalibrierverfahren, basierend auf der Methode der kleinsten Quadrate realisiert. Die Regressionsparameter werden mittels eines implementierten MATLAB-Skriptes ermittelt und für die Weiterverarbeitung in einer externen Datei gespeichert.

- Für die Funktionserprobung der entwickelten Sensoren auf der präparierten Starterbatterie wurde ein zeitgesteuerter Zyklierplan mit dem Zykliersystem konzipiert. Dieser wurde je nach SOH der Batteriezellen angepasst. Anhand der realisierten äquidistanten Zyklierschritte kann das Zeitverhalten der gemessenen Größen genauer analysiert werden
- Abschließend wurde bei der Auswertung der Messdaten aus dem Zyklierbetrieb die Änderung der Transmissionsleistung in Abhängigkeit von dem Ladezustand der Zellen erfolgreich beobachtet. Zusätzlich konnte ein Zusammenhang zwischen dem Signalhubs der optischen Signale und dem Ladegrad der Batteriezelle festgestellt werden. Darüber hinaus wurden unterschiedliche zeitliche Verhalten der optischen Signale je nach Zykliervorgang beobachtet.

  Die Wellenlängenabhängigkeit der Transmissionsleistung konnte leider anhand der einmaligen durchgeführten Messung mit der geänderten Hardware nicht beobachtet werden. Diese ist teilweise durch die mechanische Instabiltät des Steckverbindungssystems begründet.

8.2. Ausblick

An dieser Stelle werden abschließend Ansätze und Optimierungsmaßnahmen, die innerhalb dieses optischen Messverfahrens verwendet werden können, geschildert.

- Optimierung der mechanischen Stabilität des Steckverbindungssystems zwischen Faser und DSM. Im Laufe dieser Arbeit wurde festgestellt, dass der Mangel an mechanischer Stabilität an dieser Stelle die Hauptursache für die Verfälschung der optischen Messungen ist. Für den Einsatz z. B. von SMA[1]-Steckern, die eine mechanische Robustheit gegenüber die in dieser Arbeit verwendeten TOSLINK-Stecker aufweisen, wurden bereits die Grundlagen dafür auf den neu entwickelten DSM bereitgestellt.
- Einsatz von Modulationsverfahren als effizienteres Verfahren für die Unterdrückung von hochfrequenten Fremdlichtstörsignalen.
- Anhand der Analyse des zeitlichen Verhaltens der optischen Signale können mittels geeignete Referenzverfahren Alterungseffekte wie z. B. die Sulfatation und Gitterkorrosion der Batteriezelle frühzeitig erkannt werden.
- Einbringung einer zusätzlichen Referenzfaser in die Batteriezelle. Diese müsste mit demselben Sender und Empfänger der Messfaser z. B. durch ein Split-Modul verbunden werden und somit die gleichen Rahmenbedingungen (Temperaturschwankungen,

[1] *engl. Sub-Miniature-A, wird überwiegend als Hochfrequenz-Steckverbinder eingesetzt.*

Änderung der Lichtverhältnisse) ausgesetzt werden. Jedoch dürfte die Referenzfaser nicht mit dem Elektrolyten kontaktiert werden und dadurch die optischen Signale nicht durch die Dichteänderung beeinflusst werden. Störsignale und temperaturabhängige Änderungen könnten dadurch identifiziert und ausgefiltert werden.

- Untersuchung der spektralen Absorptions- und Reflexionseigenschaften des Elektrolyten bei unterschiedlichen Ladezuständen für verschiedene Wellenlängen. Besonderheiten könnten für weitere Referenz- bzw. Kalibrierverfahren genutzt werden.
- Fortführung der Kalibriermessung der optischen Signale und Entwicklung eines geeigneten Kalibrierungsverfahren, insbesondere für die Kompensation der Temperaturabhängigkeit des Elektrolyten.
- Berücksichtigung der Temperaturabhängigkeit der Lichtintensität der LEDs durch ein Kompensationsverfahren, welches in der Kalibrierungssoftware implementiert werden könnte.
- Überarbeitung der Steuergerätsoftware und Deaktivierung der Kalibrierung.
- Die Erfassung der Messdaten könnte mittels ein C-Programms realisiert werden. Die verwendete MATLAB-Software ist bei langen Messungen sehr anfällig und stützt öfters während der Messwertaufnahme ab.

Literaturverzeichnis

[1] A. Jossen and W. Weydanz, *Moderne Akkumulatoren richtig einsetzen*. Inge Reichardt Verlag, 2006.

[2] L. Xue-Long, “Global lead-acid battery market development status,” 2011. [Online]. Available: http://investtaiwan.nat.gov.tw/news/ind_news_eng_display.jsp?newsid=64

[3] A. Info-Service, “Allgemeiner deutscher automobil-club e.v. pannenstatistik 2014,” 2014. [Online]. Available: http://www.adac.de/infotestrat/unfall-schaeden-und-panne/pannenstatistik/

[4] H. W.Meyer, “Frankfurter allgemeine autobatterie das märchen von der wartungsfreiheit,” 2013. [Online]. Available: http://www.faz.net/aktuell/technik-motor/auto-verkehr/autobatterie-das-maerchen-von-der-wartungsfreiheit-12514662.html

[5] K.-R. Riemschneider and M. Schneider, “Drahtlose sensoren in den zellen von fahrzeug-batterien,” 2011. [Online]. Available: http://www.haw-hamburg.de/fileadmin/user_upload/TI-IE/Daten/Docs/ESZ-ASP/IWKM21_Batsen.pdf

[6] D. Rosenmeyer, “Ausgeglichene energiebilanz und erhöhte zuverlässigkeit energiemanagement mit dem intelligenten batteriesensor ibs,” 2005. [Online]. Available: http://www.plastverarbeiter.de/ai/resources/1d82eaada52.pdf

[7] R. Kube, “Drahtloses sensornetzwerk für fahrzeugbatterien-kanal, antennen und fehlerraten,” Masterthesis, Hochschule für Angewandte Wissenschaften Hamburg, 2011. [Online]. Available: http://edoc.sub.uni-hamburg.de/haw/volltexte/2012/1537/pdf/MasterarbeitEndversionKube.pdf

[8] A. M.Cao-Paz and J. M. Acevedo, “A multi-point sensor based on optical fiber for the measurement of electrolyte density in lead-acid batteries,” 2010. [Online]. Available: http://www.mdpi.com/1424-8220/10/4/2587/pdf

[9] W. Nasimzada, “Hard- und softwareentwicklung eines lichtleiter-sensors für die optische analyse des elektrolyten von bleibatterien,” Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg, 2013. [Online]. Available: http://edoc.sub.uni-hamburg.de/haw/volltexte/2014/2649/pdf/Wahid_Nasimzada.pdf

[10] E. Witte, *Blei-und Stahlakkumulatoren Eigenschaften und Anwendung Dritte Auflage*. Krausskopf-Verlag Mainz, 1967.

[11] S. Bräuninger, "Bleibatterien: Möglichkeiten und grenzen eines altbewährten batteriesystems," 2010. [Online]. Available: http://archiv.aktuelle-wochenschau.de/2010/w42/woche42.html

[12] V. consumer Batteries, "Varta batterie-lexikon," 2006. [Online]. Available: http://www.varta-consumer.de/~/media/Files/Local/de-DE/varta-batterielexikon-5-de.ashx

[13] W. Bludau, *Lichtwellenleiter in Sensorik und optischer Nachrichtentechnik*. Springer-Verlag Berlin Heidelberg, 1998.

[14] H. Vortisch, "Beobachtung von phasenübergängen in einzeln levitierten schwefelsäuretröpfchen mittels raman-spectroskopie und elastischer lichtstreuung," 2002. [Online]. Available: http://www.diss.fu-berlin.de/diss/receive/FUDISS_thesis_000000000808

[15] "Oxidation," 2001. [Online]. Available: http://www.chemie.de/lexikon/Oxidation.html

[16] A. Wiedmann, "Aufbau und funktionsweise des bleiakkumulators." [Online]. Available: http://www.resulf.de/pdf/bleiakkus.pdf

[17] D. Berndt, *Maintenance-free batteries : based on aqueous electrolyte lead-acid, nickel/-cadmium, nickel/metal hydride : a handbook of battery technology /*. Baldock, Hertfordshire, England : Research Studies Press ; Philadelphia, PA : Institute of Physics„ 2003.

[18] "Oxidation," 2001. [Online]. Available: http://www.chemie.de/lexikon/Oxidation.html

[19] P. kurzweil and P. Scheipers, *Chemie Grundlagen, Aufbauweisen, Anwendungen und Experimente 9.Auflage*. Vieweg+Teubner Verlag| Springer Fachmedien Wiesbaden GmbH, 2012.

[20] "Elektrolyse," 2001. [Online]. Available: http://www.chemie.de/lexikon/Oxidation.html

[21] T. Handschuh, "Untersuchung des betriebs- und alterungsverhalten von bleisäure-akkumulatoren bei hybridantriebssysteme typischen belastungen," Dissertion, Universität Ulm, 2007. [Online]. Available: http://vts.uni-ulm.de/docs/2007/5938/vts_5938_7953.pdf

[22] D. Heinemann, "Strukturen von batterie- und energiemanagementsystemen mit bleibatterien und ultracaps," Dissertion, Fakultät IV - Elektrotechnik und Informatik der Technischen Universität Berlin, 2007. [Online]. Available: opus4.kobv.de/opus4-tuberlin/files/1429/Heinemann_Detlef.pdf

[23] "Technik-tipps: Usv full calibration test," 2014. [Online]. Available: http://www.online-usv.de/de/news/2014/nl-2014-04-15.php

[24] H. Frank and Phrood, “Electromagnetic wave spectrum,” 2008. [Online]. Available: http://commons.wikimedia.org/wiki/File:Electromagnetic_spectrum_c.svg

[25] ITWissen, “Das große online-lexikon für informationstechnologie,” 2014. [Online]. Available: http://www.itwissen.info/

[26] P. Schiepel, “Messzelle für die optoelektronische analyse von lithium-ionen-batterien,” Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg, 2014. [Online]. Available: http://edoc.sub.uni-hamburg.de/haw/volltexte/2014/2684/pdf/Bachelorthesis_Schiepel.pdf

[27] A. Goldbacher, “Grundlagen der optischen sensormessung mit faser-bragg-gittern,” 2011. [Online]. Available: http://www.elektroniknet.de/messen-testen/sonstiges/artikel/77016/

[28] O. T. die verbindet, “optische spleiss und messtechnik.” [Online]. Available: http://www.opternus.de/wissen/glasfasertypen.html

[29] I. D. große online Lexikon für Informationstechnologie, “Polymerfaser pof(plastic optical fiber),” 2014. [Online]. Available: http://www.itwissen.info/definition/lexikon/Polymerfaser-POF-plastic-optical-fiber.html

[30] C. Chemapedia, “Rayleigh-streuung,” 2014. [Online]. Available: http://www.chemgapedia.de/vsengine/glossary/de/rayleigh_00045streuung.glos.html

[31] OSRAM, “Multi topled enhanced optical power led,” 2012. [Online]. Available: http://www.mouser.com/ds/2/311/LSY%20T67B%20-%20TOPLED-318390.pdf

[32] S. Giglberger, “Lock-in-verstärker,” 2009. [Online]. Available: http://www.physik.uni-regensburg.de/studium/praktika/a2/download/versuch5a.pdf

[33] S. Ilgin, “Drahtlose sensoren für batteriemodule - konzeption, kalibrierung, hard- und softwareentwicklung,” Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg, 2011.

[34] M. Meinzer, “Hard- und softwareentwicklung sowie erprobung drahtloser zellensensoren für fahrzeugbatterien,” Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg, 2013. [Online]. Available: http://edoc.sub.uni-hamburg.de/haw/volltexte/2014/2268/

[35] M. S. Albert, “In vivo validierung eines neuen verfahrens zur pulsoxymetrie im niedrigen sauerstoffsättigungsbereich,” Dissertion, Ludwig-Maximilians-Universität München, 2008. [Online]. Available: http://edoc.ub.uni-muenchen.de/8888/1/Albert_Maik.pdf

[36] J. D. Weiss, “Optical state-of-charge monitor for batteries,” 1999. [Online]. Available: http://www.google.com/patents/US5949219

[37] T. Instruments, “Mixed signal microcontroller msp430g2x53,” 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430g2553.pdf

[38] ——, “Regulated 3.3 v, 100-ma low-ripple charge pump low power dc/dc converters,” 2000. [Online]. Available: http://www.ti.com/lit/ds/symlink/tps60201.pdf

[39] S. Labs, “Si4012 crystal-less fsk/ook rf transmitter,” 2013. [Online]. Available: http://www.silabs.com/Support%20Documents/TechnicalDocs/Si4012.pdf

[40] T. Instruments, “Adjustable led driver,” 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/tl4242-q1.pdf

[41] TAOS, “Programmable light-to-frequency converters,” 2007. [Online]. Available: http://www.mouser.com/ds/2/588/TSL230RDTSL230ARDTSL230BRD-P-198494.pdf

[42] S. Püttjer, “Diagnosefunktion für automobil-starterbatterien,” Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg, 2011.

[43] M. Kusche, “Aufbau und inbetriebnahme eines zykliersystems für batteriezellen mit arm-mikrocontroller,” Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg, 2014.

A. Aufgabenstellung

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Hochschule für Angewandte Wissenschaften Hamburg
Department Informations- und Elektrotechnik
Prof. Dr.-Ing. Karl-Ragmar Riemschneider

3. Juni 2014

Bachelorthesis Maher Achour

Entwicklung eines Lichtleiter-Sensors für die optische Ladezustandsbestimmung von Bleibatterien

Motivation

Im Rahmen von Forschungsprojekten an der HAW Hamburg und der Graduiertenschule 'Key Technologies for Sustainable Energy Systems in Smart Grids' sollen Batteriesensoren entwickelt werden. Für die wirtschaftliche Ausnutzung der Batterie und die Prognose der Verfügbarkeit ist eine sichere Erkennbarkeit des Zustandes jeder einzelnen Batteriezelle entscheidend. Daher ist eine individuelle Zellüberwachung mit möglichst signifikanten Messgrößen erforderlich.

Aufgabe

Herr Maher Achour erhält die Aufgabe, eine Sensorik auf der Basis von POF-Lichtleitern zu erstellen, zu erproben und schrittweise zu verbessern. Die Sensoren sollen zeitgleich mit anderen Messgrößen die Beobachtung der Dichte des Elektrolyten (Batteriesäure) ermöglichen. Die Säurekonzentration ist etwa proportional zum Ladezustand der Zelle. Die Säurekonzentration ist über die Dichte messbar. In Vorarbeiten ist ein Verfahren realisiert worden, das die Dichte über die Änderung des Brechungsindex bestimmt. Bei gekrümmten Lichtleitfasern erfolgt eine deutliche Änderung der Transmission bei der Änderung des Brechungsindex des umgebenden Mediums durch unterschiedliche Lichtleitungs-/Transmissionsverluste.
Wesentlich ist dabei eine mechanische Konstruktion des gekrümmten Lichtleiter zu entwickeln, die nur geringe Toleranzen und Fertigungsstreuung sowie eine ausreichende mechanische Stabilität aufweist. Es soll eine LWL-Sonde entstehen, die den Lichtleiter in einem engen Krümmungsradius geometrisch fixiert, jedoch einen räumlich ausgedehnten Kontakt zum Elektrolyten gewährleistet. Hierdurch soll eine räumlich integrative Messung in verschiedenen Höhenschichten erfolgen.
Die entwickelten Sonden sind in den Batterieinnenraum einzubringen und experimentell zu erproben. Die optischen Messungen sind mit elektrischen Messungen zu kombinieren und auszuwerten. Durch Kalibrierung, Referenzmessungen und Fehlerkompensation sollen Streuungen und Störeinflüsse minimiert werden.
Für die Abschlussarbeit sind die folgenden Arbeitspakete geplant:

1. Einarbeitung und Vorarbeiten
    - Einarbeitung in Grundlagen und Effekte bei der Bleibatterie
    - Einarbeitung in die Funktionsweise der Lichtwellenleiter, insb. für optische Messverfahren
    - Darstellung des Standes eigener und fremder Vorarbeiten sowie des Verbesserungspotentials
    - Vorabmessungen mit vorhandenen Versuchsaufbauten und bestehender Software
2. Entwicklung der LWL-Sonde und des Messaufbaus

- Konstruktion der mechanischen Trägerkomponenten zum Einbringen des Lichtleiters in die Batteriezelle insb. mit den Zielstellungen:
  1) die stabile und maßgenaue Fixierung des gekrümmten Lichtleiters
  2) gut reproduzierbare Fertigung
  3) günstige und flexible Einbringung in den Zellenaufbau
  4) räumlich großflächiger Kontakt des LWL zur Batteriesäure
  5) gute Beständigkeit gegen Batteriesäure
- Präparation des Lichtleiters für den Lichtaustritt durch partielle Entfernung des Mantel/Claddings
- Umbau einer Versuchsbatterie und Aufbau von LWL-Sonden und Sensoren
- Optional: Möglichst gute Unterdrückung von Störeffekten durch Gasblasenbildung

3. Konzeption- und Entwicklung von Fehlerkompensations- und Kalibrierverfahren

   - Systematische Erfassung der Fehler- und Störeffekte bei der optischen Sensorik
   - Vorabmessungen mit Laboraufbauten, modifiziertem Sensor und LED-Reflektometer
   - Erweiterung der Sensorsoftware um Kalibrierungsmodule
   - Vorbereitung der Auswertesoftware für Kalibriermessungen, insbesonder mit Regressionsverfahren
   - Konzeption und Entwicklung eines Verfahrens für die differentielle Messung zur Offsetunterdrückung (Fremdlicht, Temperaturoffset), insbesondere durch Schaltbetrieb und/oder andere Modulationsverfahren
   - Konzeption zur Kompensation/Kalibrierung von Temperatureinflüssen
   - optional: Konzeption von Lösungsansätzen mit Referenzfaser, Darstellung von Vorteilen/Nachteilen und Limitierungen

4. Funktionserprobung auf der Starterbatterie

   - Planung von Messreihen mit Zyklierbetrieb
   - Durchführung und Auswertung der Messreihen
   - Analyse des Zeitverhaltens der optischen Messwerte
   - Auswertung von Einflussgrößen und Streuungen,
   - Durchführung von Referenzmessungen und Kalibrierungen für die Versuchsbatterie
   - Untersuchung auf Reproduzierbarkeit

5. Einordnung, Bewertung und Ausblick

   - Zusammenfassung der Ergebnisse, Beurteilung des Messkonzeptes
   - Bewertung der gewählten Konzepte und Lösungsvarianten
   - Offene Punkte und einschränkende Erfahrungen und Beobachtungen

Dokumentation

Die Fachliteratur und die kommerziellen Unterlagen bzw. Datenblätter sind zielgerichtet zu recherchieren. Dabei sind insbesondere wichtige Grundlagen der Batterieeffekte und der Lichtleiteroptik näher zu betrachten. Die gesetzten Rahmenbedingungen, die gewählten Lösungen und die Funktionsweise sind gut nachvollziehbar zu dokumentieren. Die Messergebnisse sind in aussagefähigem Umfang zu erfassen und auszuwerten. Die realisierten Lösungen und die Ergebnisse sind kritisch zu bewerten. Ansätze für Verbesserungen und weitere Arbeiten sind zu nennen.

B. Quellcodes

B.1. Sensor-Firmware

B.1.1. main.c

```
1
2
3  #define EXTERN
4
5  #define CALIBRATION
6
7  #include <msp430g2553.h>
8
9  #include "mainheader.h"
10 #include "global.h"
11 #include "Sensors\Sensor_8.h"
12 #include "tx433.h"
13 #include "adc.h"
14 #include "queue.h"
15 #include "freq.h"
16
17 int main(void) {
18     WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer
19
20
21     // Initalisierung der Grundfunktionen der Hardware
22     initFrequency();
23     initGPIO();
24     initI2C();
25     initADC();
26 //     initTimerA0();
27     initTA1();
28
29
30     // INIT sr_r
31     sr_r = (Sensoradresse | 0x08000000);
32
33     initqueue();
34
35     // TODO Framefehlermessung
36     // Counter welcher im Testfall in jedem Liveframe inkrementiert wird
37     // framecounter = 0;
38
39     // Ausschalten des TX
40     TX433_OFF;
41     // Warten bis off
42     while((P1IN & BIT5) != BIT5);
43     // wiedereinschalten
44     TX433_ON;
45
46     // Frame Initalisieren
47     frame_tx433_init(Sensoradresse);
48
49     // Zwangspause derzeit bis der TX reagiert
50     __delay_cycles(250000);
51     LED_OFF;
52
53     // Messen der Spannung vor dem Tx_Init damit Zeit vergeht bis der ADC fertig ist.
54     sample_voltage_batt();
55     tx433_init();
56
57     // Init der Variablen damit die Startbedingungen gegeben sind
58     tmp = ADC10MEM;
59
```

```
60      battvoltage_old = tmp;
61      readindex = 0;
62      writeindex = 0;
63      tx_status = 0;
64
65      // Set timervalue to zero
66      timer24 = 0;
67      timervalue = 0;
68
69
70      //Measurement state
71      int state_meas = 0;
72
73      // Init der States auf IDLE
74      transmit = IDLE;
75      ADC = ADC_idle;
76
77      // Statischer Zustand
78      Messpin_OFF;
79      LED_OFF;
80      PWM_OFF;
81
82      // Generate First Rnd Sendtime
83      // TODO RND SHIFT MAKRO
84      wait_rnd_time = ((uint16_t)(sr_r % (2 * (TXMID / TXFAC)) + ( TXMID - (TXMID / TXFAC)))) << 3;        // slow im Mittel 4 Sekunden
85      SR();
86      // Globale Interrupts zulassen
87      // Da Timer läuft
88      _EINT();
89
90        while(1){
91
92          // Alle Aktionen finden in der Timer_isr.c statt
93
94          // VoltageCal-Loop mit _Delay_cycles realisiert
95          // Ablauf
96          // warten 500ms / Dann Spg messen / Dann Werte ins Frame / Daten an Transmitter / Transmitter senden lassen / warten 50ms...
97           wait_rnd_time = ((uint16_t)(sr_r % (2 * (TXMID / TXFAC)) + ( TXMID - (TXMID / TXFAC)))) << 3;  // slow im Mittel 4 Sekunden
98           z2=wait_rnd_time;
99           __delay_cycles(2000000);                // Sollten 500ms entsprechen bei 4 MHz
100 //                  ----------MESSBLOCK 1----------
101          if (!state_meas)
102                  sample_freq(LED_1_on);                                  // 1. Frequenzmessung
103          else
104                 sample_freq(LED_2_on);                                   // 1. Frequenzmessung
105          freq1+=ticks;
106          sample_freq(LEDoff);    // 1. Freqeunzmessung Umgebungslicht subtrahieren
107          if(freq1<= ticks){
108                  freq1=2 ;
109          }else{
110          freq1-=ticks;}
111         sample_voltage_batt();                  // 1. Spannungsmessung
112         while(ADC10CTL1 & ADC10BUSY){
113         }
114         battvoltage += ADC10MEM;                // Spannungs Wert ins Framepuffer
115         sample_temperature();                   // 1. Temperautrmessung
116         while(ADC10CTL1 & ADC10BUSY){
117         }
118         temperatur += ADC10MEM;          // Temperatur Wert ins Framepuffer
119         __delay_cycles(2000000);                // Sollten 500ms entsprechen bei 4 MHz
120 //                  ----------MESSBLOCK 2----------
121         if (!state_meas)
122                 sample_freq(LED_1_on);                                  // 2. Messung Anzahl der Ticks mit eingeschalteter LED 1
123         else
124                 sample_freq(LED_2_on);                                  // 2. Messung Anzahl der Ticks mit eingeschalteter LED 2
125         freq1+=ticks;
126         sample_freq(LEDoff);                                    //  Messung Anzahl der Ticks Umgebungslicht subtrahieren
127         if(freq1<= ticks){
128              freq1=2 ;
129           }else{
130         freq1-=ticks;}
131                 sample_voltage_batt();                  // 2. Spannungsmessung
132         while(ADC10CTL1 & ADC10BUSY){
133         }
134         battvoltage += ADC10MEM;                // Spannungs Wert ins Framepuffer
135         sample_temperature();                   // 2. Temperautrmessung
136         while(ADC10CTL1 & ADC10BUSY){
137         }
138         temperatur += ADC10MEM;          // Temperatur Wert ins Framepuffer
139 //                  ----------Auswertung----------
140         freq1       >>= 1;                      //Werte mitteln
141         battvoltage >>= 1;
142         temperatur  >>= 1;
143 #ifndef CALIBRATION
144         temperatur -=TempcalAchs; //Temperatur Kalibrieren
```

```
145         temperatur *=TempcalSteig;
146         temperatur >>=10;
147         battvoltage -=VoltagecalAchs;           //Spannung kalibrieren
148         battvoltage *=VoltagecalStei;
149         battvoltage >>= 10;
150 #endif
151         warten();
152 //              ----------Übertragung----------
153
154         if (!state_meas) {
155             frame_tx433_setAddress(Sensoradresse);
156         } else {
157             frame_tx433_setAddress(Sensoradresse + SENSOR_COUNT);
158         }
159
160         frame_tx433_live();                         // Wert ins Frame
161         frame_tx433_calc_crc();                 // CRC über das Frame
162         frame_tx433_code_manchester();  // Manchestercodierung der Daten
163
164         Messpin_ON;
165         tx433_cmd_fifo_set();                   // Frame an den Transmitter senden
166         LED_ON;
167         tx433_cmd_tx_start();                   // Transmitter Commando fürs Senden schicken
168         Messpin_OFF;
169         LED_OFF;
170
171         Messpin_ON;
172         tx433_cmd_fifo_set();                   // Frame an den Transmitter senden
173         LED_ON;
174         tx433_cmd_tx_start();                   // Transmitter Commando fürs Senden schicken
175         Messpin_OFF;
176         LED_OFF;
177
178         //Werte zurücksetzen
179         battvoltage = 0;
180         freq1=0;
181         temperatur=0;
182
183         //Change LED
184         state_meas ^= 0x01;
185
186         SR();
187
188     }
189 }
```

B.1.2. mainheader.h

```
1
2
3 #ifndef MAINHEADER_H_
4 #define MAINHEADER_H_
5
6 #include <stdint.h>
7 #include <msp430g2553.h>
8
9 /*
10  * Definition für I2C
11  */
12         #define UCB0BR_BIT_CLK_100      200.0                           // UCB0 Bit Clock [kHz]
13         #define SMCLK                           1000.0                          // SubMain Clock (from DCO) [kHz]
14         #define UCB0BR_0_SET_100        ((uint16_t)( SMCLK / UCB0BR_BIT_CLK_100))
15
16 /*
17  * Allgemeine Makros und Enums
18  */
19         // TX 433 On/Off Handling
20         #define TX433_ON (P1OUT &= ~BIT4)
21         #define TX433_OFF (P1OUT |= BIT4)
22
23         // LED Handling
24         #define LED_ON (P2OUT |= BIT5)
25         #define LED_OFF (P2OUT &= ~BIT5)
26         #define LED_toggle (P2OUT ^= BIT5)
27
28         // Messpin Handling
29         #define Messpin_ON (P2OUT |= BIT3)
30         #define Messpin_OFF (P2OUT &= ~BIT3)
31         #define Messpin_toggle (P2OUT ^= BIT3)
32
33         // Enableeingang des DCDC Wandlers
34         #define DCDC_ON (P1OUT |= BIT0)
```

```
35          #define DCDC_OFF (P1OUT &= ~BIT0)
36          #define DCDC_Toggle (P1OUT ^= BIT0)
37
38          //PWM Handling
39          #define PWM_1_ON (P2OUT|= BIT6)    // Pin enable LED treiber 1
40      #define PWM_2_ON (P2OUT|= BIT7)    // Pin enable LED treiber 2
41          #define PWM_OFF (P2OUT &= ~(BIT6 | BIT7)) // Pins disable
42          #define PWM_Toggle (P2OUT ^= BIT6)
43
44          //Frequenzausgang am TSL230R
45          #define FRQ ((P2IN & BIT4))
46
47          // TX Frame Informationen
48          // gemäss neuem Frame Maximallänge 72 Bit / ohne RUN In SOF 60 Bit
49          // Makronamen von der Masterarbeit Jegenhorst übernommen
50
51          #define limitt                                  10
52          #define TX433_FRAME_BYTES                   8           // TX data bytes = 60 / 8 + 0.5 Nur ganzzahlige Bytes..
53          #define TX433_FRAME_BYTES_MCH           18          // Manchester-coded TX bytes SOF + RunIn + Framebytes ((4 + 8 +
                60)*2(BIT)) / 8
54                                                                                                  //      = (TX433_FRAME_BITS*2)
                                                                                                    / 8
55          #define TX433_FRAME_BYTES_MCH_half      9           // die hälfte der Bytes für die gesplittete Übertragung der
                Daten an den Transmitter
56
57          #define TX433_LIVEBIT                       0x10        // Kennzeichnung für ein Liveframe
58          #define TX433_BUFFERED                      0x00        // Kennzeichnung für ein Bufferedframe
59          #define TX433_LED_CLR                       0x20        // Bit is set when measurement was taken using red led
60
61          // Buffered Frame Setup
62          #define TX433_FR_PARA_BYTE0                 0               // obere 4 Bit Parameter / untere 4 Bits
                Errorcodes
63          #define TX433_FR_ADDR_BYTE0                 1               // Volle 8 Bit für die Adresse
64          #define TX433_FR_VOLTAGE_BYTE0          2               // Zellspannung
65          #define TX433_FR_TIMESTAMP24_BYTE       3               // eigentlich bei 3.5 Byte. beim Laden der Werte wird
                angepasst
66          #define TX433_FR_CRC_BYTE0                  6               // angepasst mit Korrekturrechnung von 3.5 Byte
                ..eigentlich 6.5 Byte bis 7.5 Byte
67
68          // Live Frame Setup nur zusätzliche Angaben
69          #define TX433_FR_TEMPERATUR_BYTE0       3
70          #define TX433_FR_TIMESTAMP16_BYTE       4
71
72                                                                                                  // Data-Alignment in
                                                                                                        txdata[]:
                                                                                                        Verschiebung um
                                                                                                        1 da Set_Fifo
                                                                                                        zuerst
73          #define TX433_TX_SOF_BYTE0                  1               // Set FIFO macht die Verschiebung notwendig
74          #define TX433_TX_PARA_BYTE0                 4               // Parameter Bytes Start - Verschiebung durch
                Set FIFO
75          #define TX433_TX_ERR_BYTE0              5               // Error Bytes Start - Verschiebung durch Set FIFO
76          #define TX433_TX_ADDR_BYTE0                 6               // Sensoradresse - Verschiebung durch Set FIFO
77          #define TX433_TX_DATA_BYTE0                 8               // Zellspannung - Verschiebung durch Set FIFO
78          #define TX433_TX_TIMESTAMP_BYTE         11              // Zeitstempel(Ausgangslage buffered Frame) -
                Verschiebung durch Set FIFO
79          #define TX433_TX_CRC_BYTE0                  17              // CRC Ergebnis - Verschiebung durch Set FIFO
80
81
82          // Übernommen aus der Arbeit von Puettjer
83          // RND SHIFT MAKRO
84          #define  SR() sr_r<<=1;sr_r|=(((sr_r >> 28)^(sr_r>>19))&0x00000001);sr_r&=0x0FFFFFFF
85          #define TXFAC 3
86          #define TXMID 5000
87
88          // Definitionen für die LSB Werte des ADC und untere Grenzspannung
89          // Korrektur da Floatingpointoperationen zu teuer sind
90          //      #define UBatt_LSB (2.5/1023)
91          #define LowVoltage 210          // Untere Spannungsgrenze für den Transmitter eingeführt für den Zellensimulator
92          #define Voltagediff     8               // Spannungsdifferenz in LSB 7 ca. 17mV / 9 ca. 22 mv / 6 ca. 14mv / 20 ca. 48
                mv / 8 ca. 20mv
93
94          // Queue_Length
95          #define Queue_Length 65
96
97          // Zeit pro Messwertaufnahme in die Queue ca. 50ms
98          #define queuevaluetime  750     // Wartezeit bis beim Inqueuefall der nächste Wert in die Queue kommt 750 ca. 75ms
99          #define queuereturntime 20000   // Werte Aufnahme alle  2s da das RND Makro Werte um 2s generiert
100         #define queue_fix_length 30             // Fixe Anzahl an Werte welche im Zeitraster von 50ms in die Queue kommen bevor erneut
                geprüft wird
101
102         // Sendewiederholungsbit
103         #define Sendewdh 0x01
104
105
```

```
106         // Enum für Schleifen und Status
107         enum boolean {
108                         FALSE = 0x00,
109                         TRUE  = 0x01
110         };
111
112         // Definition Transmitter Statemachine
113         typedef enum{
114                 IDLE,
115                 Standby,
116                 Frameload,
117                 Transmit,
118                 wait,
119                 Transmit_crc,
120                 Transmit_man,
121                 Transmit_data_1,
122                 Transmitter_off,
123                 Transmitter_on
124         }transmitstate;
125
126         // Definition ADC Statemachine
127         typedef enum{
128                 ADC_busy_voltage,
129                 ADC_busy_temperature,
130                 ADC_idle,
131                 ADC_inqueuewait
132         }adcstates;
133
134         // Queue-Struct
135         typedef struct {
136                 uint8_t timelabel24_1;          // oberer Teil des 24 Bit Timers
137                 uint8_t timelabel24_2;          // mittlerer Teil des 24 Bit Timers
138                 uint8_t timelabel24_3;          // unterer Teil des 24 Bit Timers
139                 uint8_t voltage12_1;            // oberer Teil des 12 Bit Spannungswertes
140                 uint8_t voltage12_2;            // unterer Teil des 12 Bit Spannungswertes und 4 Bit Status Sendewdh zb.
141         }queue;
142
143
144
145         //      Return-Codes
146         #define EXIT_SUCCESS    0x01
147         #define EXIT_FAILURE    0x00
148
149
150
151 // Funktionen
152
153 // Funktionen für die Initalisierung
154 void initFrequency(void);
155 void initGPIO(void);
156 void initI2C(void);
157 void initADC(void);
158 void initTimerA0(void);
159 void initTA1(void);
160 void warten(void);
161
162
163 // Funktionen für das Übertragungsframe
164 void frame_tx433_init(uint8_t address);
165 void frame_tx433_code_manchester(void);
166 void frame_tx433_calc_crc(void);
167 void frame_tx433_buffered(void);
168 void frame_tx433_live(void);
169 void frame_tx433_setAddress(uint8_t address);
170 void frame_tx433_setParameterBit(uint8_t position);
171 void frame_tx433_clearParameterBit(uint8_t position);
172
173 #endif /* MAINHEADER_H_ */
```

B.1.3. global.h

```
1
2
3 #ifndef GLOBAL_H_
4 #define GLOBAL_H_
5
6 #ifndef EXTERN
7                 #define EXTERN extern
8         #endif
9
10                // Struct zu grossen Teilen von Masterarbeit Jegenhorst übernommen
11                EXTERN struct {
```

```
12                      uint8_t         return_status;
13                      uint8_t         state;
14                      uint8_t         idlemode;
15                      uint8_t         actdatatsize;
16                      uint8_t         framedata[TX433_FRAME_BYTES];
17                      uint8_t         txdata[TX433_FRAME_BYTES_MCH+1];        // + 1Byte for I2C Command
18              } volatile g_tx433;
19
20
21              EXTERN volatile uint16_t battvoltage_old;                       // alter Spannungswert ADC zum Vergleich
22              EXTERN volatile uint32_t battvoltage;                           // Spannungswert fürs Liveframe
23                      // Spannungswert fürs Liveframe
24              EXTERN volatile uint16_t tmp;                                           // Spannungswert
25              EXTERN volatile uint16_t temperatur;                            // Temperaturwer fürs Liveframe
26              EXTERN volatile uint16_t queuewait_counter;                     // Counter für Inqueue Spannungsmessung
27              EXTERN volatile transmitstate transmit;                         // Statemachine Transmitter
28              EXTERN volatile adcstates ADC;                                          // Statemachine ADC
29              EXTERN volatile uint32_t timer24;                                       // TimerTicks in TimerISR
30              EXTERN volatile uint32_t timervalue;                            // Speicher für Zeitzwischenrechnungen
31
32              // Variablen für das zufällige Senden
33              EXTERN volatile long sr_r;                                                      // Zufallszahl Startwert
34              EXTERN volatile uint16_t wait_rnd_time;                         // Counter für zufälliges Senden
35              EXTERN volatile uint16_t wait_save_time;                        // Speicher für zufälliges Senden
36              EXTERN volatile uint16_t framecounter;                          // Framecounter für Framefehlertests
37              EXTERN volatile uint8_t statusbitarray;                         // Statusarray 8 Bit
38
39              // Queue Handling
40              EXTERN volatile uint8_t readindex;                                      // Leseindex Queue
41              EXTERN volatile uint8_t writeindex;                                     // Schreibindex Queue
42              EXTERN volatile queue waitingqueue[Queue_Length];       // Array der (struct)Queue
43              // Queue Handling 2
44              EXTERN volatile uint8_t queue_counter;                          // Queuezähler für das Inqueue von 30 aufeinander
                        folgendnen Werten
45              EXTERN volatile uint8_t Voltagedifferenz;                       // für das dynamische Herabsetzen der
                        Inqueuespannungsdifferenz
46
47              EXTERN volatile uint8_t tx_status;                                      // Debug Zellensimulator Transmitterstatus 1/0
                        == aus/an
48              EXTERN volatile uint16_t z1; //Zähler für flanken;
49              EXTERN volatile uint32_t z2;
50              EXTERN volatile uint32_t wartecounter;
51              EXTERN volatile uint16_t ticks;
52              EXTERN volatile uint16_t freq1;
53 //             EXTERN volatile uint16_t stopper;
54 //             EXTERN volatile uint16_t bereich;
55 //             EXTERN volatile uint16_t takt[limitt];
56
57
58
59 #endif /* GLOBAL_H_ */
```

B.1.4. init.c

```
1
2
3  #include <msp430g2553.h>
4  #include "mainheader.h"
5  #include "global.h"
6
7  // Frequenzeinstellungen
8  // 4MHz Mainclk und 1000kHz SMCLK
9  // Quelle und Ablauf der Frequenzeinstellungen Gracetool von TI
10 void initFrequency(){
11
12         /*
13      * Basic Clock System Control 2
14      *
15      * SELM_0 --- DCOCLK
16      * DIVM_1 --- Divide by 2
17      * ~SELS --- DCOCLK
18      * DIVS_3 --- Divide by 8
19      * ~DCOR --- DCO uses internal resistor
20      *
21      * Note: ~<BIT> indicates that <BIT> has value zero
22      */
23         // Quelle für MCLK -> DCOCLK / Teilung 2 bei 8 MHz / Quelle für SMCLK pauschal auf DCOCLK(alternativen nur Extern) / Teilung 8
                bei 8 MHz
24     BCSCTL2 = SELM_0 | DIVM_1 | DIVS_3;
25
26     // Quelle Gracetool
27     if (CALBC1_8MHZ != 0xFF) {
```

```
28          /* Adjust this accordingly to your VCC rise time */
29          __delay_cycles(100000);
30
31          // Follow recommended flow. First, clear all DCOx and MODx bits. Then
32          // apply new RSELx values. Finally, apply new DCOx and MODx bit values.
33          DCOCTL = 0x00;
34          BCSCTL1 = CALBC1_8MHZ;       /* Set DCO to 8MHz */
35          DCOCTL = CALDCO_8MHZ;
36      }
37
38      /*
39       * Basic Clock System Control 1
40       *
41       * XT2OFF --- Disable XT2CLK
42       * ~XTS --- Low Frequency
43       * DIVA_0 --- Divide by 1
44       *
45       * Note: ~XTS indicates that XTS has value zero
46       */
47      BCSCTL1 |= XT2OFF | DIVA_3;
48
49      /*
50       * Basic Clock System Control 3
51       *
52       * XT2S_0 --- 0.4 - 1 MHz
53       * LFXT1S_0 --- If XTS = 0, XT1 = 32768kHz Crystal ; If XTS = 1, XT1 = 0.4 - 1-MHz crystal or resonator
54       * XCAP_1 --- ~6 pF
55       */
56      //BCSCTL3 = XT2S_0 | LFXT1S_0 | XCAP_1;
57
58      /*
59          //DCOCTL  = CALBC1_1MHZ;
60          DCOCTL  = CALBC1_8MHZ;
61          // XT2 aus, 1 MHz, sowie Taktteiler für ACLK auf 1 Voller Takt auf der ACLK
62          BCSCTL1 = (XT2OFF | CALBC1_1MHZ | DIVA_0);
63          // MCLK = DCOCLK, div MCLK = X, SMCLK = DCOCLK, div SMCLK = X
64          BCSCTL2 = SELM_0 | DIVM_0 | DIVS_1;
65          */
66
67  }
68
69
70  // Portinitalisierung
71  void initGPIO(){
72          /* Port 1 */
73          P1OUT = BIT4 + BIT0;                    // GPIO Pin 1.4 für SDN Pin vom SI4012
74
75
76          // Port Selection 1 für Pin 1.6 / 1.7 beide Selectionregister auf 1 für die Funktion als I2C
77          P1SEL = BIT6 + BIT7;
78          P1SEL2 = BIT6 + BIT7;
79          // Direction Pin 1.6 / 1.7 werden vom UCI konfiguriert
80          P1DIR = BIT4 + BIT0;     // BIT 4 = SDN Funktion des SendeIC´s
81          //Save Line
82          DCDC_ON;
83          // Port 1 Interrupt Edge Select Register
84          P1IES = 0;
85          // Port 1 Interrupt Flag Register
86          P1IFG = 0;
87
88          /* Port 2 */
89          P2OUT = BIT7 + BIT6 + BIT3 + BIT5;       // GPIO Pin 2.3 für Messpin & GPIO 2.5 für LED als Ausgang
90          P2DIR = BIT7 + BIT6 + BIT3 + BIT5;       // GPIO Pin 2.3 für Messpin & GPIO 2.5 für LED als Ausgang
91
92          P2SEL &= ~(BIT6 | BIT7);
93  //      P2SEL |= BIT4; //2.4 Als TA1 für Capture
94          P2SEL2 = 0x00;
95
96
97  //      P2IE |= BIT4; //2.4 Interrupt erlauben
98  //      P2IES &= ~BIT4; //Port 2.4 Interupt bei HIGH Flanke
99
100         z1=0;
101         z2=0;
102         freq1=0;
103         ticks=0;
104 //      stopper=0;
105 }
106
107 // I2C Einstellungen übernommen von Masterarbeit Jegenhorst
108 void initI2C(){
109
110         // first software reset
111         UCB0CTL1 |= UCSWRST;
112         // Own adress 7-bit, slave adress 7-bit, single master, I2C-mode, synch mode
```

```
113         UCB0CTL0  = (UCMST | UCMODE_3 | UCSYNC);
114         // Clock source = SMCLK
115         UCB0CTL1  = (UCSSEL_2 | UCSWRST);
116         // Baud rate = 100kbps, f_bitclk = f_brclk / UCB0RX = 100kHz,
117         //       f_brclk = SMCLK = 0.5MHz
118         //       --> UCB0RX = 5
119         UCB0BR0 = UCB0BR_0_SET_100;              // low byte, UCB0RX = (UCB0R0 + UCB0R1 x 128)
120         UCB0BR1 = 0x00;                          // high byte
121         // IRQ Flags UCNACKIFG, UCSTPIFG, UCSTTIFG, UCALIFG clears by sw-reset
122         // I2C own 7-bit address = 0x7B (123)
123         UCB0I2COA |= 0x007B;
124         // no IRQs enabled for NACK, STOP, START, ARB
125         UCB0I2CIE = 0x00;
126 }
127
128 // ADC Grundeinstellungen die genauen Einstellungen erfolgen in der ADC.c für die entsprechende Messung
129 void initADC(){
130
131         // ADC10CTL0 Registersetup
132
133         // Referenz auf interne Referenz umschalten
134         ADC10CTL0 |= SREF_1;
135         // Ref- Spg. 2.5 V
136         ADC10CTL0 |= REF2_5V;
137         // Referenz Generator an
138         ADC10CTL0 |= REFON;
139         // Sample and Hold time für Temperaturmessung verlängert. Der ADC benötigt dort 30microS für die Messung.
140         ADC10CTL0 |= ADC10SHT_3;
141
142         // ADC10CTL1 Registersetup
143
144         // SHSx Sample and Hold Source Select
145         ADC10CTL1 |= SHS_0;
146         // ADC10DIV Clock divider
147         ADC10CTL1 |= ADC10DIV_5;
148         // ADC10SSEL Clock Source select
149         ADC10CTL1 |= ADC10SSEL_0;
150         // Conversion sequence mode select Singlemode.
151         ADC10CTL1 |= CONSEQ_0;
152
153         // ADC "einschalten"
154         ADC10CTL0 |= ADC10ON;
155
156
157 }
158 // TimerA Init und ISR Anmeldung
159 //void initTimerA0(){
160 //
161 //       // Funktion mit SMCLK gegeben, die ACLK ist nicht lauffähig
162 //
163 //       // Reset First
164 //       TACTL = TACLR;
165 //
166 //       // SMCLK + Inputclockteiler = 1 + Upmode
167 //       TACTL = TASSEL_2 | ID_0 | MC_1;
168 //
169 //       // Comparemode mit Interrupt Enable
170 //       TACCTL0 = CCIE;
171 //
172 //       // SMCLK läuft mit 1000kHz
173 //       // 100 entsprechen im 100micoS Kontrolle durch TogglePin in der Timerroutine
174 //       // 50 entsprechen im 50micoS Kontrolle durch TogglePin in der Timerroutine
175 //       TACCR0 = 100;
176 //
177 //}
178
179 //void initTA1(){
180 ////1Mhz Takt Durch 8 geteilt = 125kHz, Continous mode, Timer zurücksetzen, Interrupts erlauben
181 //TA1CTL=TASSEL_2 + ID_0 + MC_2 + TACLR + TAIE;
182 ////Postive Flanke auslösen, P2.4 als Eingang für Capture, Taktsynchronisation, Capturemode, Capture interrupts erlauben
183 //TA1CCTL2=OUTMOD_6 +CM_1 + CCIS_0 + SCS + CAP + CCIE;
184 //}
185
186 void initTA1(){
187 TA1CTL=TASSEL_2 + ID_3 + MC_0 + TACLR + TAIE;
188 TA1CCTL1=CCIE + OUT;
189 TA1CCR0=62499; //9999 bei Dunklerfrequenz
190 }
191
192 void warten() {
193         for(wartecounter = 10; wartecounter > 0; wartecounter--){
194
195     for (; z2 > 0; z2--) {
196     }
197 //    z2=wait_rnd_time;
```

```
198         }
199 }
```

B.1.5. adc.c

```
1
2
3  #include "mainheader.h"
4  #include "global.h"
5  #include "adc.h"
6
7
8  // Spannung messen
9  void sample_voltage_batt(){
10
11          // ADC Stoppen, damit Konfiguration der ADC Register möglich wird
12          ADC10CTL0 &= 0xFFFD;
13          // ADC Umkonfigurieren
14          ADC10CTL0 |= SREF_1;
15          // Ref– Spg. 2.5 V
16          ADC10CTL0 |= REF2_5V;
17          // Referenz Generator an
18          ADC10CTL0 |= REFON;
19
20          // Löschen der Einstellung
21          ADC10CTL1 &= 0x0FFF;
22          ADC10CTL1 |= INCH_1;
23
24          // Analog Input für A1
25          ADC10AE0 |= BIT1;
26
27          ADC10CTL0 |= ADC10ON;
28          // ADC "Scharf_schalten"
29          ADC10CTL0 |= (ENC + ADC10SC);
30
31 }
32 // Temperatur messen
33 void sample_temperature(){
34
35          // ADC Stoppen, damit Konfiguration der ADC Register möglich wird
36          ADC10CTL0 &= 0xFFFD;
37          // Interne Ref setzen
38          ADC10CTL0 |= SREF_1;
39          // ADC Ref auf 1.5 Volt gesetzt
40          ADC10CTL0 &= ~REF2_5V;
41          // Inputchannel wechseln auf Temperatursensor
42          ADC10CTL1 &= 0x0FFF;
43          ADC10CTL1 |= INCH_10;
44
45          ADC10AE0 &= 0x00;
46          // ADC "Scharf_schalten"
47          ADC10CTL0 |= (ENC + ADC10SC);
48
49 }
```

B.1.6. adc.h

```
1
2
3  #ifndef ADC_H_
4  #define ADC_H_
5
6
7  // Funktionen
8
9  void sample_voltage_batt(void);
10 void sample_temperature(void);
11
12 #endif /* ADC_H_ */
```

B.1.7. freq.c

```c
#include "mainheader.h"
#include "global.h"
#include "freq.h"


//Function LED-switch
void sample_freq(int LEDstate){
  if(LEDstate == LED_1_on)
        PWM_1_ON;
  else if(LEDstate == LED_2_on)
        PWM_2_ON;

        P2IE |= BIT4; //2.4 Interrupt erlauben
        P2IES &= ~BIT4; //Port 2.4 Interupt bei High

        if(!(P2IN&BIT4)){
                P2IFG |= BIT4;
        }
        while(P2IE&BIT4){
                //Warten bis TimerInterrupt ausgeführt wird
                }
//      PWM_OFF;
//      timervalue=freq;


}
```

B.1.8. freq.h

```c

#ifndef FREQ_H_
#define FREQ_H_
#define LED_1_on 1
#define LED_2_on 2
#define LEDoff !LED_1_on

// Function LED-switch
void sample_freq(int LEDstate);


#endif /* FREQ_H_ */
```

B.1.9. frametx.c

```c


/*

Sensor LiveFrame :



        |          SOF          | Parameters | Errorcodes |  Address  |   Cellvoltage   |   Temperatur | Timervalue | CRC |
        |-----------|---------|---------|--------|-----------|----------|---------|-----|
        | SOF       RUN-IN |            |           |          |          |          |          |
        | 4-Bit     8-Bit |  4-Bit     | 4-Bit    | 8-Bit    |  12-Bit  |   8-Bit  | 16-Bit | 8-Bit | -> 72 Bits
        |-----------|---------|--------- |--------|-----------|----------|---------|-----|
        |                     | 0        4 | 5     8 | 9    16 | 17     28 | 29          36 | 37      52 |53   60| -> in framedata
        |-----------|---------|--------- |--------|-----------|----------|---------|-----|
        | 0     7   8     23 | 24      31 | 32     39 | 40   55 | 56     79 | 80       95 | 96      127 |128 143| -> in txdata
        |-----------|---------|--------- |--------|-----------|----------|---------|-----| on 10.0kb/s
        | 800us      1.6ms |    800us   |   800us   | 1.6ms  |   2.0ms   |   1.6ms    |  3.2ms  | 1.6ms | -> 14ms


Sensor Bufferd Frame :

        |          SOF          | Parameters | Errorcodes |  Address  |   Cellvoltage   | Timervalue  |  CRC   |
        |-----------|---------|---------|--------|-----------|----------|---------|
        | SOF       RUN-IN |            |           |          |          |          |
        | 4-Bit     8-Bit |  4-Bit     | 4-Bit    | 8-Bit    |  12-Bit  |  24-Bit  | 8-Bit  | -> 72 Bits
```

```
27          |-----------|-----------|-----------|-----------|-----------|-----------|-----------|
28          | 0         4   | 5       8   | 9       16  | 17      28  | 29              52  | 53
                    60  | -> in framedata
29          |-----------|-----------|-----------|-----------|-----------|-----------|-----------|
30          | 0       7   8       23  | 24          31  | 32          39  | 40      55  | 56        79  | 80        127 | 128     143 | -> in
                txdata
31          |-----------|-----------|-----------|-----------|-----------|-----------|-----------| on 10.0kb/s
32          | 800us         1.6ms   |       800us   |       800us   |   1.6ms   |     2.0ms     |     1.6ms     |     3.2ms     | -> 14
                ms
33
34  ------------------------------------------------------------------------------
35  */
36
37
38  #include "Sensors\Sensor_5.h"           // Enthält die Sensoradresse
39  #include "mainheader.h"
40  #include "global.h"
41  #include "queue.h"
42  #include <stdint.h>
43
44  #define TX433_CMD_SET_FIFO                  0x66                    // Store data in TX FIFO
45
46  /*
47   * Initalisierung des Datenframes und des TX-Übertragungsframes
48   */
49  void frame_tx433_init(uint8_t address){
50
51          uint8_t i;
52          g_tx433.framedata[0] = 0x00;                                        // setzen der Parameter und Errorcodes auf
                Default 00
53          g_tx433.framedata[1] = address;                                     // Sensoradresse
54
55          for ( i = 2; i < TX433_FRAME_BYTES; i++){                       // Daten und CRC Prüfsumme auf 0 setzen
56                  g_tx433.framedata[i] = 0x00;
57          }
58
59          // TX Übertragungsframe
60          g_tx433.txdata[0] = TX433_CMD_SET_FIFO;                         // Command zum schreiben in den Fifo
61          g_tx433.txdata[TX433_TX_SOF_BYTE0]          = 0xFF;             // RUN-IN Seq 1111 1111 Manchestercoded
62          g_tx433.txdata[TX433_TX_SOF_BYTE0 + 1]  = 0x55;                 // Synch-Seq 0101 0101 --> "0000" Manchestercoded
63          g_tx433.txdata[TX433_TX_SOF_BYTE0 + 2]  = 0x56;                 // Synch-Seq 0101 0110 --> "0001" Manchestercoded
64
65          // Die anderen Datenbytes werden von frame_code_manchester codiert
66
67  }
68
69  // Übernommen aus Jegenhorst Masterarbeit
70  void frame_tx433_code_manchester(void) {
71
72          uint8_t byte_fr, byte_tx;
73          uint8_t shift_tx;
74          uint8_t shift_fr;
75
76          byte_tx = TX433_TX_PARA_BYTE0;    // set startbyte after SOF bytes
77                  // run through bytes of framedata:
78          for(byte_fr = TX433_FR_PARA_BYTE0; byte_fr < TX433_FRAME_BYTES; byte_fr++) {
79                  g_tx433.txdata[byte_tx] = 0x00;                         // clear databyte
80                  shift_tx = 0x80;                                                    // shift to MSB first
81                                                                                              // run through 8-bit
                                                                                                    dataword:
82                  for(shift_fr = 0x80; shift_fr > 0x00; shift_fr >>= 1) {
83                          if(g_tx433.framedata[byte_fr] & shift_fr) {     // if value = 1 --> 1/0 trans
84                                  g_tx433.txdata[byte_tx] |= shift_tx;    // shift in "10"
85                          }
86                          else {                                  // if value = 0 --> 0/1 trans
87                                  g_tx433.txdata[byte_tx] |= (shift_tx >> 1);     // shift in "01"
88                          }
89                          if(shift_fr == 0x10) {
90                                  byte_tx++;              // goto next tx databyte
91                                  shift_tx = 0x80;        // shift to MSB first
92                                  g_tx433.txdata[byte_tx] = 0x00;  // clear databyte
93                          }
94                          else {
95                                  shift_tx >>= 2;         // goto next pair
96                          }
97                  }
98                  byte_tx++;                              // goto next tx databyte
99          }
100 }
101
102 // Übernommen aus Jegenhorst Masterarbeit mit Anpassungen
103 void frame_tx433_calc_crc(void) {
104
105                 // This function calculates the CRC checksum over the frame, in dependence of
106                 // the function "parity()" from the firmware of the old sensor.
```

```
107
108         uint8_t i;
109         uint8_t tmp;
110         tmp = 0x00;                                                                    // Start mit Generatorpolynom = 0
111
112         // Schleife über die ersten 6 Bytes des Frames..
113         // XOR Byteweise mit Framedata verknüpft
114         for(i = 0; i < TX433_FRAME_BYTES-1; i++){
115                 tmp ^= g_tx433.framedata[i];
116         }
117         // CRC Wert nach g_tx433.framedata.crc schreiben
118         g_tx433.framedata[TX433_FR_CRC_BYTE0] = (tmp >> 4) + g_tx433.framedata[TX433_FR_TIMESTAMP24_BYTE + 3];
119         g_tx433.framedata[TX433_FR_CRC_BYTE0 + 1] = (tmp << 4);
120 }
121
122 // Funktionen welche die Daten ins Frame lädt :
123 void frame_tx433_buffered(void){
124
125         uint8_t i;
126         // Reset für Bitshifting notwendig
127         for(i = TX433_FR_VOLTAGE_BYTE0; i < TX433_FR_CRC_BYTE0+2;i++)
128         {
129                 g_tx433.framedata[i] = 0x00;
130         }
131
132
133         // Bufferd Framebit
134         g_tx433.framedata[TX433_FR_PARA_BYTE0] = TX433_BUFFERED;
135         // Laden des Spannungswertes 12 Bit
136         g_tx433.framedata[TX433_FR_VOLTAGE_BYTE0] = (uint8_t) (waitingqueue[readindex].voltage12_1);
137         g_tx433.framedata[TX433_FR_VOLTAGE_BYTE0 + 1] = (uint8_t) (waitingqueue[readindex].voltage12_2) & 0xF0;
138
139
140         timervalue = 0;
141         timervalue = (waitingqueue[readindex].timelabel24_1 );
142         timervalue = (timervalue << 16UL);
143         timervalue += (waitingqueue[readindex].timelabel24_2 << 8);
144         timervalue += (waitingqueue[readindex].timelabel24_3);
145
146         timervalue = (uint32_t) timer24 - timervalue;
147
148         // Der Konstante Wert von 20(2ms) repräsentiert die Zeit die benötigt wird für die Übertragung an den Transmitter und für das
                Sendecommando (MSP Seitig)
149         timervalue = timervalue + 20;
150                         //(uint32_t) timer24- timervalue + 20;
151         g_tx433.framedata[TX433_FR_TIMESTAMP24_BYTE] = (uint8_t) (timervalue >> 20) + g_tx433.framedata[TX433_FR_VOLTAGE_BYTE0 + 1];
152         g_tx433.framedata[TX433_FR_TIMESTAMP24_BYTE + 1] = (uint8_t) (timervalue >> 12);
153         g_tx433.framedata[TX433_FR_TIMESTAMP24_BYTE + 2] = (uint8_t) (timervalue >> 4);
154         g_tx433.framedata[TX433_FR_TIMESTAMP24_BYTE + 3] = (uint8_t) (timervalue << 4);
155
156 }
157
158 void frame_tx433_live(void){
159         uint8_t i;
160         uint16_t timevalue;
161         // Reset für Bitshifting notwendig
162         for(i = TX433_FR_VOLTAGE_BYTE0; i < TX433_FR_CRC_BYTE0+2;i++)
163         {
164                 g_tx433.framedata[i] = 0x00;
165         }
166
167         // TODO Framefehlermessung
168         // framecounter++;
169
170         // Livebit setzen
171         g_tx433.framedata[TX433_FR_PARA_BYTE0] = TX433_LIVEBIT;
172         // Laden des Spannungswertes 12 Bit
173         g_tx433.framedata[TX433_FR_VOLTAGE_BYTE0] = (uint8_t) (battvoltage >> 4);
174         g_tx433.framedata[TX433_FR_VOLTAGE_BYTE0 + 1] = (uint8_t) (battvoltage << 4);
175
176         // Laden des Temperaturwertes 8 Bit
177         g_tx433.framedata[TX433_FR_TEMPERATUR_BYTE0] = (uint8_t) (temperatur >> 6) + g_tx433.framedata[TX433_FR_VOLTAGE_BYTE0 + 1];
178         g_tx433.framedata[TX433_FR_TEMPERATUR_BYTE0 + 1] = (uint8_t)((temperatur & 0x003C)<<2); // maskiere Bits 11 - 8 und 3 bis 0 in
                temperatur zu 0
179
180
181         /*
182         // Framecounter wird für den Zeitwert eingefügt
183         g_tx433.framedata[TX433_FR_TIMESTAMP16_BYTE ] = (uint8_t) (framecounter >> 12) + g_tx433.framedata[TX433_FR_TEMPERATUR_BYTE0 +
                1];
184         g_tx433.framedata[TX433_FR_TIMESTAMP16_BYTE + 1] = (uint8_t) (framecounter >> 4);
185         g_tx433.framedata[TX433_FR_TIMESTAMP16_BYTE + 2] = (uint8_t) (framecounter << 4);
186
187
188         g_tx433.framedata[TX433_FR_TIMESTAMP16_BYTE ] = (uint8_t) (timer24 >> 12) + g_tx433.framedata[TX433_FR_TEMPERATUR_BYTE0 + 1];
```

```
189         g_tx433.framedata[TX433_FR_TIMESTAMP16_BYTE + 1] = (uint8_t) (timer24 >> 4);
190         g_tx433.framedata[TX433_FR_TIMESTAMP16_BYTE + 2] = (uint8_t) (timer24 << 4);*/
191
192
193         // Laden des Zeitstempels 16 Bit
194         // Umschreiben der wait_save_time sowie des festen Versatzes im Moment 2ms ca. 20Timer Ticks
195 //      timevalue = (uint16_t)wait_save_time + (timer24-wait_save_time) + 20;
196         timevalue = freq1; //Zeitstempel wird mit dem Frequenzwert des Licht-Sensors überschrieben
197         g_tx433.framedata[TX433_FR_TIMESTAMP16_BYTE ] = (uint8_t) (timevalue >> 12) + g_tx433.framedata[TX433_FR_TEMPERATUR_BYTE0 + 1];
198         g_tx433.framedata[TX433_FR_TIMESTAMP16_BYTE + 1] = (uint8_t) (timevalue >> 4);
199         g_tx433.framedata[TX433_FR_TIMESTAMP16_BYTE + 2] = (uint8_t) (timevalue << 4);
200
201
202 }
203
204 void frame_tx433_setAddress(uint8_t address) {
205         g_tx433.framedata[1] = address;
206 }
207
208 void frame_tx433_setParameterBit(uint8_t position) {
209         if (position >= 0x01)
210                 g_tx433.framedata[TX433_FR_PARA_BYTE0] |= position;
211 }
212
213 void frame_tx433_clearParameterBit(uint8_t position) {
214         if (position >= 0x01)
215                 g_tx433.framedata[TX433_FR_PARA_BYTE0] &= ~position;
216 }
```

B.1.10. Timerisr.c

```
1
2  #include "mainheader.h"
3  #include "global.h"
4  #include "queue.h"
5  #include "adc.h"
6  #include "tx433.h"
7
8  /*
9   * Zuerst wird der ADC gestartet und die Spannung gemessen
10  */
11 // ISR wird ausgelöst
12
13 //#pragma vector = TIMER0_A0_VECTOR
14 //__interrupt void Timer0_A0(void){
15 //
16 //
17 //      // Timervalue für alle Zeitwerte
18 //      timer24++;
19 //
20 //
21 //
22 //
23 //      //Messpin_toggle;
24 //
25 //      // Fehlerabsicherung Allgemein
26 //      if (timer24 >= 0x7FFFFF){
27 //              timer24 = 0;
28 //      }
29 //
30 //      switch (ADC) {
31 //      case (ADC_idle) :
32 //
33 //      break;
34 //
35 //      // Spannungsmessung wurde angetriggert
36 //      case ADC_busy_voltage :
37 //
38 //              // Prüfen ob ADC ready ist
39 //              if(ADC10CTL1 & ADC10BUSY){
40 //                      // do nothing. ADC not ready
41 //
42 //              }
43 //              else {
44 //                      // ADC fertig
45 //                      // Alten Spannungswert in ADC Wert für den Vergleich bereitstellen
46 //                      battvoltage_old = tmp;
47 //                      // Umstellung auf das Senden der ADC Werte da Multiplikationen und Divisionen von 32Bit Werten auf dem
       MSP430G2553 zu teuer in der Zeit und im Speicher sind
48 //                      tmp += ADC10MEM;
49 //                      tmp >>= 1;
50 //
```

```
51 //
52 //                      // Sonderprüfung notwendig für den Zellensimulator
53 //                      // Prüfung ob die gemessene Spannung unter 1 Volt gefallen ist
54 //                      if(tmp < LowVoltage){
55 //                              // ist unter 1 Volt Transmitter abschalten
56 //                              transmit = Transmitter_off;
57 //                      } else if ((tmp > LowVoltage) && (tx_status == 1)) {
58 //                              // ist über 1 Volt und Transmitter ist vorher ausgeschaltet worden
59 //                              transmit = Transmitter_on;
60 //                      }
61 //
62 //
63 //                      // Vergleich ob Inqueue - Bedingung zutrifft
64 //
65 ////                     if (((battvoltage_old + Voltagediff) < tmp) || (((battvoltage_old - Voltagediff) > tmp))) {
66 //                      if (1==2){
67 //                              ADC = ADC_inqueuewait;
68 //
69 //                              if(!(statusbitarray & 0x02)) {
70 //                                      // Inqueuebit setzen einmalig
71 //                                      statusbitarray |= 0x02;
72 //                                      timer24 = 0;
73 //                                      SR();
74 //                                      // TODO RND SHIFT MAKRO
75 //                                      wait_rnd_time = ((uint16_t)(sr_r % (2 * (TXMID / TXFAC)) + ( TXMID - (TXMID / TXFAC))));
            // Schnelles Senden
76 //                                      // Counter mit Queuetime laden
77 //                                      queuewait_counter = queuevaluetime;
78 //                                      writequeue();
79 //
80 //                              } else {
81 //                                      // Schutz das sich die Queue selbst überschreibt. Wenn Queue voll dann werden die Werte
       verworfen.
82 //                                      if(writeindex == readindex){
83 //
84 //                                      } else {
85 //                                              writequeue();
86 //                                      }
87 //                              }
88 //
89 //                      } else {
90 //                              // Inqueue trifft nicht zu
91 //
92 //
93 //                                      if(statusbitarray & 0x02){
94 //                                      // Erste Rückkehr
95 //                                      if(!(statusbitarray & 0x12)){
96 //                                              wait_save_time = queuereturntime;       // Sample Wiederholung auf 2s setzen temporär
97 //                                              statusbitarray |= 0x12; // Gegen erneutes Setzen schützen
98 //                                              queue_counter = 0;              // Zurücksetzen des 30 Werte Inqueues
99 //                                      }
100 //
101 //                                     // Rückkehr aus Inqueuebedingung Übergang Normalbetrieb
102 //                                     // Alle Werte aus der Queue senden, noch immer schnell.
103 //                                     if(writeindex != readindex){
104 //                                             wait_save_time--;
105 //                                             if(wait_save_time == 0){
106 //                                                     wait_save_time = queuereturntime; // Queuereturntime sollte 2 Sekunden
       entsprechen
107 //                                                     // Werte verlangsamt in die Queue schreiben, dadurch kann sich die Queue
       abbauen
108 //                                                     writequeue();
109 //                                             }
110 //
111 //                                     } else if ((((!(statusbitarray & 0x40)) && (!(statusbitarray & 0x20))) || ((statusbitarray & 0
       x40) && (statusbitarray & 0x20))) {
112 //                                     ////((statusbitarray & 0x40) && (statusbitarray & 0x20)) || (((!(statusbitarray & 0x40) && (!(
       statusbitarray & 0x20))) {
113 //                                             // Reset Status Rückkehr zu Normalbetrieb
114 //                                             statusbitarray = 0;
115 //                                             // Neue Wartezeit ermitteln
116 //                                             SR();
117 //                                             // TODO RND SHIFT MAKRO
118 //                                             wait_rnd_time = ((uint16_t)(sr_r % (2 * (TXMID / TXFAC)) + ( TXMID - (TXMID / TXFAC))))
        << 3 ; // slow im Mittel 4 Sekunden...
119 ////                                            wait_rnd_time = ((uint16_t)(sr_r % (2 * (TXMID / TXFAC)) + ( TXMID - (TXMID / TXFAC))))
        << 2 ; // slow im Mittel 2 Sekunden...
120 //                                             // Reset timer
121 //                                             timer24 = 0;
122 //                                     }
123 //                              } else {
124 //                                      // Fehlerabsicherung Normalbetrieb allerdings Wertebeschränkung auf 63000 / 0xF618
125 //                                      if(timer24 >= 0xF618){
126 //                                      timer24 = 0;
127 //                                      }
```

```
128 //                              }
129 //
130 //                              // Sicherung gegen erneutes auslösen
131 //                              if(!statusbitarray) {
132 //                                      wait_save_time = wait_rnd_time;
133 //                                      battvoltage = tmp;
134 //                                      statusbitarray = 0x01;
135 //                              }
136 //                              // Nun Temperatur messen,
137 //                              sample_temperature();
138 //                              ADC = ADC_busy_temperature;
139 //                      }
140 //              }
141 //
142 //              break;
143 //      case (ADC_inqueuewait) :
144 //                      queuewait_counter--;
145 //      // Status setzen damit die Senderoutine nicht den Ablauf verändert
146 //      statusbitarray |= 0x80;
147 //      // Neuen Wert messen
148 //      // Neuen Wert zwischenpuffern
149 //      if(queuewait_counter == 2)
150 //      {
151 //              sample_voltage_batt();
152 //
153 //      } else if (queuewait_counter == 1) {
154 //              // Überschreiben des Wertes damit die Mittelung den Wert nicht zu sehr verzerrt
155 //              tmp = ADC10MEM;
156 //              sample_voltage_batt();
157 //      } else if(queuewait_counter == 0) {
158 //              // ADC Wert einlesen und Mitteln
159 //              tmp += ADC10MEM;
160 //              tmp >>= 1;
161 //              // Sonderprüfung notwendig für den Zellensimulator
162 //              // Prüfung ob die gemessene Spannung unter 1 Volt gefallen ist
163 //              if(tmp < LowVoltage){
164 //                      // ist unter 1 Volt Transmitter abschalten
165 //                      transmit = Transmitter_off;
166 //              } else if ((tmp > LowVoltage) && (tx_status == 1)) {
167 //                      // ist über 1 Volt und Transmitter ist vorher ausgeschaltet worden
168 //                      transmit = Transmitter_on;
169 //              }
170 //
171 //              sample_voltage_batt();
172 //              // Inqueue mit 30 Werten FIX, Counter erhöhen und Wartezeit zurücksetzen
173 //              queue_counter++;
174 //              queuewait_counter = queuevaluetime;
175 //              // Schutz das sich die Queue selbst überschreibt. Wenn Queue voll dann werden die Werte verworfen.
176 //              if(writeindex == readindex){
177 //
178 //              } else {
179 //                      // Wert speichern
180 //                      writequeue();
181 //              }
182 //      } else if (queue_counter == queue_fix_length){
183 //              // Prüfen ob überhaupt noch Inqueue Bedingung vorliegt
184 //              sample_voltage_batt();
185 //              // Zurücksetzen
186 //              statusbitarray &= 0x7F;
187 //              ADC = ADC_busy_voltage;
188 //      }
189 //      break;
190 //
191 //      case (ADC_busy_temperature) :
192 //                      if(ADC10CTL1 & ADC10BUSY){
193 //                      // do nothing. ADC not ready
194 //                      }
195 //                      else {
196 //                              temperatur = ADC10MEM;
197 //                              sample_voltage_batt();
198 //                              ADC = ADC_busy_voltage;
199 //                      }
200 //      break;
201 //      default :
202 //              break;
203 //      }
204 //
205 //
206 //      switch(transmit){
207 //
208 //      case (IDLE):
209 //      // Antriggern des ADC
210 //      PWM_ON;
211 //      sample_voltage_batt();
212 //      if (statusbitarray & 0x80) {
```

```
213 //              // Rückkehr zur Inqueuewaitfunktion
214 //              ADC = ADC_inqueuewait;
215 //      } else {
216 //              ADC = ADC_busy_voltage;
217 //      }
218 //      transmit = wait;
219 //      break;
220 //      case (wait):
221 //      // Reduzierung der Sendewartezeit um 1.
222 //      // Beim erreichen eines Wertes von 0 darf erneut gesendet werden mit vorheriger Aufbereitung
223 //      wait_rnd_time--;
224 //      // Prüfung
225 //      if(wait_rnd_time == 0){
226 //              // Daten ins Frame laden lassen
227 //              transmit = Frameload;
228 //              ADC = ADC_idle;
229 //      }
230 //      break;
231 //      case (Frameload) :
232 //              // Prüfung ob Inqueue vorliegt
233 //              if(statusbitarray & 0x02){
234 //                      // liegt vor
235 //                      // Werte ins Frame
236 //                      frame_tx433_buffered();
237 //                      readindex++;
238 //                      if(readindex >= Queue_Length){
239 //                              readindex = 0;
240 //                              // Überschreibschutz Read mit Toggle wenn Ende der Queue erreicht wurde
241 //                              if((statusbitarray & 0x20)== 1){
242 //                                      statusbitarray &= 0xD0;
243 //                              } else {
244 //                                      statusbitarray |= 0x20;
245 //                              }
246 //
247 //                      }
248 //              } else {
249 //                      // liegt nicht vor
250 //                      frame_tx433_live();
251 //              }
252 //      transmit = Transmit_crc;
253 //      break;
254 //      case (Transmit_crc) :
255 //      frame_tx433_calc_crc();
256 //      transmit = Transmit_man;
257 //      break;
258 //      case (Transmit_man) :
259 //      frame_tx433_code_manchester();
260 //      transmit = Transmit_data_1;
261 //                      break;
262 //      case (Transmit_data_1) :
263 //              // Dauer der Interruptabschaltung und des Sendens des Frames an den Transmitter : 1.96ms < x < 2.08ms
264 //              _DINT();
265 //              // Daten an Transmitter senden
266 //              tx433_cmd_fifo_set();
267 //              transmit = Transmit;
268 //              //break;
269 //              // Zwangspause erzeugen sonst sendet der Transmitter nicht.
270 //              _EINT();
271 //              //transmit = Transmit;
272 //      //break;
273 //      case (Transmit) :
274 //      _DINT();
275 //      // Dem Transmitter das Sendekommando übermitteln
276 //      tx433_cmd_tx_start();
277 //      LED_toggle;
278 //      // Neue Zufallszahl
279 //      SR();
280 //      // Prüfung ob Inqueue vorliegt
281 //      if(!(statusbitarray & 0x02)){
282 //              // liegt nicht vor
283 //              // Neue Wartezeit
284 //              // TODO RND SHIFT MAKRO
285 //              wait_rnd_time = ((uint16_t)(sr_r % (2 * (TXMID / TXFAC)) + ( TXMID - (TXMID / TXFAC)))) << 2 ; // slow im Mittel 2
        Sekunden...
286 //              //wait_rnd_time = ((uint16_t)(sr_r % (2 * (TXMID / TXFAC)) + ( TXMID - (TXMID / TXFAC)))) << 3; // slow im Mittel 4
        Sekunden
287 //              timer24 = 0;
288 //              statusbitarray = 0x00;
289 //      } else {
290 //              // liegt vor
291 //              // Neue Wartezeit
292 //              wait_rnd_time = ((uint16_t)(sr_r % (2 * (TXMID / TXFAC)) + ( TXMID - (TXMID / TXFAC))));// fast
293 //      }
294 //
295 //      transmit = IDLE;
```

```
296 //      _EINT();
297 //      break;
298 //      case (Transmitter_off) :
299 //              // Flag setzen für Transmitterabschaltung
300 //              tx_status = 1;
301 //              TX433_OFF;
302 //      break;
303 //      case (Transmitter_on) :
304 //              // Globale Interrupts deaktivieren
305 //              _DINT();
306 //              // Transmitter anschalten
307 //              TX433_ON;
308 //              // Zwangspause Transmitter ON != Ready
309 //               __delay_cycles(250000);
310 //               // Initalisierung des Transmitters
311 //               tx433_init();
312 //               // Flag zurücksetzen für Transmitterabschaltung
313 //               tx_status = 0;
314 //               // Globale Interrupts aktivieren
315 //              _EINT();
316 //              // Rückkehr zum normalen Status
317 //              transmit = IDLE;
318 //              break;
319 //      default :
320 //                      break;
321 //      }
322 //
323 //
324 //}
325
326
327 #pragma vector=PORT2_VECTOR
328 __interrupt void P2_4 (void){
329         if(P2IFG&BIT4){
330                 if(z1==0){
331                         TA1CTL |=MC_1+TACLR; //Timerstarten
332                 }
333                         z1++;
334                         P2IFG &= ~BIT4;
335         }
336 }
337
338 #pragma vector=TIMER1_A1_VECTOR
339 __interrupt void TA1_ISR(void){
340         switch(TA1IV){
341                 case 0x02:{
342                         TA1CTL|=MC_0;
343                         P2IE &= ~BIT4;
344                         PWM_OFF;
345 //                       ticks=TA1CCR0+1;
346                         ticks=(z1<<1);
347                         z1=0;
348                         TA1CCTL1&=~CCIFG;
349                         //TA1CTL |= MC_1 + TACLR;
350
351                         break;
352                 }
353         }
354 }
355
356 //#pragma vector=TIMER1_A1_VECTOR
357 //__interrupt void TA1_ISR(void){
358 //      switch(TA1IV){
359 //              case 0x04:{
360 //                      PWM_OFF;
361 //                      //TA1CTL|=MC_0;
362 //                      //ticks=TA1CCR2+1;
363 ////                    freq=(ticks);
364 //                      z1=0;
365 //                      TA1CCTL2&=~CCIFG;
366 ////                    TA1CCTL2&=~COV;
367 ////                    TA1CTL |= MC_2 + TACLR;
368 //                      //LED_toggle;
369 //                      PWM_ON;
370 //                        break;
371 //              }
372 //              case 0x0A:
373 //                      PWM_OFF;
374 //                      TA1CTL|=MC_0;
375 //                      z2++;
376 //                      ticks=TA1CCR2+1;
377 //                      TA1CCTL2&=~COV;
378 //                      PWM_ON;
379 //      }
380 //}
```

B.1.11. tx433.c

```
1
2
3  #include "mainheader.h"
4  #include "global.h"
5  #include "tx433.h"
6  #include "i2cbus.h"
7
8  //---- Function: tx433_init() ------------------------------------------------
9  uint8_t tx433_init(void) {
10
11         uint8_t ok;
12
13 //      g_tx433.idlemode = TX433_IDLEMODE_TUNE;                  // --> 370us response to TX
14         g_tx433.idlemode = TX433_IDLEMODE_STANDBY;               // --> 6.6ms response to TX
15         ok = tx433_cmd_tx_stop();
16         if(ok != EXIT_SUCCESS){
17                 return EXIT_FAILURE;
18         }
19
20
21         ok = tx433_set_prop(TX433_PROP_CHIP_CONFIG);
22         if(ok != EXIT_SUCCESS){
23                 return EXIT_FAILURE;
24         }
25
26
27
28         ok = tx433_set_prop(TX433_PROP_TUNE_INTERVAL);
29         if(ok != EXIT_SUCCESS){
30                 return EXIT_FAILURE;
31                 }
32         ok = tx433_set_prop(TX433_PROP_MODULATION_FSKDEV);
33         if(ok != EXIT_SUCCESS){
34                 return EXIT_FAILURE;
35                 }
36
37         ok = tx433_set_prop(TX433_PROP_TX_FREQUENCY);
38         if(ok != EXIT_SUCCESS){
39                 return EXIT_FAILURE;
40                 }
41         ok = tx433_set_prop(TX433_PROP_PA_CONFIG);
42         if(ok != EXIT_SUCCESS){
43                 return EXIT_FAILURE;
44                 }
45         ok = tx433_set_prop(TX433_PROP_BITRATE_CONFIG);
46         if(ok != EXIT_SUCCESS){
47                 return EXIT_FAILURE;
48                 }
49 /*      ok = tx433_cmd_set_int();
50         if(ok != EXIT_SUCCESS){
51                 return EXIT_FAILURE;
52                 }
53                 */
54         ok = tx433_cmd_fifo_init();
55         if(ok != EXIT_SUCCESS){
56                 return EXIT_FAILURE;
57                 }
58
59         return EXIT_SUCCESS;
60 }
61 //---- Function: tx433_set_prop() --------------------------------------------
62 uint8_t tx433_get_prop(uint8_t property) {
63
64         // not needed
65         return 1;
66 }
67 //---- Function: tx433_set_prop() --------------------------------------------
68 uint8_t tx433_set_prop(uint8_t property) {
69
70         uint8_t ok, bytes;
71         uint8_t data[8]; // max 2+6 byte data to sent
72
73         data[0] = TX433_CMD_SET_PROPERTY;         // Command = Set_Property
74         data[1] = property;                       // Property ID
75          // Splitting of the property data into
76          // single bytes is replaced as constants
77          // by the compiler.
78          // data[2] <-- MSB of property data
79          // data[n] <-- LSB, nmax = 7
80         switch(property) {
81  case TX433_PROP_CHIP_CONFIG:
82         bytes    = TX433_PROP_CHIP_CONFIG_BCNT;
83         data[2] = TX433_PROP_CHIP_CONFIG_DATA;
```

```
84          break;
85   case TX433_PROP_LED_INTENSITY:
86          bytes    = TX433_PROP_CHIP_CONFIG_BCNT;
87          data[2] = TX433_PROP_LED_INTENSITY_DATA;
88          break;
89   case TX433_PROP_MODULATION_FSKDEV:
90          bytes    = TX433_PROP_MODULATION_FSKDEV_BCNT;
91          data[2] = (uint8_t)(TX433_PROP_MODULATION_FSKDEV_DATA >> 8);
92          data[3] = (uint8_t)(TX433_PROP_MODULATION_FSKDEV_DATA & 0xFF);
93          break;
94   case TX433_PROP_TUNE_INTERVAL:
95          bytes    = TX433_PROP_TUNE_INTERVAL_BCNT;
96          data[2] = (uint8_t)(TX433_PROP_TUNE_INTERVAL_DATA >> 8);
97          data[3] = (uint8_t)(TX433_PROP_TUNE_INTERVAL_DATA & 0xFF);
98          break;
99   case TX433_PROP_FIFO_THRESHOLD:
100         bytes    = TX433_PROP_FIFO_THRESHOLD_BCNT;
101         data[2] = (uint8_t)(TX433_PROP_FIFO_THRESHOLD_DATA >> 16);
102         data[3] = (uint8_t)(TX433_PROP_FIFO_THRESHOLD_DATA >> 8);
103         data[4] = (uint8_t)(TX433_PROP_FIFO_THRESHOLD_DATA & 0xFF);
104         break;
105  case TX433_PROP_BITRATE_CONFIG:
106         bytes = TX433_PROP_BITRATE_CONFIG_BCNT;
107         data[2] = (uint8_t)((uint32_t)TX433_PROP_BITRATE_CONFIG_DATA >> 16);
108         data[3] = (uint8_t)(TX433_PROP_BITRATE_CONFIG_DATA >> 8);
109         data[4] = (uint8_t)(TX433_PROP_BITRATE_CONFIG_DATA & 0xFF);
110         break;
111  case TX433_PROP_TX_FREQUENCY:
112         bytes = TX433_PROP_TX_FREQUENCY_BCNT;
113         data[2] = (uint8_t)(TX433_PROP_TX_FREQUENCY_DATA >> 24);
114         data[3] = (uint8_t)(TX433_PROP_TX_FREQUENCY_DATA >> 16);
115         data[4] = (uint8_t)(TX433_PROP_TX_FREQUENCY_DATA >> 8);
116         data[5] = (uint8_t)(TX433_PROP_TX_FREQUENCY_DATA & 0xFF);
117         break;
118  case TX433_PROP_LBD_CONFIG:
119  bytes = TX433_PROP_LBD_CONFIG_BCNT;
120  data[2] = (uint8_t)(TX433_PROP_LBD_CONFIG_DATA >> 24);
121  data[3] = (uint8_t)(TX433_PROP_LBD_CONFIG_DATA >> 16);
122  data[4] = (uint8_t)(TX433_PROP_LBD_CONFIG_DATA >> 8);
123  data[5] = (uint8_t)(TX433_PROP_LBD_CONFIG_DATA & 0xFF);
124  break;
125                 case TX433_PROP_PA_CONFIG:
126  bytes = TX433_PROP_PA_CONFIG_BCNT;
127  data[2] = (uint8_t)(TX433_PROP_PA_CONFIG_DATA_P2 >> 8);
128  data[3] = (uint8_t)(TX433_PROP_PA_CONFIG_DATA_P2 & 0xFF);
129  data[4] = (uint8_t)(TX433_PROP_PA_CONFIG_DATA_P1 >> 8);
130  data[5] = (uint8_t)(TX433_PROP_PA_CONFIG_DATA_P1 & 0xFF);
131  data[6] = (uint8_t)(TX433_PROP_PA_CONFIG_DATA_P0 >> 8);
132  data[7] = (uint8_t)(TX433_PROP_PA_CONFIG_DATA_P0 & 0xFF);
133  break;
134                 default:
135                         return EXIT_FAILURE;
136         }
137         bytes = bytes + 2;        // add 2 for command and property
138
139         ok = i2c_bus_write(ADDR_TX433, bytes, data); // Write Data
140         if(ok != EXIT_SUCCESS)
141                 return EXIT_FAILURE;
142
143         ok = i2c_bus_read(ADDR_TX433, 1, data);  // Read Reply
144         if(ok != EXIT_SUCCESS)
145                 return EXIT_FAILURE;
146         g_tx433.return_status = data[0];
147
148         return EXIT_SUCCESS;
149  }
150  //---- Function: tx433_cmd_tx_start() ----------------------------------------
151  uint8_t tx433_cmd_tx_start(void) {
152
153                 // Time to transmit the Start Command over SMBUS ~ 1.14ms @ 100kbit/s
154                 // Time to TX after received Start Command = 370us @ Idlemode = Tune
155                 //          = 6.6ms @ Idlemode = Standby
156
157         uint8_t ok;
158         uint8_t data[6];
159
160         data[0] = TX433_CMD_TX_START; // Command = TX_START
161         data[1] = 0x00;    // PacketSize[15:8]
162         data[2] = 0x13; // 1b  // PacketSize[ 7:0]
163                  //      = TX433_FRAME_BYTES_MCH = 27
164         data[3] = (0x02 & g_tx433.state);        // Sensor State,
165                  //      FIFO Auto-TX disable
166         data[4] = g_tx433.idlemode;                      // Idle Mode
167  //     data[5] = TX433_DTMOD_CW;                        // DataTransmission Mode = CW
168         data[5] = TX433_DTMOD_FIFO;               // DataTransmission Mode = FIFO
```

```
169
170         ok = i2c_bus_write(ADDR_TX433, sizeof(data), data);      // Write Data
171         if(ok != EXIT_SUCCESS){
172                 return EXIT_FAILURE;
173                 }else {
174                         return EXIT_SUCCESS;
175                 }
176         }
177
178         /*
179         ok = i2c_bus_read(ADDR_TX433, 2, data);           // Read Reply
180         if(ok != EXIT_SUCCESS){
181                 return EXIT_FAILURE;
182                         }
183
184
185         g_tx433.return_status = data[0];
186         g_tx433.actdatatsize  = data[1];
187         if(g_tx433.return_status == TX433_RESTATUS_CTS_OK) {
188                 return EXIT_SUCCESS;
189         }
190         else {
191                 return EXIT_FAILURE;
192         }
193         */
194 //}
195 //---- Function: tx433_cmd_tx_stop() ------------------------------------------
196 uint8_t tx433_cmd_tx_stop(void) {
197
198         uint8_t ok;
199         uint8_t data[3];
200
201         data[0] = TX433_CMD_TX_STOP; // Command = TX_STOP
202         data[1] = TX433_STATE_IDLE;       // Sensor State = Idle
203         data[2] = g_tx433.idlemode;       // Idle Mode
204
205
206         ok = i2c_bus_write(ADDR_TX433, sizeof(data), data);      // Write Data
207         if(ok != EXIT_SUCCESS)
208                 return EXIT_FAILURE;
209
210         ok = i2c_bus_read(ADDR_TX433, 1, data);  // Read Reply
211         if(ok != EXIT_SUCCESS)
212                 return EXIT_FAILURE;
213         g_tx433.return_status = data[0];
214
215         return EXIT_SUCCESS;
216 }
217 //---- Function: tx433_cmd_fifo_init() ----------------------------------------
218 uint8_t tx433_cmd_fifo_init(void) {
219
220         uint8_t ok;
221         uint8_t data[1];
222
223         data[0] = TX433_CMD_INIT_FIFO;    // Command = Init_FIFO
224
225         ok = i2c_bus_write(ADDR_TX433, sizeof(data), data);      // Write Data
226         if(ok != EXIT_SUCCESS)
227                 return EXIT_FAILURE;
228
229         ok = i2c_bus_read(ADDR_TX433, 1, data);           // Read Reply
230         if(ok != EXIT_SUCCESS)
231                 return EXIT_FAILURE;
232         g_tx433.return_status = data[0];
233
234         return EXIT_SUCCESS;
235 }
236 //---- Function: tx433_cmd_fifo_set() -----------------------------------------
237 uint8_t tx433_cmd_fifo_set(void) {
238
239         uint8_t ok;
240         uint8_t data[1];
241          // Write Data
242         ok = i2c_bus_write(ADDR_TX433, TX433_FRAME_BYTES_MCH+1, (uint8_t *) g_tx433.txdata);
243         if(ok != EXIT_SUCCESS)
244                 return EXIT_FAILURE;
245
246         ok = i2c_bus_read(ADDR_TX433, 1, data);  // Read Reply
247         if(ok != EXIT_SUCCESS)
248                 return EXIT_FAILURE;
249         g_tx433.return_status = data[0];
250
251         return EXIT_SUCCESS;
252 }
253
```

```
254 uint8_t tx433_cmd_fifo_set_part1(void){
255
256                 uint8_t ok;
257  // Write Data
258                 //ok = i2c_bus_writedata(1);
259                 ok = i2c_bus_write(ADDR_TX433, TX433_FRAME_BYTES_MCH+1, (uint8_t *) g_tx433.txdata);
260                 if(ok != EXIT_SUCCESS)
261                         return EXIT_FAILURE;
262
263                 /*
264                 ok = i2c_bus_read(ADDR_TX433, 1, data);                         // Read Reply
265                 if(ok != EXIT_SUCCESS)
266                         return EXIT_FAILURE;
267                 g_tx433.return_status = data[0];
268                 */
269                 return EXIT_SUCCESS;
270
271 }
272 /*
273 uint8_t tx433_cmd_fifo_set_part2(void){
274
275                 uint8_t ok;
276                                                                                                         // Write Data
277                 ok = i2c_bus_writedata(!1);
278                 if(ok != EXIT_SUCCESS)
279                         return EXIT_FAILURE;
280
281
282                 ok = i2c_bus_read(ADDR_TX433, 1, data);                         // Read Reply
283                 if(ok != EXIT_SUCCESS)
284                         return EXIT_FAILURE;
285                 g_tx433.return_status = data[0];
286
287                 return EXIT_SUCCESS;
288
289
290 }
291
292 //--- Function: tx433_cmd_set_int() ------------------------------------------
293 uint8_t tx433_cmd_set_int(void) {
294
295         uint8_t ok;
296         uint8_t data[2];
297
298         data[0] = TX433_CMD_SET_INT;                                            // Command = SET_Interrupt
299         data[1] = TX433_IRQ_PACKET_SENT;                                        // Set IRQ:
300                                                                                                 //      - Packet Sent
301
302         ok = i2c_bus_write(ADDR_TX433, sizeof(data), data);     // Write Data
303         if(ok != EXIT_SUCCESS)
304                 return EXIT_FAILURE;
305
306         ok = i2c_bus_read(ADDR_TX433, 1, data);                         // Read Reply
307         if(ok != EXIT_SUCCESS)
308                 return EXIT_FAILURE;
309         g_tx433.return_status = data[0];
310
311         return EXIT_SUCCESS;
312 }
313 //--- Function: tx433_cmd_set_int() ------------------------------------------
314 uint8_t tx433_cmd_get_int_status(void) {
315
316         uint8_t ok;
317         uint8_t data[2];
318
319         data[0] = TX433_CMD_GET_INT_STATUS;                                     // Command = GET_Interrupt_Status
320
321         ok = i2c_bus_write(ADDR_TX433, 1, data);                        // Write Command
322
323
324         ok = i2c_bus_read(ADDR_TX433, 2, data);                         // Readout Response
325         if(ok == 1) {
326                 g_tx433.return_status = data[0];
327                 g_tx433.irq_status = data[1];
328                 return 1;
329         }
330         else
331                 return 0;
332 }
333 //--- Function: tx433_cmd_get_state() ----------------------------------------
334 uint8_t tx433_cmd_get_state(void) {
335
336         uint8_t ok;
337         uint8_t data[6];
338
```

```
339         data[0] = TX433_CMD_GET_STATE;                                  // Command = GET_State
340
341         ok = i2c_bus_write(ADDR_TX433, 1, data);                        // Write Command
342         if(ok != EXIT_SUCCESS)
343                 return EXIT_FAILURE;
344
345         ok = i2c_bus_read(ADDR_TX433, 6, data);                         // Readout Response
346         if(ok == EXIT_SUCCESS) {
347  g_tx433.error = data[0] & 0x7F;
348  g_tx433.cts       = data[0] & 0x80;
349  g_tx433.state = data[1] & 0x03;
350  g_tx433.idlemode = data[2] & 0x07;                             // if state = TX --> DTMode
351  g_tx433.acttxpktsize  = (uint16_t) data[3] << 8;
352  g_tx433.acttxpktsize |= data[4];
353  g_tx433.prverror = data[5];
354  return EXIT_SUCCESS;
355         }
356         else
357                 return EXIT_FAILURE;
358 }
359
360 //--- Function: tx433_change_state() ------------------------------------------
361 uint8_t tx433_cmd_change_state(void) {
362
363         uint8_t ok;
364         uint8_t data[3];
365
366         data[0] = TX433_CMD_CHANGE_STATE;                               // Command = Change_State
367         data[1] = g_tx433.state;
368         data[2] = g_tx433.idlemode;
369
370         ok = i2c_bus_write(ADDR_TX433, 3, data);                        // Write Command
371         if(ok != EXIT_SUCCESS)
372                 return EXIT_FAILURE;
373
374         ok = i2c_bus_read(ADDR_TX433, 1, data);                         // Read Reply
375         if(ok != EXIT_SUCCESS)
376                 return EXIT_FAILURE;
377         g_tx433.return_status = data[0];
378
379         return EXIT_SUCCESS;
380 }*/
```

B.1.12. tx433.h

```
1
2  #include <stdint.h>
3
4  #ifndef TX_433_H_
5  #define TX_433_H_
6
7          #define ADDR_TX433                                      0x70            // Address of 433MHz transmitter
8
9          //___ Commands: ______________________________________
10         #define TX433_CMD_GET_REV                       0x10            // Return product and revision info
11         #define TX433_CMD_SET_PROPERTY          0x11            // Set property
12         #define TX433_CMD_GET_PROPERTY          0x12            // Get property
13         #define TX433_CMD_LED_CTRL                      0x13            // Turn LED on/off
14         #define TX433_CMD_CHANGE_STATE          0x60            // Change power state
15         #define TX433_CMD_GET_STATE                     0x61            // Get state
16         #define TX433_CMD_TX_START                      0x62            // Start transmission
17         #define TX433_CMD_TX_STOP                       0x67            // Stop transmission
18         #define TX433_CMD_SET_INT                       0x63            // Interrupt control
19         #define TX433_CMD_GET_INT_STATUS        0x64            // Get interrupt status
20         #define TX433_CMD_INIT_FIFO                     0x65            // Clear TX FIFO
21         #define TX433_CMD_SET_FIFO                      0x66            // Store data in TX FIFO
22         #define TX433_CMD_GET_BAT_STATUS        0x68            // Get battery status (Vdd voltage)
23
24         //___ Properties: ____________________________________
25         #define TX433_PROP_CHIP_CONFIG                  0x10    // FSK dev, LSB first, ext. XO
26         #define TX433_PROP_LED_INTENSITY                0x11    // LED current drive strength
27         #define TX433_PROP_MODULATION_FSKDEV    0x20    // MOD type and FSK dev
28         #define TX433_PROP_TUNE_INTERVAL                0x21    // Tuning interval in sec
29         #define TX433_PROP_FIFO_THRESHOLD               0x30    // FIFO threshold
30         #define TX433_PROP_BITRATE_CONFIG               0x31    // Date rate and ramp if OOK
31         #define TX433_PROP_TX_FREQUENCY                 0x40    // Carrier if OOK, upper if FSK
32         #define TX433_PROP_LBD_CONFIG                   0x41    // Low battery voltage threshold
33         #define TX433_PROP_XO_CONFIG                    0x50    // XO config
34         #define TX433_PROP_PA_CONFIG                    0x60    // PA config
35
36         //___ Data Values for Properties: (MSB in DATA1) _____
37         #define TX433_PROP_CHIP_CONFIG_DATA                     0x00    // stand for: intern oscillator,
```

```
38 //                          MSB first
39         #define TX433_PROP_CHIP_CONFIG_BCNT                    1               // Number of Bytes = 1
40
41         #define TX433_PROP_LED_INTENSITY_DATA          0x03    // stand for: LED current =0.97mA
42         #define TX433_PROP_LED_INTENSITY_BCNT          1               // Number of Bytes = 1
43
44         #define TX433_PROP_MODULATION_FSKDEV_DATA      0x0000  // stand for: OOK-Modulation
45 //      #define TX433_PROP_MODULATION_FSKDEV_DATA      0x013F  // stand for: FSK-Modulation
46 ////                        biFSKDev = 63
47         #define TX433_PROP_MODULATION_FSKDEV_BCNT      2               // Number of Bytes = 2
48
49         #define TX433_PROP_TUNE_INTERVAL_DATA          0x000A  // stand for: 10s tuning interval
50         #define TX433_PROP_TUNE_INTERVAL_BCNT          2               // Number of Bytes = 2
51
52         #define TX433_PROP_FIFO_THRESHOLD_DATA 0x701020        // stand for: almost full = 240
53 //                          almost empt = 16
54 //                          auto tx          = 32
55         #define TX433_PROP_FIFO_THRESHOLD_BCNT 3                       // Number of Bytes = 3
56
57         #define TX433_PROP_BITRATE_CONFIG_DATA 0x006402        // stand for: 10.000bps,
58 //      #define TX433_PROP_BITRATE_CONFIG_DATA 0x01F402        // stand for: 50.000bps,
59 //                          ramp rate = 2us
60         #define TX433_PROP_BITRATE_CONFIG_BCNT 3                       // Number of Bytes = 3
61
62         #define TX433_PROP_TX_FREQUENCY_DATA   0x19DDC7C8      // stand for: 433.965.000 Hz
63         #define TX433_PROP_TX_FREQUENCY_BCNT   4                       // Number of Bytes = 4
64
65         #define TX433_PROP_LBD_CONFIG_DATA             0x09C4003C      // stand for: threshold = 2500mV
66 //                          interval  = 60s
67         #define TX433_PROP_LBD_CONFIG_BCNT             4 // Number of Bytes = 4
68
69 // TX433_PROP_PA_CONFIG_DATA = 0x014600807D7F
70 // splitted into 3 parts, stands for:
71         #define TX433_PROP_PA_CONFIG_DATA_P2   0x0146          //              max current drive,
72 //                          PA level    = 70
73 //      #define TX433_PROP_PA_CONFIG_DATA_P2   0x0023          //                      limit current drive
74 //                          PA level    = 35
75
76         #define TX433_PROP_PA_CONFIG_DATA_P1   0x0080          //                      PA cap      = 128,
77         #define TX433_PROP_PA_CONFIG_DATA_P0   0x7D7F          //                      fAlphaSteps = 125,
78 //                          fBetaSteps  = 127,
79         #define TX433_PROP_PA_CONFIG_BCNT      6                               // Number of Bytes = 6
80
81         //___ States & IdleModes: _____________________
82         #define TX433_STATE_IDLE                       0x00
83         #define TX433_STATE_SHUTDOWN           0x01
84         #define TX433_STATE_TX                         0x10
85         #define TX433_IDLEMODE_STANDBY         0x00
86         #define TX433_IDLEMODE_SENSOR          0x01
87         #define TX433_IDLEMODE_TUNE                    0x02
88
89         //___ DataTransmission Mode: __________________
90         #define TX433_DTMOD_FIFO                       0x00
91         #define TX433_DTMOD_CW                         0x01
92         #define TX433_DTMOD_PN90                       0x02
93         #define TX433_DTMOD_PN91                       0x03
94
95         //___ Interrupts: _____________________________
96         #define TX433_IRQ_FIFO_UFLOW           0x80
97         #define TX433_IRQ_FIFO_OFLOW           0x10
98         #define TX433_IRQ_FIFO_AEMTY           0x20
99         #define TX433_IRQ_FIFO_AFULL           0x40
100        #define TX433_IRQ_PACKET_SENT          0x08
101        #define TX433_IRQ_LOW_BAT                      0x04
102        #define TX433_IRQ_TUNE                         0x02
103        #define TX433_IRQ_POR                          0x01
104
105        //___ Return Status: __________________________
106        #define TX433_RESTATUS_CTS_OK          0x80
107
108        //--- Prototyps: ------------------------------------------
109        uint8_t tx433_init(void);
110        uint8_t tx433_set_prop(const uint8_t);
111        uint8_t tx433_get_prop(const uint8_t);
112        uint8_t tx433_cmd_tx_start(void);
113        uint8_t tx433_cmd_tx_stop(void);
114        uint8_t tx433_cmd_led_ctrl(const uint8_t);
115        uint8_t tx433_cmd_fifo_init(void);
116        uint8_t tx433_cmd_fifo_set(void);
117        uint8_t tx433_cmd_set_int(void);
118        uint8_t tx433_cmd_get_int_status(void);
119        uint8_t tx433_cmd_get_state(void);
120        uint8_t tx433_cmd_change_state(void);
121
122        uint8_t tx433_cmd_fifo_set_part1(void);
```

```
123         uint8_t tx433_cmd_fifo_set_part2(void);
124
125
126 #endif /* TX_433_H_ */
```

B.1.13. queue.c

```
 1
 2
 3 // Handling mit der Queue
 4
 5 #include "mainheader.h"
 6 #include "global.h"
 7 #include "queue.h"
 8
 9 // Init der Queue mit 0 Werten
10 void initqueue(){
11
12         // Freerun des Randomshift Makros und init Queue
13         for (timer24 = 0;timer24 < Queue_Length;timer24++){
14             SR();
15             waitingqueue[timer24].timelabel24_1 = 0;
16             waitingqueue[timer24].timelabel24_2 = 0;
17             waitingqueue[timer24].timelabel24_3 = 0;
18             waitingqueue[timer24].voltage12_1 = 0;
19             waitingqueue[timer24].voltage12_2 = 0;
20             }
21         timer24 = 0;
22
23  }
24
25
26 // Schreiben in die Queue an die nächste verfügbare Queueposition
27 void writequeue(){
28
29         waitingqueue[writeindex].timelabel24_1 = (uint8_t) (timer24 >> 16UL);    // 16UL = 16Bit System und LongValue
30         waitingqueue[writeindex].timelabel24_2 = (uint8_t) (timer24 >> 8);
31         waitingqueue[writeindex].timelabel24_3 = (uint8_t) timer24;
32         // Voltagevalue
33         waitingqueue[writeindex].voltage12_1 = (uint8_t) (tmp >> 4);
34         waitingqueue[writeindex].voltage12_2 = (uint8_t)(tmp << 4);
35         //waitingqueue[writeindex].voltage12_2 |= Sendewdh;      // WDH Bit setzen
36         writeindex++;
37
38         // Reset des Schreibindexes
39         if(writeindex >= Queue_Length){
40         writeindex = 0;
41         // Überschreibschutz Write
42         if ((statusbitarray & 0x40) == 1){
43                 statusbitarray &= 0xB0;
44         } else {
45                 statusbitarray |= 0x40;
46         }
47
48         }
49 }
50
51  void readqueue(){
52
53  }
54
55
56  // Kopieren des aktuellen Queuewertes an die nächste verfügbare Schreibposition und löschen des WDH Bits
57  void copyqueue(){
58
59          waitingqueue[writeindex].timelabel24_1 = waitingqueue[readindex].timelabel24_1;
60          waitingqueue[writeindex].timelabel24_2 = waitingqueue[readindex].timelabel24_2;
61          waitingqueue[writeindex].timelabel24_3 = waitingqueue[readindex].timelabel24_3;
62          waitingqueue[writeindex].voltage12_1 = waitingqueue[readindex].voltage12_1;
63          waitingqueue[writeindex].voltage12_2 = waitingqueue[readindex].voltage12_2 & 0xF0; // Wdh Bit löschen
64  }
65
66  void clearentry() {
67          waitingqueue[readindex].timelabel24_1 = 0;
68          waitingqueue[readindex].timelabel24_1 = 0;
69          waitingqueue[readindex].timelabel24_1 = 0;
70          waitingqueue[readindex].voltage12_1 = 0;
71          waitingqueue[readindex].voltage12_2 = 0;
72
73  }
```

B.1.14. queue.h

```
1
2
3  #ifndef QUEUE_H_
4  #define QUEUE_H_
5
6  void writequeue(void);
7  void readqueue(void);
8  void initqueue(void);
9  void copyqueue(void);
10 void clearentry(void);
11
12
13 #endif /* QUEUE_H_ */
```

B.1.15. i2cbus.c

```
1
2
3  #include "i2cbus.h"
4  #include "mainheader.h"
5  #include "tx433.h"
6  #include "global.h"
7
8  //---- Function: i2c_bus_write() ----------------------------------------------------
9  uint8_t i2c_bus_write(uint8_t div_address, uint8_t data_length, uint8_t *data) {
10
11         uint8_t i;
12
13         if((UCB0CTL1 & UCSWRST) != UCSWRST) {                    // when I2C is activ
14                 return EXIT_FAILURE;                                        //      then exit
15         }
16         UCB0CTL1 |= UCTR;                                                   // set transmitter-mode
17         UCB0I2CSA = div_address;                                            // set slave address of sensor
18         UCB0CTL1 &= ~UCSWRST;                                               // unset SWRESET
19         UCB0CTL1 |= UCTXSTT;                                                // send START-CON
20         UCB0TXBUF = data[0];                                                // then write data
21
22         for(i = 1; i < data_length; i++) {
23                 while(TRUE) {                                               // wait until ...
24                         if((UCB0STAT & UCNACKIFG) == UCNACKIFG){ // NACK from slave
25                                 UCB0CTL1 |= UCTXSTP;                        //      then send STOP-CON
26                                 UCB0STAT &= ~UCNACKIFG;                     //      reset flag
27                                 UCB0CTL1 |= UCSWRST;                        //      set SWRESET
28                                 return EXIT_FAILURE;                        //      and exit
29                         }
30                         if((IFG2 & UCB0TXIFG) == UCB0TXIFG) {    // data / start-con was send
31                                 UCB0TXBUF = data[i];                        //      then write data
32                                 break;
33                         }
34                 }
35         }
36         while((IFG2 & UCB0TXIFG) != UCB0TXIFG);                  // wait until data/start-con was send
37         UCB0CTL1 |= UCTXSTP;                                                // send STOP-COND
38
39         while((UCB0CTL1 & UCTXSTP) == UCTXSTP);                  // wait until STOP-con was send
40         UCB0CTL1 |= UCSWRST;                                                // set SWRESET
41
42         return EXIT_SUCCESS;
43 }
44
45 //---- Function: i2c_bus_read() -----------------------------------------------------
46 uint8_t i2c_bus_read(uint8_t div_address, uint8_t data_length, uint8_t *data) {
47
48         uint8_t i;
49
50         if((UCB0CTL1 & UCSWRST) != UCSWRST) {                    // when I2C is activ
51                 return EXIT_FAILURE;                                        //      then exit
52         }
53         UCB0CTL1 &= ~UCTR;                                                  // reset transmitter-mode
54         UCB0I2CSA = div_address;                                            // set slave address of sensor
55         UCB0CTL1 &= ~UCSWRST;                                               // unset SWRESET
56         UCB0CTL1 |= UCTXSTT;                                                // send START-CON
57
58         if(data_length > 1) {
59                 for(i = 0; i < data_length; i++) {
60                         while(TRUE) {                                       // wait until ...
61                                 if((UCB0STAT & UCNACKIFG) == UCNACKIFG){    // NACK from slave
62                                         UCB0CTL1 |= UCTXSTP;                //      then send STOP-CON
63                                         UCB0STAT &= ~UCNACKIFG;             //      reset flag
```

```
64                                      UCB0CTL1 |= UCSWRST;                    //      set SWRESET
65                                      return EXIT_FAILURE;                    //      and exit
66                              }
67                              if((IFG2 & UCB0RXIFG) == UCB0RXIFG) {           // data / start-con was send
68                                      data[i] = UCB0RXBUF;                    //      readout data
69                                      if(i == data_length-2)                  //      if next byte will be the last one
70                                              UCB0CTL1 |= UCTXSTP;            //      send STOP-CON after next receive
71                                      break;
72                              }
73                      }
74              }
75      }
76      else {
77              while((UCB0CTL1 & UCTXSTT) == UCTXSTT);         // wait for acknowledge of slave,
78              UCB0CTL1 |= UCTXSTP;                                            // then send STOP-COND immediately
79              if((UCB0STAT & UCNACKIFG) == UCNACKIFG) {       // if NACK from slave
80                      UCB0CTL1 |= UCSWRST;                                    //      set SWRESET
81                      return EXIT_FAILURE;                                    //      and exit
82              }
83              data[0] = UCB0RXBUF;                                            // readout data
84      }
85
86      while((UCB0CTL1 & UCTXSTP) == UCTXSTP);                 // wait until STOP-con was send
87      UCB0CTL1 |= UCSWRST;                                                    // set SWRESET
88
89      return EXIT_SUCCESS;
90 }
91 //------------------------------------------------------------------------------
```

B.1.16. i2cbus.h

```
1
2
3  #include <stdint.h>
4
5  #ifndef I2C_BUS_H_
6  #define I2C_BUS_H_
7
8          uint8_t i2c_bus_write(uint8_t div_address, uint8_t data_length, uint8_t *data);
9          uint8_t i2c_bus_read(uint8_t div_address, uint8_t data_length, uint8_t *data);
10
11         // Verändertes Schreiben auf den I2C Bus Grund : Länge der Datenübertragung grösser 100uS
12         //uint8_t i2c_bus_writedata(uint8_t part);
13
14
15 #endif /* I2C_BUS_H_ */
```

B.1.17. Sensor0.h

```
1
2  #ifndef SENSOR_0_H_
3  #define SENSOR_0_H_
4
5  #define Sensoradresse    0x10
6  #define Voltagecal               0x01
7  #define Temperaturcal    0x01
8
9  #endif /* SENSOR_0_H_ */
```

B.1.18. Sensor1.h

```
1
2
3  #ifndef SENSOR_1_H_
4  #define SENSOR_1_H_
5
6  #define Sensoradresse    0x01        //  tatsächliche Sensoradresse
7  #define SENSOR_COUNT     10          //entspricht die Anzahl der verwendeten Sensoren (wird für die Manipulation der Sensoradresse
       benötigt)
8
9  /* werden hier nicht mehr gebraucht (Kalibrierung findet der MATLAB-Auswertesoftware statt)*/
10 //------Kalibrierwerte Spannung------  //
11 #define VoltagecalStei   1 //1013
12 #define VoltagecalAchs 0 // -3
13 //------Kalibrierwerte Temperatur------
```

```
14 #define TempcalSteig 1
15 #define TempcalAchs 0
16 #define Temperaturcal    0x01
17
18 #endif /* SENSOR_1_H_ */
```

B.1.19. Sensor2.h

```
1
2
3 #ifndef SENSOR_2_H_
4 #define SENSOR_2_H_
5
6 #define Sensoradresse    0x02    //  tatsächliche Sensoradresse
7 #define SENSOR_COUNT     10      //entspricht die Anzahl der verwendeten Sensoren (wird für die Manipulation der Sensoradresse benötigt)
8
9 /*werden hier nicht mehr gebraucht (Kalibrierung findet der MATLAB-Auswertesoftware statt)*/
10 //------Kalibrierwerte Spannung------ //
11 #define VoltagecalStei   1              //1017
12 #define VoltagecalAchs   0             //-5
13 //------Kalibrierwerte Temperatur-----//
14 #define TempcalSteig 1
15 #define TempcalAchs  0
16 #define Temperaturcal    0x01
17
18
19
20 #endif /* SENSOR_2_H_ */
```

B.1.20. Sensor3.h

```
1
2 #ifndef SENSOR_3_H_
3 #define SENSOR_3_H_
4
5 #define Sensoradresse    0x03        //  tatsächliche Sensoradresse
6 #define SENSOR_COUNT     10         //entspricht die Anzahl der verwendeten Sensoren (wird für die Manipulation der Sensoradresse
      benötigt)
7
8 /*werden hier nicht mehr gebraucht (Kalibrierung findet der MATLAB-Auswertesoftware statt)*/
9 //------Kalibrierwerte Spannung-----//
10 #define VoltagecalStei   1        //1007
11 #define VoltagecalAchs   0       //-1
12 //------Kalibrierwerte Temperatur-----//
13 #define TempcalSteig 1
14 #define TempcalAchs  0          //-4
15 #define Temperaturcal    0x01
16
17
18
19 #endif /* SENSOR_3_H_ */
```

B.1.21. Sensor4.h

```
1
2
3 #ifndef SENSOR_4_H_
4 #define SENSOR_4_H_
5 #define SENSOR_COUNT     6
6
7 #define Sensoradresse    0x04    // Die Sensoradresse 4 wird von der Basisstation nicht erkannt
8 //------Kalibrierung Spannung-----//
9 #define VoltagecalStei   1
10 #define VoltagecalAchs   0
11 // #define Temperaturcal         0x01
12 //------Kalibrierwerte Temperatur------
13 #define TempcalSteig 1
14 #define TempcalAchs 0
15 #define Temperaturcal    0x01
16
17
18 #endif /* SENSOR_4_H_ */
```

B.1.22. Sensor5.h

```
1
2
3  #ifndef SENSOR_5_H_
4  #define SENSOR_5_H_
5
6  #define Sensoradresse    0x05  //  tatsächliche Sensoradresse
7  #define SENSOR_COUNT     10    //entspricht die Anzahl der verwendeten Sensoren (wird für die Manipulation der Sensoradresse benötigt)
8
9  /* werden hier nicht mehr gebraucht (Kalibrierung findet der MATLAB-Auswertesoftware statt)*/
10 //------Kalibrierwerte Spannung------
11 #define VoltagecalStei   1024
12 #define VoltagecalAchs   3
13 //------Kalibrierwerte Temperatur-----//
14 #define TempcalSteig 1
15 #define TempcalAchs -5
16 #define Temperaturcal    0x01
17
18
19
20
21
22 #endif /* SENSOR_5_H_ */
```

B.1.23. Sensor6.h

```
1
2
3  #ifndef SENSOR_6_H_
4  #define SENSOR_6_H_
5
6  #define Sensoradresse    0x06   //  tatsächliche Sensoradresse
7  #define SENSOR_COUNT     10     //entspricht die Anzahl der verwendeten Sensoren (wird für die Manipulation der Sensoradresse benötigt)
8
9  /* werden hier nicht mehr gebraucht (Kalibrierung findet der MATLAB-Auswertesoftware statt)*/
10 //------Kalibrierwerte Spannung-----//
11 #define VoltagecalStei   1     //1002
12 #define VoltagecalAchs   0
13 //------Kalibrierwerte Temperatur-----//
14 #define TempcalSteig 1
15 #define TempcalAchs  0
16 #define Temperaturcal    0x01
17
18
19
20 #endif /* SENSOR_6_H_ */
```

B.1.24. Sensor7.h

```
1
2
3  #ifndef SENSOR_7_H_
4  #define SENSOR_7_H_
5
6  #define Sensoradresse    0x07      // tatsächliche Sensoradresse
7  #define SENSOR_COUNT     10        //entspricht die Anzahl der verwendeten Sensoren (wird für die Manipulation der Sensoradresse
       benötigt)
8
9  /* werden hier nicht mehr gebraucht (Kalibrierung findet der MATLAB-Auswertesoftware statt)*/
10 //------Kalibrierwerte Spannung------
11 #define VoltagecalStei   1       //1020
12 #define VoltagecalAchs   0       //4
13 //------Kalibrierwerte Temperatur-----//
14 #define TempcalSteig 1
15 #define TempcalAchs  0
16 #define Temperaturcal    0x01
17
18
19
20 #endif /* SENSOR_7_H_ */
```

B.1.25. Sensor8.h

```
1
2
3  #ifndef SENSOR_8_H_
4  #define SENSOR_8_H_
5
6  #define Sensoradresse    0x08 // tatsächliche Sensoradresse
7  #define SENSOR_COUNT     10   //entspricht die Anzahl der verwendeten Sensoren (wird für die Manipulation der Sensoradresse benötigt)
8
9  /* werden hier nicht mehr gebraucht (Kalibrierung findet der MATLAB-Auswertesoftware statt)*/
10 #define VoltagecalAchs 0
11 #define VoltagecalStei 1
12
13 #define Temperaturcal    0x01
14 #define Tempcal 0
15 #define TempcalAchs 0
16
17
18 #endif /* SENSOR_8_H_ */
```

B.1.26. Sensor9.h

```
1
2
3  #ifndef SENSOR_9_H_
4  #define SENSOR_9_H_
5
6  #define Sensoradresse    0x09  // tatsächliche Sensoradresse
7  #define SENSOR_COUNT     10    //entspricht die Anzahl der verwendeten Sensoren (wird für die Manipulation der Sensoradresse benötigt)
8
9  /* werden hier nicht mehr gebraucht (Kalibrierung findet der MATLAB-Auswertesoftware statt)*/
10 #define VoltagecalStei  1
11 #define VoltagecalAchs  0
12
13 #define Temperaturcal    0x01
14 #define Tempcal 0
15 #define TempcalAchs 0
16
17 #endif /* SENSOR_9_H_ */
```

B.2. MATLAB-Software

B.2.1. StgReaderX.m

```
1  function varargout = StgReaderX(varargin)
2  % STGREADERX MATLAB code for StgReaderX.fig
3  %      STGREADERX, by itself, creates a new STGREADERX or raises the existing
4  %      singleton*.
5  %
6  %      H = STGREADERX returns the handle to a new STGREADERX or the handle to
7  %      the existing singleton*.
8  %
9  %      STGREADERX('CALLBACK',hObject,eventData,handles,...) calls the local
10 %      function named CALLBACK in STGREADERX.M with the given input arguments.
11 %
12 %      STGREADERX('Property','Value',...) creates a new STGREADERX or raises the
13 %      existing singleton*.  Starting from the left, property value pairs are
14 %      applied to the GUI before StgReaderX_OpeningFcn gets called.  An
15 %      unrecognized property name or invalid value makes property application
16 %      stop.  All inputs are passed to StgReaderX_OpeningFcn via varargin.
17 %
18 %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
19 %      instance to run (singleton)".
20 %
21 % See also: GUIDE, GUIDATA, GUIHANDLES
22
23 % Edit the above text to modify the response to help StgReaderX
24
25 % Last Modified by GUIDE v2.5 25-Oct-2013 15:57:23
26
27 % Begin initialization code - DO NOT EDIT
28 gui_Singleton = 1;
29 gui_State = struct('gui_Name',       mfilename, ...
30                    'gui_Singleton',  gui_Singleton, ...
31                    'gui_OpeningFcn', @StgReaderX_OpeningFcn, ...
```

```
32                      'gui_OutputFcn',  @StgReaderX_OutputFcn,  ...
33                      'gui_LayoutFcn',  [] , ...
34                      'gui_Callback',   []);
35 if nargin && ischar(varargin{1})
36     gui_State.gui_Callback = str2func(varargin{1});
37 end
38
39 if nargout
40     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
41 else
42     gui_mainfcn(gui_State, varargin{:});
43 end
44 % End initialization code - DO NOT EDIT
45
46
47 %--- Executes just before StgReaderX is made visible.
48 function StgReaderX_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version of MATLAB
52 % handles    structure with handles and user data (see GUIDATA)
53 % varargin   command line arguments to StgReaderX (see VARARGIN)
54
55 % Choose default command line output for StgReaderX
56 handles.output = hObject;
57
58 % Update handles structure
59 guidata(hObject, handles);
60
61 % UIWAIT makes StgReaderX wait for user response (see UIRESUME)
62 % uiwait(handles.figure1);
63
64 set(handles.figure1, 'UserData',[]);
65 startstop = false;
66 userData.stop = false;
67 dateiname='data';
68 set(handles.figure1, 'UserData',userData);
69 % set(handles.stoptaste,'Enable','off')
70 serialInfo = instrhwinfo('serial');
71 ports = serialInfo.AvailableSerialPorts;
72
73 set(handles.portSel, 'String', ports)
74
75 set(handles.portSel, 'Value', 1)
76 set(handles.text1,'Visible','off')
77 set(handles.auswertungTaste,'Enable','off');
78
79 %Ch1
80 % set(handles.portSel,'Enable','off')
81 % set(handles.popupmenu3,'Enable','off')
82 % set(handles.text2,'Enable','off')
83 % set(handles.ch1,'Value',1)
84
85
86 %--- Outputs from this function are returned to the command line.
87 function varargout = StgReaderX_OutputFcn(hObject, eventdata, handles)
88 % varargout  cell array for returning output args (see VARARGOUT);
89 % hObject    handle to figure
90 % eventdata  reserved - to be defined in a future version of MATLAB
91 % handles    structure with handles and user data (see GUIDATA)
92
93 % Get default command line output from handles structure
94 varargout{1} = handles.output;
95
96
97
98 function dateiText_Callback(hObject, eventdata, handles)
99 % hObject    handle to dateiText (see GCBO)
100 % eventdata  reserved - to be defined in a future version of MATLAB
101 % handles    structure with handles and user data (see GUIDATA)
102
103 % Hints: get(hObject,'String') returns contents of dateiText as text
104 %        str2double(get(hObject,'String')) returns contents of dateiText as a double
105
106
107 %--- Executes during object creation, after setting all properties.
108 function dateiText_CreateFcn(hObject, eventdata, handles)
109 % hObject    handle to dateiText (see GCBO)
110 % eventdata  reserved - to be defined in a future version of MATLAB
111 % handles    empty - handles not created until after all CreateFcns called
112
113 % Hint: edit controls usually have a white background on Windows.
114 %       See ISPC and COMPUTER.
115 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
116     set(hObject,'BackgroundColor','white');
```

```
117 end
118
119
120 %--- Executes on selection change in portSel.
121 function portSel_Callback(hObject, eventdata, handles)
122 % hObject    handle to portSel (see GCBO)
123 % eventdata  reserved - to be defined in a future version of MATLAB
124 % handles    structure with handles and user data (see GUIDATA)
125
126 % Hints: contents = cellstr(get(hObject,'String')) returns portSel contents as cell array
127 %        contents{get(hObject,'Value')} returns selected item from portSel
128
129
130 %--- Executes during object creation, after setting all properties.
131 function portSel_CreateFcn(hObject, eventdata, handles)
132 % hObject    handle to portSel (see GCBO)
133 % eventdata  reserved - to be defined in a future version of MATLAB
134 % handles    empty - handles not created until after all CreateFcns called
135
136 % Hint: popupmenu controls usually have a white background on Windows.
137 %       See ISPC and COMPUTER.
138 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
139     set(hObject,'BackgroundColor','white');
140 end
141
142
143 %--- Executes on button press in starttaste.
144 function starttaste_Callback(hObject, eventdata, handles)
145 global dats  dateiname timename xtitel1 ylabel1 tempcheck1 xtitel2 ylabel2 tempcheck2 xtitel3 ylabel3 tempcheck3 pfad pfad2 foldername
        blockfolder
146 % hObject    handle to starttaste (see GCBO)
147 % eventdata  reserved - to be defined in a future version of MATLAB
148 % handles    structure with handles and user data (see GUIDATA)
149
150 % Hint: get(hObject,'Value') returns toggle state of starttaste
151 set(handles.starttaste,'Enable','off')
152 set(handles.auswertungTaste,'Enable','off')
153 % set(handles.stoptaste,'Enable','on')
154 set(handles.stoptoggle,'Enable','on')
155 set(handles.stoptoggle,'Value',0)
156 set(handles.dateiText,'Enable','off')
157 % set(handles.text9,'Enable','off')
158 % set(handles.text10,'Enable','off')
159 % set(handles.text9,'String','XXXXXXXXXXXXXXXX');
160 % set(handles.text2,'Enable','off')
161 % set(handles.text3,'Enable','off')
162 % set(handles.edit2,'Enable','off')
163 set(handles.portSel,'Enable','off')
164 % set(handles.popupmenu2,'Enable','off')
165 % set(handles.popupmenu3,'Enable','off')
166 % set(handles.popupmenu4,'Enable','off')
167 % set(handles.popupmenu5,'Enable','off')
168 % set(handles.popupmenu6,'Enable','off')
169 % set(handles.ch1,'Enable','off')
170 % set(handles.ch2,'Enable','off')
171 % set(handles.ch3,'Enable','off')
172
173 xtitel1='Licht->Frequenz';
174 xtitel2='Zellspannung';
175 xtitel3='Zelltemperatur';
176 ylabel1='Frequenz / Hz';
177 ylabel2='Spannung / mV';
178 ylabel3='Temperatur / °C';
179 % tempcheck1=get(handles.popupmenu3,'value');
180 % tempcheck2=get(handles.popupmenu4,'value');
181 % tempcheck3=get(handles.popupmenu5,'value');
182 % aktuname1=get(handles.text2,'String');
183 % aktuname2=get(handles.text3,'String');
184 % aktuname3=get(handles.edit2,'String');
185 % set(handles.text15,'String',aktuname1);
186 % set(handles.text16,'String',aktuname2);
187 % set(handles.text22,'String',aktuname3);
188
189 % dats=zeros(500,30);
190 % dats=[];
191 % data2=zeros(500,30);
192 % data2=[];
193 % data3=zeros(500,30);
194 % data3=[];
195 blockl=10;
196 dats=zeros(1,13);
197 data3=zeros(blockl,11);
198
199 ii=0;
200 ik=0;
```

```
201 counter2=1;
202 datacounter=1;
203 beginn=now;
204 dateiname=get(handles.dateiText,'String');
205 timename=strcat(datestr(beginn,'YYYY'),datestr(beginn,'mm'),datestr(beginn,'dd'),'-',datestr(beginn,'HH'),datestr(beginn,'MM'),'_',
        dateiname);
206
207 werte=strcat(timename,'.txt');
208 workspc=strcat(timename,'.mat');
209 tmpspc=strcat(timename,'_DATstemp.mat');
210
211 set(handles.text3,'String',datestr(beginn));
212
213 userData = get(handles.figure1, 'UserData');
214 userData.stop = false; %reset for next time
215 set(handles.figure1,'UserData',userData);
216 bedingung = true;
217 splitting=false;
218
219 foldername=timename;
220 blockfolder='blocks';
221 pfad=fullfile(pwd,foldername);
222 mkdir(pfad);
223 pfad2=fullfile(pfad,blockfolder);
224 mkdir(pfad2);
225
226 % files = dir(fullfile(pwd,foldername));
227 % filecount = length(files);
228 %% Ports konfigurieren
229 % if get(handles.ch1,'Value')==true;
230 value = get(handles.portSel,'value');
231 strings = get(handles.portSel,'string');
232 string = strings(value);
233 port2 = serial(string,'Baudrate',115200,'Terminator','CR');
234 fopen(port2);
235 set(port2,'Timeout',50);
236
237 % port2.ReadAsyncMode = 'continuous';
238 % port2.ReadAsyncMode ='manual';
239 % end;
240
241 %% if get(handles.ch2,'Value')==true;
242 % value2 = get(handles.popupmenu2,'value');
243 % strings = get(handles.popupmenu2,'string');
244 % string = strings(value2);
245 % port1 = serial(string, 'BaudRate', 9600,'BytesAvailableFcnMode','terminator','Terminator','CR/LF');
246 % fopen(port1);
247 % port1.ReadAsyncMode = 'manual';
248 %% end;
249 %
250 %% if get(handles.ch3,'Value')==true;
251 % value3 = get(handles.popupmenu6,'value');
252 % strings = get(handles.popupmenu6,'string');
253 % string = strings(value3);
254 % port3 = serial(string, 'BaudRate', 9600,'BytesAvailableFcnMode','terminator','Terminator','CR/LF');
255 % fopen(port3);
256 % port3.ReadAsyncMode = 'manual';
257 %% end;
258 set(handles.text1,'Visible','on')
259 % pause(4);
260 tic;
261 x=0;
262 %% Programmablauf bis Stoptaste gedrückt wird
263 while (get(handles.stoptoggle,'Value')==0)
264
265 ii=ii+1; %Zähler für Matrixzeilen
266 ik=ik+1;
267 dauer=datestr(toc/86400,'dd␣Tage␣HH:MM:SS');
268 % pause(4);
269 set(handles.text5,'String',dauer);
270
271 [data,count,msg] = binblockread(port2,'uint8');
272 data2(ii,:) = uint8(data');
273 data3(ii,:) = [uint32(data2(ii,1)),uint32(data2(ii,2)),uint32(swapbytes(typecast(data2(ii,3:6),'uint32'))),uint32(data2(ii,7)),uint32(
        swapbytes(typecast(data2(ii,8:11),'uint32'))),uint32(data2(ii,12)),uint32(data2(ii,13)),uint32(data2(ii,14)),uint32(data2(ii,15)
        ),uint32(swapbytes(typecast(data2(ii,16:17),'uint16'))),uint32(swapbytes(typecast(data2(ii,18:21),'uint32')))];
274 dats(ii,:) = [double(data3(ii,:)),now,ik]
275 % tempMul4(ii,1)=dats(ii,4)*4;
276 % dats(ii,4)=tempMul4(ii,1);
277 set(handles.text13,'String',ik)
278 %% LivePlot
279 %       if mod(ik,55)==0
280 if get(handles.plotTasteFRQ,'Value')==true;
281       subplot(handles.axes1)
282       plot(dats(:,12),dats(:,5),'b','linewidth',2);
```

```
283         legend(xtitel1,'Location','NorthEast')
284         grid minor
285         datetick('x','DD_HH:MM:SS', 'keepticks')
286         title(xtitel1);
287         ylabel(ylabel1(tempcheck1));
288         xlabel('Zeit');
289         end
290 %
291         if get(handles.plotTasteCV,'Value')==true;
292         subplot(handles.axes2)
293         plot(dats(:,12),dats(:,3),'r','linewidth',2);
294         legend(xtitel2,'Location','NorthEast')
295         grid minor
296         datetick('x','DD_HH:MM:SS', 'keepticks')
297         title(xtitel2);
298         ylabel(ylabel2(tempcheck2));
299         xlabel('Zeit');
300         end
301
302         if get(handles.plotTasteT,'Value')==true;
303         subplot(handles.axes3)
304         plot(dats(:,12),dats(:,4),'k','linewidth',2);
305         legend(xtitel3,'Location','NorthEast')
306         grid minor
307         datetick('x','DD_HH:MM:SS', 'keepticks')
308         title(xtitel3);
309         ylabel(ylabel3(tempcheck3));
310         xlabel('Zeit');
311         end
312 %
313 %       drawnow;
314 %       end
315       drawnow;
316 % x=x+1*1/1;
317       userData = get(handles.figure1, 'UserData');
318 %       if userData.stop == true
319 %           bedingung = false;
320 %       end
321 %% Temporäres Abspeichern
322
323 if mod(ik,blockl)==0
324 dlmwrite(fullfile(pfad,'temp.txt'),dats,'delimiter','\t','newline', 'pc','precision', '%.6f')
325 tempspcdats=strcat('tmpspc',num2str(counter2));
326 save(fullfile(pfad2,tempspcdats),'dats')
327 counter2=counter2+1;
328 ii=0;
329 splitting=true;
330 dats=[];
331 end
332
333 end
334
335 %% Ports schließen
336 fclose(port2);
337 delete(port2);
338 clear port2;
339
340 set(handles.text1,'Visible','off')
341 endzeit=now;
342 set(handles.text7,'String',datestr(endzeit));
343
344 if splitting==true
345 tempspcdats=strcat('tmpspc',num2str(counter2));
346 save(fullfile(pfad2,tempspcdats),'dats')
347 %% Gesamtdaten sichern
348 dlmwrite(fullfile(pfad,werte),dats,'delimiter','\t','newline','pc','precision', '%.6f')
349 block=strcat(timename,'BLOCK.mat');
350 save(fullfile(pfad,workspc));
351 %% Dateien zusammensetzten
352 dats=[];
353 files = dir(fullfile(pfad2,'*.mat'));
354 filecount = length(files);
355 for co3=1:filecount
356 tempdatsload=load(fullfile(pfad2,files(co3).name),'dats');
357 [z s]=size(tempdatsload.dats);
358 if s ==13
359     tempdatsload.dats(:,14)=0;
360 end
361 dats=[dats;tempdatsload.dats];
362 end
363 a=sortrows(dats,13);
364 dats=a;
365 end
366 block2=strcat(timename,'BLOCKSORT.mat');
367 save(fullfile(pfad,block2),'dats');
```

```
368 set(handles.auswertungTaste,'Enable','on');
369
370 %--- Executes on button press in stoptaste.
371 function stoptaste_Callback(hObject, eventdata, handles)
372 % hObject    handle to stoptaste (see GCBO)
373 % eventdata  reserved - to be defined in a future version of MATLAB
374 % handles    structure with handles and user data (see GUIDATA)
375
376 % Hint: get(hObject,'Value') returns toggle state of stoptaste
377
378 userData = get(handles.figure1, 'UserData');
379 userData.stop = true;
380 startstop=true;
381 set(handles.figure1,'UserData',userData);
382 % set(handles.stoptaste,'Enable','off')
383 set(handles.stoptoggle,'Enable','off')
384 set(handles.starttaste,'Enable','on')
385 set(handles.dateiText,'Enable','on')
386 set(handles.portSel,'Enable','on')
387
388 % set(handles.ch1,'Enable','on')
389
390 %%--- Executes on button press in stoptaste.
391 % function stoptaste_Callback(hObject, eventdata, handles)
392 %% hObject    handle to stoptaste (see GCBO)
393 %% eventdata  reserved - to be defined in a future version of MATLAB
394 %% handles    structure with handles and user data (see GUIDATA)
395 %
396 %% Hint: get(hObject,'Value') returns toggle state of stoptaste
397 %
398 % userData = get(handles.figure1, 'UserData');
399 % userData.stop = true;
400 % startstop=true;
401 % set(handles.figure1,'UserData',userData);
402 %% set(handles.stoptaste,'Enable','off')
403 % set(handles.stoptoggle,'Enable','off')
404 % set(handles.starttaste,'Enable','on')
405 % set(handles.dateiText,'Enable','on')
406 % set(handles.portSel,'Enable','on')
407 %
408 %% set(handles.ch1,'Enable','on')
409
410
411 %--- Executes on button press in stoptoggle.
412 function stoptoggle_Callback(hObject, eventdata, handles)
413 % hObject    handle to stoptoggle (see GCBO)
414 % eventdata  reserved - to be defined in a future version of MATLAB
415 % handles    structure with handles and user data (see GUIDATA)
416
417 % Hint: get(hObject,'Value') returns toggle state of stoptoggle
418 userData = get(handles.figure1, 'UserData');
419 userData.stop = true;
420 startstop=true;
421 set(handles.figure1,'UserData',userData);
422 % set(handles.stoptaste,'Enable','off')
423 set(handles.stoptoggle,'Enable','off')
424 set(handles.starttaste,'Enable','on')
425 set(handles.dateiText,'Enable','on')
426 set(handles.portSel,'Enable','on')
427
428
429 %--- Executes on button press in auswertungTaste.
430 function auswertungTaste_Callback(hObject, eventdata, handles)
431 global dats dateiname timename xtitel1 ylabel1 tempcheck1 xtitel2 ylabel2 tempcheck2 xtitel3 ylabel3 tempcheck3 pfad pfad2 foldername
        blockfolder
432 % hObject    handle to auswertungTaste (see GCBO)
433 % eventdata  reserved - to be defined in a future version of MATLAB
434 % handles    structure with handles and user data (see GUIDATA)
435 % clear all;
436 % file='20131012-AlleSensoren-3000mV.dat';
437 % file='20131012-AlleSensoren-3000mV-Reihenfolge213756.dat'
438 % file='20131012-2800mV-Reihenfolge213756.dat';
439 % file='20131012-2904mV-Reihenfolge213756.dat';
440 % file='20131012-BLOCK1-5Sensoren.dat'
441 % file='20131012-AlleSensoren-3000mV-Reihenfolge213756.dat';
442 % file='20131012-31013mV-Reihenfolge213756.dat';
443 % file='20131012-3201mV-Reihenfolge213756.dat';
444 % file='20131012-3303mV-Reihenfolge213756.dat';
445 % file='20131012-3402mV-Reihenfolge213756.dat';
446 % file='20131014-BatterieBLOCK1_CALI-5+1Sensoren.dat';
447 % file='20131015-BatterieBLOCK1_CALI-5+1Sensoren.dat'
448 % file='20131015-BatterieBLOCK1_CALI-5+1SensorenMitFasern.dat'
449 % file='20131015-BatterieBLOCK1_CALI-5+1SensorenMitFasern1000.dat'
450 % file='s8-4h-2211V40.dat';
451 % file='s8-1800V10.dat';
```

```
452 % file='s8-1909V20.dat';
453 % file='s8-2211V20.dat';
454 % file='S8-TKurve105010-2002V.dat';
455 % file='S8-TKurve105010Rampe2002V.dat';
456 % file='Sall.dat';
457 % file='SALL-20000Daten-2002mV-RAMPE105010.dat';
458 % file='SALL-10000Daten-3006mV-RAMPE5010.dat';
459
460 load(fullfile(pfad,strcat(timename,'BLOCKSORT.mat')),'dats');
461 i=1;
462 file=dateiname;
463 file2=timename;
464 %% file='20131012-BLOCK1-5Sensoren.dat';
465 % sourcetemp=[];
466 % file2=strrep(file,'.dat','');
467 %% source=dlmread(file);
468 source=dats;
469 sourcetemp(:,1)=(((source(:,4)*4*0.001466)-0.986)/0.00355);%Temperatur Umrechnung
470 sourcetemp(:,2)=source(:,2);
471 source(:,4)=sourcetemp(:,1);
472 heute=datevec(fix(now));
473
474 sensor_count=10; %Je nach gesamt Anzahl von Sensoren anpassen
475 s=cell(1, sensor_count);
476 s_red=cell(1, sensor_count);
477 s_ylw=cell(1, sensor_count);
478
479 s_plot_color = cellstr(['m'; 'c'; 'k'; 'b'; 'y'; 'g';'-m'; '-c'; '-k'; '-b'; '-y'; '-g';'+m'; '+c'; '+k'; '+b'; '+y'; '+g' ])
480
481 sensor_red = cell(1, sensor_count);
482 sensor_ylw = cell(1, sensor_count);
483 sensor = cell(1, sensor_count);
484
485 for count=1:sensor_count
486     s{count}=strcat('Sensor ', num2str(count));
487     s_red{count}=strcat('Sensor ', num2str(count), ' red');
488     s_ylw{count}=strcat('Sensor ', num2str(count), ' ylw');
489
490     %Split data to seperate sensor values
491     corr=find(source(:,2)==count);%Sensor red abfrage
492     sensor_red{count}=(source(corr,:));
493     corr=find(source(:,2)==count+sensor_count);%Sensor ylw abfrage
494     sensor_ylw{count}=(source(corr,:));
495     corr=find(source(:,2)==count | source(:,2)==count+sensor_count);%Sensor alle abfrage
496     sensor{count}=(source(corr,:));
497 end
498
499 %
500 % zeitachse1=[];
501 % zeitachse2=[];
502 % zeitachse3=[];
503 % zeitachse4=[];
504 % zeitachse5=[];
505 % zeitachse6=[];
506 % zeitachse7=[];
507 % zeitachse8=[];
508 %
509 % zeitNum1=[];
510 % zeitNum2=[];
511 % zeitNum3=[];
512 % zeitNum4=[];
513 % zeitNum5=[];
514 % zeitNum6=[];
515 % zeitNum7=[];
516 % zeitNum8=[];
517
518 %sensoranz=0;
519
520 %% Temperaturumrechnung
521
522 % ta(:,1)=(((sensor1(:,4)*4*0.001466)-0.986)/0.00355);
523
524 % zeitNum7=datenum(zeitVec7);
525 % Mittelwert
526 % mittel=[mean(sensor1(:,3)) mean(sensor2(:,3)) mean(sensor3(:,3)) mean(sensor5(:,3)) mean(sensor6(:,3)) mean(sensor7(:,3))]
527 % fuerKali=[mittel
528 %% PLOT AllInOne
529 %
530 % figureALLE=figure('Name','Alle Sensoren')
531 %
532 % if (length(sensor1))
533 % plot(zeitNum1,sensor1(:,3),'-+m','LineWidth',2);hold on;legend(s1);
534 % plot(zeitNum1,sensor1(:,4),'-.+m','LineWidth',2);hold on;legend(s1);
535 % plot(zeitNum1,sensor1(:,5),'--+m','LineWidth',2);hold on;legend(s1);
536 % end
```

```
537 % if (length(sensor2))
538 % plot(zeitNum2,sensor2(:,3),'-+c','LineWidth',2);hold on;legend(s2);
539 % plot(zeitNum2,sensor2(:,4),'-.+c','LineWidth',2);hold on;legend(s2);
540 % plot(zeitNum2,sensor2(:,5),'--+c','LineWidth',2);hold on;legend(s2);
541 % end
542 %
543 % if (length(sensor3))
544 % plot(zeitNum3,sensor3(:,3),'-+r','LineWidth',2);hold on;legend(s3);
545 % plot(zeitNum3,sensor3(:,4),'-.+r','LineWidth',2);hold on;legend(s3);
546 % plot(zeitNum3,sensor3(:,5),'--+r','LineWidth',2);hold on;legend(s3);
547 % end
548 %
549 %% if (length(sensor4))
550 %% plot(zeitNum1,sensor4(:,3),'-r');hold on;
551 %% plot(ZeitNum1,sensor4(:,5),'-b');hold on;
552 %% end
553 % if (length(sensor5))
554 % plot(zeitNum5,sensor5(:,3),'-+g','LineWidth',2);hold on;legend(s5);
555 % plot(zeitNum5,sensor5(:,4),'-.+g','LineWidth',2);hold on;legend(s5);
556 % plot(zeitNum5,sensor5(:,5),'--+g','LineWidth',2);hold on;legend(s5);
557 % end
558 % if (length(sensor6))
559 % plot(zeitNum6,sensor6(:,3),'-+b','LineWidth',2);hold on; legend(s6);
560 % plot(zeitNum6,sensor6(:,4),'-.+b','LineWidth',2);hold on; legend(s6);
561 % plot(zeitNum6,sensor6(:,5),'--+b','LineWidth',2);hold on;legend(s6);
562 % end
563 % if (length(sensor7))
564 % plot(zeitNum7,sensor7(:,3),'-+k','LineWidth',2);hold on;legend(s7);
565 % plot(zeitNum7,sensor7(:,4),'-.+k','LineWidth',2);hold on;legend(s7);
566 % plot(zeitNum7,sensor7(:,5),'--+k','LineWidth',2);hold on;legend(s7);
567 % end
568 % if (length(sensor8))
569 % plot(zeitNum8,sensor8(:,3),'-+y','LineWidth',2);hold on;legend(s8);
570 % plot(zeitNum8,sensor8(:,4),'-.+y','LineWidth',2);hold on;legend(s8);
571 % plot(zeitNum8,sensor8(:,5),'--+y','LineWidth',2);hold on;legend(s8);
572 % end
573 % grid minor;
574 %
575 %% plot(zeitNum8,smooth(sensor8(:,3),5),'-b','LineWidth',2);hold on;
576 %% plot(zeitNum8,smooth(sensor8(:,4),5),'--b','LineWidth',2);hold on;
577 %% plot(zeitNum8,smooth(sensor8(:,5),5),'--b','LineWidth',2);hold on;
578 %
579 % title(file2);
580 %
581 %% legend(s1,s1,s2,s2,s3,s3,s5,s5,s6,s6,s7,s7)
582 % legend(s1,s1,s1,s2,s2,s2,s3,s3,s3,s5,s5,s5,s6,s6,s6,s7,s7,s7,s8,s8,s8)
583 %% '20131012-AlleSensoren-3000mV.dat'
584 % datetick('x','keepticks');
585 % set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
586 % dcm_obj = datacursormode(figure(figureALLE));
587 % set(dcm_obj,'UpdateFcn',@myupdatefcn)
588 % beginn=now;
589 % dateiname=file2;
590 % timename=strcat(datestr(beginn,'YYYY'),datestr(beginn,'mm'),datestr(beginn,'dd'),'-',datestr(beginn,'HH'),datestr(beginn,'MM'),'_',
        dateiname);
591 %
592 % saveas(figureALLE,strcat(timename,'1'),'fig')
593 % saveas(figureALLE,strcat(timename,'1'),'bmp')
594
595 %% SUBPLOT
596 figureALLEP = figure('Name',file);
597 lg=cell(1, sensor_count);
598 i=0;
599
600 %SPANNUNG
601 ax(1)=subplot(4,1,1);
602 for count=1:sensor_count
603     if (length(sensor{count}))
604         active_sensor=sensor{count};
605         plot(active_sensor(:,12),active_sensor(:,3),s_plot_color{count},'LineWidth',1);hold on; %TODO: Change colors
606         % liste(1)=['Sensor' int2str(1)];
607         i=i+1;
608         lg{i}=s{count};
609     end
610 end
611 grid minor
612 datetick('x','keepticks');
613 xlabel('Time / s');
614 ylabel('Cell-Voltage / mV');
615 legend(s);
616 title('Zellspannung');
617
618 % TEMPERATUR
619 ax(2)=subplot(4,1,4);
620 lg=cell(1, sensor_count);
```

```
621 i=0;
622 for count=1:sensor_count
623     if (length(sensor{count}))
624         active_sensor=sensor{count};
625         plot(active_sensor(:,12),active_sensor(:,4),s_plot_color{count},'LineWidth',1);hold on; %TODO: Change colors
626         % liste(1)=['Sensor' int2str(1)];
627         i=i+1;
628         lg{i}=s{count};
629     end
630 end
631 grid minor
632 datetick('x','keepticks');
633 xlabel('Time / s');
634 ylabel('Temperature / ADC');
635 legend(s);
636 title('Temperaturmessung des ADC');
637
638 % DICHTE BZW FREQUENZ BEI WELLENLAENGE 1
639 ax(3)=subplot(4,1,2);
640 %lg=cell(length(unique(source(:,2))),1);
641 i=0;
642
643 for count=1:sensor_count
644     if (length(sensor_red{count}))
645         active_sensor=sensor_red{count};
646         plot(active_sensor(:,12),active_sensor(:,5),s_plot_color{count},'LineWidth',1);hold on; %TODO: Change colors
647         % liste(1)=['Sensor' int2str(1)];
648         i=i+1;
649         lg{i}=s_red{count};
650     end
651 end
652 grid minor
653 datetick('x','keepticks');
654 xlabel('Time / s');
655 ylabel('Light-Frequency Sensor / Hz');
656 legend(s);
657 title('Dichte mit Wellenlänge 633nm');
658
659 % DICHTE BZW FREQUENZ BEI WELLENLAENGE 2
660 ax(3)=subplot(4,1,3);
661 %lg=cell(length(unique(source(:,2))),1);
662 i=0;
663
664 for count=1:sensor_count
665     if (length(sensor_ylw{count}))
666         active_sensor=sensor_ylw{count};
667         plot(active_sensor(:,12),active_sensor(:,5),s_plot_color{count},'LineWidth',1);hold on; %TODO: Change colors
668         % liste(1)=['Sensor' int2str(1)];
669         i=i+1;
670         lg{i}=s_ylw{count};
671     end
672 end
673 grid minor
674 datetick('x','keepticks');
675 xlabel('Time / s');
676 ylabel('Light-Frequency Sensor / Hz');
677 legend(s);
678 title('Dichte mit Wellenlänge 587nm');
679
680 set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
681 dcm_obj = datacursormode(figure(figureALLEP));
682 set(dcm_obj,'UpdateFcn',@myupdatefcn)
683 beginn=now;
684 % dateiname=file2;
685 % timename=strcat(datestr(beginn,'YYYY'),datestr(beginn,'mm'),datestr(beginn,'dd'),'-',datestr(beginn,'HH'),datestr(beginn,'MM'),'_',
        dateiname);
686 linkaxes([ax(1) ax(2) ax(3)],'x')
687 saveas(figureALLEP,fullfile(pfad,strcat(timename,'2')),'fig');
688 saveas(figureALLEP,fullfile(pfad,strcat(timename,'2')),'bmp');
689 save(fullfile(pfad,strcat(timename,'_workspace')));
690 %%
691 % grenze=size(sensor1);
692 % for i=1:grenze(1,1)
693 % s1k(i,1)=sensor1(i,3)-reg(2,1);
694 % s1k(i,1)=sensor1(i,3)/reg(1,1);
695 % end
696 %
697 % grenze=size(sensor2);
698 % for i=1:grenze(1,1)
699 % s2k(i,1)=sensor2(i,3)-reg(2,2);
700 % s2k(i,1)=sensor2(i,3)/reg(1,2);
701 % end
702 %
703 % grenze=size(sensor3);
704 % for i=1:grenze(1,1)
```

```
705 % s3k(i,1)=sensor3(i,3)-reg(2,3);
706 % s3k(i,1)=sensor3(i,3)/reg(1,3);
707 % end
708 %
709 % grenze=size(sensor5);
710 % for i=1:grenze(1,1)
711 % s5k(i,1)=sensor5(i,3)-reg(2,4);
712 % s5k(i,1)=sensor5(i,3)/reg(1,4);
713 % end
714 %
715 % grenze=size(sensor7);
716 % for i=1:grenze(1,1)
717 % s7k(i,1)=sensor7(i,3)-reg(2,6);
718 % s7k(i,1)=sensor7(i,3)/reg(1,6);
719 % end
720 %
721 %
722 %
723 % figureALLE1=figure('Name','Alle Sensoren')
724 % if (length(sensor1))
725 % plot(zeitNum1,s1k(:,1),'*m');hold on;
726 % plot(zeitNum1,sensor1(:,5),'*m');hold on;
727 % end
728 % if(length(sensor2))
729 % plot(zeitNum2,s2k(:,1),'*c');hold on;
730 % plot(zeitNum2,sensor2(:,5),'*c');hold on;
731 % end
732 % if(length(sensor3))
733 % plot(zeitNum3,s3k(:,1),'*r');hold on;
734 % plot(zeitNum3,sensor3(:,5),'*r');hold on;
735 % end
736 % if(length(sensor4))
737 % plot(zeitNum1,sensor4(:,3),'-r');hold on;
738 % plot(ZeitNum1,sensor4(:,5),'-b');hold on;
739 % end
740 % if(length(sensor5))
741 % plot(zeitNum5,s5k(:,1),'*g');hold on;
742 % plot(zeitNum5,sensor5(:,5),'*g');hold on;
743 % end
744 % if(length(sensor6))
745 % plot(zeitNum6,sensor6(:,3),'*b');hold on;
746 % plot(zeitNum6,sensor6(:,5),'*b');hold on;
747 % end
748 % if(length(sensor7))
749 % plot(zeitNum7,s7k(:,1),'*k');hold on;
750 % plot(zeitNum7,sensor7(:,5),'*k');hold on;
751 % end
752 % if(length(sensor8))
753 % plot(zeitNum8,s8k(:,1),'*k');hold on;
754 % plot(zeitNum8,sensor8(:,5),'*k');hold on;
755 % end
756 %
757 % title('kalibriert');
758 %% legend(s1,s1,s2,s2,s3,s3,s5,s5,s6,s6,s7,s7)
759 % legend(s1,s1,s2,s2,s3,s3,s5,s5,s7,s7)
760 %% '20131012-AlleSensoren-3000mV.dat'
761 % datetick('x','keepticks');
762 % set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
763 % dcm_obj = datacursormode(figure(figureALLE1));
764 % set(dcm_obj,'UpdateFcn',@myupdatefcn)
765 % beginn=now;
766 % dateiname=file;
767 % timename=strcat(datestr(beginn,'YYYY'),datestr(beginn,'mm'),datestr(beginn,'dd'),'-',datestr(beginn,'HH'),datestr(beginn,'MM'),'_',
        dateiname);
768 %
769 % saveas(figureALLE1,strcat(timename,'Kalibriert'),'fig')
770 % saveas(figureALLE1,strcat(timename,'Kalibriert'),'bmp')
771 %
772 %%%
773 % figureALLE3=figure('Name','Alle Sensoren')
774 % plot(zeitNum1,sensor1(:,3),'-m','LineWidth',2);hold on;
775 % plot(zeitNum1,s1k(:,1),'--m','LineWidth',2);hold on;
776 %
777 % plot(zeitNum2,sensor2(:,3),'-c','LineWidth',2);hold on;
778 % plot(zeitNum2,s2k(:,1),'--c','LineWidth',2);hold on;
779 %
780 % plot(zeitNum3,sensor3(:,3),'-r','LineWidth',2);hold on;
781 % plot(zeitNum3,s3k(:,1),'--r','LineWidth',2);hold on;
782 %
783 %% plot(zeitNum1,sensor4(:,3),'-r');hold on;
784 %% plot(ZeitNum1,sensor4(:,5),'-b');hold on;
785 %
786 % plot(zeitNum5,sensor5(:,3),'-g','LineWidth',2);hold on;
787 % plot(zeitNum5,s5k(:,1),'--g','LineWidth',2);hold on;
788 %% plot(zeitNum6,sensor6(:,3),'*b');hold on;
```

```
789 %% plot(zeitNum6,sensor6(:,5),'*b');hold on;
790 %
791 % plot(zeitNum7,sensor7(:,3),'-k','LineWidth',2);hold on;
792 % plot(zeitNum7,s7k(:,1),'--k','LineWidth',2);hold on;
793 %
794 % title(file2);
795 %% legend(s1,s1,s2,s2,s3,s3,s5,s5,s6,s6,s7,s7)
796 % legend(s1,s1,s2,s2,s3,s3,s5,s5,s7,s7)
797 %% '20131012-AlleSensoren-3000mV.dat'
798 % datetick('x','keepticks');
799 % set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
800 % dcm_obj = datacursormode(figure(figureALLE3));
801 % set(dcm_obj,'UpdateFcn',@myupdatefcn)
802 % beginn=now;
803 % dateiname=file;
804 % timename=strcat(datestr(beginn,'YYYY'),datestr(beginn,'mm'),datestr(beginn,'dd'),'-',datestr(beginn,'HH'),datestr(beginn,'MM'),'_',
        dateiname);
805 %
806 % saveas(figureALLE3,strcat(timename,'Unterschied'),'fig')
807 % saveas(figureALLE3,strcat(timename,'Unterschied'),'bmp')
808
809 %%
810 % stunden = sensor1(:,7);
811 % minuten = sensor1(:,8);
812 % sekunden = sensor1(:,9);
813 % [stunden minuten sekunden] = Dateiname(:,(7:9))
814 % zeitachse=horzcat(sensor1(:,7), sensor1(:,8), sensor1(:,9));
815
816
817
818 % for i=1:length(zeitachse)
819 % neu(i,:)=horzcat(heute(1,1),heute(1,2),heute(1,3),zeitachse(i,:));
820 % end
821 %
822 % zeitNum=datenum(neu);
823 % plot(zeitNum,Dateiname(:,3),'r');
824 % hold on;
825 % plot(zeitNum,Dateiname(:,5),'b');
826 % datetick('x','keepticks');
827 %%
828 % figure1=figure('Name','Zellsensor');
829 % [haxes,hline1,hline2]=plotyy(zeitNum,x1000Werte(:,3),zeitNum,x1000Werte(:,5),'plot');
830 %
831 % grid minor; xlabel('Zeit');%ylabel('Temperatur / [°C]');
832 % legend('Spannung','Licht->Frq');
833 % title(sprintf('Zellsensor'));
834 % axes(haxes(1))
835 %% datetick('x','yy mmm dd HH:MM:SS','keepticks')
836 % datetick('x','keepticks')
837 % ylabel('Spannung / [V]')
838 % set(gca,'YTickMode','auto')
839 % set(hline1,'marker','.','color','b')%,'Line','none')
840 % axes(haxes(2))
841 %% datetick('x','dd mmm yyyy HH:MM:SS','keepticks')
842 % datetick('x','keepticks')
843 % ylabel('Licht->Frequenz / [Hz]')
844 % set(gca,'YTickMode','auto')
845 % set(hline2,'color','r','LineWidth',2)
846 % set(haxes(2),'ycolor','r')
847 % set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
848 % dcm_obj = datacursormode(figure(figure1));
849 % set(dcm_obj,'UpdateFcn',@myupdatefcn)
850 % saveas(figure1,fullfile(pfad,strcat(timename,1)),'fig')
851 % saveas(figure1,fullfile(pfad,strcat(timename,1)),'bmp')
852 % saveas(figure1,strcat(timename,'1'),'fig')
853 % saveas(figure1,strcat(timename,'1'),'bmp')
854
855
856
857
858 %--- Executes on button press in plotTasteFRQ.
859 function plotTasteFRQ_Callback(hObject, eventdata, handles)
860 % hObject    handle to plotTasteFRQ (see GCBO)
861 % eventdata  reserved - to be defined in a future version of MATLAB
862 % handles    structure with handles and user data (see GUIDATA)
863
864 % Hint: get(hObject,'Value') returns toggle state of plotTasteFRQ
865
866 %--- Executes on button press in plotTasteCV.
867 function plotTasteCV_Callback(hObject, eventdata, handles)
868 % hObject    handle to plotTasteCV (see GCBO)
869 % eventdata  reserved - to be defined in a future version of MATLAB
870 % handles    structure with handles and user data (see GUIDATA)
871
872 % Hint: get(hObject,'Value') returns toggle state of plotTasteCV
```

```
873
874
875 %--- Executes on button press in plotTasteT.
876 function plotTasteT_Callback(hObject, eventdata, handles)
877 % hObject    handle to plotTasteT (see GCBO)
878 % eventdata  reserved - to be defined in a future version of MATLAB
879 % handles    structure with handles and user data (see GUIDATA)
880
881 % Hint: get(hObject,'Value') returns toggle state of plotTasteT
```

B.2.2. Kalibration.m

```
1  clear all;
2  close all;
3  clc;
4
5  %% Parameter
6
7  % Anzahl der Sensoren
8  noSensors = 9;
9  % Fehlerhafte Sensoren / nicht vergebene Adressen
10 defectSensors = [4 1 8];
11 % Ideale Kalibrierspannungen
12 voltIdeal = [1700 1750 1800 1850 1900 1950 2000 2050 2100 2150 2200 2250 2300 2350 2400];
13 % Reale Temperaturen (vom Temperaturnormal)
14 tempReal  = [ -20  -10    0   10   20   30   40   50];
15 % Reale Spannungen (vom Spannungsnormal)
16 voltReal  = [1703 1700 1700 1701 1701 1701 1701 1701;...
17              1753 1753 1749 1749 1751 1752 1751 1754;...
18              1802 1803 1797 1797 1802 1800 1803 1799;...
19              1850 1847 1853 1849 1851 1851 1846 1852;...
20              1902 1904 1902 1899 1900 1905 1903 1901;...
21              1952 1952 1950 1949 1950 1950 1950 1952;...
22              2003 2001 2002 2000 1999 2002 2000 1999;...
23              2050 2050 2049 2049 2052 2051 2052 2048;...
24              2102 2101 2099 2099 2101 2100 2100 2101;...
25              2150 2150 2153 2149 2150 2151 2151 2152;...
26              2202 2199 2203 2204 2199 2201 2203 2200;...
27              2255 2249 2247 2251 2252 2251 2252 2253;...
28              2302 2303 2303 2308 2303 2301 2301 2301;...
29              2352 2349 2349 2350 2351 2349 2351 2351;...
30              2402 2402 2402 2397 2398 2398 2399 2398];
31
32 % Rohdaten des Batteriesteuergeräts
33 files      = {'20141119-1731_-20_1700BLOCKSORT.mat',...
34              '20141119-1741_-20_1750BLOCKSORT.mat',...
35              '20141119-1751_-20_1800BLOCKSORT.mat',...
36              '20141119-1801_-20_1850BLOCKSORT.mat',...
37              '20141119-1811_-20_1900BLOCKSORT.mat',...
38              '20141119-1821_-20_1950BLOCKSORT.mat',...
39              '20141119-1832_-20_2000BLOCKSORT.mat',...
40              '20141119-1841_-20_2050BLOCKSORT.mat',...
41              '20141119-1850_-20_2100BLOCKSORT.mat',...
42              '20141119-1859_-20_2150BLOCKSORT.mat',...
43              '20141119-1909_-20_2200BLOCKSORT.mat',...
44              '20141119-1923_-20_2250BLOCKSORT.mat',...
45              '20141119-1933_-20_2300BLOCKSORT.mat',...
46              '20141119-1944_-20_2350BLOCKSORT.mat',...
47              '20141119-1954_-20_2400BLOCKSORT.mat',...
48              '20141119-2039_-10_1700BLOCKSORT.mat',...
49              '20141119-2052_-10_1750BLOCKSORT.mat',...
50              '20141119-2104_-10_1800BLOCKSORT.mat',...
51              '20141119-2116_-10_1850BLOCKSORT.mat',...
52              '20141119-2127_-10_1900BLOCKSORT.mat',...
53              '20141119-2138_-10_1950BLOCKSORT.mat',...
54              '20141119-2149_-10_2000BLOCKSORT.mat',...
55              '20141119-2158_-10_2050BLOCKSORT.mat',...
56              '20141119-2209_-10_2100BLOCKSORT.mat',...
57              '20141119-2220_-10_2150BLOCKSORT.mat',...
58              '20141119-2230_-10_2200BLOCKSORT.mat',...
59              '20141119-2241_-10_2250BLOCKSORT.mat',...
60              '20141119-2251_-10_2300BLOCKSORT.mat',...
61              '20141119-2301_-10_2350BLOCKSORT.mat',...
62              '20141119-2314_-10_2400BLOCKSORT.mat',...
63              '20141120-1135_0_1700BLOCKSORT.mat',...
64              '20141120-1149_0_1750BLOCKSORT.mat',...
65              '20141120-1200_0_1800BLOCKSORT.mat',...
66              '20141120-1209_0_1850BLOCKSORT.mat',...
67              '20141120-1219_0_1900BLOCKSORT.mat',...
68              '20141120-1231_0_1950BLOCKSORT.mat',...
69              '20141120-1241_0_2000BLOCKSORT.mat',...
70              '20141120-1250_0_2050BLOCKSORT.mat',...
```

```
71                  '20141120-1301_0_2100BLOCKSORT.mat',...
72                  '20141120-1314_0_2150BLOCKSORT.mat',...
73                  '20141120-1326_0_2200BLOCKSORT.mat',...
74                  '20141120-1338_0_2250BLOCKSORT.mat',...
75                  '20141120-1348_0_2300BLOCKSORT.mat',...
76                  '20141120-1401_0_2350BLOCKSORT.mat',...
77                  '20141120-1417_0_2400BLOCKSORT.mat',...
78                  '20141120-1505_10_1700BLOCKSORT.mat',...
79                  '20141120-1519_10_1750BLOCKSORT.mat',...
80                  '20141120-1533_10_1800BLOCKSORT.mat',...
81                  '20141120-1546_10_1850BLOCKSORT.mat',...
82                  '20141120-1607_10_1900BLOCKSORT.mat',...
83                  '20141120-1623_10_1950BLOCKSORT.mat',...
84                  '20141120-1638_10_2000BLOCKSORT.mat',...
85                  '20141120-1657_10_2050BLOCKSORT.mat',...
86                  '20141120-1716_10_2100BLOCKSORT.mat',...
87                  '20141120-1732_10_2150BLOCKSORT.mat',...
88                  '20141120-1747_10_2200BLOCKSORT.mat',...
89                  '20141120-1827_10_2250BLOCKSORT.mat',...
90                  '20141120-1839_10_2300BLOCKSORT.mat',...
91                  '20141120-1849_10_2350BLOCKSORT.mat',...
92                  '20141120-1901_10_2400BLOCKSORT.mat',...
93                  '20141119-1353_20_1700BLOCKSORT.mat',...
94                  '20141119-1403_20_1750BLOCKSORT.mat',...
95                  '20141119-1412_20_1800BLOCKSORT.mat',...
96                  '20141119-1421_20_1850BLOCKSORT.mat',...
97                  '20141119-1430_20_1900BLOCKSORT.mat',...
98                  '20141119-1439_20_1950BLOCKSORT.mat',...
99                  '20141119-1449_20_2000BLOCKSORT.mat',...
100                 '20141119-1458_20_2050BLOCKSORT.mat',...
101                 '20141119-1536_20_2100BLOCKSORT.mat',...
102                 '20141119-1545_20_2150BLOCKSORT.mat',...
103                 '20141119-1556_20_2200BLOCKSORT.mat',...
104                 '20141119-1607_20_2250BLOCKSORT.mat',...
105                 '20141119-1618_20_2300BLOCKSORT.mat',...
106                 '20141119-1629_20_2350BLOCKSORT.mat',...
107                 '20141119-1639_20_2400BLOCKSORT.mat',...
108                 '20141125-1336_30_1700BLOCKSORT.mat',...
109                 '20141125-1349_30_1750BLOCKSORT.mat',...
110                 '20141125-1359_30_1800BLOCKSORT.mat',...
111                 '20141125-1410_30_1850BLOCKSORT.mat',...
112                 '20141125-1421_30_1900BLOCKSORT.mat',...
113                 '20141125-1430_30_1950BLOCKSORT.mat',...
114                 '20141125-1439_30_2000BLOCKSORT.mat',...
115                 '20141125-1450_30_2050BLOCKSORT.mat',...
116                 '20141125-1500_30_2100BLOCKSORT.mat',...
117                 '20141125-1512_30_2150BLOCKSORT.mat',...
118                 '20141125-1524_30_2200BLOCKSORT.mat',...
119                 '20141125-1534_30_2250BLOCKSORT.mat',...
120                 '20141125-1542_30_2300BLOCKSORT.mat',...
121                 '20141125-1552_30_2350BLOCKSORT.mat',...
122                 '20141125-1605_30_2400BLOCKSORT.mat',...
123                 '20141125-1717_40_1700BLOCKSORT.mat',...
124                 '20141125-1732_40_1750BLOCKSORT.mat',...
125                 '20141125-1750_40_1800BLOCKSORT.mat',...
126                 '20141125-1804_40_1850BLOCKSORT.mat',...
127                 '20141125-1825_40_1900BLOCKSORT.mat',...
128                 '20141125-1838_40_1950BLOCKSORT.mat',...
129                 '20141125-1852_40_2000BLOCKSORT.mat',...
130                 '20141125-1902_40_2050BLOCKSORT.mat',...
131                 '20141125-1917_40_2100BLOCKSORT.mat',...
132                 '20141125-1930_40_2150BLOCKSORT.mat',...
133                 '20141125-1943_40_2200BLOCKSORT.mat',...
134                 '20141125-1957_40_2250BLOCKSORT.mat',...
135                 '20141125-2012_40_2300BLOCKSORT.mat',...
136                 '20141125-2030_40_2350BLOCKSORT.mat',...
137                 '20141125-2049_40_2400BLOCKSORT.mat',...
138                 '20141126-1257_50_1700BLOCKSORT.mat',...
139                 '20141126-1308_50_1750BLOCKSORT.mat',...
140                 '20141126-1318_50_1800BLOCKSORT.mat',...
141                 '20141126-1328_50_1850BLOCKSORT.mat',...
142                 '20141126-1340_50_1900BLOCKSORT.mat',...
143                 '20141126-1353_50_1950BLOCKSORT.mat',...
144                 '20141126-1403_50_2000BLOCKSORT.mat',...
145                 '20141126-1413_50_2050BLOCKSORT.mat',...
146                 '20141126-1425_50_2100BLOCKSORT.mat',...
147                 '20141126-1434_50_2150BLOCKSORT.mat',...
148                 '20141126-1444_50_2200BLOCKSORT.mat',...
149                 '20141126-1456_50_2250BLOCKSORT.mat',...
150                 '20141126-1506_50_2300BLOCKSORT.mat',...
151                 '20141126-1515_50_2350BLOCKSORT.mat',...
152                 '20141126-1531_50_2400BLOCKSORT.mat'};
153
154 %% Variablen und Konstanten initialisieren
155 noTemps = size(tempReal,2);
```

```
noVolts = size(voltReal,1);

sensor = struct( 'volt', {},...                          % alle Spannungsmesswerte
                 'temp', {},...                          % alle Temperaturmesswerte
                 'avgVolt', zeros(noVolts, noTemps),... % mittlerer Spannungsmesswert
                 'avgTemp', zeros(noVolts, noTemps),... % mittlerer Temperaturmesswert
                 'aV', zeros(noTemps,1),...              % Regression (Spannung): Steigungen
                 'bV', zeros(noTemps,1),...              % Regression (Spannung): Achsenabschnitte
                 'resV', zeros(noVolts, noTemps),...     % Regression (Spannung): Residuen
                 'aT', 0,...                             % Regression (Temperatur): Steigung
                 'bT', 0,...                             % Regression (Temperatur): Achsenabschnitt
                 'resT', zeros(noTemps,1),...            % Regression (Temperatur): Residuen
                 'aKS', 0,...                            % Regression (Koeffizienten/Steigung): Steigung
                 'bKS', 0,...                            % Regression (Koeffizienten/Steigung): Achsenabschnitt
                 'resKS', zeros(noTemps,1),...           % Regression (Koeffizienten/Steigung): Residuen
                 'aKA', 0,...                            % Regression (Koeffizienten/Achsenabschnitt): Steigung
                 'bKA', 0,...                            % Regression (Koeffizienten/Achsenabschnitt): Achsenabschnitt
                 'resKA', zeros(noTemps,1),...           % Regression (Koeffizienten/Achsenabschnitt): Residuen
                 'regVolt', zeros(noVolts, noTemps));    % Datenpunkte der Regressionsgeraden (Spannung)

for n=1:1:noSensors
    sensor(n).volt = cell(noVolts,noTemps);
    sensor(n).temp = cell(noVolts,noTemps);
end
clear n;

%------------------------------------------------------------------------
% Kontrastreiche Farben fuer Plots definieren
% Quelle:
% http://eleanormaclure.files.wordpress.com/2011/03/colour-coding.pdf
colors = [    0,117,220; 240,163,255; 153, 63,   0;  76,   0,  92;...
             25, 25, 25;   0, 92, 49;  43,206, 72; 255,204,153;...
            128,128,128; 148,255,181; 143,124,   0; 157,204,   0;...
            194,   0,136;   0, 51,128; 255,164,   5; 255,168,187;...
             66,102,   0; 255,   0, 16;  94,241,242;   0,153,143;...
            224,255,102; 116, 10,255; 153,   0,   0; 255,255,128;...
            255,255,   0; 255, 80,   5] ./255;

%% Rohdaten einlesen und Mittelwerte berechnen
for l=1:1:noTemps
  for m=1:1:noVolts
    load(char(files(noVolts*(l-1)+m)), 'dats');
    for n=1:1:noSensors
      idx                 = find(dats(:,2)==n);
      sensor(n).temp(m,l) = {dats(idx,4)};
      sensor(n).volt(m,l) = {dats(idx,3)};
      sensor(n).avgVolt(m,l) = mean(dats(idx,3));
      sensor(n).avgTemp(m,l) = mean(dats(idx,4));
      if isempty(idx) && isempty(find(defectSensors==n))
        disp(['No Sensor ' num2str(n) ' data in: ' char(files(noVolts*(l-1)+m))]);
      end
      clear idx;
    end
    clear dats;
  end
end
clear l m n;

%% Regressionsrechnung durchfuehren
for n=1:1:noSensors
  if isempty(find(defectSensors==n))
    for l=1:1:noTemps
      [k b r] = regress(voltReal(:,l), [sensor(n).avgVolt(:,l) ones(noVolts,1)]);
      sensor(n).aV(l)       = k(1);
      sensor(n).bV(l)       = k(2);
      sensor(n).resV(:,l)   = r;
      sensor(n).regVolt(:,l) = sensor(n).aV(l) * sensor(n).avgVolt(:,l) + sensor(n).bV(l);
    end
    [k b r] = regress(tempReal', [nanmean(sensor(n).avgTemp)' ones(noTemps,1)]);
    sensor(n).aT    = k(1);
    sensor(n).bT    = k(2);
    sensor(n).resT  = r;
    [k b r] = regress(sensor(n).aV', [tempReal' ones(noTemps,1)]);
    sensor(n).aKS   = k(1);
    sensor(n).bKS   = k(2);
    sensor(n).resKS = r;
    [k b r] = regress(sensor(n).bV', [tempReal' ones(noTemps,1)]);
    sensor(n).aKA   = k(1);
    sensor(n).bKA   = k(2);
    sensor(n).resKA = r;
    clear k b r;

  % Koeffizienten ausgeben
    disp(' '); disp(' ');
    disp(['Sensor ' num2str(n)]);
```

```
241       disp('␣Temperatur:␣␣␣␣␣␣␣␣␣␣Steigung:␣␣␣␣␣␣␣␣␣␣Achsenabschnitt:');
242       disp(['tempReal' sensor(n).aV' sensor(n).bV']);
243       disp(['Temperaturgerade␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=>␣Steigung:␣' num2str(sensor(n).aT, '%+0.5e') '␣␣␣Achsenabschnitt:␣' num2str(sensor(
              n).bT, '%+0.5e')]);
244       disp(['Koeffizientengerade␣(Steigung)␣␣␣␣␣␣␣␣=>␣Steigung:␣' num2str(sensor(n).aKS, '%+0.5e') '␣␣␣Achsenabschnitt:␣' num2str(sensor(
              n).bKS, '%+0.5e')]);
245       disp(['Koeffizientengerade␣(Achsenabschnitt)␣=>␣Steigung:␣' num2str(sensor(n).aKA, '%+0.5e') '␣␣␣Achsenabschnitt:␣' num2str(sensor(
              n).bKA, '%+0.5e')]);
246     end
247   end
248   clear n l;
249
250   %% Spannungskennlinien
251   set(0,'DefaultAxesColorOrder',colors);
252   for n=1:1:noSensors
253     if isempty(find(defectSensors==n))
254     % Spannungskennlinien
255       figure('units','normalized','outerposition',[0 0 .66 .66]);
256       plot(sensor(n).avgVolt, voltReal, '.');
257       hold on;
258       plot(sensor(n).avgVolt, sensor(n).regVolt, '-');
259       legend(strcat(num2str(tempReal'), '␣°C'));
260       grid;
261       hold off;
262       xlabel('U_{Sensor}␣␣[mV]');
263       ylabel('U_{Normal}␣␣[mV]');
264       title(['Sensor␣' num2str(n) '␣-␣Spannungskennlinien']);
265
266     % Residuen (Spannung)
267       figure('units','normalized','outerposition',[0 0 .66 .66]);
268       bar(voltIdeal, sensor(n).resV);
269       legend(strcat(num2str(tempReal'), '␣°C'));
270       xlabel('U_{Ideal}␣␣[V]');
271       ylabel('\DeltaU␣␣[mV]');
272       title(['Sensor␣' num2str(n) '␣-␣Residuen␣der␣Spannungskennlinien']);
273     end
274   end
275   clear n;
276
277   %% Temperaturkennlinien
278   for n=1:1:noSensors
279     if isempty(find(defectSensors==n))
280     % Temperaturkennlinien
281       figure('units','normalized','outerposition',[0 0 .66 .66]);
282       t = nanmean(sensor(n).avgTemp);
283       plot(t, tempReal, '.');
284       hold on;
285       plot(t, sensor(n).aT * t + sensor(n).bT, '-');
286       grid;
287       hold off;
288       xlabel('ADC_{Sensor}');
289       ylabel('T_{Normal}␣␣[°C]');
290       title(['Sensor␣' num2str(n) '␣-␣Temperaturkennlinie']);
291
292     % Residuen (Temperatur)
293       figure('units','normalized','outerposition',[0 0 .66 .66]);
294       bar(tempReal, sensor(n).resT);
295       xlabel('T_{Normal}␣␣[°C]');
296       ylabel('\DeltaT␣␣[°C]');
297       title(['Sensor␣' num2str(n) '␣-␣Residuen␣der␣Temperaturkennlinie']);
298     end
299   end
300   clear n t;
301
302   %% Zusammenhang der Regressionskoeffizienten (Steigung)
303   for n=1:1:noSensors
304     if isempty(find(defectSensors==n))
305     % Zusammenhang der Regressionskoeffizienten (Steigung) der Spannungskennlinie
306       figure('units','normalized','outerposition',[0 0 .66 .66]);
307       plot(tempReal, sensor(n).aV, '.');
308       hold on;
309       plot(tempReal, sensor(n).aKS * tempReal + sensor(n).bKS, '-');
310       grid;
311       hold off;
312       xlabel('T_{Normal}␣␣[°C]');
313       ylabel('Regressionskoeffizient');
314       title(['Sensor␣' num2str(n) '␣-␣Temperaturabhängigkeit␣der␣Regressionskoeffizienten␣(Steigung)']);
315
316     % Residuen (Koeffizienten)
317       figure('units','normalized','outerposition',[0 0 .66 .66]);
318       bar(tempReal, sensor(n).resKS);
319       xlabel('T_{Normal}␣␣[°C]');
320       ylabel('Abweichung');
321       title(['Sensor␣' num2str(n) '␣-␣Residuen␣(Regressionskoeffizienten␣der␣Steigung)']);
322     end
```

```
end
clear n;

%% Zusammenhang der Regressionskoeffizienten (Achsenabschnitt)
for n=1:1:noSensors
  if isempty(find(defectSensors==n))
  % Zusammenhang der Regressionskoeffizienten (Achsenabschnitt) der Spannungskennlinie
    figure('units','normalized','outerposition',[0 0 .66 .66]);
    plot(tempReal, sensor(n).bV, '.');
    hold on;
    plot(tempReal, sensor(n).aKA * tempReal + sensor(n).bKA, '-');
    grid;
    hold off;
    xlabel('T_{Normal} [°C]');
    ylabel('Regressionskoeffizient [mV]');
    title(['Sensor ' num2str(n) ' - Temperaturabhängigkeit der Regressionskoeffizienten (Achsenabschnitt)']);

  % Residuen (Koeffizienten)
    figure('units','normalized','outerposition',[0 0 .66 .66]);
    bar(tempReal, sensor(n).resKA);
    xlabel('T_{Normal} [°C]');
    ylabel('Abweichung [mV]');
    title(['Sensor ' num2str(n) ' - Residuen (Regressionskoeffizienten des Achsenabschnitts)']);
  end
end
clear n;

%% Kalibrationskoeffizienten in Datei speichern
coeffs = struct( 'aV', zeros(noTemps,1),... % Regression (Spannung): Steigungen
                 'bV', zeros(noTemps,1),... % Regression (Spannung): Achsenabschnitte
                 'aT',  0,...               % Regression (Temperatur): Steigung
                 'bT',  0,...               % Regression (Temperatur): Achsenabschnitt
                 'aKS', 0,...               % Regression (Koeffizienten/Steigung): Steigung
                 'bKS', 0,...               % Regression (Koeffizienten/Steigung): Achsenabschnitt
                 'aKA', 0,...               % Regression (Koeffizienten/Achsenabschnitt): Steigung
                 'bKA', 0);                 % Regression (Koeffizienten/Achsenabschnitt): Achsenabschnitt

for n=1:1:noSensors
  coeffs(n).aV  = sensor(n).aV;
  coeffs(n).bV  = sensor(n).bV;
  coeffs(n).aT  = sensor(n).aT;
  coeffs(n).bT  = sensor(n).bT;
  coeffs(n).aKS = sensor(n).aKS;
  coeffs(n).bKS = sensor(n).bKS;
  coeffs(n).aKA = sensor(n).aKA;
  coeffs(n).bKA = sensor(n).bKA;
end

save('calibration_data.mat', 'coeffs', 'noSensors', 'defectSensors', 'tempReal');
clear coeffs;
```

B.2.3. Auswertung.m

```
clc
clear all;
close all;

%% Parameter
[data_f data_p] = uigetfile('', 'Messdaten');
[cal_f cal_p]   = uigetfile('calibration_data.mat', 'Kalibrationsdaten');
offset          = str2num(char(inputdlg( {'Offset'}, 'Parameter', 1, {'10'})));


%-------------------------------------------------------------------------
% Kontrastreiche Farben fuer Plots definieren
% Quelle:
% http://eleanormaclure.files.wordpress.com/2011/03/colour-coding.pdf
colors = [   0,117,220; 240,163,255; 153, 63,  0;  76,  0, 92;...
            25, 25, 25;   0, 92, 49;  43,206, 72; 255,204,153;...
           128,128,128; 148,255,181; 143,124,  0; 157,204,  0;...
           194,  0,136;   0, 51,128; 255,164,  5; 255,168,187;...
            66,102,  0; 255,  0, 16;  94,241,242;   0,153,143;...
           224,255,102; 116, 10,255; 153,  0,  0; 255,255,128;...
           255,255,  0; 255, 80,  5] ./255;

sensor = struct( 'volt',     0,...
                 'volt_y',   0,...
                 'temp',     0,...
                 'temp_y',   0,...
                 'red',      0,...
                 'ylw',      0,...
                 'time',     0,...
```

```
30                  'time_y',    0,...
31                  'temp_idx', 0 );
32
33
34 %% Daten einlesen
35 load([data_p data_f],'dats');
36 load([cal_p cal_f]);
37 clear data_p cal_f cal_p;
38
39 for n=1:1:noSensors
40   idx               = find(dats(:,2)==n);
41   idx2              = find(dats(:,2)==n+offset);
42   sensor(n).temp    = dats(idx,4);
43   sensor(n).temp_y  = dats(idx2,4);
44   sensor(n).volt    = dats(idx,3);
45   sensor(n).volt_y  = dats(idx2,3);
46   sensor(n).red     = dats(idx,5);
47   sensor(n).ylw     = dats(idx2,5);
48   sensor(n).time    = dats(idx,12);
49   sensor(n).time_y  = dats(idx2,12);
50   clear idx idx2;
51 end
52 clear dats n;
53
54 %% Korrektur mit Kalibrationsdaten
55 for n=1:1:noSensors
56   if isempty(find(defectSensors==n))
57    % Temperatur kompensieren
58     sensor(n).temp = sensor(n).temp * coeffs(n).aT + coeffs(n).bT;
59    % Indices des dichtesten Temperaturwertes
60     [~, idx]  = min(abs(bsxfun(@minus, (sensor(n).temp   * ones(size(tempReal))), tempReal)),[],2);
61     [~, idx2] = min(abs(bsxfun(@minus, (sensor(n).temp_y * ones(size(tempReal))), tempReal)),[],2);
62    % Spannung kompensieren
63     sensor(n).volt = sensor(n).volt .* (sensor(n).temp * coeffs(n).aKS + coeffs(n).bKS) + coeffs(n).bV(idx)';
64     sensor(n).volt_y = sensor(n).volt_y .* (sensor(n).temp_y * coeffs(n).aKS + coeffs(n).bKS) + coeffs(n).bV(idx2)';
65   end
66 end
67 clear n idx idx2;
68
69 %% Darstellung - Messdaten gegen Zeit
70 set(0,'DefaultAxesColorOrder',colors);
71 figure('units','normalized','outerposition',[0 0 1 1],'Name',data_f);
72
73 % Spannung
74 hnd_plot(1) = subplot(4,1,1);
75 hold on;
76 for n=1:1:noSensors
77   if isempty(find(defectSensors==n))
78     plot(sensor(n).time, sensor(n).volt, 'LineStyle','.', 'Color',colors(n,:), 'LineWidth',2);
79   end
80 end
81 hold off;
82 ylabel('Zellspannung [mV]');
83 title('Zellspannung');
84
85 % Transmissionsleistung (rot)
86 hnd_plot(2)=subplot(4,1,2);
87 hold on;
88 for n=1:1:noSensors
89   if isempty(find(defectSensors==n))
90     plot(sensor(n).time, sensor(n).red, 'LineStyle','.', 'Color',colors(n,:), 'LineWidth',2);
91   end
92 end
93 hold off;
94 ylabel('Light-Frequency Sensor [Ticks]');
95 title('Dichte bei Wellenlänge 633nm');
96
97 % Transmissionsleistung (gelb)
98 hnd_plot(3)=subplot(4,1,3);
99 hold on;
100 for n=1:1:noSensors
101   if isempty(find(defectSensors==n))
102     plot(sensor(n).time_y, sensor(n).ylw, 'LineStyle','.', 'Color',colors(n,:), 'LineWidth',2);
103   end
104 end
105 hold off;
106 ylabel('Light-Frequency Sensor [Ticks]');
107 title('Dichte bei Wellenlänge 587nm');
108
109 % Temperatur
110 hnd_plot(4)=subplot(4,1,4);
111 hold on;
112 for n=1:1:noSensors
113   if isempty(find(defectSensors==n))
114     plot(sensor(n).time, sensor(n).temp, 'LineStyle','.', 'Color',colors(n,:), 'LineWidth',2);
```

```
115   end
116 end
117 hold off;
118 ylabel('Temperatur [°C]');
119 title('Temperaturmessung des ADC');
120
121 % gemeinsame Eigenschaften setzen
122 linkaxes(hnd_plot,'x');
123 for n=1:1:size(hnd_plot,2)
124   grid(hnd_plot(n), 'minor');
125   datetick(hnd_plot(n), 'x', 'keepticks');
126   xlabel(hnd_plot(n), 'Zeit [hh:mm]');
127   legend(hnd_plot(n), strread(sprintf('Sensor %d\n', setdiff([1:1:noSensors], defectSensors)),'%s',noSensors,'delimiter','\n'));
128 end
129
130 clear hnd_plot n;
131
132
133 %% Darstellung - Transmissionsleistung gegen Zellspannung (rot)
134 set(0,'DefaultAxesColorOrder',colors);
135 figure('units','normalized','outerposition',[0 0 1 1],'Name',data_f);
136 hold on;
137 for n=1:1:noSensors
138   if isempty(find(defectSensors==n))
139     plot(sensor(n).volt, sensor(n).red, 'LineStyle','.', 'Color',colors(n,:), 'LineWidth',2);
140   end
141 end
142 hold off;
143 ylabel('Light-Frequency Sensor [Ticks]');
144 title('Transmissionsleistung vs. Spannung bei Wellenlänge 633nm');
145 grid on;
146 xlabel('Zellspannung [mV]');
147 legend(strread(sprintf('Sensor %d\n', setdiff([1:1:noSensors], defectSensors)),'%s',noSensors,'delimiter','\n'));
148 clear n;
149
150 %% Darstellung - Transmissionsleistung gegen Zellspannung (gelb)
151 set(0,'DefaultAxesColorOrder',colors);
152 figure('units','normalized','outerposition',[0 0 1 1],'Name',data_f);
153 hold on;
154 for n=1:1:noSensors
155   if isempty(find(defectSensors==n))
156     plot(sensor(n).volt_y, sensor(n).ylw, 'LineStyle','.', 'Color',colors(n,:), 'LineWidth',2);
157   end
158 end
159 hold off;
160 ylabel('Light-Frequency Sensor [Ticks]');
161 title('Transmissionsleistung vs. Spannung bei Wellenlänge 587nm');
162 grid on;
163 xlabel('Zellspannung [mV]');
164 legend(strread(sprintf('Sensor %d\n', setdiff([1:1:noSensors], defectSensors)),'%s',noSensors,'delimiter','\n'));
165 clear n;
```

B.3. Konfig-Datei

```
1  //*********************************************
2  //Konfigurationsdatei Prototyp von Maher Achour
3  //Erstellungsdatum: 03.09.2014
4  //*********************************************
5
6  quantity 6  //Anzahl Zellen
7  cyclestep 3  //Messdauer
8
9  //Kalibrierwerte
10 gain 01 79070
11 gain 02 79069
12 gain 03 79071
13 gain 04 79078
14 gain 05 79100
15 gain 06 79093
16 gain 07 79090
17 gain 08 79111
18 gain 09 79117
19 gain 10 79119
20 gain 11 79087
21 gain 12 79095
22
23 offset 01 165274
24 offset 02 165285
25 offset 03 165171
26 offset 04 164931
27 offset 05 164395
```

```
28 offset 06 164521
29 offset 07 164661
30 offset 08 164110
31 offset 09 164046
32 offset 10 163943
33 offset 11 163943
34 offset 12 164483
35
36 //Zeitgrenzen in Minuten
37 meastime 5000
38 maxcycletime 5000
39
40 //Grenzwerte zur Sicherheit in mV
41 vlimits 1800 2350
42 maxcurr 10000
43 maxtemp 60
44 maxcharge 100000
45
46 //Zyklierablauf
47 relsel 01 wait 120 0  //Ruhe
48 relsel 02 wait 360 2  //Laden
49 relsel 03 wait 360 0  //Ruhe
50 relsel 04 wait 360 2  //Laden
51 relsel 05 wait 360 0  //Ruhe
52 relsel 06 wait 360 2  //Laden
53 relsel 08 wait 360 0  //Ruhe
54 relsel 09 wait 360 1  //Entladen
55 relsel 10 wait 360 0  //Ruhe
56 relsel 11 wait 360 2  //Laden
57 relsel 12 wait 360 0  //Ruhe
58 relsel 13 wait 360 1  //Entladen
59 relsel 14 wait 360 0  //Ruhe
60 relsel 15 wait 360 2  //Laden
61 relsel 16 wait 360 0  //Ruhe
62 relsel 17 wait 360 1  //Entladen
63 relsel 18 wait 360 0  //Ruhe
64 relsel 19 wait 360 2  //Laden
65 relsel 20 wait 360 0  //Ruhe
66 relsel 21 wait 360 1  //Entladen
67 relsel 22 wait 360 0  //Ruhe
68 relsel 23 wait 360 2  //Laden
69 relsel 24 wait 360 0  //Ruhe
70 relsel 25 wait 360 1  //Entladen
```

C. Hardware

C.1. Zellensensor

C.1.1. Schaltplan

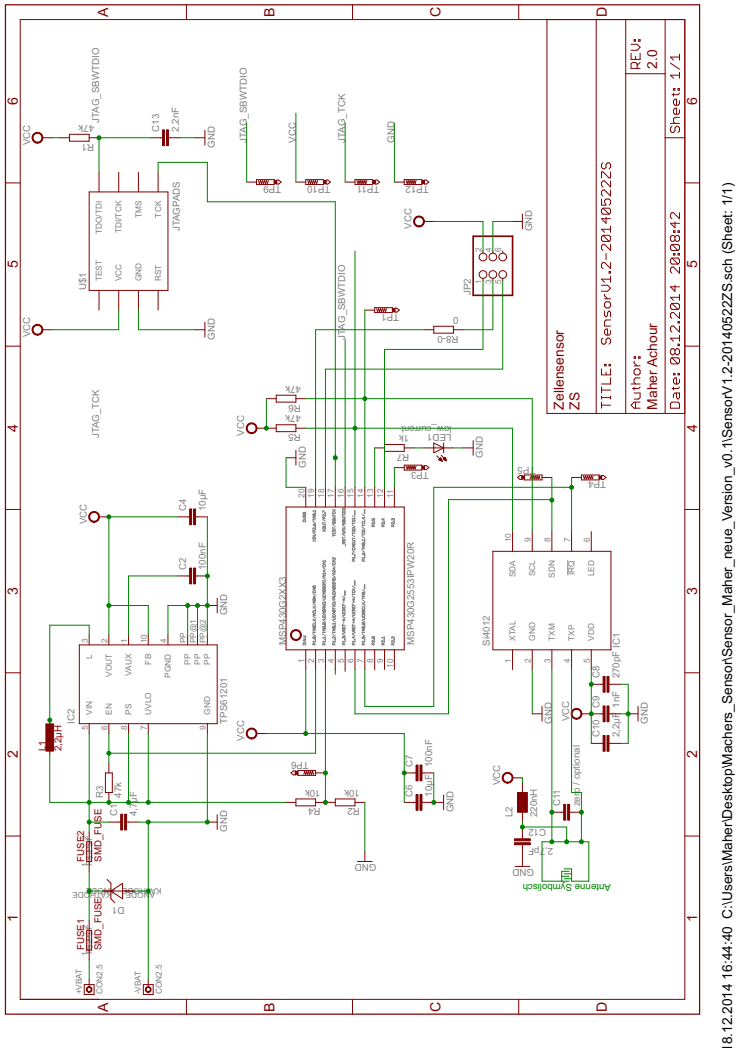


Abbildung C.1.: Schaltplan des erweiterten Zellensensors

C.1.2. Platinenlayout

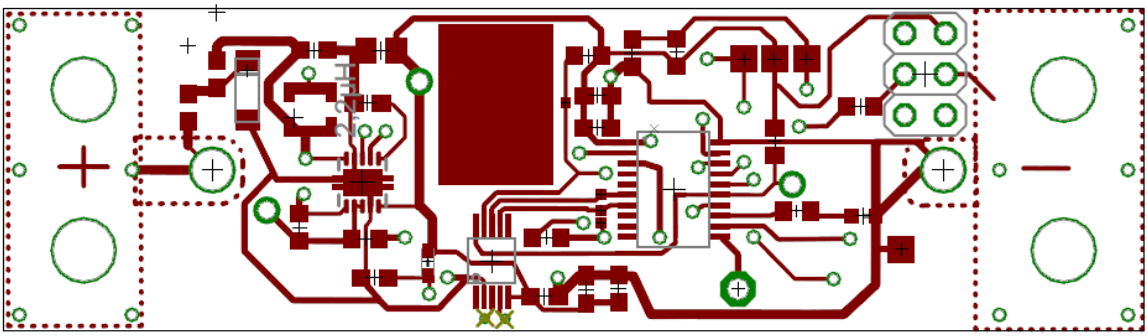


Abbildung C.2.: Platinenlayout des erweiterten Zellensensors (Top-Layer)

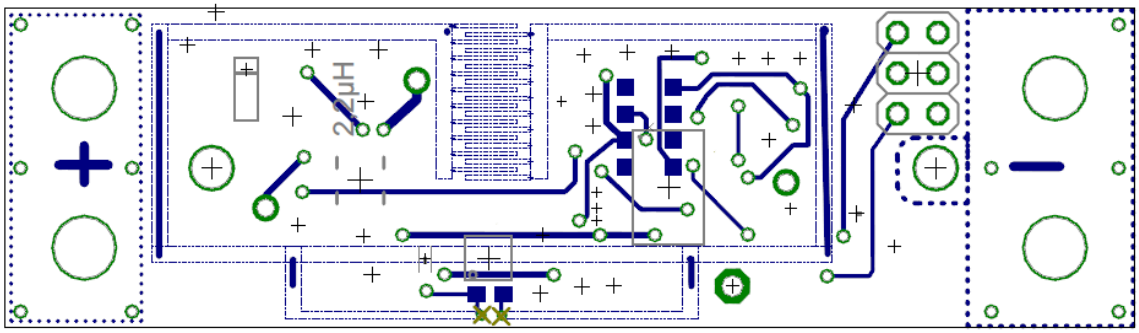


Abbildung C.3.: Platinenlayout des erweiterten Zellensensors (Bottom-Layer)

C.2. Dichte-Sensor-Modul

C.2.1. Schaltplan

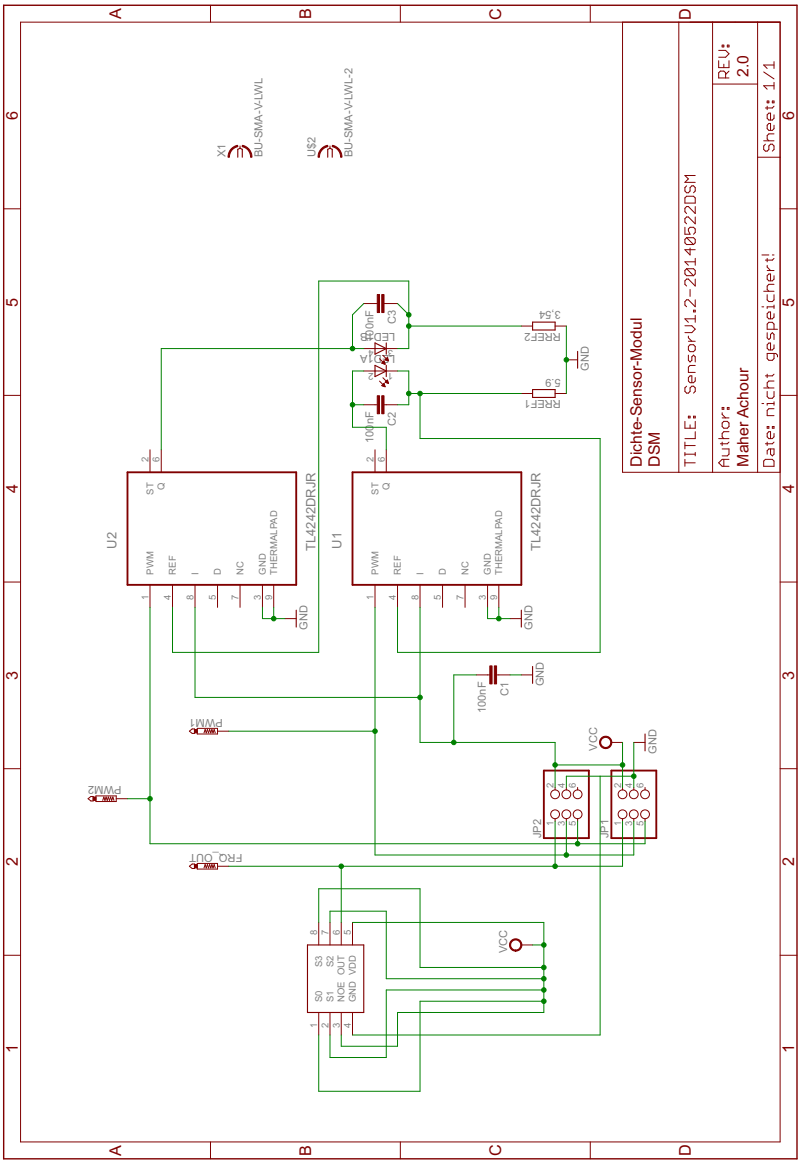


Abbildung C.4.: Schaltplan des erweiterten Dichte-Sensor-Moduls

C.2.2. Platinenlayout

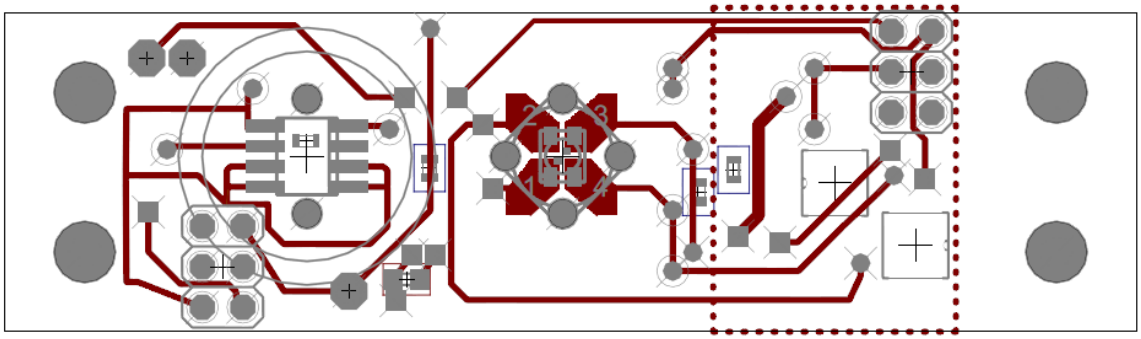


Abbildung C.5.: Platinenlayout des erweiterten Dichte-Sensor-Moduls (Top-Layer)

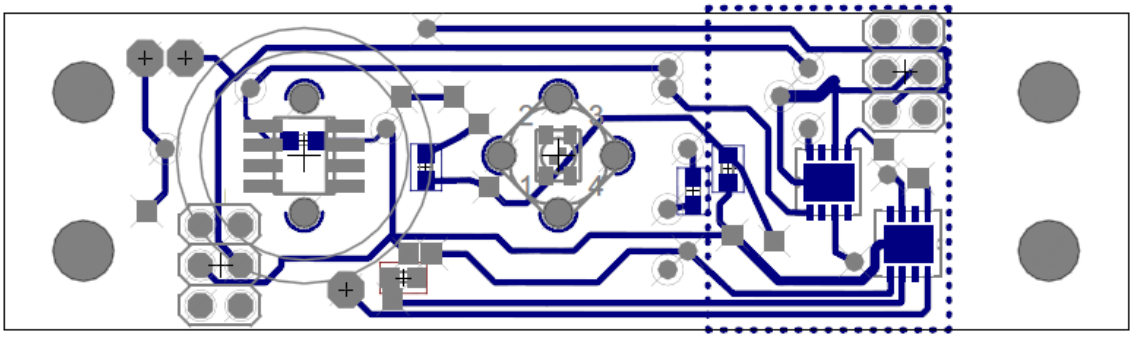


Abbildung C.6.: Platinenlayout des erweiterten Dichte-Sensor-Moduls (Bottom-Layer)

D. Auswertung Kalibrierung

D.1. Temperaturkennlinien

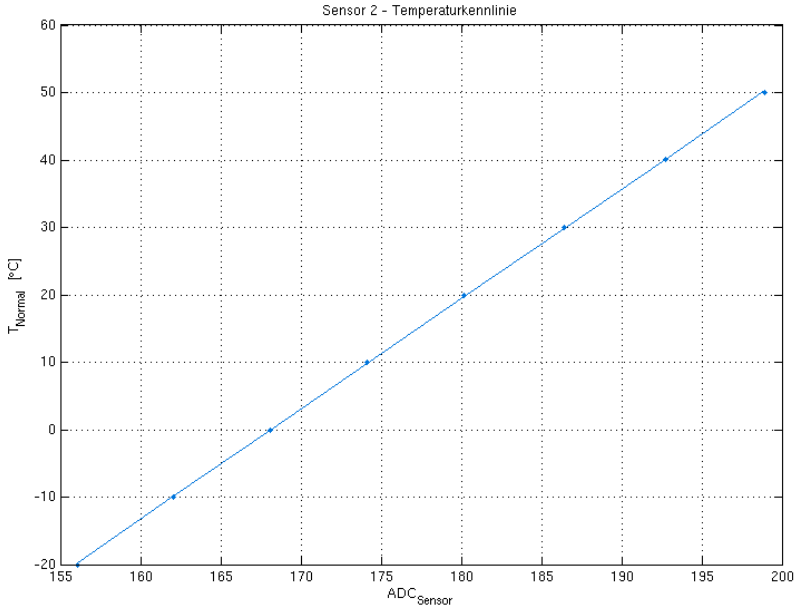


Abbildung D.1.: Temperaturkennlinie Sensor 2

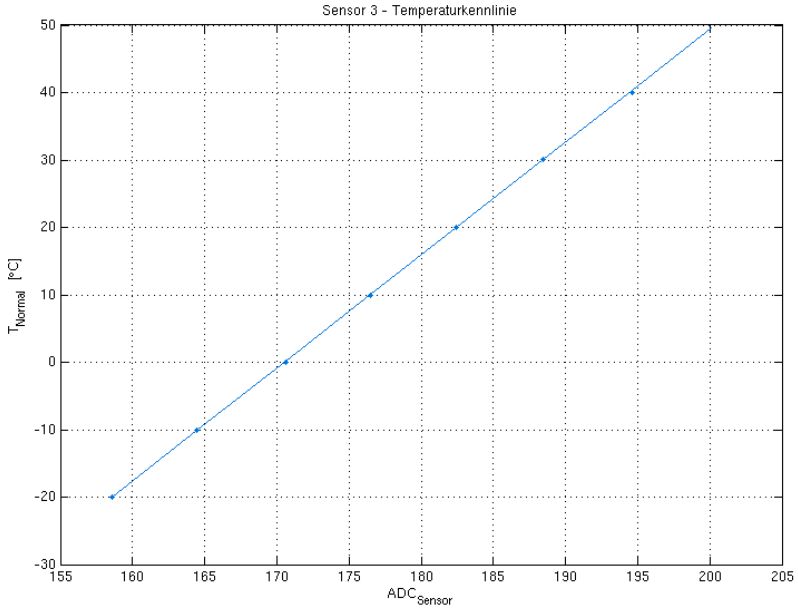


Abbildung D.2.: Temperaturkennlinie Sensor 3

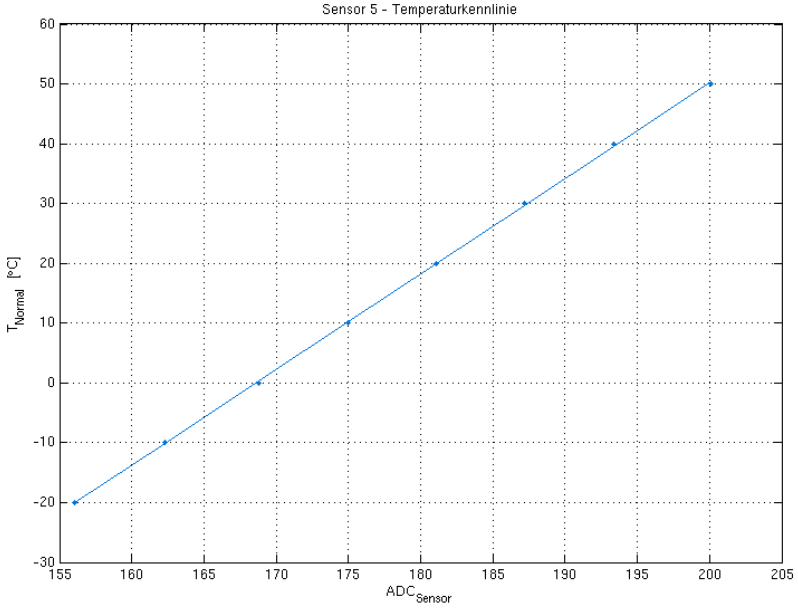


Abbildung D.3.: Temperaturkennlinie Sensor 5

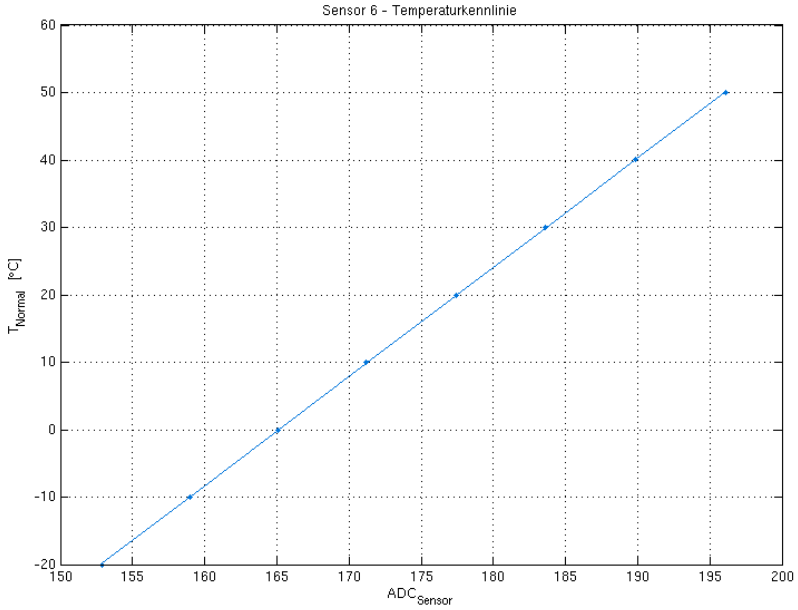


Abbildung D.4.: Temperaturkennlinie Sensor 6

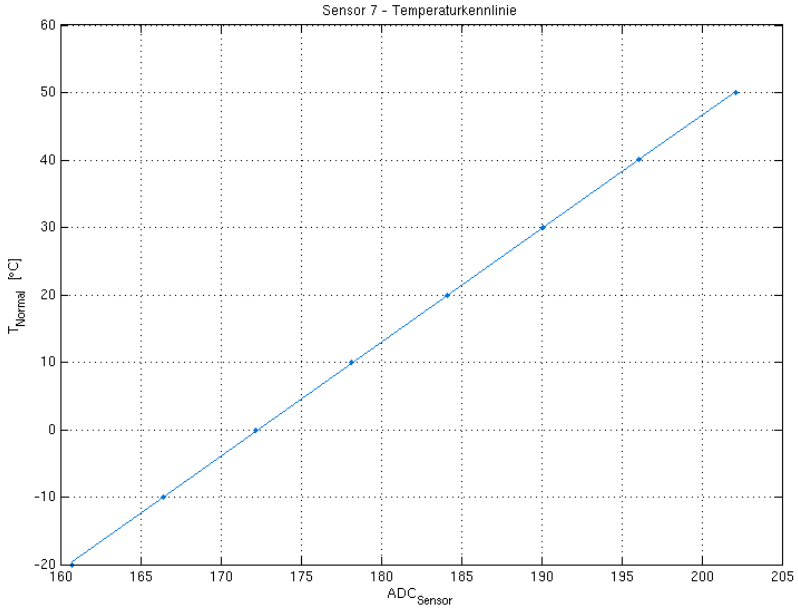


Abbildung D.5.: Temperaturkennlinie Sensor 7

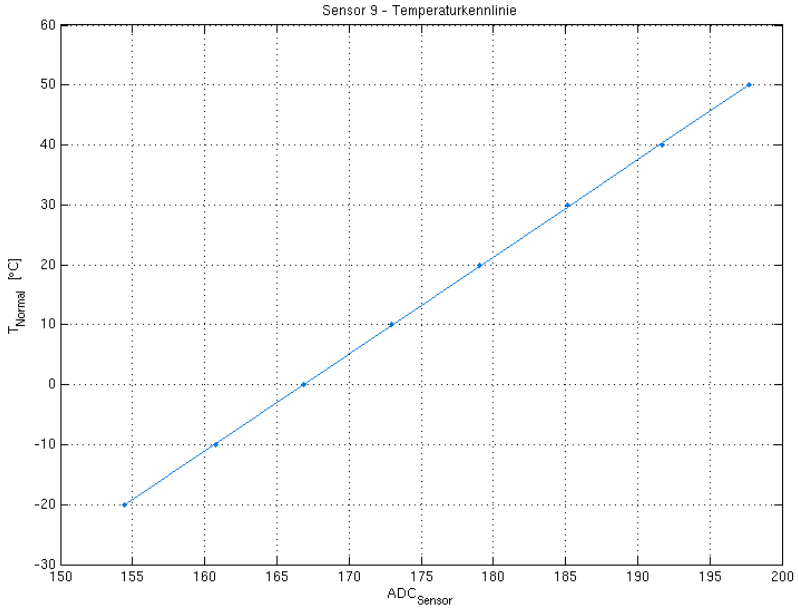


Abbildung D.6.: Temperaturkennlinie Sensor 9

D.2. Residuen der Temperaturkennlinien

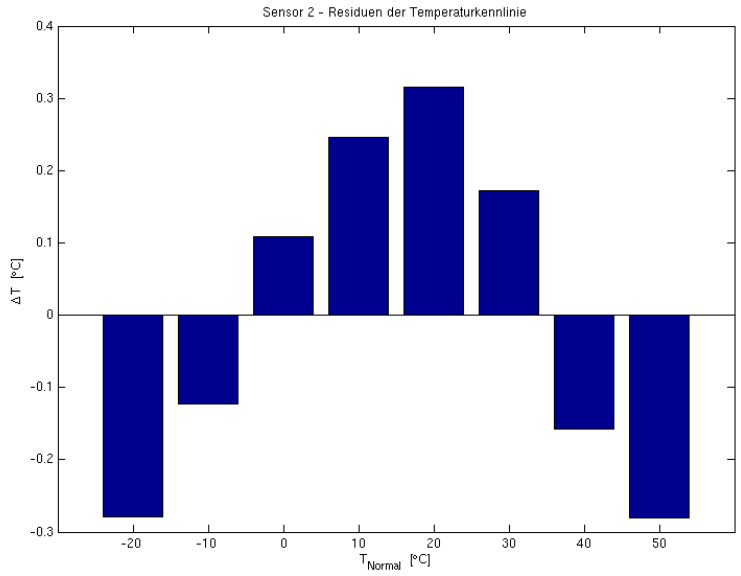


Abbildung D.7.: Residuen der Temperaturkennlinie Sensor 2

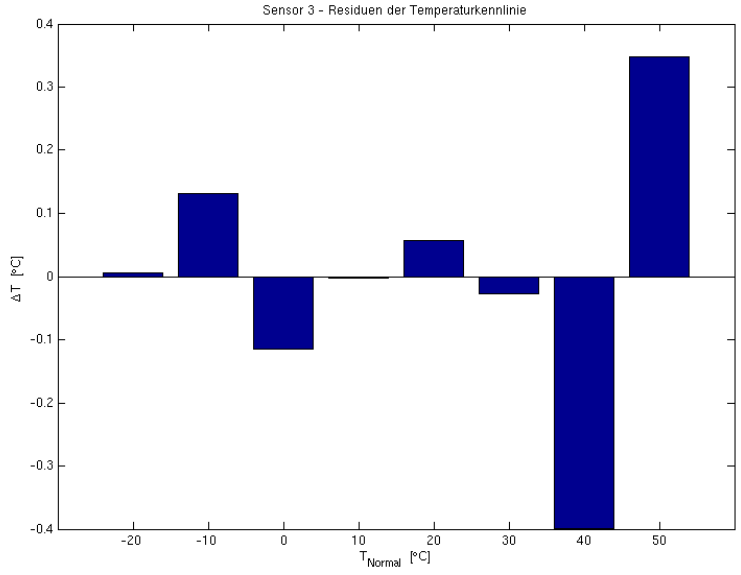


Abbildung D.8.: Residuen der Temperaturkennlinie Sensor 3

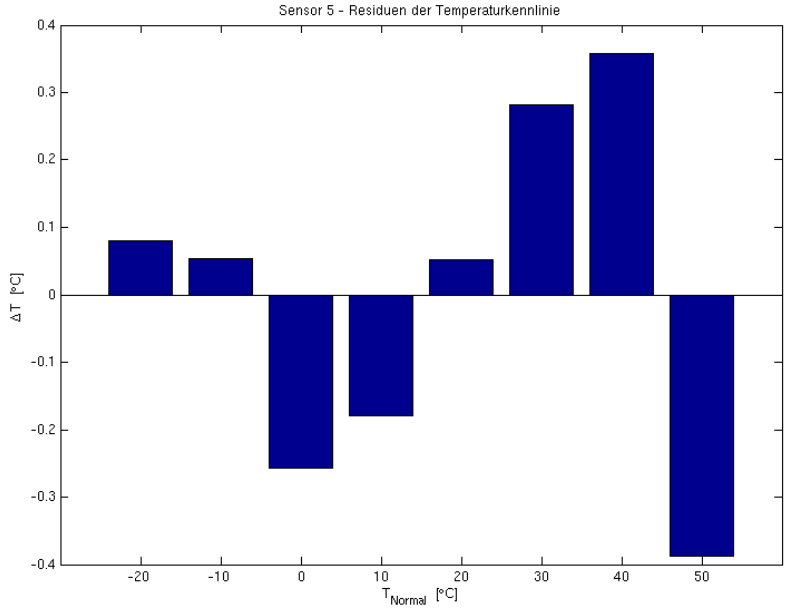


Abbildung D.9.: Residuen der Temperaturkennlinie Sensor 5

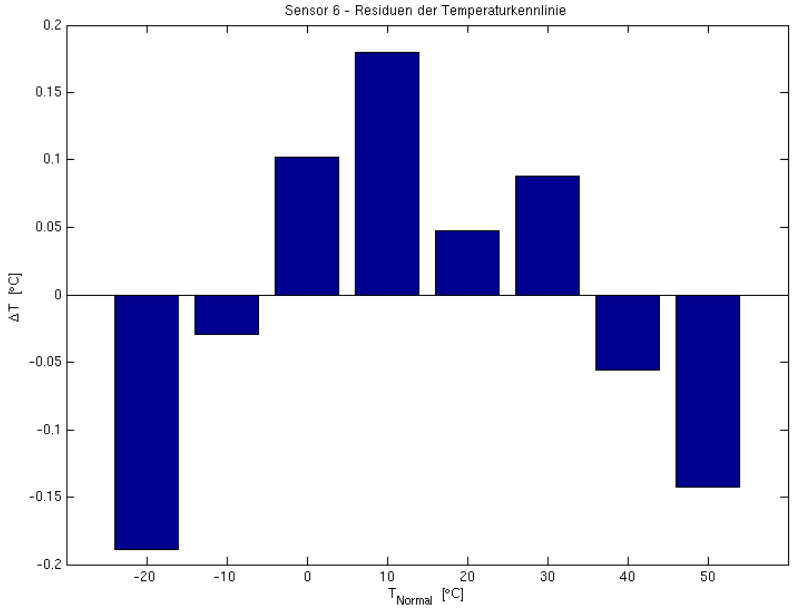


Abbildung D.10.: Residuen der Temperaturkennlinie Sensor 6

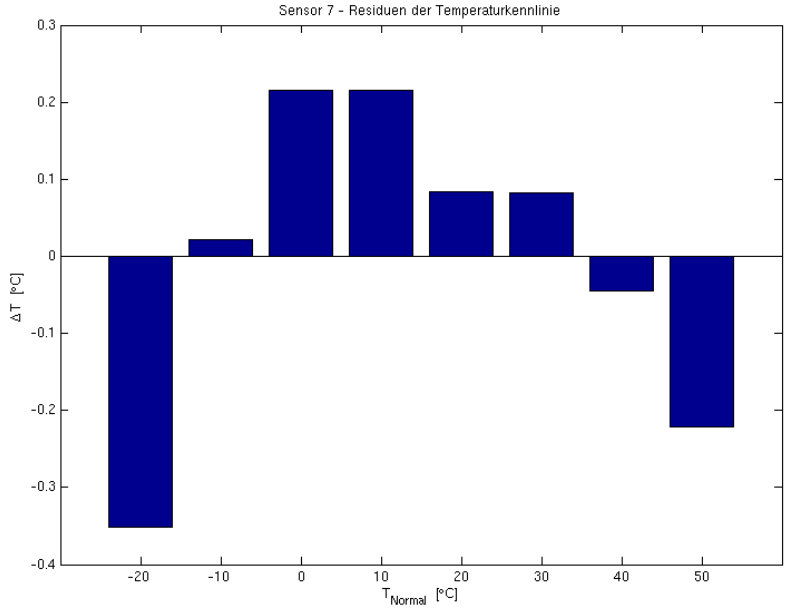


Abbildung D.11.: Residuen der Temperaturkennlinie Sensor 7

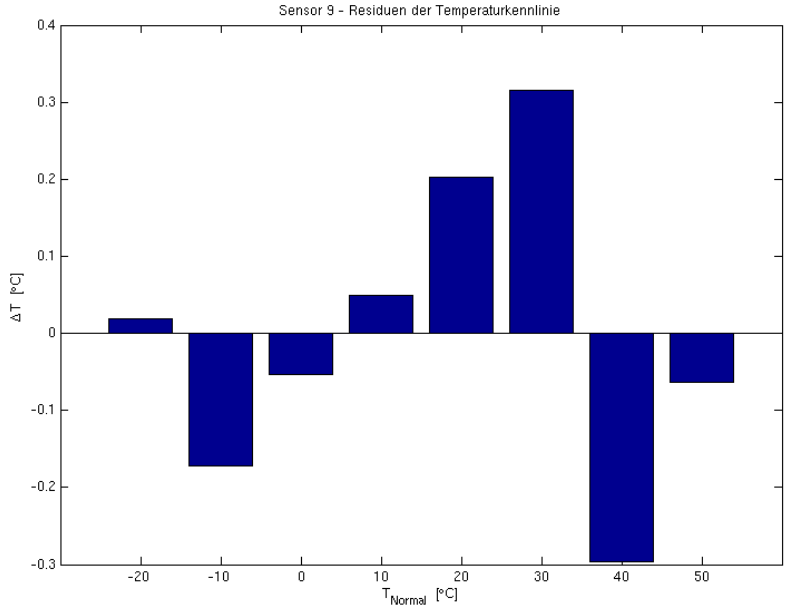


Abbildung D.12.: Residuen der Temperaturkennlinie Sensor 9

D.3. Spannungskennlinien

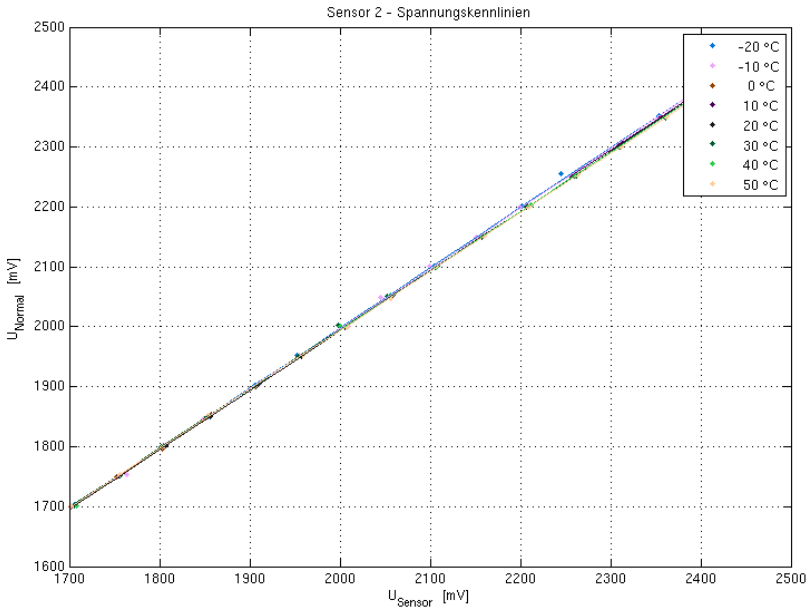


Abbildung D.13.: Spannungskennlinien Sensor 2

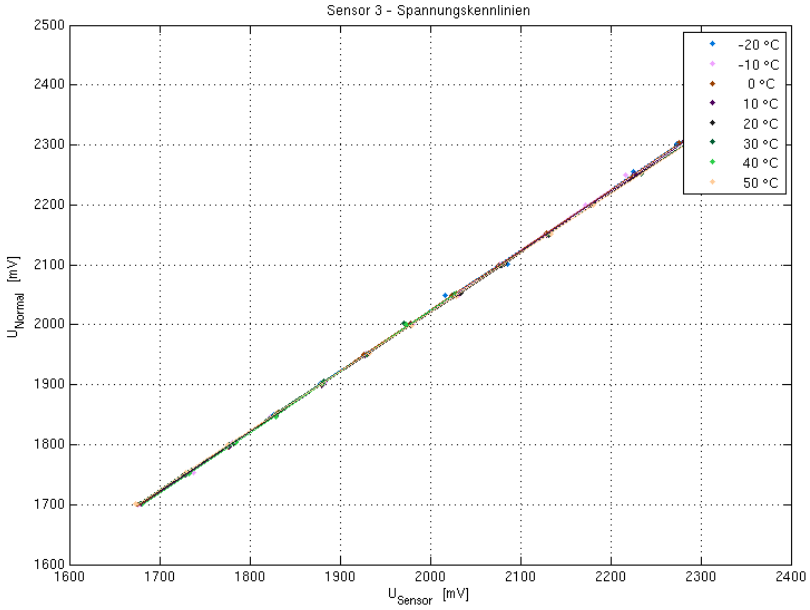


Abbildung D.14.: Spannungskennlinien Sensor 3

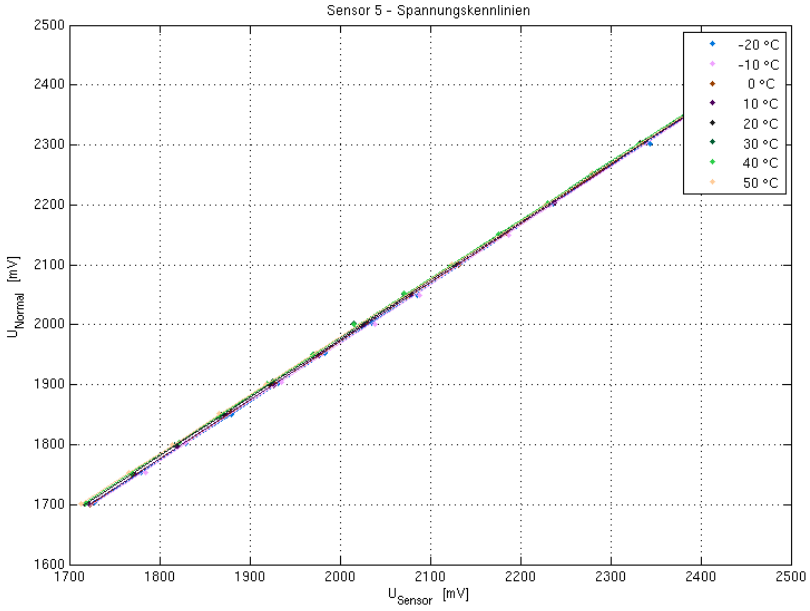


Abbildung D.15.: Spannungskennlinien Sensor 5

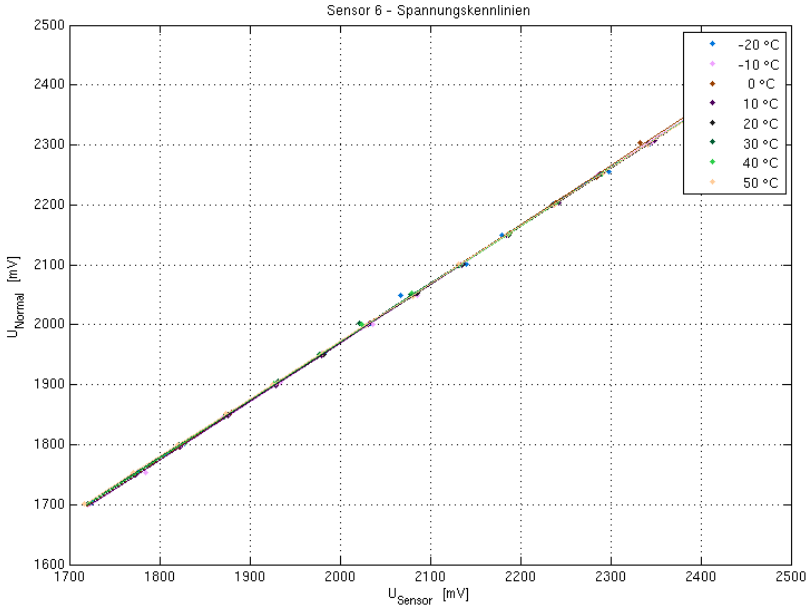


Abbildung D.16.: Spannungskennlinien Sensor 6

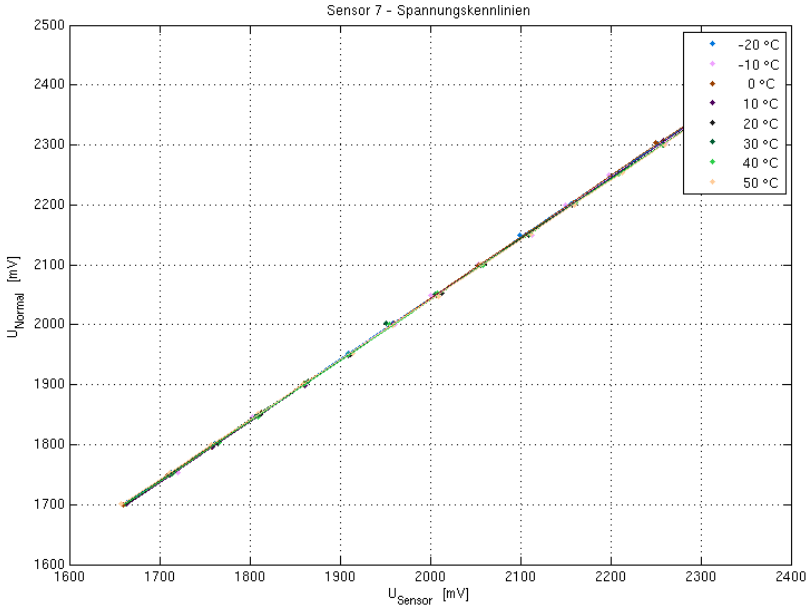


Abbildung D.17.: Spannungskennlinien Sensor 7

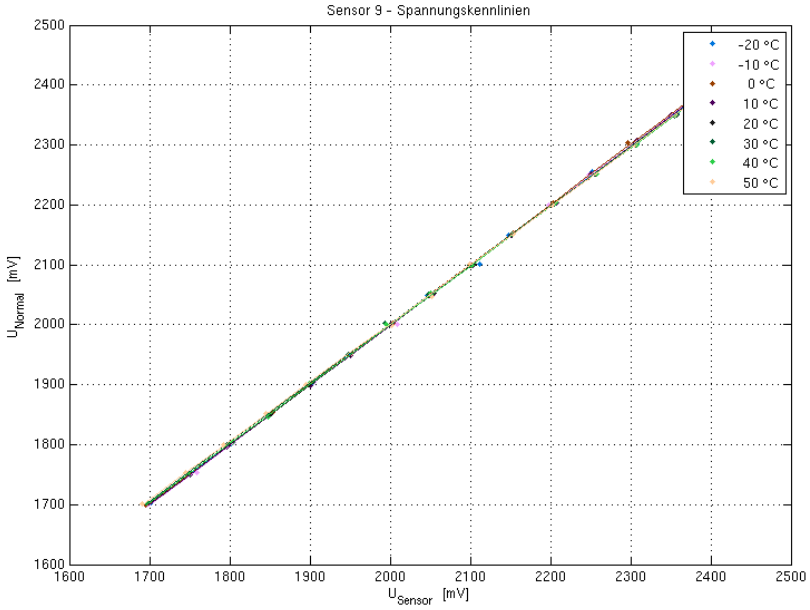


Abbildung D.18.: Spannungskennlinien Sensor 9

D.4. Residuen der Spannungskennlinien

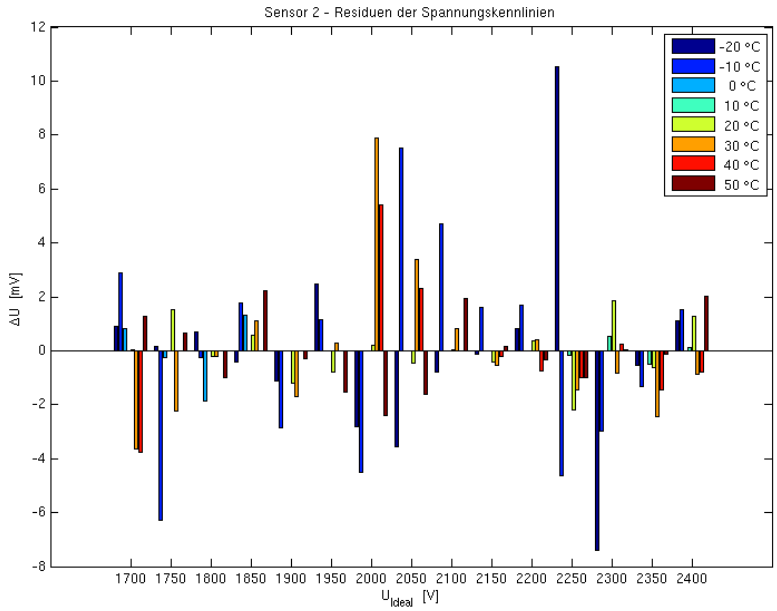


Abbildung D.19.: Residuen der Spannungskennlinien Sensor 2

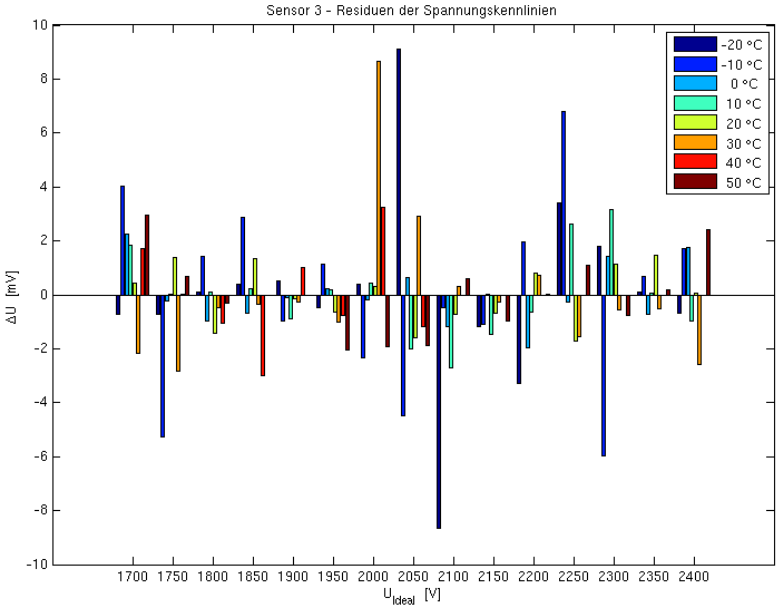


Abbildung D.20.: Residuen der Spannungskennlinien Sensor 3

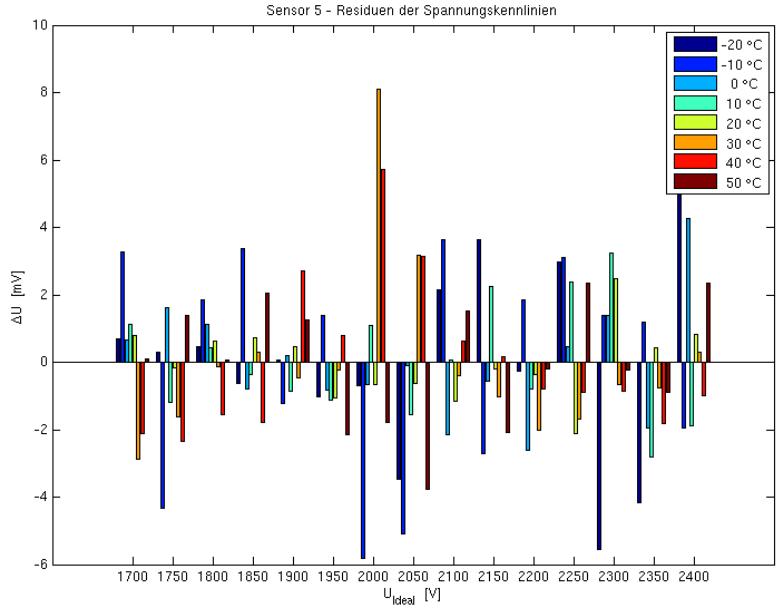


Abbildung D.21.: Residuen der Spannungskennlinien Sensor 5

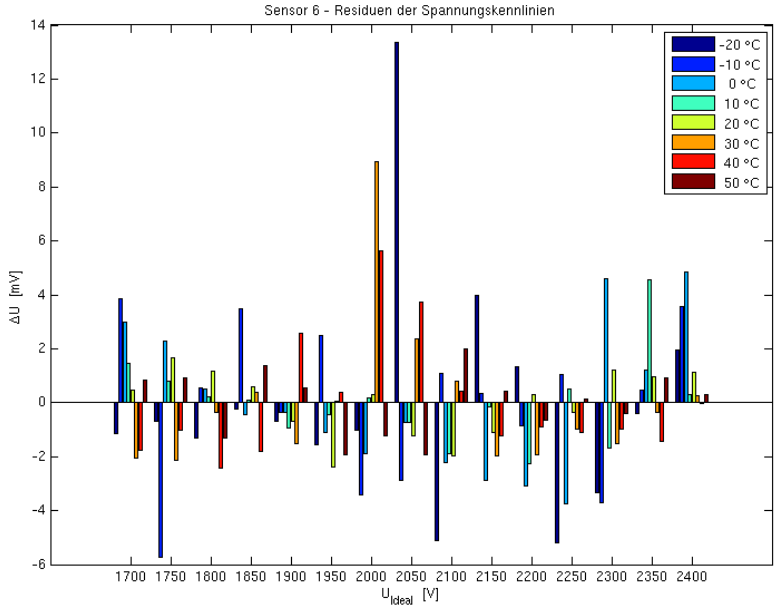


Abbildung D.22.: Residuen der Spannungskennlinien Sensor 6

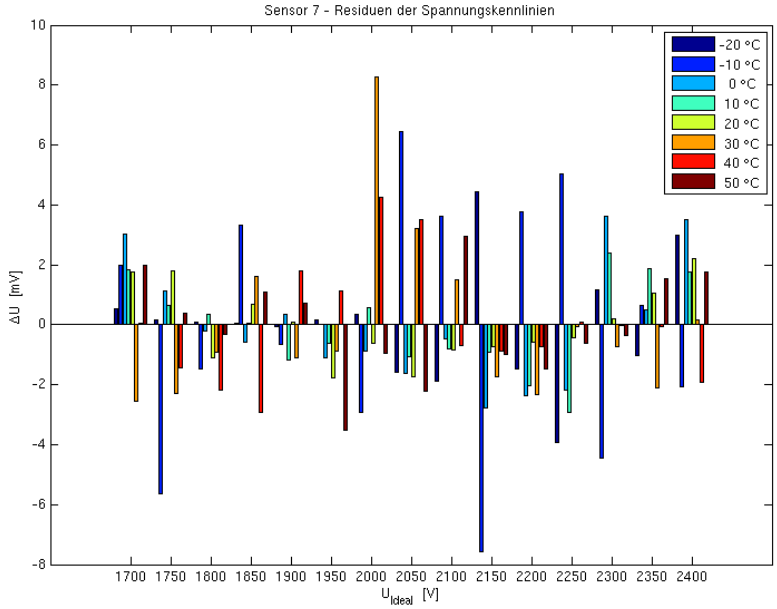


Abbildung D.23.: Residuen der Spannungskennlinien Sensor 7

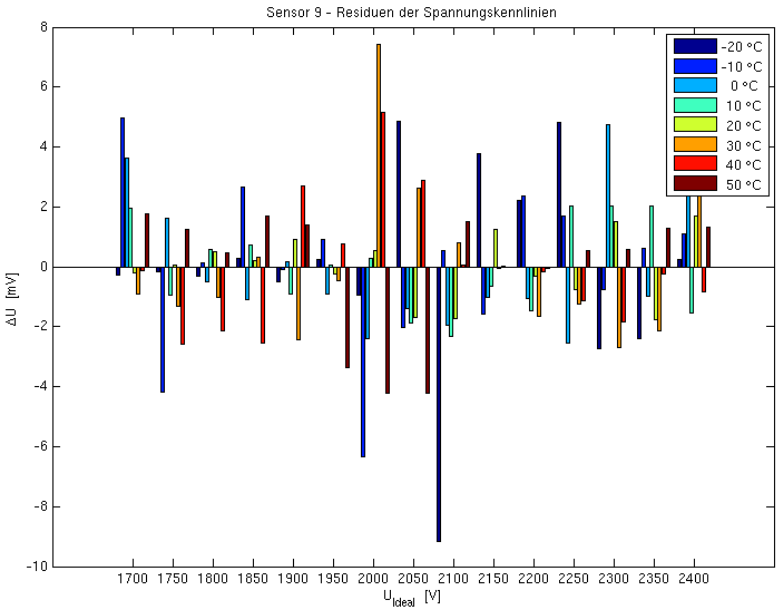


Abbildung D.24.: Residuen der Spannungskennlinien Sensor 9

D.5. Temperaturabhängigkeit der Regressionskoeffizienten (Steigung)

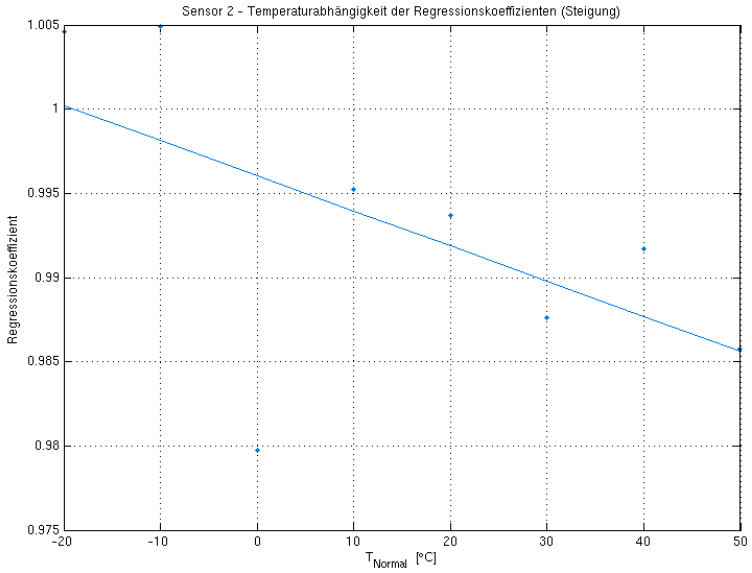


Abbildung D.25.: Temperaturabhängigkeit der Regressionskoeffizienten (Steigung) Sensor 2

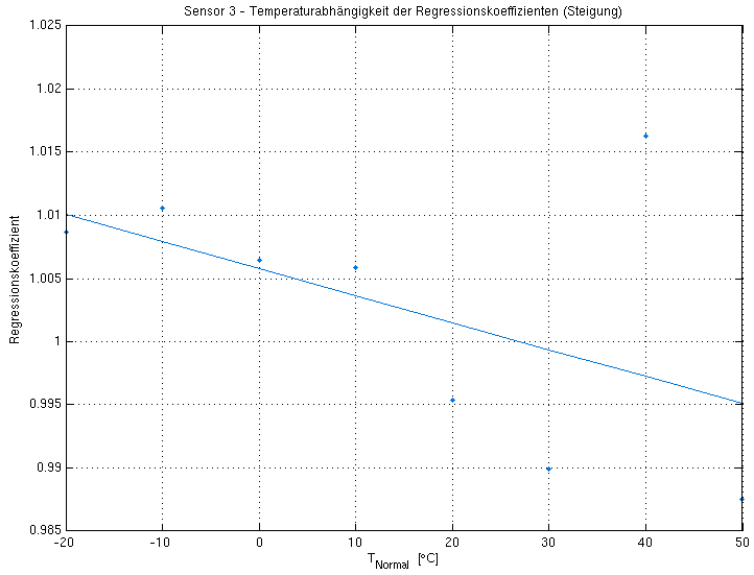


Abbildung D.26.: Temperaturabhängigkeit der Regressionskoeffizienten (Steigung) Sensor 3

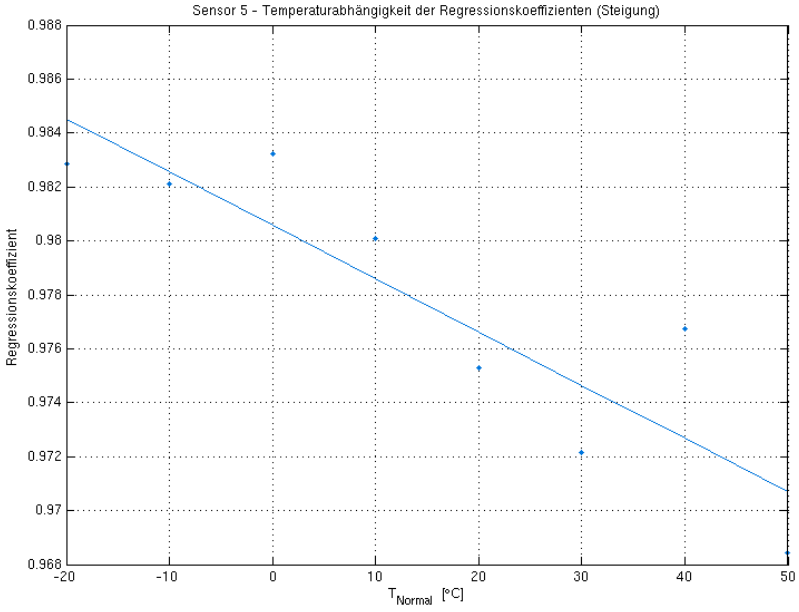


Abbildung D.27.: Temperaturabhängigkeit der Regressionskoeffizienten (Steigung) Sensor 5

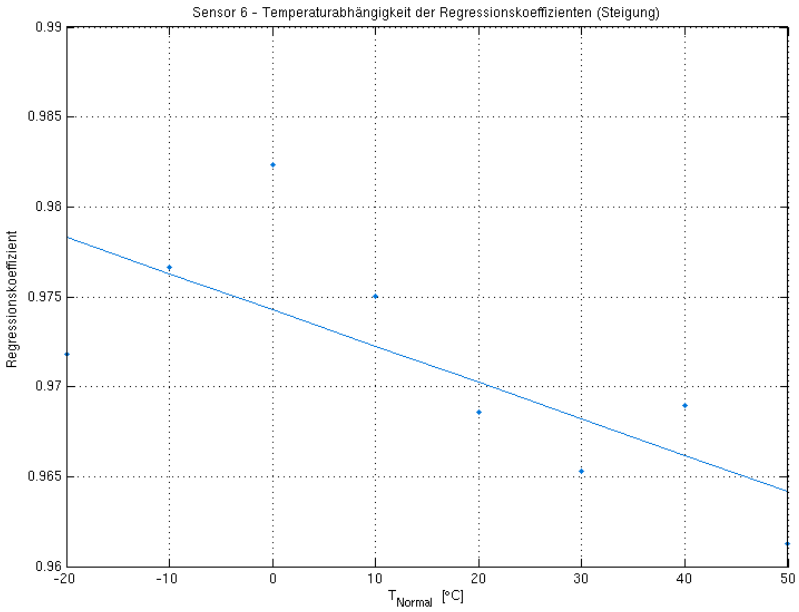


Abbildung D.28.: Temperaturabhängigkeit der Regressionskoeffizienten (Steigung) Sensor 6

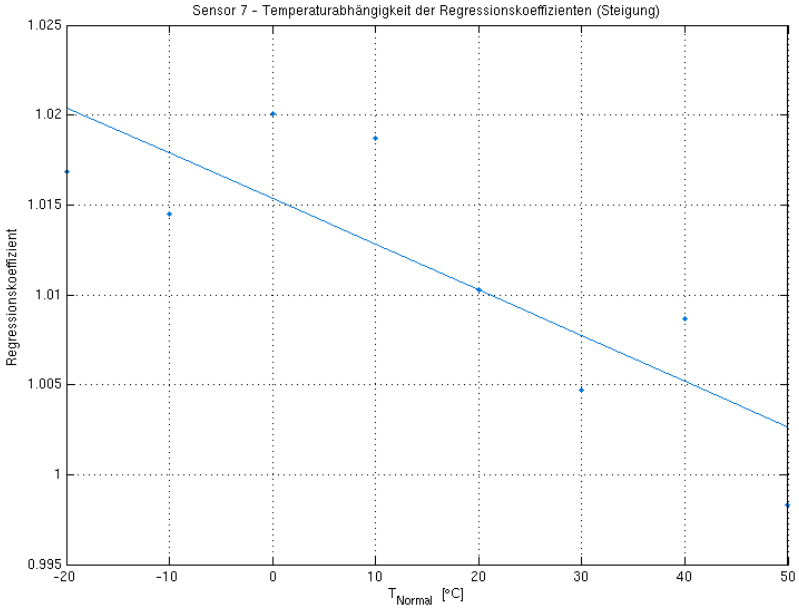


Abbildung D.29.: Temperaturabhängigkeit der Regressionskoeffizienten (Steigung) Sensor 7

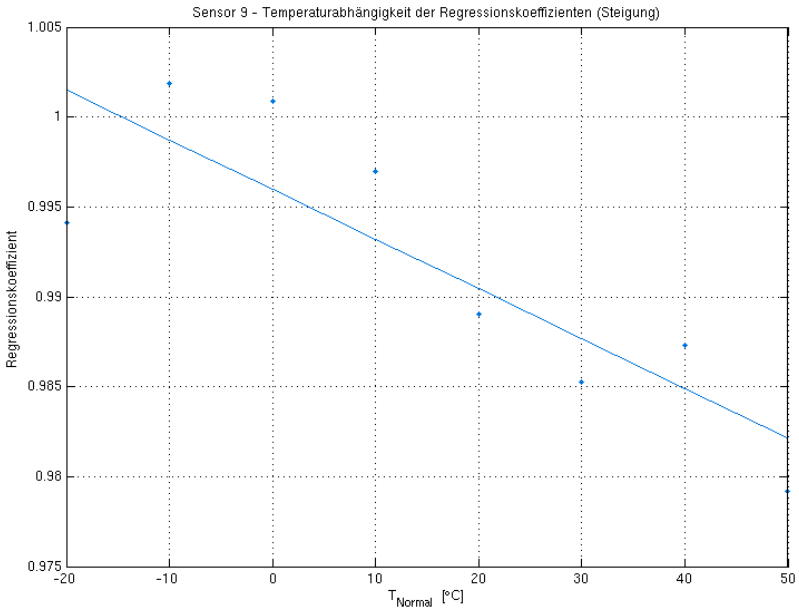


Abbildung D.30.: Temperaturabhängigkeit der Regressionskoeffizienten (Steigung) Sensor 9

D.6. Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Steigung)

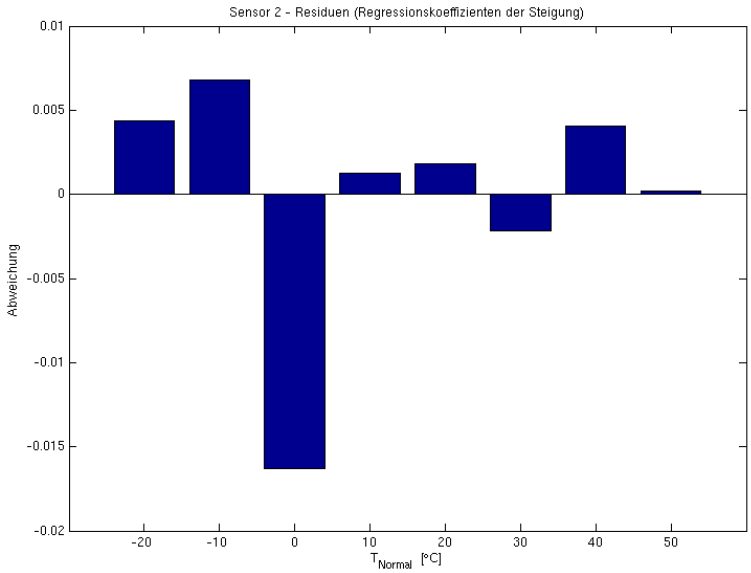


Abbildung D.31.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Steigung) Sensor 2

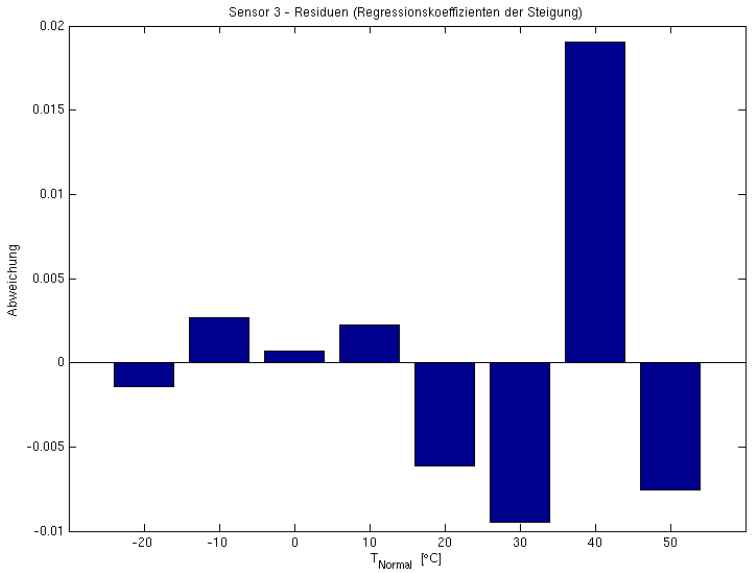


Abbildung D.32.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Steigung) Sensor 3

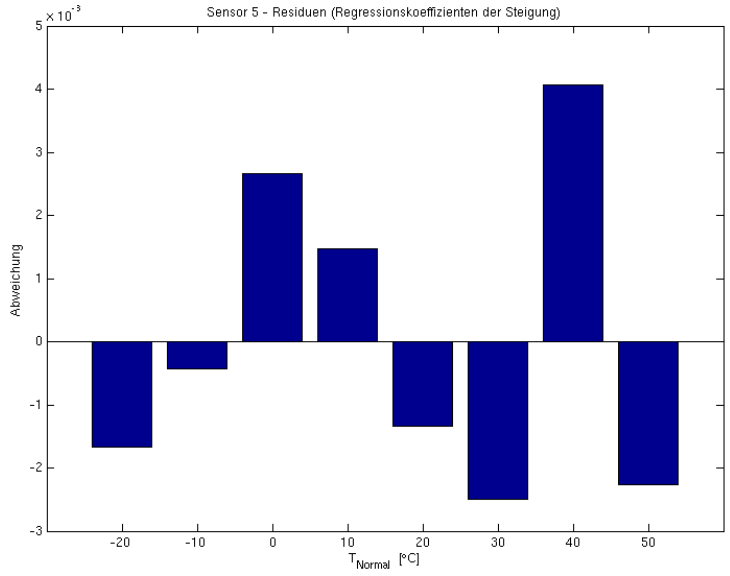


Abbildung D.33.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Steigung) Sensor 5

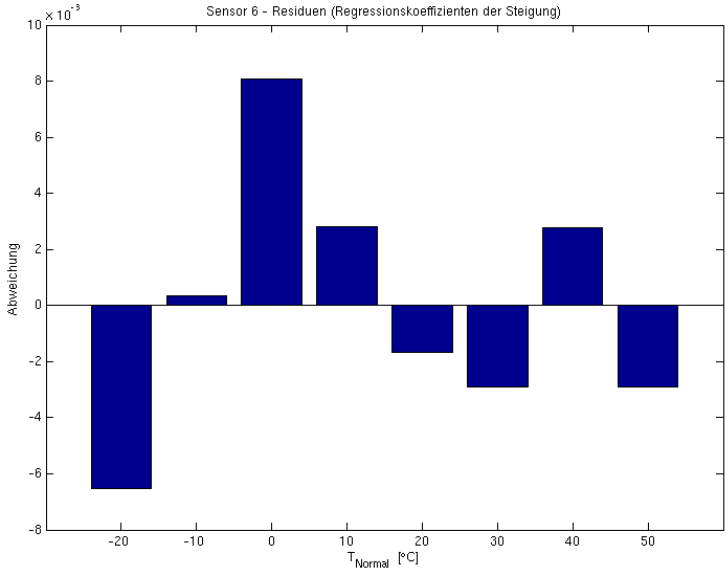


Abbildung D.34.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Steigung) Sensor 6

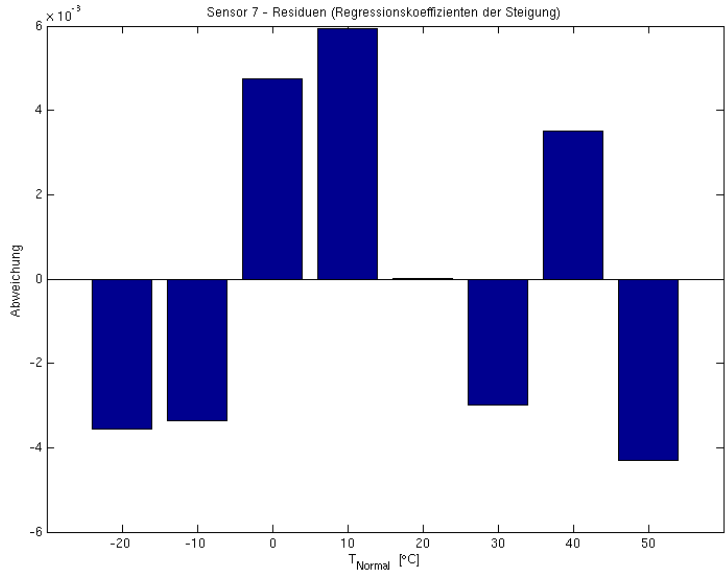


Abbildung D.35.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Steigung) Sensor 7

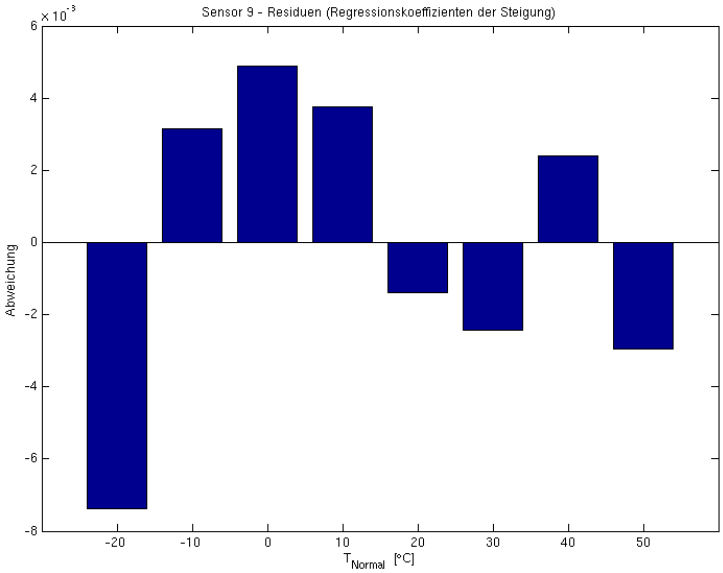


Abbildung D.36.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Steigung) Sensor 9

D.7. Temperaturabhängigkeit der Regressionskoeffizienten (Achsenabschnitt)

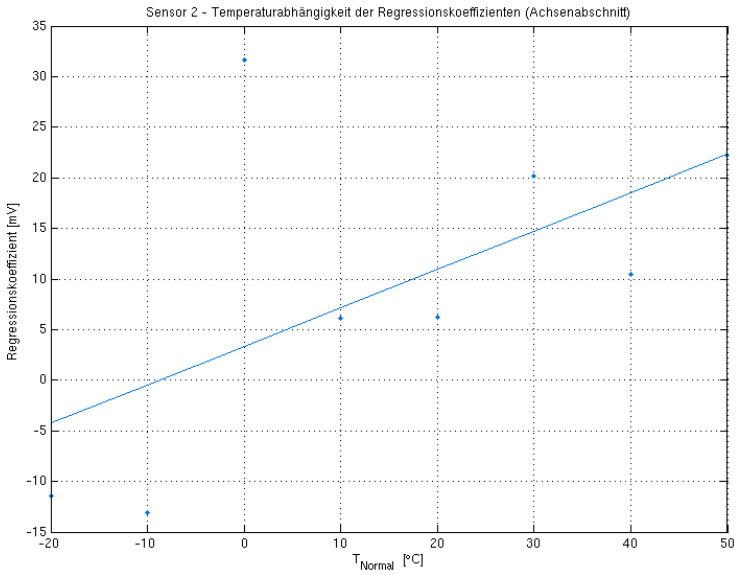


Abbildung D.37.: Temperaturabhängigkeit der Regressionskoeffizienten (Achsenabschnitt) Sensor 2

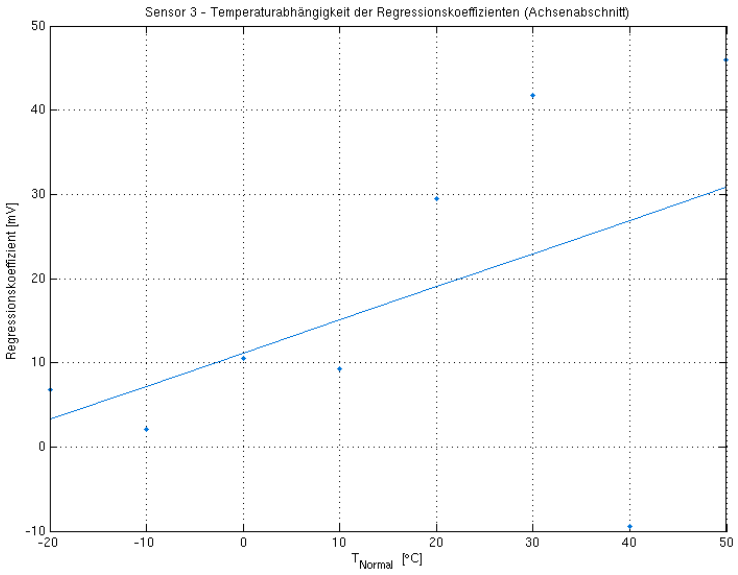


Abbildung D.38.: Temperaturabhängigkeit der Regressionskoeffizienten (Achsenabschnitt) Sensor 3

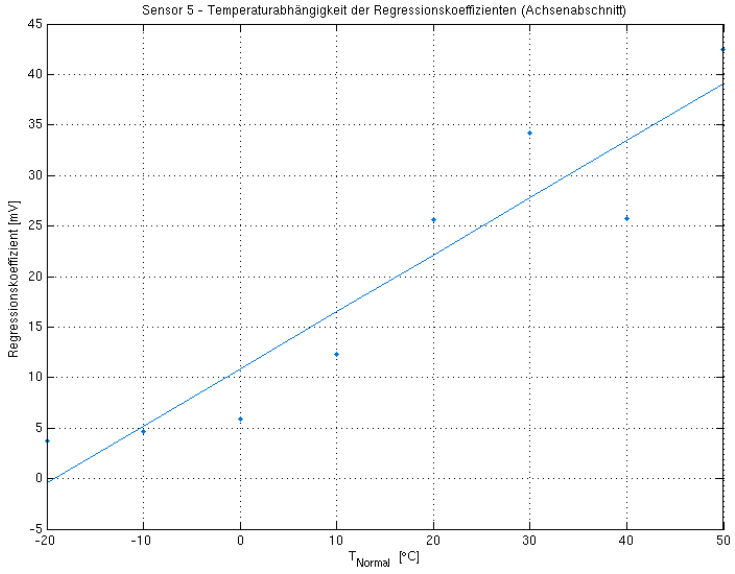


Abbildung D.39.: Temperaturabhängigkeit der Regressionskoeffizienten (Achsenabschnitt) Sensor 5

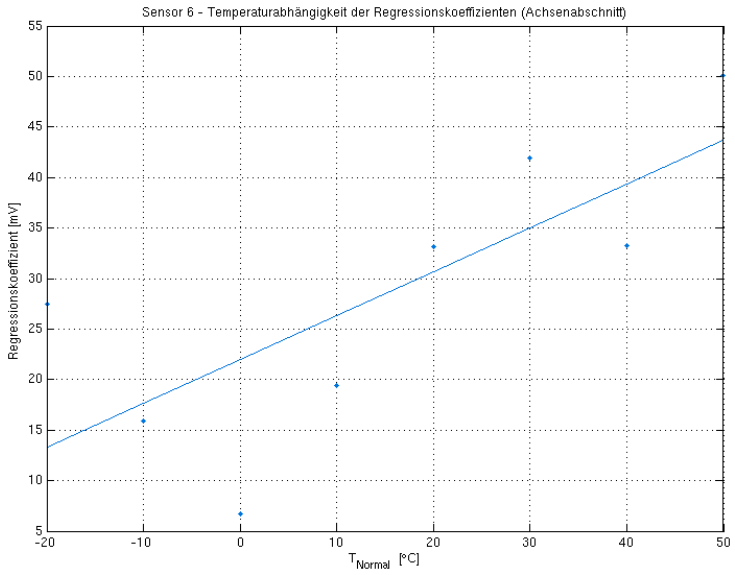


Abbildung D.40.: Temperaturabhängigkeit der Regressionskoeffizienten (Achsenabschnitt) Sensor 6

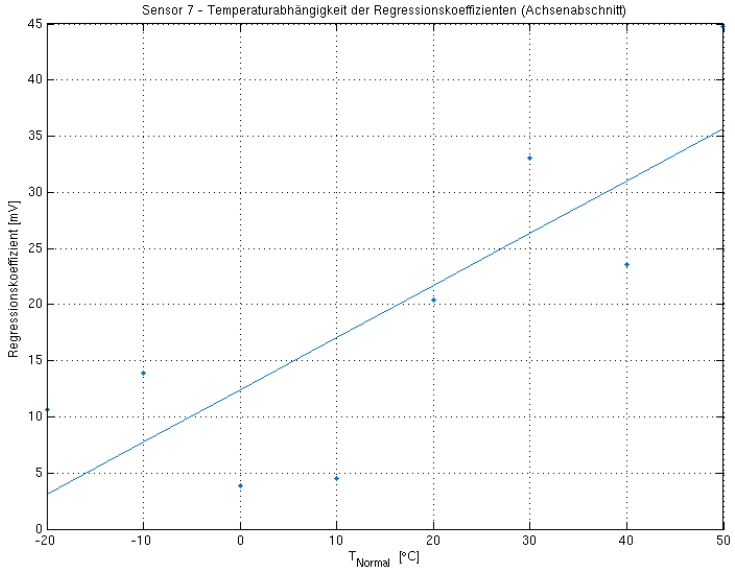


Abbildung D.41.: Temperaturabhängigkeit der Regressionskoeffizienten (Achsenabschnitt) Sensor 7

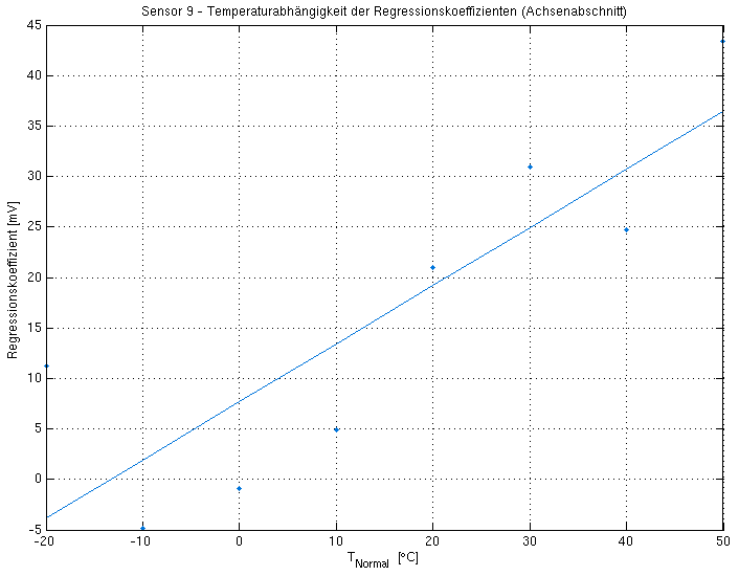


Abbildung D.42.: Temperaturabhängigkeit der Regressionskoeffizienten (Achsenabschnitt) Sensor 9

D.8. Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Achsenabschnitt)

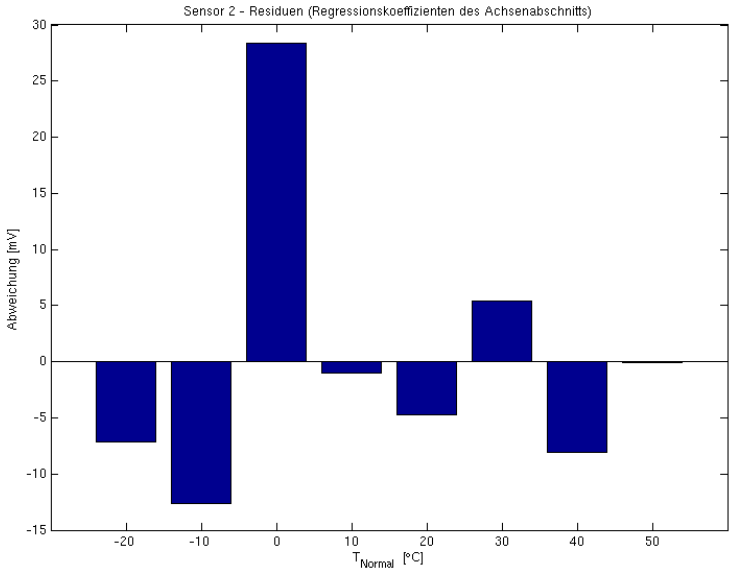


Abbildung D.43.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Achsenabschnitt) Sensor 2

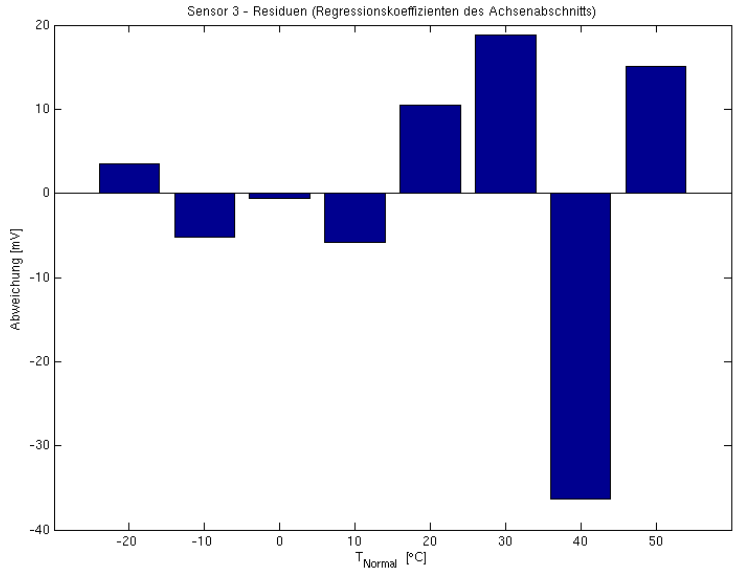


Abbildung D.44.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Achsenabschnitt) Sensor 3

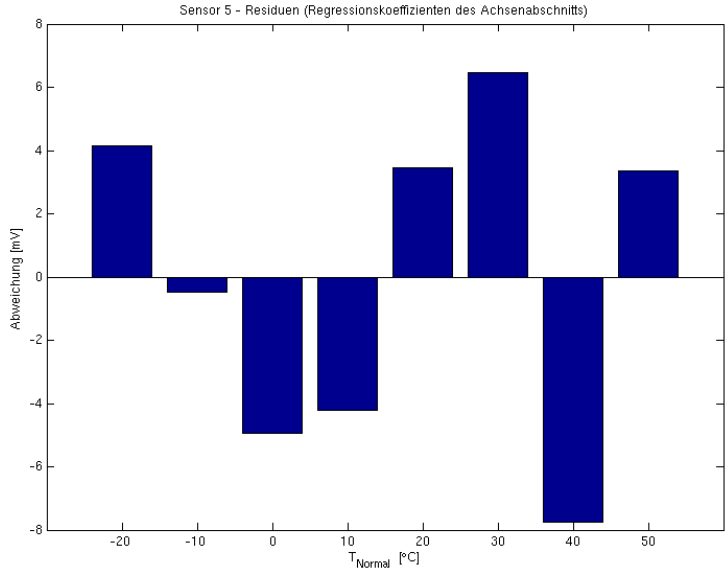


Abbildung D.45.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Achsenabschnitt) Sensor 5

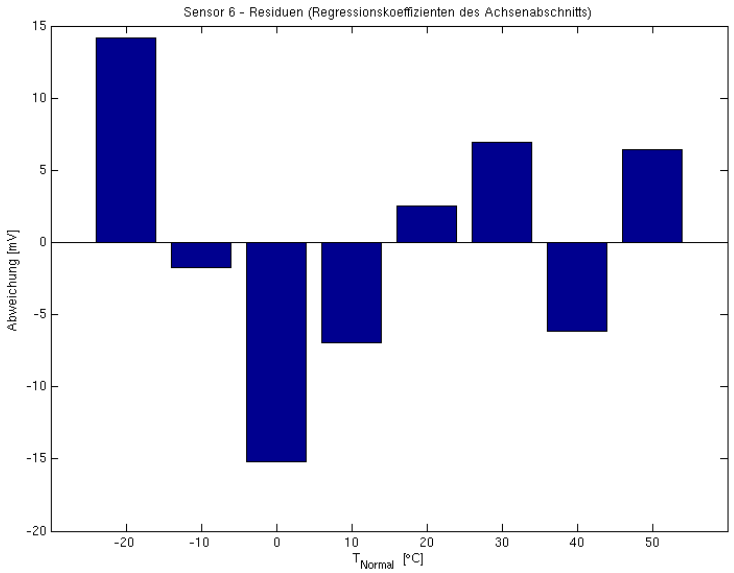


Abbildung D.46.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Achsenabschnitt) Sensor 6

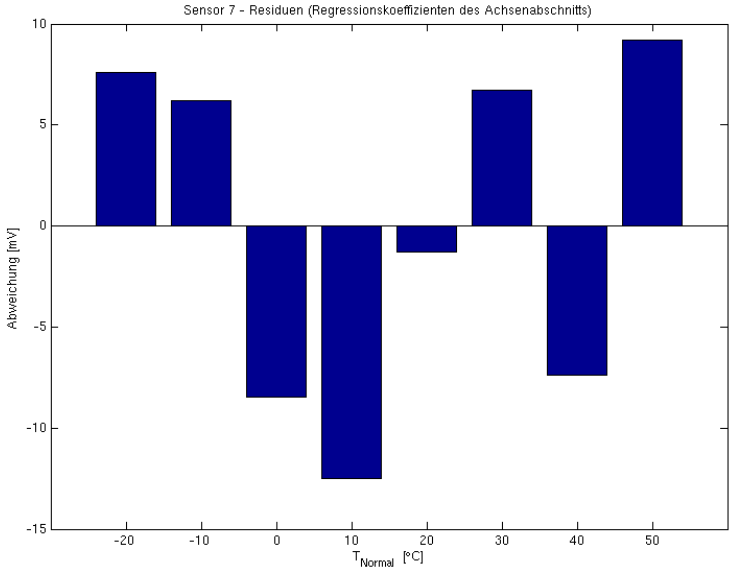


Abbildung D.47.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Achsenabschnitt) Sensor 7

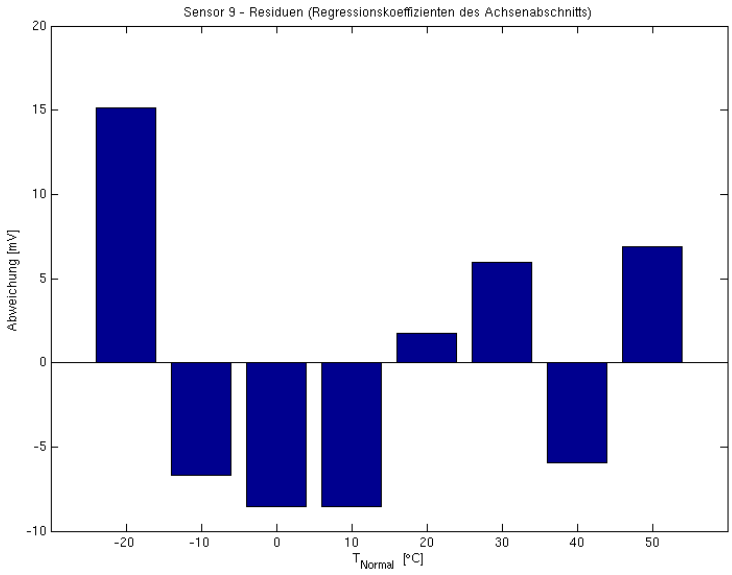


Abbildung D.48.: Residuen der Regressionskoeffizienten der Temperaturabhängigkeit (Achsenabschnitt) Sensor 9

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 4. Januar 2015	
Ort, Datum	Unterschrift