## Integrator (->zu Theoretischer Hintergrund)

Wie oben beschrieben, sind die zwei wichtigsten elektronischen Bauteile des Nephelometers eine LED und eine Photodiode (PD). Wenn das Signal, das an der PD abgegriffen wird, sehr klein ist (ein niedriger Strom), bedeutet dies, dass die PD nur wenig Licht aufnimmt. Daraus kann man schließen, dass nur wenige Partikel in der Luft innerhalb der Kammer sind, sonst würden diese das Licht der LED streuen und die PD würde einen höheren Strom ausgeben.

Die gemessene momentane Dichte des Aerosols in der Dunkelkammer, die durch diese LED-PD-Anordnung aufgenommen werden kann, ist eine Zufallsvariable mit starker Varianz. Für die Dämpfung, die mit dem Nephelometer bestimmt werden soll, ist dieses Signal zu stark verrauscht. Hier ist deutlich interessanter, was der momentane Mittelwert der gemessenen Dichte im Zeitraum bis ist, wobei man ein geeignetes finden muss. Man könnte diesen Mittelwert bilden, indem man das Ausgangssignal der PD kontinuierlich in Intervallen der Periodendauer abfragt und numerisch einen Mittelwert über Werte berechnet. Dann würde gelten:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

Da die genaue physikalische Abhängigkeit zwischen tatsächlicher Partikeldichte und gemessenem Ausgansstrom nicht bekannt ist, spielen genauen Stromwerte hierbei keine wichtige Rolle. Deshalb kommt es auf einen konstanten Faktor nicht an und bei der obigen Rechnung kann man sich den Faktor sparen, sodass man nur die Summe der gemessenen Ströme berechnet, die dieselbe Information wie der Mittelwert enthält.

Eine Alternative zu dieser digitalen Berechnung ist die Verwendung eines analogen Integrators. In der Berechnung des Mittelwerts hat dies den Effekt, dass die Zeit zwischen zwei Messungen gegen 0 geht:

In der oberen Grenze der Summe konnte die vernachlässigt werden, da . Auch hier ist aus dem oben diskutierten Grund der konstante Faktor uninteressant. Tatsächlich wird von der Integratorschaltung ein konstanter Faktor dazu gefügt, der ebenso ignoriert werden kann. Von dem Integrator wird also ein Signal nach der folgenden Formel in ein Signal transformiert:

wird hier als Integrationszeit beschrieben. Damit diese Gleichung stimmt, muss zum Zeitpunkt der Kondensator komplett entladen sein. Wie oben beschrieben, enthält dieser Ausdruck dieselbe Information wie der Mittelwert des Signals (tatsächlich mehr Information als ein mit endlich großem berechneter Mittelwert).

Neben der minimalen Zeit hat der analoge Integrator noch einen weiteren großen Vorteil gegenüber der digitalen Alternative. Für die digitale Berechnung der Summe müssen die analogen Signale der PD mit einem Analog-Digital-Umsetzer transformiert werden. Hier kann es bei geringen Auflösungen zu großen Problemen kommen. Ist beispielsweise das Ausgangssignal der PD im Normalfall so gering, dass der ADU als Spannungswert immer 0 registriert, funktioniert die Schaltung nicht. Mit einem Integrator kann dieses Problem behoben werden, da die Integrationszeit so gewählt werden kann, dass im Normalfall im optimalen Bereich des ADUs liegt.

## X.X Elektronische Umsetzung des Nephelometers

Aus den oben beschriebenen Gründen haben wir uns in unserer Schaltung für einen analogen Integrator entschieden. Die benutzte Schaltung stammt aus [o1], der Schaltplan des Nephelometers kann in Anhang A1 gefunden werden.

Die Stromversorgung für kommt von einem Arduino Nano (der über USB von einem Computer versorgt wird), an den auch die Anschlüsse „Output“ (Spannungswert am Integrator), „Reset“ (Transistorsteuerung zum Entladen des Integrationskondensators) und „Pulse“ (Transistorsteuerung zur Bedienung der LED) angeschlossen sind.

Im Schaltplan ist auf der linken Seite die LED-Schaltung zu sehen. Hier musste anhand der LED- und Transistorkennlinien ein geeigneter Wert für den Vorwiderstand gefunden werden. Die Überlegungen hierzu werden im nächsten Abschnitt diskutiert.

In der Mitte, im oberen Bereich des Schaltplans, ist ein Festspannungsregler zu sehen. Dieser wandelt die 0 V / +5 V Versorgungsspannung des Arduinos in eine -5 V / 0 V / +5 V Spannung um, mit der der Operationsverstärker in der Integratorschaltung betrieben wird. Zwei 100 nF Kondensatoren werden zur Spannungsstabilisierung verwendet.

Rechts neben dieser Spannungsversorgung ist die Photodiode, welche im Gehäuse mit einem Transimpedanzwandler verschaltet ist. Unter der PD ist der Rest der Integratorschaltung zu sehen. Der hier verwendete Kondensator legt die Zeitkonstante des Integrators fest. Da die Zeit zwischen dem periodischen Entleeren des Kondensators am Arduino eingestellt werden kann, ist der genaue Wert der Zeitkonstante nicht von großer Bedeutung. Wichtig ist jedoch, dass der Integrator beim Integrieren nicht an seine Grenze kommt, dann wäre der Kondensator bei jedem Abfragen auf 5 V aufgeladen und das Signal enthält keine Information über die Aerosoldichte mehr.

Dass dies nicht passiert, kann durch zwei verschiedene Maßnahmen sichergestellt werden: eine größere Zeitkonstante oder eine kleine Integrationszeit. Da die Integrationszeit nicht zu klein sein darf, sonst würde der Integrator seinen Zweck der Glättung der Messungen nicht erreichen, ist es sinnvoll, einen großen Kondensator zu verwenden, um eine ausreichende Zeitkonstante zu erhalten. In unserem Fall haben wir uns für einen 100 nF Kondensator entschieden.

Der Widerstand, der sich auf dem Schaltplan unter dem Kondensator befindet und mit dem Transistor verschaltet ist, dient der Strombegrenzung durch den Transistor. Auch hier ist der genaue Wert nicht ausschlaggebend, es muss jedoch darauf geachtet werden, dass er die Funktion der Strombegrenzung erfüllt, aber nicht zu groß ist. Bei zu einem hohen Widerstand ist der Entladevorgang des Kondensators sehr langsam, was die Gesamtgeschwindigkeit des Systems beeinträchtigt, da der Kondensator bei jeder Messung vor der Integration entladen werden muss. Ein 100 Ω Widerstand erfüllt beide Bedingungen.

Es müssen also noch zwei verschiedene Werte gefunden werden: der Widerstand zur Strombegrenzung in der LED-Schaltung, sowie eine geeignete Integrationszeit. Diesen beiden Fragen widmen sich die nächsten zwei Unterkapitel.

## X.X Vorwiderstand und Arbeitspunktanalyse in der LED-Schaltung

Der hier betrachtete Ausschnitt der Schaltung ist in Abbildung 1 dargestellt.

A picture containing laptop, white

Description automatically generated

Abbildung : Die LED-Schaltung. Es muss ein geeigneter Wert für gefunden werden.

Es soll der Widerstand so bestimmt werden, dass der in dem Zweig fließende Strom knapp unter dem Maximalstrom der LED liegt, damit diese mit möglichst hoher Helligkeit leuchtet, aber nicht beschädigt wird. Der Widerstand lässt sich mit der folgenden Formel berechnet:

Hier ist  V die Versorgungsspannung und  mA ist der zulässige Maximalstrom durch die LED, der aus [o2] bestimmt wurde. Bei und handelt es sich um die Spannungen, die sich im Arbeitspunkt an der LED bzw. am Transistor (zwischen Drain und Source), einstellen. Bei dem hier betrachteten Arbeitspunkt ist die LED an, es gilt also . Da sich die Spannungen und aus den in den Datenblättern [o2] und [o3] zur Verfügung gestellten Kennlinien nicht genau abgelesen lassen, haben wir uns für eine SPICE-Simulation der Bauteile entschieden, um die Arbeitspunkte genau zu bestimmen.

Die SPICE-Modelle der LED und des Transistors sind [o4] und [o5] entnommen und der entsprechende Code ist im Anhang A2 hinterlegt. Wir haben die grafische SPICE-Umgebung LTspice verwendet. Die Schaltungen zur Bestimmung der Spannungen und sind in Abbildung 2 (links und mittig) dargestellt. Die im oberen Bereich der Abbildung stehenden SPICE-Befehle gehören jeweils zur darunterliegenden Schaltung. Alle drei Befehle wurden unabhängig voneinander ausgeführt und beschreiben unterschiedliche Simulationen.

A screenshot of a cell phone

Description automatically generated

Abbildung : SPICE-Simulationsschaltungen zur Bestimmung der genauen Kennlinien von LED und FET. Links: Simulationsschaltung der LED, hier wird die Abhängigkeit Diodenspannung von dem Diodenstrom aufgenommen. Mitte: Simulationsschaltung des MOSFETs mit , hier wird die Abhängigkeit des Drain-Stroms von aufgenommen. Rechts: Hier wird dieselbe Abhängigkeit wie im mittleren Bild untersucht, jedoch bei , also im „Aus“-Arbeitspunkt.

Um die Spannung zu bestimmen, die die LED bei 100 mA aufnimmt, wird der Strom durch die LED von 1 nA bis 1 A in logarithmischen Abständen variiert und die Resultierende Diodenspannung wird abgegriffen (Abb. 2, links). Das Ergebnis der Simulation ist in Abbildung 3 dargestellt. Natürlich hätte es in diesem Fall auch ausgereicht, die Schaltung nur mit dem konstanten Strom 100 mA zu simulieren. Stattdessen wird hier die gesamte U-I-Kennlinie erstellt, aus der abgelesen werden kann, dass sich bei der Diode im „Ein“-Arbeitspunkt eine Spannung von 1,495 V einstellt.

Anschließend muss die Spannung, die sich im „Ein“-Arbeitspunkt am Transistor zwischen Drain und Source einstellt, bestimmt werden. Hier kann man sehr ähnlich verfahren, die benutzte Simulationsschaltung ist mittig in Abbildung 2 dargestellt. wird konstant bei 5 V gehalten und wird von 0 V bis 5 V variiert. Das Ergebnis dieser Simulation ist die Ausgangskennlinie des MOSFETs, dargestellt in Abbildung 4. Man kann sehr leicht ablesen, dass sich am Transistor im Arbeitspunkt eine Spannung von einstellt.

Somit sind alle notwendigen Werte bestimmt und es lässt sich der optimale Vorwiderstand berechnen:

Man wählt also den nächstgrößeren verfügbaren Widerstand aus der gewählten Widerstandsreihe, in diesem Fall ist der nächste Wert 33 Ω.

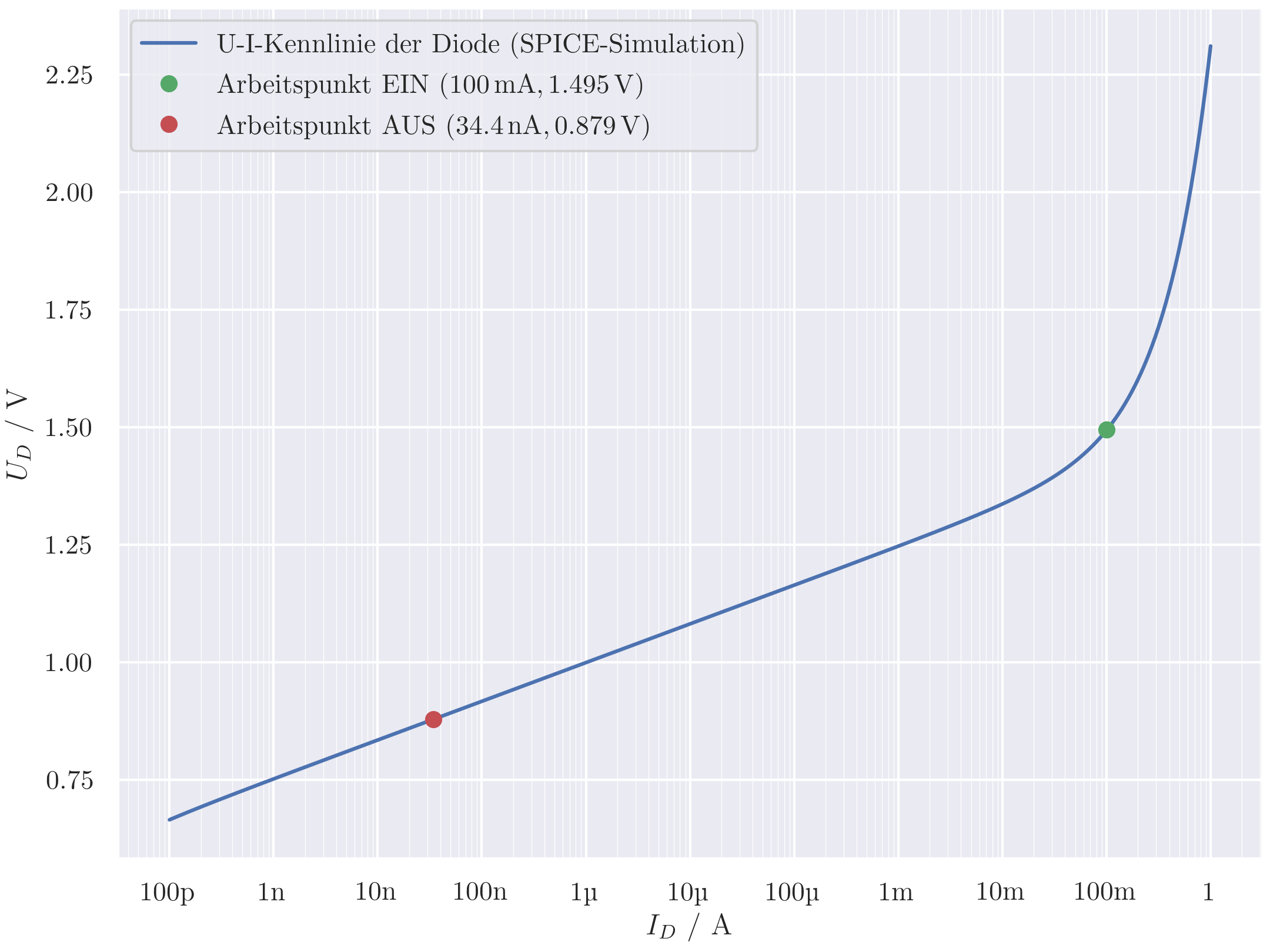


Abbildung : U-I-Kennlinie der LED. Der „Ein“-Arbeitspunkt kann direkt abgelesen werden: Bei dem soll-Strom stellt sich eine Spannung von ein. Der „Aus“-Arbeitspunkt wird weiter unten behandelt.

A screenshot of a cell phone

Description automatically generated

Abbildung 4: Ausgangskennlinie des Transistors (). Der „Ein“-Arbeitspunkt kann wieder direkt abgelesen werden: Bei dem soll-Strom stellt sich eine Spannung von ein. Der „Aus“-Arbeitspunkt, sowie die Arbeitsgerade wird weiter unten behandelt.

Bei der Simulation der U-I-Kennlinie der LED fällt eine Eigenart in auf, die in Abbildung 3 zu sehen ist. Selbst bei sehr niedrigen Strömen (nA-Bereich) fällt noch eine große Spannung an der LED ab. Auf den Betrieb im „Ein“-Modus ( V) hat dies keinen Einfluss, da hier der Arbeitspunkt durch den Vorwiderstand festgelegt wird. Man würde erwarten, dass der Arbeitspunkt „Aus“ (also  V) sehr simpel ist, nämlich  
 A. Die in Abbildung 3 erkennbare starke Nichtlinearität der LED im unteren Strombereich verkompliziert die Realität jedoch.

Es ist zu erwarten, dass der Transistor selbst bei  V noch einen sehr geringen Strom durchlässt, wenn eine Drain-Source-Spannung von  V anliegt. Tatsächlich scheint der Transistor bei einer Gate-Source-Spannung von 0 V eine lineare Abhängigkeit des Stroms von aufzuweisen, sodass bei  V statt des gewünschten Nullstroms ein Strom von  nA fließt.

Dieses Ergebnis kommt aus einer weiteren Simulation, in der  V konstant gehalten wurde und von 0 V bis 5 V erhöht wurde (Abb. 2, rechts). In dem verwendeten SPICE-Modell ist der Drain-Strom bei  V (Sperrbereich) linear von abhängig. Da es sich hier um einen Edge-Case (Ströme im nA-Bereich) handelt, kann es gut sein, dass das Modell die Realität nicht akkurat wiederspiegelt. Dasselbe gilt für das SPICE-Modell der LED. Da diese minuziösen Ströme aber dennoch große Spannungsabfälle an der LED zu bewirken scheinen, ist die Analyse nicht unbegründet.

Wie in Abbildung 3 zu erkennen ist, bedeutet der eben bestimmte Strom  nA, dass eine erhebliche Spannung an der LED abfallen würde, die somit nicht mehr am Transistor abfallen kann. Die Ströme sind hier so gering, dass die Spannung am Widerstand als vernachlässigbar klein angenommen werden kann, es gilt also näherungsweise  
. Diese Änderung der Spannung am Transistor führt wiederum dazu, dass der Strom durch den Transistor auch abfällt, was wieder die Spannung an der LED beeinflusst. Dieses dynamische System lässt sich leicht iterativ lösen, mit dem Anfangsschätzwert  V und  A. Die Iteration ist in Tabelle 1 zu sehen, eine graphische Darstellung ist in Abbildung 5 gezeigt. Die Iteration konvergiert sehr schnell und es stellt sich ein „Aus“-Arbeitspunkt von ( ein. Dieser Arbeitspunkt ist auch in Abbildung 3 und Abbildung 4 eingezeichnet. In Abbildung 4 ist auch die Arbeitsgerade eingezeichnet, die beim Ein- und Ausschalten des Transistors passiert wird.

Um das Schaltverhalten mit dem berechneten Widerstandswertes von zu überprüfen, wird noch eine letzte Simulation der Gesamtschaltung durchgeführt, wobei (Pulse) nicht, wie in der praktischen Umsetzung der Schaltung, binär zwischen 0 V und 5 V geschaltet wird, sondern kontinuierlich geändert wird, damit sich die glatte Kurve bildet, die in Abbildung 7 zu sehen ist. Die verwendete Simulationsschaltung ist in Abbildung 6 dargestellt.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iterationsschritt | 0 | 1 | 2 | 3 |
|  | 0 | 41,7 | 34,3 | 34,4 |
|  | 0 | 0,886 | 0,879 | 0,879 |
|  | 5 | 4,11 | 4,12 | 4,12 |

Tabelle 1: Der „Aus“-Arbeitspunkt wird durch eine numerische Iteration gefunden. Die Iteration wird so durchgeführt, dass erst aus , mithilfe der linearen Kennlinie , die im Sperrbereich des Transistors gültig ist, der Strom bestimmt wird. Aus wird dann mithilfe der in Abbildung 3 gezeichneten Kennlinie der LED die Spannung , die an der LED abfällt bestimmt. Aus dem Verhältnis wird der neue Wert für bestimmt und die Iteration fängt von vorne an.

A close up of a piece of paper

Description automatically generated

Abbildung 5: Dieselbe Iteration aus Tabelle 1 in graphischer Darstellung. Die blaue Kurve zeigt , die Umkehrfunktion der in Abbildung 3 dargestellten U-I-Kurve der LED. Hierbei wird verwendet, dass gilt. Die grüne Kurve zeigt das lineare Verhältnis , das beim Transistor im Fall gilt. Der rechte Graph zeigt eine Vergrößerung des liken im Bereich um den „Aus“-Arbeitspunkt. Die Iteration hat die graphische Interpretation, den Schnittpunkt der zwei Gleichungen zu finden.

A screenshot of a cell phone

Description automatically generated

Abbildung 6: Simulationsschaltung mit der der Graph in Abbildung 7 erstellt wurde.

Wie zu sehen ist, erreicht der Diodenstrom nicht den gewünschten Wert von 100 mA, sondern nur knapp über 90 mA. Dies liegt daran, dass statt dem berechneten Widerstandswert von 29,25 Ω ein 33 Ω Widerstand verwendet wird. Unsere Tests haben gezeigt, dass der so erreichte Strom durch die LED für unseren Anwendungszweck ausreichend ist.

A picture containing sitting, white, man

Description automatically generated

Abbildung 7: Schaltverhalten mit bei .

## X.X Auswahl einer geeigneten Integrationszeit

Wie bereits erwähnt, darf die Integrationszeit weder zu klein noch zu groß sein. Wird zu lange integriert, tritt eine Sättigung auf. Der Kondensator lädt sich auf 5 V auf und eine genaue Messung ist nicht mehr möglich. Wird zu kurz integriert, sinkt das Detail der Messergebnisse. Der verwendete Arduino unterstützt nur 1024 diskrete Spannungswerte; bei kleiner Integrationszeit liegen alle Messungen im unteren Wertebereich und die Auflösung kommt an ihre Grenzen, es kann nicht mehr zwischen verschiedenen Werten differenziert werden. Ist die Integrationszeit beispielsweise so klein, dass alle im Normalfall auftretenden Messungen im unteren Viertel des Messbereichs liegen, sind nur noch 256 diskrete Spannungswerte unterscheidbar (die oberen 768 treten nie auf). Eine optimale Integrationszeit nutzt also den kompletten Wertebereich (0 V bis 5 V) so weit wie möglich aus, ohne dass im normalen Betrieb eine Sättigung an der oberen Spannungsgrenze auftritt.

In Abbildung 8 ist eine Messung dargestellt, bei der 6 verschiedene Integrationszeiten miteinander verglichen werden. Zur Referenz ist auch noch eine Messung aufgetragen, bei der die LED nicht eingeschaltet wurde (dunkelblaue Kurve). Mit dieser Messung kann das konstante Störsignal abgeschätzt werden. Der Arduino-Code für diese Messung ist in Anhang A3 hinterlegt. Das Nephelometer wurde in dem konstruierten Kasten platziert und die Nebelmaschine wurde etwa 70 s nach Beginn der Messung eingeschaltet. Bei etwa  s wurde die Nebelmaschine wieder ausgeschaltet und die Filteranlage eingeschaltet. Der Peak, der zu diesem Zeitpunkt zu sehen ist, ist durch die temporäre Verdichtung des Nebels innerhalb der Dunkelkammer entstanden, hervorgerufen durch Turbolenzen, die vom Sog der Filteranlage ausgingen.

A close up of a map

Description automatically generated

Abbildung 8: Testmessung zur Bestimmung der optimalen Integrationszeit . Diese Abbildung stellt die unveränderte Ausgabe des Programms in Anhang A3 dar und wurde mit dem Arduino Serial Plotter erstellt.

Es ist zu erkennen, dass die oberen beiden Integrationszeiten () zu groß sind. Bei   
=  ms (rote Kurve) ist die Kurve bereits vor dem oben genannten Peak gesättigt. Bei   
 ms (grüne Kurve) kommt die Kurve bei dem Peak an ihre Grenzen und es ist hier bereits weniger Detail zu erkennen als in den unteren Kurven. Die naheliegende Wahl ist deshalb  ms (gelbe Kurve), da wie oben erläutert der verfügbare Wertebereich so weit wie möglich ausgenutzt werden soll. In dem Programm für das Nephelometer, hinterlegt in Anhang A5, wird deshalb auch  ms einprogrammiert.

Der bei allen Messungen zu erkennende Konstantwert entsteht dadurch, dass nicht die gesamte Strahlung der LED vom Lichtfang eingefangen wird, sondern ein erheblicher Teil auch ohne in der Luft gelöste Partikel zur PD gelangt. Dieses Phänomen schränkt den verwendbaren Werteberich und damit die effektive Auflösung weiter ein. Bei  ms ist zu erkennen, dass das untere Viertel des Wertebereichs nie erreicht wird.

In Abbildung 8 ist gut die Linearität des Integrators zu erkennen: die gelbe Kurve   
( ms) lässt sich konstruieren, indem die violette Kurve ( ms) verdoppelt wird. Diese Eigenschaft wird im nächsten Abschnitt bei der Referenzmessung ausgenutzt, wo aber auch gezeigt wird, dass die Linearität aufgrund des Programms nicht perfekt ist.

In der Zukunft kann man sich als Verbesserung des Programms eine variable Integrationszeit vorstellen. So könnte bei einer niedrigen Aerosoldichte automatisch eine große Integrationszeit und bei einer hohen Dichte eine kleine Integrationszeit gewählt werden. Dies würde bewirken, dass die Auflösung immer bestmöglich ausgenutzt wird. Diese Verbesserung kann direkt in der Arduino-Software implementiert werden. Es müsste dann jedoch auch die aktuelle Integrationszeit auf die serielle Schnittstelle geschrieben werden, damit diese im Bedienungsprogramm (siehe unten) berücksichtigt werden kann. Zusätzlich müsste ein Modell erstellt werden, dass die -Faktoren (siehe unten) für Integrationszeiten berechnet, die nicht bei der Referenzmessung auftreten. Die Referenzmessung wird im nächsten Abschnitt behandelt.

## X.X Umrechnung der Messwerte in dB mithilfe einer Referenzmessung

Als Ausgabe des Nephelometers möchte man nicht eine ganze Zahl zwischen 0 und 1023 bekommen. Stattdessen soll die *Dämpfung* in dB angegeben werden. Um die Abhängigkeit zwischen dem Messwert und der Dämpfung zu bestimmen, wird eine Referenzmessung durchgeführt.

Bei der Referenzmessung wurde das Nephelometer in einen Rauchkanal gelassen. Hier wurde die Rauchdichte kontrolliert erhöht und die Dämpfung wurde von einem Referenzmessgerät (MIREX) aufgezeichnet. Das Nephelometer hat hierbei dasselbe Programm wie in der obigen Messung ausgeführt. Die Ergebnisse sind in Abbildung 9 dargestellt.

A screenshot of a cell phone

Description automatically generated

Abbildung 9: Ergebnis der Referenzmessung. Links: Messwerte des Referenzmessgeräts (MIREX) in dB. Rechts: Messwerte des Nephelometers mit verschiedenen Integrationszeiten. Geplottet ist genau wie in Abbildung 8 die unveränderte Ausgabe des Programms, das in Anhang A3 zu finden ist.

Zur Auswertung der Messergebnisse müssen als erstes die Verhältnisse zwischen den verschiedenen Integrationszeiten untereinander untersucht werden. Beim verwendeten Integrator wird, wie bereits ausführlich beschrieben, die Integrationszeit vom Arduino-Programm festgelegt. Bei einem Eingangssignal wird deshalb das folgende folgendes Ausgangssignal zurückgegeben:

wobei eine Konstante ist. Es kann erkannt werden, dass das Integral mit dem -fachen des im Kapiel *Theoretischer Hintergrund* beschriebenen Mittelwerts äquivalent ist:

Daraus folgt, dass das Verhältnis zweier unserer Messungen bei verschiedenen Integrationszeiten durch die folgende Relation ausgedrückt werden kann:

Dieses Verhältnis wurde bereits im letzten Abschnitt durch die Linearität des Integrators begründet. Leider stimmt es in unserem Fall nicht ganz, da die tatsächlichen Integrationszeiten nicht exakt mit den angegebenen übereinstimmen. Dies liegt an dem Arduino-Programm, welches die Messungen steuert. Die Integrationszeit wird im Programm dadurch umgesetzt, dass der Prozessor nach Beginn der Integration die vorgegebene Zeit wartet und anschließend die Integration stoppt. Durch die Umsetzung solcher Befehle auf dem Arduino kommt es zu Nichtlinearitäten zwischen der angegebenen Zeit und der tatsächlichen Integrationszeit. Bei jeder Messung wird zum Beispiel zusätzlich zur vorgegebenen Zeit eine weitere, konstante Zeit integriert, die von den ausgeführten Befehlen abhängt. Je größer ist, desto weniger fällt diese konstante Zeit ins Gewicht, bei kleinem ist sie jedoch ausschlaggebend. Wir modellieren diese Nichtlinearitäten in den Verhältnissen zwischen Messungen verschiedener Integrationszeiten und mit dem folgenden allgemeinen Modell:

Im Idealfall gilt also . Die richtigen Werte für wurden durch eine einfache numerische Optimierung des quadratischen Fehlers gefunden. Die Ergebnisse sind in Abbildung 11 dargestellt. Es gilt , deshalb mussten -Faktoren mit (die unteren Dreiecksmatrizen) nicht dargestellt werden.

Nachdem die -Faktoren berechnet wurden, kann mit der Auswertung fortgefahren werden. Als erstes wird von jeder aufgenommenen Messung der konstante Hintergrundstörwert abgezogen. Dies ist der Wert, der auch ohne eingeschaltete LED aufgenommen wird. Im Fall  ms ist dies leicht, da das Hintergrundsignal hier aufgenommen wurde (‚off\_100ms‘). Es wird also einfach der Mittelwert des Hintergrundsignals von der Messung abgezogen. Da das Hintergrundsignal ohnehin konstant ist, wird durch das Mitteln nur das Rauschen entfernt. Bei allen anderen Signalen muss das Hintergrundsignal anhand der Messung bei  ms abgeschätzt werden. Dies ist jedoch mit den oben bestimmten -Parametern leicht und es wird für jede Messung bestimmt:

wobei die verschobene Messung der Integrationszeit ist, die entsprechende Originalmessung und das Hintergrundsignal (‚off\_100ms‘). Das Ergebnis der Verschiebungen ist in Abbildung 10 dargestellt. Hier kann erkannt werden, dass die Messwerte jetzt natürlich immer noch nur diskrete Werte, aber nicht mehr wie vorher ganze Zahlen annehmen. Sehr auffällig ist, was im vorherigen Abschnitt ausführlich diskutiert wurde: je kleiner die Integrationszeit, desto schlechter ist die effektive Auflösung. Deshalb wird für die restliche Analyse die Messung bei  ms verwendet und die Ergebnisse werden zum Schluss auf die eigentlich wichtige Integrationszeit von  ms übertragen. Es ist ersichtlich, dass bei der Referenzmessung eine Messung bei einer noch größeren Integrationszeit, etwa  ms noch besser Ergebnisse hätte liefern können. Da in der Testmessung (Abb. 8) aber bereits bei  ms Sättigung aufgetreten ist, haben wir für die Referenzmessung keine solche größere Integrationszeit eingebaut.

A screenshot of a cell phone

Description automatically generated

Abbildung 10: Die Messdaten und die jeweils verschobenen Daten .

A screenshot of a cell phone

Description automatically generated

Abbildung 11: Umrechnungsfaktoren zwischen den Messwerten verschiedener Integrationszeiten. Im Idealfall gilt . In den oberen zwei Graphen sind die Faktoren selbst dargestellt, links für alle Kombinationen, bei denen gilt und rechts nur für , im Vergleich mit dem Idealfall . Unten sind die relativen Fehler der -Faktoren dargestellt. Links wieder für alle notwendigen Kombinationen und rechts für den Spezialfall . Hier ist gut zu erkennen, dass die entstehenden Nichtlinearitäten bei kleinen Integrationszeiten besonders ausschlaggebend sind.

Als nächstes soll die Messung geglättet werden, damit aus ihr eine stetige Funktion erstellt werden kann, die nicht zu stark rauscht. Normalerweise kann man so eine Kurve gut glätten, indem man sie mit einer symmetrischen Fensterfunktion (z.B. Gauß-Glocke) faltet. Da die Glättung, die hier angewendet wird, aber auch später im Echtzeitbetrieb durchgeführt werden soll, muss eine Funktion verwendet werden, die zur Berechnung des Funktionswerts am Zeitpunkt nur die vorherigen Zeitpunkte () heranzieht. Berechnet wird ein durch diese Fensterfunktion gewichteter Mittelwert aller im Zeitraum bis aufgenommener Messwerte. Konkret bedeutet das, dass die Funktion nur auf dem Intervall von Null verschieden sein darf. Drei solche Funktionen sind links in Abbildung 12 dargestellt.

Alle drei haben die Form:

mit s. Der Faktor im Normalisierungskoeffizienten der Formel taucht auf, da die Messung in Intervallen aufgenommen wird, die Zeit aber in Sekunden gemessen wird. Die geglätteten Messwerte können dann wie folgt berechnet werden (gewichteter Mittelwert):

oder, da so definiert ist, dass es außerhalb des betrachteten Bereichs Null wird, ist die folgende Rechnung äquivalent:

Diese Berechnung entspricht einer diskreten Faltung von mit , und wird auch programmatisch als solche beschrieben.

In Abbildung 12 (links) ist mit drei verschiedenen Werten für gezeichnet. Auffällig ist, dass ein kleines (hier 0,2) eine sehr flache Kurve erzeugt. Der Extremfall stellt den normalen, ungewichteten Mittelwert dar (Boxcar-Fenster). Ein großer Wert für (hier 5) hingegen, bedeutet eine sehr steile Kurve. Hier ist der Extremfall ein Dirac-Impuls, womit gar kein Mittelwert mehr gebildet wird, sondern nur der aktuellste Wert mit dem Faktor 1 in das Ergebnis einfließt.

In Abbildung 12 sind rechts die Ergebnisse der Glättung mit diesen drei Filtern gezeigt. Man kann erkennen, dass ein kleines eine glattere Kurve bedeutet, wohingegen ein großes noch viel Rauschen durchlässt. Ein kleines hat jedoch auch eine entsprechend größere Reaktionszeit, schließlich hinkt der Mittelwert der letzten 20 s dem Momentanwert hinterher. Als Mittelweg, mit dem Bonus, dass es sehr leicht zu berechnen ist, entscheiden wir uns für .

A screenshot of a cell phone

Description automatically generated

Abbildung 12: Links: Die Fensterfunktion mit verschiedenen Werten des Parameters . Rechts: Die verschobenen Messdaten () und die geglätteten Daten . Für das glätten wurden die drei Funktionen verwendet, die links dargestellt sind, die farbliche Kennzeichnung in der Legende gilt für beide Graphen.

Jetzt wo die Messung als glatte Funktion betrachtet werden kann, kann versucht werden, die Parameter zu finden, die nach einem vorgegebenen Modell in eine Funktion transformieren, welche möglichst nah an der Referenzmessung (Abb. 9, links) liegt. Dazu muss als erstes das Modell aufgestellt werden. Da sich die Kurven bereits sehr ähnlich sehen (siehe auch Abbildung 13, oben und mittig), sollte das folgende simple Modell hinreichen:GLEICHUNG 1

Die Zeitverschiebung um kommt daher, dass die Reaktionszeiten der zwei Messgeräte (MIREX und Nephelometer) identisch sind. Ansonsten sollen also nur ein linearer Faktor und ein konstanter additiver Parameter gefunden werden.

A close up of a map

Description automatically generated

Abbildung 13: Oben: Die Messung des Referenzmessgeräts. Mittig: Die verschobene und geglättete Messung des Nephelometers bei der Integrationszeit . Es ist gut zu erkennen, dass die zwei Kurven eine sehr ähnliche Form aufweisen, die absoluten Werte jedoch sehr unterschiedlich sind. Unten: Die zwei Kurven und mit den gefundenen Parametern. entspricht einer im Zeit- und Wertebereich verschobenen, sowie skalierten Funktion

Für die numerische Optimierung, mit der nach den Parametern gesucht wird, wird die folgende Fehlerfunktion verwendet:

wobei die gesamte Zeit der Messung (ca. 20 min) und  ms der zeitliche Abstand zwischen zwei Messwerten ist. Es wird also die Methode der kleinsten Quadrate verwendet. Zusätzlich wird von der Fehlerfunktion der Zeitbereich unter 100 s doppelt gewichtet, damit es möglichst keine systematische Abweichung im 0 dB-Bereich gibt. Damit bestimmt werden kann, werden die verwendeten Funktionen so erweitert, dass sie außerhalb des Definitionsbereichs einen Fehler von 0 zurückgeben. Damit dies nicht dazu führt, dass der Optimierungsalgorithmus riesig wählt, sodass die resultierende Funktion nirgendwo definiert ist (und der Fehler Null ist), wird der Suchbereich für auf das Intervall [-50 s, 50 s] begrenzt. Die gefundenen Parameter sind in Tabelle 2 zu sehen. Das Resultat der Transformation ist in Abbildung 13 zu erkennen, wo und im gleichen Graphen gezeichnet sind.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 18,25 s | 0,0372 dB | -21,70 dB |

Tabelle 2: Die gefundenen Parameter für das Modell aus GLEICHUNG 1

A close up of a map

Description automatically generated

Abbildung 14: Die resultierenden Transformationen (nach GLEICHUNG 1) mit den gefundenen Parametern (Tab. 2). Links: Transformation der geglätteten Messwerte aller Integrationszeiten. Rechts: Transformation der Messwerte bei .

Als letztes muss der Parameter noch auf die zu benutzende Integrationszeit  ms angepasst werden. Dies ist leicht mit dem oben definierten -Faktor gemacht:

In Abbildung 14 ist rechts das Ergebnis der Transformation für die originalen (um den Mittelwert des gemessenen Hintergrundstörsignals verschobenen) Messdaten, sowie die geglätteten Messdaten mit  ms dargestellt. Es ist zu sehen, dass die transformierte Kurve, abgesehen von etwas Rauschen, gut auf der Referenzkurve liegt. Links in Abbildung 14 ist dasselbe für alle Integrationszeiten geplottet. Wie zu erwarten war, sind die kleinsten Integrationszeiten  ms und  ms sehr stark verrauscht. Dennoch scheint es bei allen Kurven der Fall zu sein, dass die Parameter, welche nur mit den  ms Messwerten gefunden wurden, nah am Optimum liegen.

Der Parameter wird im Folgenden keine Rolle mehr spielen. Er war nur ein Hilfsmittel, um die anderen beiden Parameter zu finden, selbst hat er aber keinen Einfluss auf den Funktionswert von und kann deshalb ignoriert werden. Wichtig ist, dass man sich darüber im Klaren ist, dass bei Messungen eine Verzögerung von ca. 20 s zu erwarten ist. Im Bedienungsprogramm, welches gleich beschrieben wird, werden nur die zwei Parameter und verwendet.

Der Python-Code zur Erstellung aller Grafiken dieses und des vorletzten Abschnittes, sowie zur Bestimmung des „Aus“-Arbeitspunktes, der -Parameter und der Parameter und ist in Anhang A4 hinterlegt.

## X.X Bedienungsprogramm und Endprodukt

Im Betrieb läuft auf dem Arduino ein anderes Programm als in den obigen Abschnitten, da nun nur noch die Integrationszeit  ms eine Rolle spielt. Es wird aber weiterhin neben der Messung, in der die LED eingeschaltet ist, der Hintergrundwert aufgenommen. Dieser wird dann wie oben von der eigentlichen Messung abgezogen. Das Programm ist in Anhang A5 hinterlegt.

Die Ausgabe des Programms wird vom Arduino über die serielle Schnittstelle an den per USB verbundenen Computer weitergegeben. Hier wird die Schnittstelle von dem Bedienungsprogramm eingelesen. Es werden dann die Messungen ausgewertet und grafisch dargestellt. Der komplette Code des Bedienungsprogramms ist in Anhang A6 hinterlegt. Ein Screenshot des Programms ist in Abbildung 15 dargestellt. Vom Arduino werden alle 500 ms der Hintergrundmesswert (LED ausgeschaltet,  ms) und der eigentliche Messwert (LED eingeschaltet,  ms) aufgenommen und an den Computer geschickt. Um die Dämpfung zu bestimmen, wird die Rechnung aus dem vorherigen Abschnitt mit denselben Parametern durchgeführt, wobei nicht beachtet wird. Zur Glättung wird wieder gesetzt und es wird ein gewichteter Mittelwert über   
 s, bei  ms also Messwerte berechnet. Insgesamt wird berechnet:

Mit den geglätteten Messwerten:

und den geglätteten Hintergrundwerten:

Die Fensterfunktion ist in Abbildung 12 zu sehen (die grüne Linie). Die Ergebnisse werden anschließend geplottet, wie in Abbildung 15 erkennbar. Im obersten Diagramm sind die Werte der letzten Minute über der Zeit geplottet. Darunter sind alle Werte seit Beginn der Messung geplottet. Unten im Fenster sind vier Diagramme zu sehen, die die aktuellen Messungen mit der Referenzmessung vergleichen. In dem Diagramm ganz links ist in blau der Teil der Referenzmessung, bei dem die LED aus war (bei   
 ms), aufgetragen. Diese Werte entsprechen der pinken Linie in Abbildung 9. Zusätzlich sind zwei horizontale Linien zu sehen; die rote ist der aktuellste Wert von und die grüne ist ein „korrigierter“ Wert, der im Folgenden noch besprochen wird. In dem zweiten Diagramm von links sind diese Werte in Histogrammform dargestellt. Das blaue Histogramm beschreibt die relative Häufigkeit einzelner Messwerte bei der Referenzmessung, das rote Histogramm beschreibt die relative Häufigkeit der Messwerte bei der aktuellen Messung und das dünne bzw. gelbe Histogramm beschreibt dasselbe bei den „korrigierten“ Werten. Rechts daneben sind zwei weitere Diagramme, die exakt analog zu diesen beiden sind, nur dass hier die Messwerte, bei denen die LED an ist, betrachtet werden. Die entsprechende Linie in Abbildung 9 (und in Abbildung 10) ist grün.

Wie schon in Abbildung 15 zu sehen ist, liegen die Messwerte im Normalfall, angedeutet durch die roten horizontalen Linien, bzw. durch die roten Histogramme, außerhalb des Bereichs, der bei der Referenzmessung aufgetreten ist. Der Grund hierfür ist uns nicht bekannt. Es scheint jedoch nicht an der Temperatur zu liegen, wie aus Abbildung 16 ersichtlich wird.

A screenshot of a computer

Description automatically generated

Abbildung 15: Die grafische Oberfläche des Bedienungsprogramms.

A screenshot of a cell phone

Description automatically generated

Abbildung 16: Links: Messung bei niedriger Temperatur (Nephelometer im Eisfach). Rechts: Messung bei hoher Temperatur (Nephelometer auf der Heizung).

Es ist zu erkennen, dass die Temperatur scheinbar einen kleinen Unterschied macht, jedoch nicht erklären kann, warum die Messung so stark von der Referenzmessung abweicht.

Um das Problem zu beheben, ist in das Programm eine Ruhewertmitführung eingebaut. Wenn die aktuelle Aerosoldämpfung 0 dB betragen soll, kann der „Calibrate“-Button dafür benutzt werden. Solange dieser betätigt ist, wird ein Mittelwert der Messungen (LED an und LED aus) aufgenommen. Das Kalibrieren wird durch erneutes betätigen des Buttons wieder beendet. Nach Kalibration sind zwei Mittelwerte und verfügbar, die nach der folgenden Gleichung in die Rechnung eingehen:

In den vier unteren Diagrammen auf der Bedienoberfläche sind auch die „korrigierten“ Werte und als grüne horizontale Linien bzw. als dünne / gelbe Histogramme dargestellt.

Mit dem Button „Reset Calibration” werden die Mittelwerte und wieder gelöscht. Um das Programm zu beenden muss der Button „Stop Measurement“ betätigt werden, anschließend kann das Fenster geschlossen werden. Mit dem Button „Clear History“ wird die gesamte aufgezeichnete Messung gelöscht. Dies kann eventuell notwendig werden, wenn die Messung eine sehr lange Zeit gelaufen ist, dann kann das Programm theoretisch an die Leistungsgrenzen des Computers kommen. Im Normalfall sollte es aber nicht notwendig sein.

# Quellen

[o1] Lunar Rauchmelder..

[o2] LED <https://www.osram.com/media/resource/hires/osram-dam-2496286/SFH%204550.pdf> , abgerufen am 28.01.2020

[o3] transistor <https://www.onsemi.com/pub/Collateral/BS170-D.PDF> , abgerufen am 28.01.2020

[o4] SPICE led <https://www.osram.com/ecat/Radial%20T1%203-4%20SFH%204550/com/en/class_pim_web_catalog_103489/global/prd_pim_device_2219775/>

[o5] SPICE transistor <http://ltwiki.org/index.php?title=BS170>

Anhang:

A1: Nephelometer Schaltplan

A2: SPICE-Modelle LED und Transistor

A3: plotttttt

A4: Python

A5: Programm Nephelometer-Steuerung (Arduino)

A6: Bedienungsprogramm (MATLAB)

A5:

/\*

Nephelometer Steuerung

Bachelorprojekt Brinkmann, Eberhard, Stock 2020

\*/

// Pins

#define RESET 7

#define PULSE 8

#define SENSOR A1

// Integrationszeit [ms]

#define T 100

void setup() {

pinMode(RESET, OUTPUT); // Pin for discharging the capacitor

pinMode(PULSE, OUTPUT); // Pin for turning the LED on/off

Serial.begin(9600);

}

int record(bool led\_on) {

digitalWrite(RESET, HIGH); // Reset capacitor (close bypass connection)

delayMicroseconds(100); // Let capacitor discharge (R = 100 Ohm, C = 100 nF => RC = 10 us)

digitalWrite(RESET, LOW); // Start integration (open bypass connection)

if (led\_on)

digitalWrite(PULSE, HIGH); // Turn LED on!

delay(T); // Integrate for T milliseconds

if (led\_on)

digitalWrite(PULSE, LOW); // Turn LED off!

return analogRead(SENSOR); // Record sensor measurement

}

void loop() {

// Record time to adjust delay for next measurement

unsigned long time\_start = millis();

// Make the measurements

int led\_off = record(false); // Background measurement

int led\_on = record(true); // Measurement with LED on

// Print result to Serial

char str[64];

sprintf(str, "%010lu,%04d,%04d", time\_start, led\_off, led\_on);

Serial.println(str);

// Repeat exactly every 500ms

unsigned long time\_diff = (millis() - time\_start);

delay(500 - time\_diff);

}

A6:

clear all; close all; clc;

% TODO: Change port accordingly, e.g. 'COM3' or '/dev/tty.usbserial-A106PX8C'

port = '/dev/tty.usbserial-A106PX8C';

arduino = serialport(port, 9600);

% Global variables

global run cali offset\_m offset\_b off\_count X Y c;

% Control variables

run = true;

cali = false;

offset\_m = 0;

offset\_b = 0;

off\_count = 0;

X = []; % Matrix to store measurements

T = 40; % Amount of datapoints to use for convolution / smoothing (T/2 seconds setup time!)

Y = []; % Matrix to store smoothed values

% Window function for smoothing

w = linspace(0, 1, T);

w = w / sum(w); % Normalization, the sum is theoretically T/2, but it needs to be exact

% Parameters for 100ms integration

k = 0.07300424260104722;

b = -21.70479394467475;

t0 = -36.5046091840224; % This is the delay between our measurement and MIREX (response time in 500ms -> ca. 20s)

% Import reference data

ref = readtable('reference\_100ms.csv');

ref = ref{:, 2:end};

% Setup figure with calibrate and reset button

c = uicontrol('Style', 'togglebutton', 'String', 'Calibrate', 'Callback', @calibrate, 'Position', [10 10 100 20]);

r = uicontrol('String', 'Reset Calibration', 'Callback', @reset, 'Position', [120 10 100 20]);

q = uicontrol('String', 'Stop Measurement', 'Callback', @stop, 'Position', [230 10 100 20]);

h = uicontrol('String', 'Clear History', 'Callback', @clear, 'Position', [340 10 100 20]);

while run

% Read data from serial port

x = strip(arduino.readline());

disp(x);

x = str2num(x);

X = [X ; x];

% Compute offset when in calibration mode

if cali

off\_count = off\_count + 1;

offset\_m = mean(X(end-off\_count+1:end, 3)) - mean(ref(1:200, 2));

offset\_b = mean(X(end-off\_count+1:end, 2)) - mean(ref(1:200, 1));

end

% Superimpose current measurements onto reference data

subplot(3, 4, 9);

plot(ref(:, 1), '.')

yline(x(2), 'r');

if offset\_b ~= 0

yline(x(2) - offset\_b, 'g');

end

xlim([0 inf])

title('off');

subplot(3, 4, 10);

histogram(ref(:, 1), 'Normalization', 'probability');

hold on;

histogram(X(:, 2), 'Normalization', 'probability');

if offset\_b ~= 0

histogram(X(:, 2) - offset\_b, 'Normalization', 'probability');

end

hold off;

title('off hist');

subplot(3, 4, 11);

plot(ref(:, 2), '.');

yline(x(3), 'r');

if offset\_m ~= 0

yline(x(3) - offset\_m, 'g');

end

xlim([0 inf])

title('on');

subplot(3, 4, 12);

histogram(ref(:, 2), 'Normalization', 'probability');

hold on;

histogram(X(:, 3), 'Normalization', 'probability');

if offset\_m ~= 0

histogram(X(:, 3) - offset\_m, 'Normalization', 'probability');

end

hold off;

title('on hist');

% At least T measurements are needed to start evaluation

if length(X) < T

continue;

end

% Smooth measurements

z1 = sum(X(end-T+1:end, 3)' .\* w); % Smoothed measurement

z2 = sum(X(end-T+1:end, 2)' .\* w); % Smoothed background

Y = [Y ; [x(1), z1, z2]]; % Store newly smoothed values

% Calculate dB values. For greater efficiency, this matrix could be

% updated row by row for every new measurement but would have to be

% recomputed every time the offset values are changed.

Z = [Y(:, 1), (Y(:, 2) - offset\_m - (Y(:, 3) - offset\_b))\*k + b];

% Plot dB values

if length(Z) < 120 % Plot everything in one graph until 60s have passed

subplot(3, 4, 1:8);

plot(Z(:, 1)/1000, Z(:, 2));

xlim([20 inf])

title('[dB]')

else % After 60s plot complete history and last minute separately

subplot(3, 4, 1:4);

plot(Z(end-120+1:end, 1)/1000, Z(end-120+1:end, 2));

title('Last minute [dB]')

xlim([Z(end-120+1, 1)/1000 inf])

subplot(3, 4, 5:8);

plot(Z(:, 1)/1000, Z(:, 2));

xlim([20 inf])

title('Compete history [dB]')

end

end

function calibrate(~, ~)

global cali off\_count c;

cali = ~cali;

if cali

c.String = 'Calibrating...';

off\_count = 0;

else

c.String = 'Calibrate';

end

end

function reset(~, ~)

global offset\_b offset\_m;

offset\_b = 0;

offset\_m = 0;

end

function stop(~, ~)

global run;

run = false;

end

function clear(~, ~)

global X Y;

X = [];

Y = [];

end

A4:

import numpy as np

import matplotlib as mpl

import matplotlib.pyplot as plt

from matplotlib.ticker import EngFormatter

import seaborn as sns

import pandas as pd

from scipy.interpolate import interp1d

from scipy.signal import convolve

from scipy.optimize import minimize

from sklearn.linear\_model import LinearRegression

# Configuration

sns.set()

plt.rcParams.update({

'figure.figsize': (4\*1.8, 3\*1.8),

'figure.dpi': 300,

'font.family': 'serif',

'font.serif': ['Computer Modern Roman'],

'text.usetex': True,

'text.latex.preamble': [

r'\usepackage[T1]{fontenc}',

r'\usepackage{siunitx}',

r'\usepackage{physics}',

r'\usepackage{amsmath}'

]

})

## Resistor and operating points

# Load spice data

ledvi = pd.read\_csv('led\_v\_i.txt', '\t').values

afk\_on = pd.read\_csv('fet\_id\_uds\_ugs5.txt', '\t').values

afk\_off = pd.read\_csv('fet\_id\_uds\_ugs0.txt', '\t').values

operation = pd.read\_csv('operation.txt', '\t').values

# Compute "off" operating point

led\_v = interp1d(ledvi[:, 0], ledvi[:, 1])

fet\_id = interp1d(afk\_off[:, 0], afk\_off[:, 1])

fet\_v = 5

i\_d = 0

points = np.zeros((4, 2))

print("Arbeitspunkt-Iteration:")

for i in range(4):

print(f"{i\_d:.5e} {5 - fet\_v:.5e} {fet\_v:.5e}")

points[i, :] = np.r\_[[fet\_v, i\_d]]

i\_d = fet\_id(fet\_v)

fet\_v = 5 - led\_v(i\_d)

# Plot LED V-I diagram and operating points

fig, ax = plt.subplots(constrained\_layout=True)

ax.semilogx(ledvi[:, 0], ledvi[:, 1], label="U-I-Kennlinie der Diode (SPICE-Simulation)")

ax.plot([0.1], [1.495], 'go', label=r'Arbeitspunkt EIN $(\SI{100}{\mA}, \SI{1.495}{\V})$')

ax.plot(points[-1][1], 5 - points[-1][0], 'ro', label=r'Arbeitspunkt AUS $(\SI{34.4}{\nA}, \SI{0.879}{\V})$')

ax.set(xlabel='$I\_D$ / A', ylabel='$U\_D$ / V')

ax.set\_xticks(10.\*\*np.arange(-10, 1))

ax.xaxis.grid(True, which='minor', linewidth=.3)

ax.xaxis.set\_minor\_locator(mpl.ticker.LogLocator(base=10. ,subs=(.1, .2, .3, .4, .5, .6, .7, .8, .9), numticks=12))

ax.xaxis.set\_major\_formatter(EngFormatter(places=0, sep=""))

ax.legend()

fig.savefig('AP.png')

# Plot FET output curve and operating points / line

fig, ax = plt.subplots(constrained\_layout=True)

ax.plot(afk\_on[:, 0], afk\_on[:, 1]\*1000, label=r"Ausgangskennlinie des FET bei $U\_{GS} = \SI{5}{\V}$ (SPICE-Sim.)")

p = 100 + (100 - points[-1, 1])/(points[-1, 0] - 0.58)\*0.58

ax.plot([0, points[-1, 0]], [p, points[-1, 1]], label='Arbeitsgerade')

ax.plot([0.58], [100], 'o', label=r'Arbeitspunkt EIN $(\SI{0.58}{\V}, \SI{100}{\mA})$')

ax.plot(points[-1, 0], points[-1, 1], 'o', label=r'Arbeitspunkt AUS $(\SI{4.12}{\V}, \SI{34.4}{\nA})$')

ax.set\_xticks(list(range(6)) + [0.58])

ax.set(xlabel='$U\_{DS}$ / V', ylabel='$I\_D$ / mA')

ax.legend(loc='upper left')

fig.savefig('AKF.png')

# Plot switching characteristic

fig, ax = plt.subplots(constrained\_layout=True)

ax.plot(operation[:, 0], operation[:, 1]\*1000, label="SPICE-Simulation")

ax.plot([5], [90.67], 'go', label='Arbeitspunkt EIN')

ax.plot([0], [0], 'ro', label='Arbeitspunkt AUS')

ax.set\_yticks(list(range(0, 90, 20)) + [90.67])

ax.set(xlabel='$U\_{GS}$ / V', ylabel='$I\_D$ / mA')

ax.legend()

fig.savefig('sim.png')

# Plot iteration

fig, ax = plt.subplots(1, 2, constrained\_layout=True)

ax[0].semilogy(5 - ledvi[:, 1], ledvi[:, 0], label="$I\_{D}$ (LED)")

ax[0].semilogy(afk\_off[:, 0], afk\_off[:, 1], "C2", label="$I\_D$ (FET)")

ax[0].step(points[:, 0], points[:, 1], 'C3', label="Iteration")

ax[0].set\_xticks(np.arange(0, 6))

ax[0].set\_yticks(10.\*\*np.arange(-10, 1))

ax[0].yaxis.grid(True, which='minor', linewidth=.3)

ax[0].yaxis.set\_minor\_locator(mpl.ticker.LogLocator(base=10. ,subs=(.1, .2, .3, .4, .5, .6, .7, .8, .9), numticks=12))

ax[0].yaxis.set\_major\_formatter(EngFormatter(unit="A", places=0))

ax[0].set\_xlabel("$U\_{DS} = \SI{5}{\V} - U\_{\mathrm{LED}}$")

ax[0].legend(loc='upper left')

ax[1].plot(5 - ledvi[:len(ledvi)//2, 1], ledvi[:len(ledvi)//2, 0], label="$I\_{D}$ (LED)")

ax[1].plot(afk\_off[len(afk\_off)//2:, 0], afk\_off[len(afk\_off)//2:, 1], "C2", label="$I\_D$ (FET)")

ax[1].step(points[:, 0], points[:, 1], 'C3', label="Iteration")

ax[1].yaxis.set\_major\_formatter(EngFormatter(unit="A", places=0))

ax[1].set\_xlim(4.11, 4.13)

ax[1].set\_ylim(3e-8, 4.5e-8)

ax[1].set\_xlabel("$U\_{DS} = \SI{5}{\V} - U\_{\mathrm{LED}}$")

ax[1].legend(loc='upper right')

fig.savefig('iteration.png')

## Reference measurement

# Import and process MIREX data

reference = pd.read\_csv('nephelometer\_002.dat', '\t|;', index\_col=0, skiprows=19, engine='python')

reference = reference.apply(lambda c: c.str.replace(',', '.')).astype(float).rename(columns=lambda c: c.strip())

reference.index = pd.to\_datetime(reference.index) - pd.to\_datetime(reference.index[0])

reference = reference.resample('500ms').interpolate('time')

reference.index.name = None

reference['sec'] = reference.index.values.astype(float) \* 1e-9

# Import and process nephelometer data

nephelometer = pd.read\_csv('messung.txt', '\ |,', index\_col=0, engine='python').iloc[:, 1:] # Import data

nephelometer.index = (pd.to\_datetime(nephelometer.index) # Normalize timestamps

- pd.to\_datetime(nephelometer.index[0]))

nephelometer = nephelometer.resample('500ms').ffill() # Upsample for synchronous data

nephelometer.index.name = None

nephelometer['sec'] = nephelometer.index.values.astype(float) \* 1e-9

# The relation between integration times is not precisely linear (due to arduino program)

def scale(data\_from, data\_to):

return LinearRegression(fit\_intercept=False).fit(data\_from[:, np.newaxis], data\_to[:, np.newaxis]).coef\_[0][0]

# Remove background measurement from measurement with LED turned on and normalize values.

for i in ['200', '150', '100', '50', '20', '10']:

nephelometer[f'i{i}'] = (nephelometer[f'on\_{i}ms']

- (scale(nephelometer.on\_100ms, nephelometer[f'on\_{i}ms'])

\* nephelometer.off\_100ms.mean()))

# Make both the MIREX and nephelometer measurements the same length

reference = reference[:len(nephelometer)]

assert len(reference) == len(nephelometer)

# Plot measurements

fig, ax = plt.subplots(1, 2, constrained\_layout=True)

reference.plot(x='sec', y='MIREX', ax=ax[0])

ax[0].set\_xlabel("Zeit [s]")

ax[0].set\_ylabel("Dämpfung [dB]")

nephelometer.plot(x='sec', y=['on\_200ms', 'on\_150ms', 'on\_100ms', 'on\_50ms', 'on\_20ms', 'on\_10ms', 'off\_100ms'],

ax=ax[1])

ax[1].legend([r'\texttt{on\\_200ms}', r'\texttt{on\\_150ms}', r'\texttt{on\\_100ms}', r'\texttt{on\\_50ms}',

r'\texttt{on\\_20ms}', r'\texttt{on\\_10ms}', r'\texttt{off\\_100ms}'])

ax[1].set\_xlabel("Zeit [s]")

ax[1].set\_ylabel("Messwert [0-1023]")

fig.savefig('messung.png')

# Plot shifted measurements

fig, ax = plt.subplots(2, 3, sharex=True, constrained\_layout=True)

# Lighter colors

c = np.zeros((6, 3))

for i in range(6):

x = mpl.colors.rgb\_to\_hsv(mpl.colors.to\_rgb(f"C{i}"))

x[1] \*\*= 3

x[2] \*\*= 1/3

c[i, :] = mpl.colors.hsv\_to\_rgb(x)

nephelometer.plot(x='sec', y='on\_200ms', c=c[0, :], style='.', ax=ax[0, 0],

title=r"$T = \SI{200}{\ms}$", label="Original", alpha=1)

nephelometer.plot(x='sec', y='on\_150ms', c=c[1, :], style='.', ax=ax[0, 1],

title=r"$T = \SI{150}{\ms}$", label="Original", alpha=1)

nephelometer.plot(x='sec', y='on\_100ms', c=c[2, :], style='.', ax=ax[0, 2],

title=r"$T = \SI{100}{\ms}$", label="Original", alpha=1)

nephelometer.plot(x='sec', y='on\_50ms', c=c[3, :], style='.', ax=ax[1, 0],

title=r"$T = \SI{50}{\ms}$", label="Original", alpha=1)

nephelometer.plot(x='sec', y='on\_20ms', c=c[4, :], style='.', ax=ax[1, 1],

title=r"$T = \SI{20}{\ms}$", label="Original", alpha=1)

nephelometer.plot(x='sec', y='on\_10ms', c=c[5, :], style='.', ax=ax[1, 2],

title=r"$T = \SI{10}{\ms}$", label="Original", alpha=1)

nephelometer.plot(x='sec', y='i200', style='C0.', ax=ax[0, 0], title=r"$T = \SI{200}{\ms}$", label="Verschoben")

nephelometer.plot(x='sec', y='i150', style='C1.', ax=ax[0, 1], title=r"$T = \SI{150}{\ms}$", label="Verschoben")

nephelometer.plot(x='sec', y='i100', style='C2.', ax=ax[0, 2], title=r"$T = \SI{100}{\ms}$", label="Verschoben")

nephelometer.plot(x='sec', y='i50', style='C3.', ax=ax[1, 0], title=r"$T = \SI{50}{\ms}$", label="Verschoben")

nephelometer.plot(x='sec', y='i20', style='C4.', ax=ax[1, 1], title=r"$T = \SI{20}{\ms}$", label="Verschoben")

nephelometer.plot(x='sec', y='i10', style='C5.', ax=ax[1, 2], title=r"$T = \SI{10}{\ms}$", label="Verschoben")

ax[0, 0].set\_ylabel("Messwert [0-1023]")

ax[1, 0].set\_ylabel("Messwert [0-1023]")

for a in ax.flat:

a.yaxis.grid(True, 'minor', linewidth=.3)

a.yaxis.set\_minor\_locator(mpl.ticker.MultipleLocator(1))

a.yaxis.set\_major\_formatter(mpl.ticker.FormatStrFormatter('%d'))

a.set\_xlabel("Zeit [s]")

a.legend(fontsize=8, loc='upper left')

fig.savefig('verschiebung.png')

# Window function

def window(T=40, alpha=1):

w = np.r\_[np.linspace(0, 1, T)\*\*alpha, T\*[0]]

return w / w.sum() # Normalization

assert np.allclose([window(alpha=a).sum() for a in [.2, 1, 5]], [1]\*3) # Test normalization

# Smoothing function

def conv(data, T=40, alpha=1, pad=True):

win = window(T, alpha) # Window function

res = np.r\_[T\*[np.nan], convolve(data, win[::-1], mode='same')[T:]] # Convolution

if pad:

res = np.r\_[T\*[data[:T].mean()], res[T:]] # Pad beginning

return res

# Plot window functions and smoothed measurements

fig, ax = plt.subplots(1, 2, constrained\_layout=True)

ax[0].plot(np.linspace(-20, 20, 80), window(alpha=5), 'C1', label=r'$\alpha = 5$') # almost f (dirac)

ax[0].plot(np.linspace(-20, 20, 80), window(alpha=1), 'C2', label=r'$\alpha = 1$') # linear

ax[0].plot(np.linspace(-20, 20, 80), window(alpha=.2), 'C3', label=r'$\alpha = 0.2$') # almost mean (boxcar)

ax[0].set\_xlabel(r"Zeit [s]")

ax[0].legend()

ax[1].plot(nephelometer.sec, nephelometer.i200.values, '.', alpha=.5, label='Messwerte ($T = \SI{200}{\ms}$)')

ax[1].plot(nephelometer.sec, conv(nephelometer.i200.values, alpha=5))

ax[1].plot(nephelometer.sec, conv(nephelometer.i200.values, alpha=1))

ax[1].plot(nephelometer.sec, conv(nephelometer.i200.values, alpha=.2))

ax[1].set\_xlabel(r"Zeit [s]")

ax[1].legend()

fig.savefig('glatt.png')

# Find parameters using the 200ms curve to get the cleanest fit.

neph = conv(nephelometer.i200.values)

ref = reference['MIREX'].values

# Optimization to find actual parameters

t = np.arange(len(neph))

f\_ref = interp1d(t, ref, bounds\_error=False)

f\_neph = interp1d(t, neph, bounds\_error=False)

f\_model = lambda t, a, b, c: c\*f\_neph(t - a) + b

def error(x):

E = (f\_ref(t) - f\_model(t, x[0], x[1], x[2])) \*\* 2 # Least squares error

E[:200] \*= 2 # Especially penalize bias at 0

return E[~np.isnan(E)].sum()

# Find optimal parameters using `error` as the objective function

x = None

for a in range(-50, 50):

x\_ = \*minimize(error, np.r\_[a, 0, 1], bounds=((-100, 100), (None, None), (None, None))).x,

if x is None or error(x\_) < error(x):

x = x\_

a, b, c = \*x, # Optimal parameters

print("\nGefundene Parameter:")

print(a, b, c)

# Plot fit with found parameters

fig, ax = plt.subplots(3, 1, sharex=True, constrained\_layout=True)

ax[0].plot(t/2, ref, label='MIREX')

ax[0].legend()

ax[0].set\_ylabel("Dämpfung [dB]")

ax[1].plot(t/2, neph, label='Messwerte ($T = \SI{200}{\ms}$, geglättet)')

ax[1].legend()

ax[1].set\_ylabel("Messwert [0-1023]")

ax[2].plot(t/2, f\_ref(t), label='MIREX')

ax[2].plot(t/2, f\_model(t, a, b, c), label='Messwerte (Verschoben und Skaliert)')

ax[2].legend()

ax[2].set\_ylabel("Dämpfung [dB]")

ax[2].set\_xlabel("Zeit [s]")

fig.savefig('optim.png')

# Rescale parameters for T = 100ms

k = c\*scale(nephelometer.i100.values, nephelometer.i200.values)

b = b

t0 = a

print(f"\nParameters for 100ms integration:\nScale factor: k = {k}\nOffset: b = {b}\nTime delay: t0 = {t0}"

"\nf(t) [dB] = k\*m(t - t\_0) + b")

# Test model on all measurements

def model(data):

f = interp1d(t, conv(scale(data, nephelometer.i200.values)\*data), bounds\_error=False)

return lambda t, a, b, c: c\*f(t - a) + b

# Plot results

fig, ax = plt.subplots(1, 2, sharey=True, constrained\_layout=True)

ax[0].plot(t, f\_ref(t), label='MIREX')

ax[0].plot(t, model(nephelometer.i200.values)(t, a, b, c), alpha=.3, label=r'$\SI{200}{ms}$')

ax[0].plot(t, model(nephelometer.i150.values)(t, a, b, c), alpha=.3, label=r'$\SI{150}{ms}$')

ax[0].plot(t, model(nephelometer.i100.values)(t, a, b, c), alpha=.3, label=r'$\SI{100}{ms}$')

ax[0].plot(t, model(nephelometer.i50.values)(t, a, b, c), alpha=.3, label=r'$\SI{50}{ms}$')

ax[0].plot(t, model(nephelometer.i20.values)(t, a, b, c), alpha=.3, label=r'$\SI{20}{ms}$')

ax[0].plot(t, model(nephelometer.i10.values)(t, a, b, c), alpha=.3, label=r'$\SI{10}{ms}$')

ax[0].legend()

ax[1].plot(t, f\_ref(t), label='MIREX')

ax[1].plot(k\*nephelometer.i100.values[int(-t0):] + b, '.', # Result without smoothing

alpha=.5, label=r'Messwerte ($T = \SI{100}{\ms}$)')

ax[1].plot(k\*interp1d(t, conv(nephelometer.i100.values), # Result with smoothing

bounds\_error=False)(t - t0) + b, label='Geglättete Messwerte')

ax[1].legend()

ax[0].set\_ylabel("Dämpfung [dB]")

ax[0].set\_xlabel("Zeit [s]")

ax[1].set\_xlabel("Zeit [s]")

fig.savefig('final.png')

# Scale factors (lambda-factors)

sizes = [200, 150, 100, 50, 20, 10]

scales = np.full((6, 6), np.nan)

scales2 = np.full((6, 6), np.nan)

for i, a\_ in enumerate(sizes):

for j, b\_ in enumerate(sizes[i:], i):

scales[i, j] = scale(nephelometer[f"i{a\_}"], nephelometer[f"i{b\_}"])

scales2[i, j] = scales[i, j] / (sizes[j] / sizes[i])

# Plot scale factors

fig, ax = plt.subplots(2, 2, sharex='col', constrained\_layout=True)

ax[0, 1].plot(sizes, np.r\_[sizes] / 200, 'C1-', label=r"Ideal ($\frac{T\_2}{T\_1}$)")

ax[0, 1].plot(sizes, scales[0, :], 'C0.-', label="Tatsächlich")

ax[0, 1].text(120, 0.25, r"$T\_1 = \SI{200}{\ms}$")

ax[0, 1].legend()

ax[0, 1].set\_ylabel(r"$\lambda\_{T\_1\rightarrow T\_2}$")

ax[1, 1].plot(sizes, len(sizes)\*[1], 'C1-', label="Ideal (1)")

ax[1, 1].plot(sizes, scales2[0, :], 'C0.-', label="Tatsächlich")

ax[1, 1].text(120, 1.06, r"$T\_1 = \SI{200}{\ms}$")

ax[1, 1].legend()

ax[1, 1].set\_ylabel(r"$\lambda\_{T\_1\rightarrow T\_2}\cdot\frac{T\_1}{T\_2}$")

ax[1, 1].set\_xticks(sizes)

ax[1, 1].set\_xlabel(r"$T\_2$ [ms]")

im = ax[0, 0].imshow(scales, cmap='viridis', vmin=ax[0, 1].get\_ylim()[0], vmax=ax[0, 1].get\_ylim()[1])

cbar = fig.colorbar(im, ax=ax[0, 0])

ax[0, 0].grid(False)

ax[0, 0].set\_xticks(np.arange(0, 6))

ax[0, 0].yaxis.set\_major\_formatter(mpl.ticker.IndexFormatter(sizes))

ax[0, 0].set\_ylabel(r"$T\_1$ [ms]")

im = ax[1, 0].imshow(scales2, cmap='viridis', vmin=ax[1, 1].get\_ylim()[0], vmax=ax[1, 1].get\_ylim()[1])

cbar = fig.colorbar(im, ax=ax[1, 0])

ax[1, 0].grid(False)

ax[1, 0].set\_xticks(np.arange(0, 6))

ax[1, 0].yaxis.set\_major\_formatter(mpl.ticker.IndexFormatter(sizes))

ax[1, 0].xaxis.set\_major\_formatter(mpl.ticker.IndexFormatter(sizes))

ax[1, 0].set\_xlabel(r"$T\_2$ [ms]")

ax[1, 0].set\_ylabel(r"$T\_1$ [ms]")

fig.savefig('linear.png')

# Export 100ms data for error checking in matlab

export = nephelometer[['off\_100ms', 'on\_100ms']]

export.columns = ['off', 'on']

export.to\_csv('reference\_100ms.csv')

A3:

// Pins

#define RESET 7

#define PULSE 8

#define SENSOR A1

void setup() {

pinMode(RESET, OUTPUT); // Pin for discharging the capacitor

pinMode(PULSE, OUTPUT); // Pin for turning the LED on/off

Serial.begin(9600);

Serial.println("off\_100ms,on\_200ms,on\_150ms,on\_100ms,on\_50ms,on\_20ms,on\_10ms");

}

int record(int T, bool led\_on) {

digitalWrite(RESET, HIGH); // Reset capacitor (close bypass connection)

delayMicroseconds(100); // Let capacitor decharge (R = 100 Ohm, C = 100 nF => tau = 10 us)

if (led\_on)

digitalWrite(PULSE, HIGH); // Turn LED on!

digitalWrite(RESET, LOW); // Start integration (open bypass connection)

delay(T); // Integrate for T milliseconds

if (led\_on)

digitalWrite(PULSE, LOW); // Turn LED off!

return analogRead(SENSOR); // Record sensor measurement

}

void loop() {

char str[64];

sprintf(str, "%d,%d,%d,%d,%d,%d,%d", record(100, false), record(200, true), record(150, true),

record(100, true), record(50, true), record(20, true), record(10, true));

Serial.println(str);

}

A2:

SPICE-Modell LED:

.MODEL SFH4550 D

+ IS = 7.06922E-19

+ N = 1.381203754

+ RS = 0.775721745

+ IKF = 0.341537995

+ IBV = 0.00000003

+ NBV = 2

+ IBVL = 0.000000003

+ NBVL = 150

+ BV = 14

+ CJO = 4.19E-11

+ VJ = 0.23

+ M = 0.07

+ FC = -0.33

+ TT = 0.000000009

+ EG = 1.46

SPICE-Modell Transistor:

.MODEL BS170 VDMOS

+ VTO = 1.824

+ RG = 270

+ RS = 1.572

+ RD = 1.436

+ RB = .768

+ KP = .1233

+ Cgdmax = 20p

+ Cgdmin = 3p

+ CGS = 28p

+ Cjo = 35p

+ Rds = 1.2E8

+ IS = 5p

+ Bv = 60

+ Ibv = 10u

+ Tt = 161.6n