Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jan Huesmann
[redacted]

Automatisierte robotergestützte Messung und Protokollierung photometrischer Daten von DMX-fähigen Scheinwerfern

Fakultät Design, Medien und Information
Department Medientechnik

Faculty of Design, Media and Information
Department of Media Technology

Jan Huesmann

Automatisierte robotergestützte Messung und Protokollierung photometrischer Daten von DMX-fähigen Scheinwerfern

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Medientechnik
am Department Medientechnik
der Fakultät Design, Medien und Information
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuende Prüferin: Prof. Dr.-Ing Carolin Liedtke
Zweitgutachter: B. Sc. Fabian Oving

Eingereicht am: 21. Juli 2022

Jan Huesmann

Thema der Arbeit
Automatisierte robotergestützte Messung und Protokollierung photometrischer Daten von DMX-fähigen Scheinwerfern

Stichworte
Spektrometer, Photometer, Automatisierung, Robotik, DMX, Scheinwerfer

Kurzzusammenfassung
Gegenstand dieser Bachelorarbeit ist die Entwicklung einer Software zur automatischen Messung und Protokollierung photometrischer Daten von Scheinwerfern, die über das DMX-512-Protokoll steuerbar sind. Die Softwareentwicklung basiert auf einem Spektrometer und einem Roboter und ist für den Einsatz im Lichtlabor der HAW Hamburg entwickelt worden. Zur Bedienung der Software wird lediglich ein Browser verwendet, die Messdaten werden automatisch in einer benutzerfreundlichen XLSX-Datei gespeichert und visualisiert.

Jan Huesmann

Title of the paper
Automated robotic measurement and logging of photometric data of DMX-capable spotlights

Keywords
spectrometer, photometer, automation, robotic, DMX, spotlights

Abstract
The subject of this bachelor thesis is the development of a software for the automatic measurement and logging of photometric data of spotlights controllable via DMX-512 protocol. The software development is based on a spectrometer and a robot and has been developed for use in the lighting laboratory of HAW Hamburg. Only a browser is used to operate the software and the measurement data is automatically stored and visualized in a user-friendly XLSX file.

Danksagung

Fabian Oving, für die tollen Jahre im Lichtlabor, das für mich und viele andere zu einem zweiten Zuhause geworden ist.

Prof. Dr. Carolin Liedtke, für die Betreuung dieser Arbeit.

Allen anderen Menschen im Lichtlabor, die die Zeit meines Bachelorstudiums so bereichert haben.

Inhaltsverzeichnis

1 Einleitung

„Bei einer Bearbeitung lichttechnischer Fragen darf heute die Verwendung des künstlichen Lichts für Reproduktionszwecke nicht unberücksichtigt bleiben. Früher meist nur aushilfsweise neben dem natürlichen Tageslicht benutzt, ist das Kunstlicht schon seit längerer Zeit zum ständigen und unentbehrlichen Arbeitsgerät des Reproduktionstechnikers geworden. Hierzu hat besonders die neuere Entwicklung der elektrischen Lichtquellen beigetragen.“ (Bloch 2019, S. 523)

Diesen Abschnitt schrieb der Ingenieur L. Bloch bereits 1921 in einem Werk zur Lichttechnik. Zu jener Zeit war die Erzeugung von Licht durch Elektrizität noch keine Selbstverständlichkeit, stattdessen war die Verbrennung von Gas oder Feststoffen zur Lichterzeugung noch sehr geläufig. L. Bloch erkannte jedoch früh die Vorteile der elektrischen Lichterzeugung, um unabhängig vom Tageslicht etwa Fotografien zu erstellen, die immer gleich aussehen konnten. Heute eine wahrgenommene Selbstverständlichkeit, die doch nicht mehr immer eine ist. Mit dem Einzug der LED-Technik in die Lichterzeugung veränderte sich diese Selbstverständlichkeit der Reproduzierbarkeit drastisch und die technischen Neuerungen stellten die Hersteller von Leuchten vor viele neue Herausforderungen. Hohe Wärmeentwicklung auf sehr kleiner Fläche, schmale emittierte Wellenlängenbereiche und neue Ansteuerungstechniken sind nur eine Auswahl an Herausforderungen, die der Einzug der LED-Technik in die Industrie mit sich brachte.

Das Wissen darüber, wie gut diese Herausforderungen von Herstellern gelöst werden und welche Kompromisse dafür eingegangen werden müssen, ist für die Kunden:innen und die Nutzer:innen von modernen Scheinwerfern in der Veranstaltungs- und Theaterbranche essenziell wichtig. Bei heute üblichen vier- bis fünfstelligen Preisen für einzelne Scheinwerfer müssen Investitionen genau durchdacht werden und die eingegangenen Kompromisse dem Einsatzzweck entsprechen.

Photometrische Messdaten der Scheinwerfer sind bei diesen Entscheidungsfindungen einer der wichtigsten Anhaltspunkte, da diese objektive Argumente liefern. Welcher Scheinwerfer ist heller, welche Farbwiedergabequalität ist besser oder welche Farbtemperaturen können

wirklich erreicht werden? Und welche Einflüsse haben bestimmte Einstellungen auf die anderen Werte? Diesen Fragen kann mit photometrischen Messungen nachgegangen werden. Soll die Aufnahme dieser Messdaten effizient erfolgen, stellt sich spätestens bei der Veränderung einer Variablen während der Messung die Aufgabe einer Automatisierung. Diese Arbeit fokussiert sich auf die Entwicklung und die Realisierung einer Software zur automatisierten Aufnahme von Messreihen photometrischer Daten.

1.1 Motivation

Die Motivation für diese Bachelorarbeit kommt durch meine persönlichen Erfahrungen in der Erstellung von Messreihen zu photometrischen Daten von Scheinwerfern.

Bei meiner Anstellung für einen Scheinwerferhersteller war es Teil meiner Aufgaben Messreihen zu erstellen und zu visualisieren, die von Kund:innen oder den internen Vertriebler:innen angefragt wurden. Die manuelle Aufnahme der Messreihen war oft sehr zeitaufwendig, jedoch dabei wenig herausfordernd. Wurde beispielsweise der Verlauf der maximalen Helligkeit in den ersten 30 Minuten angefragt, bedeutete dies einen Testaufbau, bei dem die Beleuchtungsstärke eines Scheinwerfers jede 60 Sekunden gemessen und die Messung dabei händisch ausgelöst werden musste. Danach wurde der Messwert vom Display abgelesen und manuell in eine Tabellenkalkulation eingetragen. Nebenbei wurde das Spektrum auf dem Messgerät in einer einzelnen Datei gespeichert. Dieser Vorgang wurde 30-mal wiederholt und anschließend manuell in eine Grafik zu diesem zeitlichen Verlauf und dem Spektrum zu Beginn und nach 30 Minuten übertragen. Ähnlich waren die Messungen von beispielsweise Dimmerkurven. Hier musste nach der geforderten Genauigkeit in einer gewissen Anzahl von Schritten der DMX-Wert für den Dimmerkanal stufenweise durchlaufen und durch händische Messungen ausgelöst werden, bis die gesamte Dimmerkurve aufgenommen war. Insbesondere bei Messungen mit vielen Messschritten oder etwa bei genauer Betrachtung der Farbwiedergabequalitäten, war dies aufgrund der Datenmengen eine sehr ineffiziente Arbeit.

Die Entwicklung einer automatisierten Erfassung der Messreihen mit erweitertem Funktionsumfang war daher für mich der logische Schritt zur effizienteren Bewerkstelligung dieser notwendigen Arbeit. Benutzerfreundlich, flexibel und trotzdem weitestgehend automatisiert soll bestenfalls das Ergebnis dieser Arbeit sein, um auf einfache Weise mit der Aufnahme von photometrischen Messdaten Scheinwerfer vergleichbar zu machen und objektive Aussagen über Stärken und Schwächen von Scheinwerfern zu ermöglichen.

1.2 Aufbau der Arbeit

Der Aufbau dieser Arbeit orientiert sich an dem Entwicklungsprozess der im Rahmen dieser Arbeit realisierten Software.

Dafür wird diese Arbeit in fünf weitere Kapitel unterteilt. In dem folgenden Kapitel werden die Anforderungen an die im Rahmen dieser Arbeit entstehenden Software erfasst, grundlegende Funktionen festgehalten und in ihren Grundzügen definiert.
Im nachfolgenden Kapitel 3 wird basierend auf der Anforderungsanalyse aus Kapitel 2 die verfügbare Hardware analysiert und erste Ideen zur Umsetzung der Anforderungen mit der konkreten Hardware werden erklärt. Kapitel 4 befasst sich mit der Realisierung der Software und beschreibt in acht Unterkapiteln die Softwareentwicklung. Dabei werden verschiedene Arbeitsschritte betrachtet, die beim physischen technischen Aufbau und der Verkabelung des Gesamtsystems beginnen und über die Struktur der Software bis abschließend hin zur Visualisierung der aufgezeichneten Messdaten gehen. Dieses Kapitel bildet den Schwerpunkt dieser Arbeit und soll getroffene Entscheidungen und technische Lösungen erklären, die im Laufe der Softwareentwicklung auftraten.

Im 5. und vorletzten Kapitel wird die entstandene Software getestet. Dafür wird zu Beginn eine durchgeführte Messung inhaltlich analysiert und überprüft, ob sich anhand der Messergebnisse konkrete Aussagen über den vermessenen Scheinwerfer treffen lassen. Danach wird die Hardware auf Fehlerquellen hin geprüft und eine Abschätzung der entstehenden Fehler vorgenommen. Das letzte Kapitel 6 bildet den Abschluss der Arbeit und zieht ein Fazit zu den vorherigen Kapiteln.

2 Anforderungsanalyse

Folgend werden die gesetzten Anforderungen an die im Rahmen dieser Arbeit entstehenden hardware-nahen Software aufgeführt und kurz beschrieben.

— Photometrische Messungen im zeitlichen Verlauf

Bei der Aufnahme von Messreihen ist eine wichtige Variable die Zeit.

Zur Beobachtung zeitlicher Veränderung photometrischer Daten wie beispielsweise Beleuchtungsstärke, ähnlichster Farbtemperatur, Farbort oder Spektrum ist eine Funktion zur automatischen, zeitgesteuerten Messungen in definierten Abständen essenziell. Dazu gehört auch die Funktion einer optionalen, definierten Wartezeit vor der ersten Messung, da zeitlich bedingte Schwankungen besonders in den ersten Minuten zu erwarten sind. So kann beispielsweise bei Aufnahme einer Dimmerkurve dieser Einfluss minimiert werden.

— Photometrische Messungen im räumlichen Verlauf

Besonders bei scharfen Abbildungen von Profilscheinwerfern ist die Homogenität der Abbildung interessant. Mit Hilfe eines Roboters soll eine Messung auf der horizontalen Achse des Lichtkegels möglich gemacht werden, um mit einer benutzerdefinierten Anzahl von Schritten den Lichtkegel im räumlichen Verlauf zu vermessen und so die Homogenität bewertbar zu machen.

— Photometrische Messungen im Verlauf der DMX-Ansteuerung von Funktionen

Essenziell ist die automatische Ansteuerung von Funktionen des zu vermessenen Scheinwerfers. Die Ansteuerung dieser Funktionen funktioniert per DMX-Protokoll. Somit muss die Software benutzerdefinierte DMX-Kanäle ansteuern können und die Werte automatisiert anhand der von den Nutzer:innen eingestellten Anzahl an Messschritten bestimmen und ausgeben.

Außerdem sind Minimal- und Maximalwerte sowie eine 16-Bit Unterstützung wünschenswert.

— Einbindung eines Lichtstellpultes per IP-DMX-Protokoll

DMX-fähige Scheinwerfer haben in der Regel definierte DMX-Default-Werte, die den Scheinwerfer in eine Grundstellung bringen. Diese sind bei jedem Scheinwerfer, sowie nochmal bei jedem DMX-Modus, sehr unterschiedlich und müssen für eine sinnvolle Messung ausgegeben werden. Um nicht eine gesamte Bibliothek für alle Scheinwerfer aller Hersteller zu erstellen, die diese Werte beinhaltet und einstellt, empfiehlt sich die Einbindung eines Lichtstellpults, bei dem die Hersteller die Bibliotheken pflegen und regelmäßig aktualisieren. Das Lichtstellpult gibt per IP-Protokoll die DMX-Werte an einen DMX-Merger, in dem dann diese Werte von den variablen Werten der Software überschrieben werden. So ist eine einfache und gewohnte Bedienung des Scheinwerfers per Lichtstellpult und gleichzeitig eine DMX-Ausgabe der Software mit Priorität gegeben.

— Steuerung und Auslesung der Messdaten des Gossen MAVOSPEC BASE-Spektrometer

Das *Gossen MAVOSPEC BASE* ist das gewählte Spektrometer zur Messung der photometrischen Daten. Das Messgerät ist ein Standalone-Messgerät, das somit auch autark für simplere Messungen unterwegs genutzt werden kann. Gleichzeitig bietet es jedoch auch eine USB-Schnittstelle, mit der per serieller Kommunikation das Gerät gesteuert werden kann und Rohdaten sowie berechnete Messwerte ausgelesen werden können. Diese Funktionen sollen automatisch im Messablauf ausgelöst werden, so dass keine Interaktion von Nutzer:innen mit dem Messgerät nötig ist.

— Steuerung des fruitcore HORST900-Roboter

Die Software soll den vorhandenen *fruitcore HORST900* Roboter steuern, um die händische Einstellung des Messgerätes in der Mitte des Lichtkegels zu übernehmen und außerdem eine räumliche Messung zu ermöglichen. Dafür soll der Roboter in der horizontalen Achse annähernd ideale, berechnete Positionen im Raum anfahren, die eine Messung der Homogenität des Lichtkegels ermöglichen.

— Befestigung des Spektrometers am Roboter

Zur Umsetzung des vorherigen Punktes ist die Befestigung des Spektrometers am Roboter nötig. Dafür ist eine speziell konstruierte Halterung für diesen Zweck notwendig, die mit Hilfe eines 3D-Druckers produziert werden soll.

— Protokollierung der Messdaten

Die Messdaten sollen in einem einheitlichen Format protokolliert werden, das benutzerfreundlich einsehbar ist, jedoch ebenso möglichst viele Rohdaten abspeichert. Dafür soll bei jeder Messung auch das Spektrum gespeichert werden, um eine Weiterverarbeitung der Daten unabhängig der vom Spektrometer errechneten Daten zu ermöglichen. Die angedachte Zielgruppe für die Messdaten sind Student:innen.

— Visualisierung der Messdaten

Die Messdaten sollen automatisch in einfachen, übersichtlichen Grafiken visualisiert werden. Außerdem sollen diese generierten Grafiken noch anpassbar sein, um Informationen noch nachträglich hinzufügen und Wertebereiche der Visualisierung anpassen zu können.
Die Grafiken sollen außerdem simpel in verschiedene Formate exportierbar sein.

3 Hardwareanalyse

Im folgenden Kapitel wird die vorliegende Hardware genauer betrachtet und überprüft, ob die Anforderungen aus dem vorhergehenden Kapitel mit den Spezifikationen der Hardware umsetzbar sind.

3.1 Gossen MAVOSPEC BASE

Das MAVOSPEC Base von dem Hersteller Gossen ist ein Spektrometer zur professionellen Messung von photometrischen Daten. Es ist durch seine Größe von 39x60x30mm, dem Gewicht von 150g sowie dem verbauten Bildschirm und Akku auch für den mobilen Einsatz gedacht. Der Sensor des MAVOSPEC BASE ist ein CMOS-Sensor mit 256 Pixeln (Gossen 2021a, S. 35–36).

Abbildung 3.1: Gossen MAVOSPEC BASE (Gossen 2022)

Der messbare Spektralbereich ist für 380-780 nm definiert und deckt so das gesamte VIS-Spektrum ab (Baer, Barfuß und Seifert 2020, S. 24). Der Bereich der messbaren Beleuchtungsstärke ist für 10 bis 100 000 lux angegeben, der Bereich der ähnlichsten Farbtemperatur (CCT) von 1600 bis 50 000 K, der Abstand des Farborts zum plank'schen-Kurvenzug (Duv) in diesem Bereich mit $\geq$ -0,1 (Gossen 2021a, S. 35). Diese Werte sind für den Einsatzzweck der Vermessung von Scheinwerfern ausreichend. Die photometrischen Werte professioneller Scheinwerfer liegen in der Regel deutlich innerhalb dieser Grenzbereiche.

Das MAVOSPEC BASE nutzt zur Kosinuskorrektur einen diffusen Zylinderaufsatz mit einer Lichteintrittsfläche von 38,48 mm² zur Überbewertung der seitlichen Lichtanteile sowie eine zusätzliche radiale Abschattung zur Kompensierung des künstlichen Fehlers (Baer, Barfuß und Seifert 2020, S. 169–171). Gossen gibt die Fehlergrenze der Kosinuskorrektur mit < 3% an.

Die Fehlertoleranzangaben des Herstellers bei Normlichtart A (2856 K bei 1000 lux) finden sich in folgender Tabelle:

Messunsicherheit Beleuchtungsstärke	± 3%
Messunsicherheit Farbort	/
Reproduzierbarkeit Farbort	± 0,0005%
Messunsicherheit CCT	± 2%
Messunsicherheit TM30	± 1,5%
Messunsicherheit CRI	± 1,5%
Messunsicherheit Flicker	± 1,5%

Tabelle 3.1: Herstellerangaben Fehlertoleranzen - Gossen MAVOSPEC BASE (Gossen 2021a, S. 35)

Das MAVOSPEC BASE bietet eine Mini-USB 2.0 Schnittstelle zur Verbindung mit einem Computer. Dort meldet sich das Gerät als „USB composite device", einem USB-Gerät, das mehr als eine Verbindung mit dem Betriebssystem aufbaut. So meldet sich das Gerät gleichzeitig als Wechselspeicher zur Übertragung von auf der microSD-Karte gespeicherten Messdaten, als auch als HID (Human Interface Device) sowie als CDC-Device (Communications Device Class) zur Ansteuerung und Abfrage des Messgeräts.

Zur Steuerung per Software wird in dieser Arbeit das CDC Protokoll als serielle Schnittstelle verwendet. Da das CDC-Protokoll als serielle Kommunikation auf UART basiert, sind einige

Einstellungen nötig, die von Gossen definiert sind (Gossen 2021b, S. 11) und von der Software übernommen werden müssen, um eine Kommunikation aufzubauen:

Baudrate	9600 Bits / s
Datenbits	8
Stopbits	1
Paritätskontrolle	/
Flusskontrolle	/

Tabelle 3.2: UART Einstellungen - Gossen MAVOSPEC BASE (Gossen 2021b, S. 11)

Per CDC-Protokoll ist so eine komplette Steuerung des Messgeräts möglich, in dem definierte ASCII-Commands als Strings seriell an das Spektrometer gesendet werden und eine Antwort mit definierter Länge erwartet wird, die wiederum seriell ausgelesen und im nächsten Schritt zur Weiterverarbeitung geparsed werden kann.

3.2 fruitcore robotics HORST900

Der HORST900 ist ein 6-Achsen Industrieroboter des Herstellers fruitcore robotics. Der Roboter hat einen Arbeitsbereich von 1265 x 905 x 1810 mm, benötigt eine Aufstellfläche von 380 x 380 mm und wiegt 55kg. Die maximale Reichweite beträgt 905 mm. Die maximale Traglast gibt der Hersteller mit 5 kg an, die Nennlast mit 3 kg. (fruitcore robotics GmbH 2022b, 1 ff.)

Abbildung 3.2: fruitcore robotics HORST900 (fruitcore robotics GmbH 2021)

Die Wiederholgenauigkeit von ± 0,05 mm und die Geschwindigkeit von bis zu 1,6 m/s bieten sich für den geplanten Einsatzzweck besonders an, um zielsicher und schnell die geforderten Messpositionen anzufahren und bei wiederholten Messungen die Vergleichbarkeit zu gewährleisten.

HORST900 bietet folgende Bewegungsbereiche (fruitcore robotics GmbH 2022b, S. 2):

Achse	Bewegungsbereich
1	± 176°
2	+85,5° / -16°
3	+42° / -67°
4	± 171°
5	± 119 °
6	± 300°

Tabelle 3.3: Bewegungsbereiche Achswinkel - HORST900

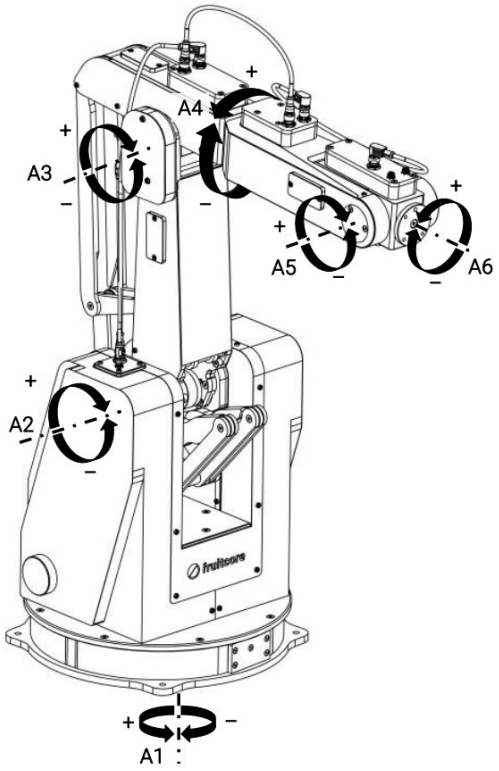

Abbildung 3.3: Roboterachsen - Horst900 (fruitcore robotics GmbH 2022a, S. 21)

Da für die Messungen das Spektrometer immer in die Richtung des Scheinwerfers ausgerichtet sein muss, kann diese maximale Reichweite nicht komplett ausgenutzt werden und die maximale horizontale Spannweite begrenzt sich auf ca. 1200 mm. Dies ist folglich auch der maximale Durchmesser des Lichtkegels, der bei einer Messung der Homogenität erreicht werden kann.

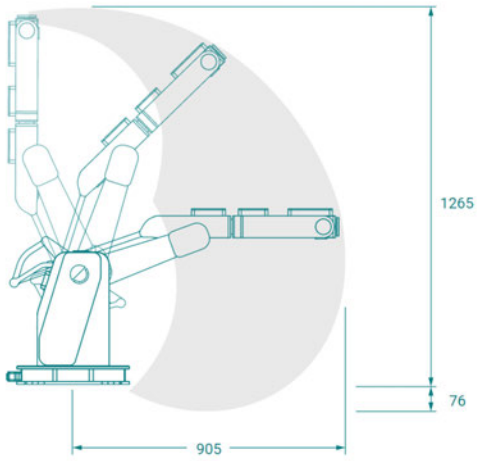

Abbildung 3.4: Bewegungsbereich Vertikal (fruitcore robotics GmbH 2022b, S. 6)

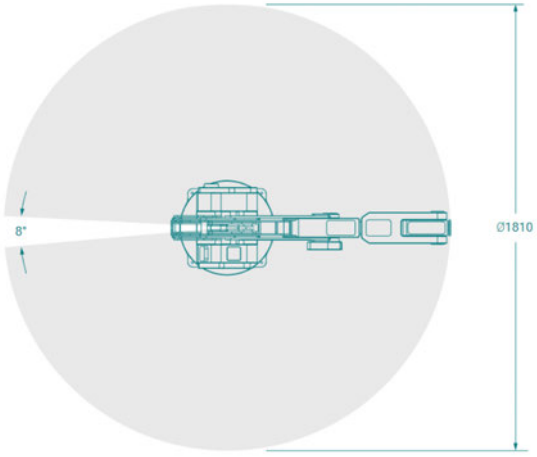

Abbildung 3.5: Bewegungsbereich Horizontal (fruitcore robotics GmbH 2022b, S. 6)

Die Ansteuerung des HORST900 ist auf verschiedene Arten möglich. Die einfachste Weise ist die Steuerung per herstellereigener Programmierumgebung „HorstFX“, die sowohl auf dem angeschlossenen Touch-Steuertableau läuft, als auch auf externen Computern genutzt werden kann. Diese Steuerung ist jedoch proprietär und ein Zugriff aus einer externen Programmiersprache auf die Schnittstelle ist so nicht verfügbar.
Für eine externe Steuerung des Roboters bietet fruitcore robotics neben *Modbus* und *Profinet* die Ansteuerung per *„Extensible Markup Language Remote Procedure Call“*, kurz *XML-RPC* an. Dieses programmiersprachenunabhängige Protokoll als Kommunikationsformat eignet sich aufgrund seiner Einfachheit und der stabilen Implementierung in viele Programmiersprachen sehr gut für die Kommunikation mit kritischer Hardware.
XML-RCP nutzt XML als Dokumentenformat zur Übermittlung der Steuerdaten und Antworten zwischen Software und Roboter. Der HORST900 fungiert dabei als Server, die Software als Client. Die Übertragung selbst funktioniert dabei per HTTP, genauer per POST-Requests als HTTP-Anfragemethode (Abts 2010, S. 205–206).

Struktur-Beispiel - Client:

```
<?xml version="1.0" encoding="UTF8"?>
<methodCall>
  <methodName>echo.getEcho</methodName>
  <params>
    <param>
      <value> Hello </value>
    </param>
  </params>
```

```
</methodCall>
```

Listing 3.1: XML-RPC - HTTP-POST-Request Client

Der Client fügt ein „getEcho“ hinzu und fordert so den Server zu einer Antwort auf.

Struktur-Beispiel - Antwort Server:

```
<?xml version="1.0" encoding="UTF8"?>
<methodCall>
  <params>
    <param>
      <value> World </value>
    </param>
  </params>
</methodCall>
```

Listing 3.2: XML-RPC - HTTP-POST-Request Server

So können auch komplexere Befehle einfach und schnell per Ethernet-Verbindung übertragen werden und der Roboter kann Bestätigungen, Positionsdaten oder Fehlermeldung ebenso simpel zurück an den Client melden. Voraussetzung ist dabei lediglich, dass Client und Server sich im selben Netzwerk befinden. Die IP-Adressen des Clients und des Roboters müssen dementsprechend angepasst werden.

3.2.1 Halterung

Zur Befestigung des Spektrometers an dem Roboter ist eine speziell konstruierte Halterung notwendig.

Der Roboter bietet dafür eine Werkzeugaufnahme mit vier metrischen M6-Gewinden in einer 63 mm Grundplatte (fruitcore robotics GmbH 2022b, S. 5).

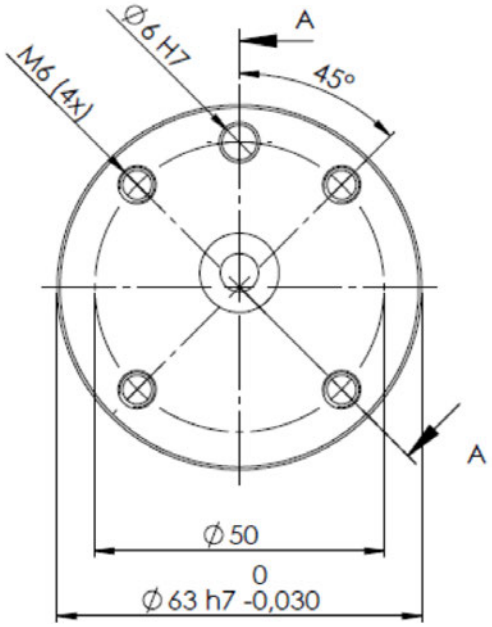

Abbildung 3.6: Werkzeugaufnahme-Flansch - HORST900 (fruitcore robotics GmbH 2022b, S. 5)

An diesen Flansch kann mit M6-Schrauben eine Halterung für das MAVOSPEC BASE geschraubt werden, die das Messgerät stabil und gerade direkt am Roboter befestigt. Da das MAVOSPEC BASE keine Schraubverbindung bietet, muss die Halterung das Messgerät festklemmen. Dabei soll das Gerät weiter bedienbar bleiben, um spezielle Einstellungen am Gerät selbst vornehmen zu können. Außerdem soll die Fertigungsart der Halterung möglichst genau sein, um eine sehr stabile Position des Messgeräts während der Bewegungen des Roboters zu gewährleisten.

Für die Fertigungsart bietet sich dementsprechend 3D-Druck an, da bei dieser Methode die Genauigkeit der Maße bei etwa 0.1 mm liegt und auch die Kosten der Fertigung sehr niedrig bleiben. Bei einem Preis von etwa 20€ für ein Kilogramm Plastikfilament und einem Gewicht des Drucks von etwa 110g liegt der Fertigungspreis bei 2,20€.
Die Halterung muss für den Druck virtuell in einer CAD-Software konstruiert werden und kann daraufhin mit einem beliebigen Plastikfilament gedruckt werden. Polylactide (PLA) bieten sich hier wegen der Einfachheit der Verarbeitung und den geringen Kosten an. Die Stabilität von PLA ist für den Einsatzzweck ausreichend.

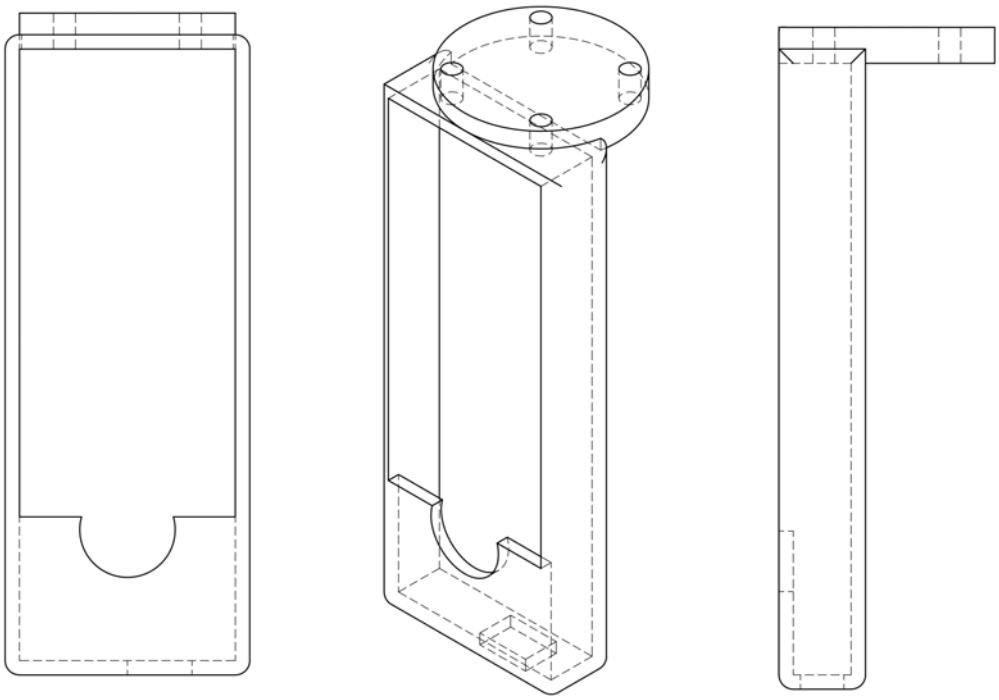

Abbildung 3.7: Front-, Perspektiv- und Seitenansicht - Halterung Spektrometer

Die Halterung ist so konstruiert, dass das Messgerät lediglich von oben eingeschoben werden muss und danach fest in der Halterung sitzt. Dafür entsprechen die Innenmaße der Halterung den Maßen des Messgerätes, um einen gewissen benötigten Kraftaufwand beim Einsetzten des Messgerätes zu erreichen. Durch die leichte Flexibilität von PLA kann so das Messgerät sehr stabil in die Halterung gesetzt werden und ein Verrutschen ausgeschlossen werden.

Um Einflüsse der Halterung auf Messergebnisse zu minimieren und eine korrekte Kosinuskorrektur zu gewährleisten, muss der zylindrische Diffusor inklusive Abschattung vor der Halterung selbst sitzen. Daher wurde die Halterung so konstruiert, dass das Messgerät direkt unter der vorderen Kante der Aufnahme und der vorstehende Messkopf frei vor der Halterung sitzt.

Umgesetzt wurde die CAD-Zeichnung in *Autodesk Fusion 360*.
Im Anhang finden sich eine bemaßte Zeichnung(6.1) und Fotos des 3D-Drucks (6.2).

3.3 DMX-Protokoll

DMX-512

Publiziert wurde DMX-512 erstmals 1986. In den USA bereits 1990 verbreitet, entwickelte sich *DMX-512/1990* Ende der Neunzigerjahre auch in Deutschland als Norm *DMX-512/DIN-56930* zum Standard für Scheinwerferansteuerung (Greule 2021, S. 175 ff.).

DMX-512 ist ein Steuerprotokoll zur Ansteuerung von 512 Kanälen mit jeweils einem 8-Bit Wert (0-255). Bei der Übertragung eines DMX-Pakets werden dabei jeweils folgende Sequenzen übertragen:
„Break (logisch 0) – Mark After Break (logisch 1) – Startbyte (Wert 00) – Kanal 1 (1 Byte) - … - Kanal 512 (1 Byte)“ (Greule 2021, S. 175 ff.).

Übertragen wird DMX-512 nach Norm über 5-polige XLR-Stecker. In der Praxis finden sich jedoch oft auch die aus der Tontechnik bekannten 3-pol XLR-Stecker zur Analog-Übertragung, jedoch mit invertiertem Gender für In- und Outputs. Die Übertragungsrate von DMX-512 ist mit 250 Kbit/s und einer Übertragungsrate von 44,1 Hz im Vergleich zu modernen Protokollen sehr langsam. Dies ermöglicht jedoch eine sehr ressourcensparende und somit günstige Implementierung in Soft- und Hardware.

IP-DMX-Protokolle

Für die Übertragung von DMX-kompatiblen Daten wurden seit der Jahrtausendwende verschiedene Ethernet-basierte Protokolle entwickelt, die mehrere sogenannte „DMX-Universen“ übertragen können. Diese Protokolle können als Container-Format viele DMX-Universen je 512 Kanälen übertragen und sind somit bedeutend leistungsstärker als das ursprüngliche DMX-512 Protokoll.

Mit dem TCP/IP-Protokoll *ArtNet* sind so beispielsweise 32768 DMX-Universen gleichzeitig möglich, das UDP-Protokoll sACN erlaubt sogar 63999 simultane DMX-Universen (Greule 2021, S. 175 ff.).

USB-DMX512-Interface

Für die Implementierung des DMX-Protokolls bietet sich ein USB-DMX512-Interface an, vorliegend ist ein *„eurolite USB-DMX512-PRO Interface“*. Dieses meldet sich als serielle Schnittstelle

beim Betriebssystem und kann so mit einer beliebigen Programmiersprache angesteuert werden, um DMX-Daten direkt auf ein XLR-f Anschluss zu geben. Diese USB-DMX512-Interfaces nutzen in der Regel zwei verbaute Chips: Einen FTDI-Chip zur seriellen USB-Kommunikation mit dem Computer und außerdem einen, von dem FTDI-Chip angesteuerten, ST485-Chip zur Ausgabe des DMX512-Protokolls.

4 Realisierung

In dem folgenden Kapitel wird die Umsetzung der Anforderungen und die Implementierung der Hardware in eine funktionsfähige Software behandelt. Dabei wird nicht der gesamte Programmcode betrachtet, sondern sich auf speziellere Abschnitte beschränkt. Zu diesen wird teilweise die Herleitung erklärt und auf die theoretischen Grundlagen eingegangen. Außerdem werden die Gedankengänge zu der Benutzeroberfläche und der Darstellung der Messdaten erläutert. In dem folgenden Beginn des Kapitels wird zudem kurz auf den technischen Aufbau, die Programmiersprache und die verwendete Entwicklungsumgebung eingegangen.

4.1 Technischer Aufbau

Der Aufbau und die Verkabelung zur Umsetzung der Anforderungen sieht wie folgt aus:

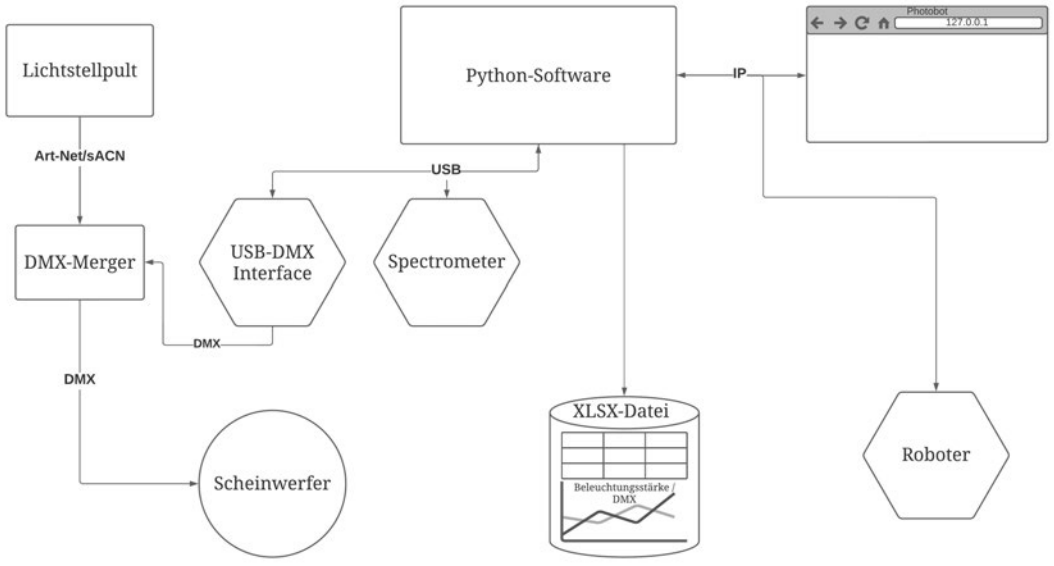

Abbildung 4.1: Grafik - Technischer Aufbau

Die Python-Software läuft dabei auf einem Computer mit Windows-Betriebssystem, der mit

USB-Schnittstellen und einem Ethernet-Port ausgestattet ist. Statt dem Lichtstellpultes ist auch eine Lichtstellsoftware auf dem Computer zur Steuerung des Scheinwerfers möglich. Wird in diesem Fall ArtNet als IP-DMX-Container Protokoll verwendet sollte sich das LAN-Netzwerk in der IP-Range von 2.x.x.x oder 10.x.x.x befinden. In diese IP-Range müssen dementsprechend auch Computer sowie Roboter konfiguriert werden, um eine Kommunikation zu ermöglichen. Insgesamt ist der technische Aufbau bewusst möglichst simpel gehalten und erlaubt eine schnelle Verkabelung mit handelsüblichen CAT- sowie DMX-Kabeln. Benötigt wird lediglich noch ein einfacher Netzwerkswitch, wenn der Computer sich gleichzeitig mit Lichtstellpult und Roboter in einem LAN befinden soll.

4.2 Programmiersprache

Python ist eine interpretierte und objekt-orientierte Programmiersprache (*General Python FAQ — Python 3.10.4 documentation* 20.05.2022), die 1991 von dem niederländischen Softwareentwickler Guido van Rossum entwickelt wurde. Es setzt dabei statt auf Klammern auf eine Syntax, die die Einrückungen interpretiert und diese somit Teil der Logik sind. So werden beispielsweise Schleifen durch eine Einrückung begonnen und durch Ausrücken wieder beendet, was eine gute Lesbarkeit der Programmiersprache ermöglicht. Als höhere Programmiersprache hat Python viele komplexere Funktionen bereits eingebaut und erlaubt so eine effiziente Programmierung. Python ist in der Forschung und speziell im MINT-Bereich sehr beliebt und verbreitet. Daher existieren viele herunterladbare Bibliotheken, die Python nochmal deutlich funktionsreicher machen und die bei der Umsetzung dieser Arbeit genutzt werden können.

4.3 Verwendete Entwicklungsumgebung

Die Entwicklungsumgebung der Wahl ist die IDE *„PyCharm“* des Softwareherstellers *„JetBrains“* in der Softwareversion *„2021.2.2“*. Als mir bereits vertraute Entwicklungsumgebung für Python ist *„PyCharm“* die naheliegendste Lösung, die mir einen schnellen Projektstart ermöglicht und außerdem weitere Features bietet, die ein zügiges Entwickeln ermöglichen. So unterstützt die IDE das Webframework *„Flask“* nativ und kann die Projektstruktur dafür automatisch vordefinieren. Besonders hilfreich ist außerdem die sehr einfache Integration von externen Bibliotheken. So kann mithilfe weniger Klicks eine Bibliothek in einer gewählten Softwareversion installiert und eingebunden werden. Die Bibliotheken werden dabei nicht

global, sondern lediglich in der Projektstruktur des jeweiligen Projektes abgelegt und nur lokal in diesem Projekt verwendet. Dies erlaubt eine aufgeräumte Entwicklungsumgebung zu Beginn der Entwicklung und verhindert Komplikationen durch vorherige Python-Projekte in derselben Umgebung. Zusätzlich bietet *„PyCharm“* diverse Tools zur Unterstützung der programmierenden Personen wie beispielsweise automatische Codevervollständigungen, Syntaxkorrektur und einen umfassenden Debugger.

4.4 Verwendete Bibliotheken

Für die Realisierung der Software wurden verschiedene öffentlich verfügbare Python-Bibliotheken verwendet. Im folgenden Abschnitt sollen diese benannt und Ihre Funktionen kurz erklärt werden.

4.4.1 Flask

https://github.com/pallets/flask

„Flask“ ist ein WSGI-Web-Framework (Pallets 30.05.2022). WSGI steht dabei für „Web Server Gateway Interface“. Dieser Spezifikation standardisiert die Kommunikation zwischen Webserver und Webapplikation in Python (*What is WSGI? — WSGI.org* 29.01.2021). „Flask“ basiert auf der Bibliothek „Werkzeug“ und der Template-Engine „Jinja2“. Dabei fungiert die Bibliothek „Werkzeug“ als WSGI-Implementierung, während „Jinja2“ die HTML und CSS-Verarbeitung übernimmt.

Einfach ausgedrückt kümmert sich „Werkzeug“ um die Implementierungen der WSGI - Spezifikationen, um einer in Python entwickelten Webanwendung die Kommunikation mit einem Webserver wie „Apache“ oder „Nginx“ zu ermöglichen. Diese Webserver kümmern sich dabei um die komplexe Kommunikation mit den Browser-Clients, so dass diese unabhängig von der Anwendung gehandhabt werden kann und in die Entwicklung einer Webanwendung nicht mit einfließen muss. Zur freien Auswahl des Webservers gibt es den WSGI-Standard und die Integration von „Werkzeug“ in „Flask“. Die Template-Engine „Jinja2“ setzt wiederum die Integration von HTML und CSS in die Webanwendung um. So können in der darzustellenden HTML-Seiten Variablen dargestellt werden und User-Eingaben als Variablen wieder abgespeichert werden, um hierdurch eine Interaktion der Nutzer:innen mit der Anwendung über die Webseite zu ermöglichen.

„Flask“ kombiniert diese Funktionen und erlaubt damit die Entwicklung von Webanwendungen, die dann auf einem Webserver gehostet werden können. Außerdem beinhaltet „Flask“ selbst einen rudimentären Webserver, der für die Entwicklung genutzt werden kann.

Folgender Programmcode ist ein Minimalbeispiel:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World"

if __name__ == "__main__":
    app.run()
```

Listing 4.1: Flask Minimalbeispiel

4.4.2 pySerial

https://github.com/pyserial/pyserial

Zur Kommunikation mit seriellen Schnittstellen wie die des MAVOSPEC BASE bietet sich die Python-Bibliothek „pySerial“ an. Diese Bibliothek ermöglicht die Implementierung der seriellen Kommunikation in Windows, Linux, OSX und BSD an und unterstützt alle nötigen Einstellungen, um mit diversen seriellen Schnittstellen kommunizieren zu können. Einstellbar sind etwa die Bytegröße, die Definition von Stopbits, Parität und Flusskontrolle, die Baudrate und Timeout-Zeiten. Dies ermöglicht eine einfache Umsetzung der Vorgaben aus SDKs der Hersteller und eine zuverlässige sowie einfache Implementierung.

4.4.3 NumPy

https://github.com/numpy/numpy

„NumPy“ ist eine weitverbreitete Python-Bibliothek zur Verarbeitung von Datenstrukturen wie Matrizen und Arrays sowie numerischen Berechnungen. „NumPy“ ist Open-Source und wird gemeinschaftlich von der Community weiterentwickelt. Die Bibliothek bietet viele nützliche Funktionen für den Umgang mit Datensätzen, die in Form von Arrays abgespeichert werden. So

können Arrays nach Definitionen erstellt, verändert und kombiniert und miteinander beliebig verrechnet werden. Außerdem bietet „NumPy“ mathematische Funktionen wie Sinus und Cosinus und deren Umkehrfunktionen an.

4.4.4 Math

https://docs.python.org/3/library/math.html

Integriert in Python ist bereits das Modul „math“, welches ebenso einige mathematische Grundfunktionen für Python liefert und außerdem Konstanten wie π bereitstellt. Zusätzlich unterstützt das Modul einfachere Umrechnungen wie etwa von Winkeleinheiten und andere einfachere mathematische Operationen.

4.4.5 threading

https://docs.python.org/3/library/threading.html

Ein weiteres Standard-Modul in Python ist „threading“. Diese Bibliothek erlaubt es einen weiteren „Thread“ zu starten, also einen weiteren Ausführungsstrang von Programmcode neben dem Hauptprogrammcode auszuführen. Dies ermöglicht es Teile des Programmcodes unabhängig von Lauf- und Wartezeiten anderer Teile des Programmcodes laufen zu lassen und beispielsweise parallel zu längeren Berechnungen Kommunikation auszuführen, die eine dauerhaften Datenfluss voraussetzt und so auf zeitlicher Ebene nicht flexibel ausgeführt ist.

4.4.6 time; datetime

https://docs.python.org/3/library/time.html https://docs.python.org/3/library/datetime.html

Die Python-Module „time“ und „datetime“ ermöglichen eine Einbindung der Zeit in Python. Mit diesen Modulen kann sowohl die aktuelle Uhrzeit als auch das aktuelle Datum in Variablen gespeichert werden. Außerdem können Umwandlungen des Formates sowie Berechnungen wie Addition oder Subtraktion von Zeit vorgenommen werden.

4.4.7 python-pause

https://github.com/jgillick/python-pause

Die Bibliothek „python-pause“ ermöglicht es den Programmcode an einer exakten Stelle anzuhalten und nach einer definierten Zeit wieder weiterlaufen zu lassen. Unterstützt werden verschiedenen Zeiteinheiten und Formate. Der Entwickler gibt eine Soll-Genauigkeit von 1 ms an. Dies kann jedoch von System zu System minimal unterschiedlich sein. Diese Genauigkeit ist für den Anwendungszweck jedoch ausreichend.

4.4.8 XlsxWriter

https://github.com/jmcnamara/XlsxWriter

„XlsxWriter“ ist ein Python-Modul mit dem Dateien im XLSX-Format erstellt werden können. Das XLSX-Format ist bekannt geworden durch „Microsoft Excel“ und fungiert heutzutage als übliches Format für die meisten Tabellenkalkulationen. Das Dateiformat XLSX ist grundlegend eine ZIP-komprimierte Datei mit lediglich anderer Endung und einer festgelegten Dateistruktur innerhalb der ZIP-Datei (data2type GmbH 2022).

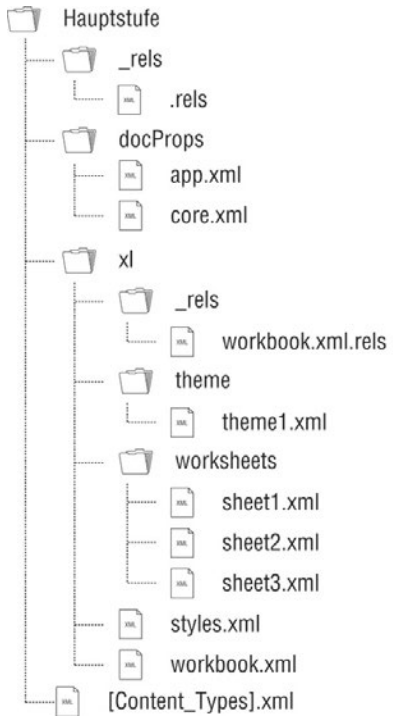

Abbildung 4.2: Datenstruktur XLSX-Format (data2type GmbH 2022)

Innerhalb dieser Ordnerstruktur werden die verschiedenen „Parts“-genannten Daten als XML-Format abgespeichert. Folgend ein Abschnitt aus einem Minimalbeispiel zur Speicherung von „1234“ in der Zelle A1:

```
[...]
<sheetData>
    <row r="1" spans="1:1" x14ac:dyDescent="0.3">
        <c r="A1">
            <v>1234</v>
        </c>
    </row>
</sheetData>
[...]
```

Listing 4.2: XML XLSX Minimalbeispiel

Die Bibliothek „XlsxWriter“ ermöglicht es per Python automatisch Dateien für Tabellenkalkulation zu erstellen und mit Variablen zu füllen. Folgend ein weiteres Minimalbeispiel:

```
import xlsxwriter

workbook = xlsxwriter.Workbook('Example.xlsx')
worksheet = workbook.add_worksheet()
worksheet.write('A1', 'Hello World')
workbook.close()
```

Listing 4.3: XlsxWriter Minimalbeispiel

Es lassen sich mit der Bibliothek zudem auch viele weitere Einstellungen wie etwa Zellenbreiten, -formatierungen oder Grafiken erstellen. Besonders die Grafikenerstellung ist sehr umfangreich und ermöglicht es, individualisierte Plots zu den abgespeicherten Daten zu generieren.

4.4.9 DmxPy

https://github.com/davepaul0/DmxPy

Die Ausgabe von einem DMX512-Signal (siehe Kapitel 3.3) über USB kann mit der Bibliothek „DmxPy“ realisiert werden. „DmxPy"basiert dabei auf PySerial. An die Bibliothek muss lediglich der serielle Port des USB-Adapters übergeben werden und mit wenigen Befehlen liegt ein DMX-Signal an dem Ausgang des Adapters an.

Folgend ein weiteres Minimalbeispiel:

```
from DmxPy import DmxPy

mxmx = DmxPy('serial port')
dmx.setChannel(channel, value)
dmx.render()
```

Listing 4.4: DmxPy Minimalbeispiel

4.5 Struktur der Software

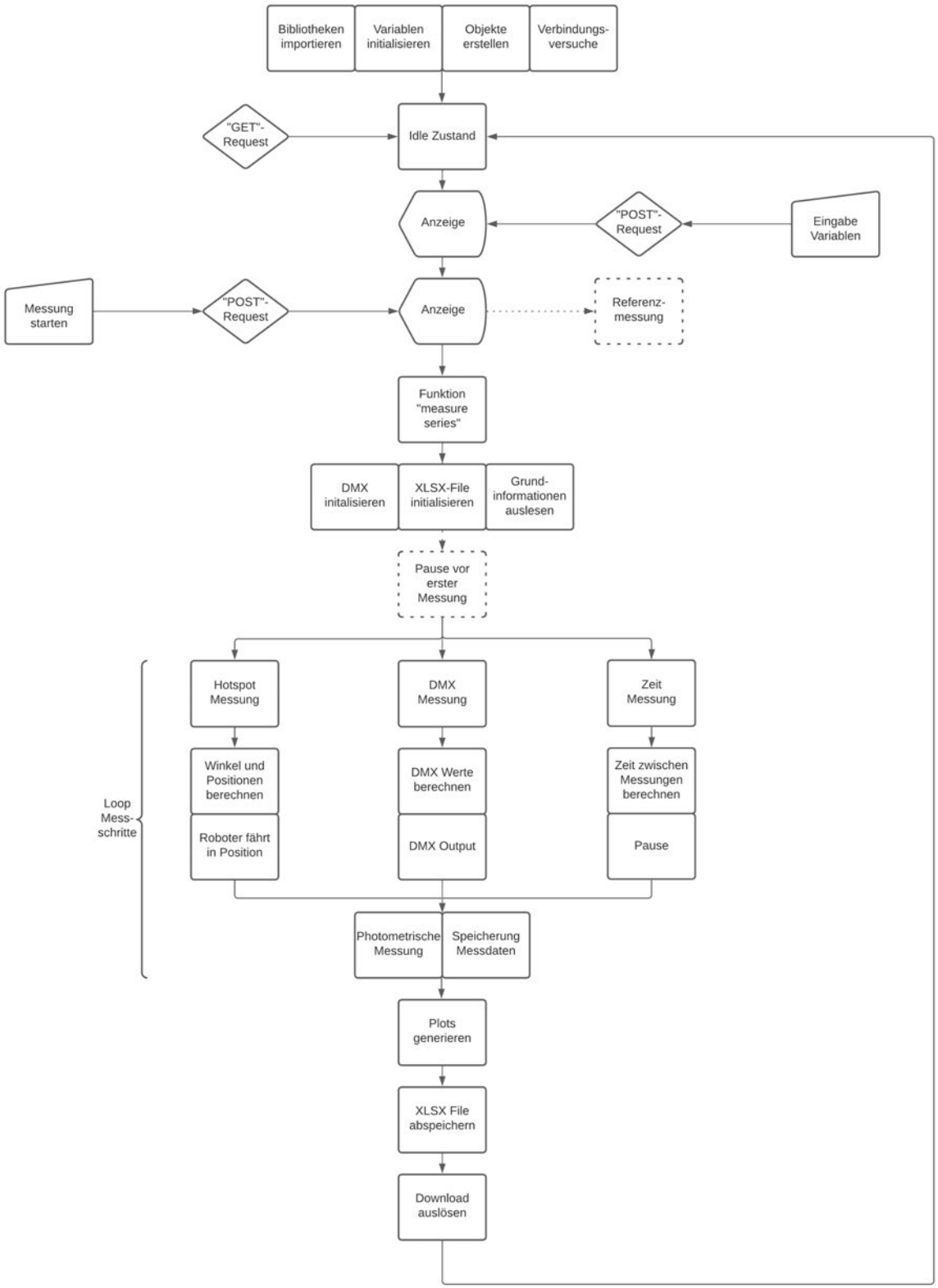

Abbildung 4.3: Flussdiagramm Softwarestruktur

Das in Abbildung 4.3 dargestellte Flussdiagramm stellt den Programmablauf vereinfacht dar. Gestrichelte Linien sind optionale Wege, die übersprungen werden können. Blöcke die direkt nebeneinander dargestellt sind, werden in der Reihenfolge von Links nach Rechts ausgeführt.

Der Programmablauf basiert auf einer typischen Struktur für Webanwendungen. Bei Start der Anwendung werden benötigte Bibliotheken importiert und alle benötigten Variablen und Objekte definiert. Nach erfolgreicher Verbindung mit der Hardware geht die Anwendung in einen Idle-Modus, in dem auf ein „GET“-Request durch das Aufrufen des Webservers aus einem Webbrowser heraus gewartet wird. Rufen die Nutzer:innen die Adresse des Webserver auf, antwortet die Anwendung mit einer Funktion, die die Index-Seite mit den vordefinierten Variablen übergibt. Geben Nutzer:innen nun auf der Webpage ihre Einstellungen ein und bestätigen diese, senden Sie ein „POST“-Request an die Anwendung und die Variablen werden umdefiniert. Auch das Auslösen der optionalen Referenzmessung und das Starten der Messreihe funktioniert mit einem „POST“-Request. Dieses startet die interne Funktion „measure_series“, die den Ablauf der Messreihen-Aufnahme anhand der Einstellungen der Nutzer:innen automatisch vornimmt.

Nach notwendigen Initialisierungen und dem Auslesen von grundlegenden Informationen, wie beispielsweise Uhrzeit, Sensortemperatur und Softwareversion des Spektrometers, beginnt eine Schleife, die sich anhand der eingestellten Messschritte beliebig oft wiederholt. Nach dem erfolgreichen Durchlauf der ausgewählten Messreihe generiert die Anwendung die grafischen Plots anhand der Messdaten und speichert diese in dem XLSX-File. Nach der Speicherung auf dem Server wird noch ein Download im Browser ausgelöst, mit dem die Datei an einem weiteren, Nutzer:innen-definierten Speicherort abgelegt werden kann.
Anschließend lädt die Anwendung erneut in den Ausgangszustand zurück, behält dabei jedoch die bereits getätigten Einstellungen der vorherigen Messung.

4.6 Softwareumsetzung

In den folgenden Unterkapiteln sollen verschiedene Funktionen und bestimmte Codeteile genauer betrachtet werden. Dabei werden sowohl theoretische Grundlagen als auch konkrete Umsetzungen betrachtet und erklärt.

4.6.1 Hardware-Detektion

Da die angeschlossene Hardware fest definiert ist und die Software die Verwendung bestimmter Geräte voraussetzt, sollen diese von der Software automatisch erkannt werden und die Nutzer:innen visuell darüber informieren, wenn eines der Geräte nicht korrekt von der Software erkannt wird.

Praktisch ist für die USB-Geräte hierbei die *Hardware-ID*. Diese Identifikationszeichenfolge wird vom Hersteller definiert und von den Geräten beim Anschluss an einen Computer an diesen kommuniziert. Die Betriebssysteme nutzen die Hardware-ID etwa für die Zuordnung von Treibern zu den jeweiligen Geräten. Diese automatische Erkennung erlaubt Plug-and-Play Lösungen, wie sie heute bei modernen Betriebssystemen üblich sind (Microsoft 2022).
Bei USB-Geräten, die sich als serielle Schnittstelle anmelden, ordnet Windows diesen einen sogenannten *COM-Port* zu. Ursprünglich bezeichneten diese die physikalischen RS-232-Schnittstellen eines Computers. Heute werden diese jedoch immer noch als interne Bezeichnung für serielle Geräte benutzt und ermöglichen einen kurzen und individuellen Namen für den Zugriff auf die angeschlossene serielle Hardware. Da die COM-Ports jedoch von Windows ohne ein bestimmtes Muster vergeben werden und dasselbe Gerät an einem anderen USB-Port möglicherweise einen anderen COM-Port zugeordnet bekommt, braucht es die Hardware-ID oder kurz *HWID*, zur genauen Zuordnung eines Gerätes zu einem COM-Port.
Diese HWID besteht aus zwei 16-Bit Zahlen, der *Vendor ID (VID)*, die dem Hersteller zugeordnet ist und der *Product ID (PID)*, die das Produkt definiert.

In Python kann eine automatische Zuordnung der COM-Ports zu gegebenen HWIDs mit folgender Funktion realisiert werden:

```
1 def findComPort(pid,vid):
2     com_port = "ERROR"
3     ports = list(serial.tools.list_ports.comports())
4 
5     for p in ports:
6         if vid and pid in p.hwid:
```

```
7               com_port = p.device
8
9       return com_port
10
11  dmx_com  = findComPort(pid="04D8",vid="FA63")
12  mavo_com = findComPort(pid="1CD7",vid="4001")
13
```

Listing 4.5: Funktion - Hardware-Detektion

Als Variablen nimmt die Funktion die VID und PID der Hardware entgegen. In Zeile 2 wird nun die return-Variable mit dem String ERROR initialisiert, so dass, falls das Gerät nicht gefunden werden kann, der Fehler eindeutig anhand der Variable erkennbar ist. In Zeile 3 wird nun eine Liste in der Variable *ports* gespeichert, die alle verfügbaren Informationen der angeschlossenen USB-Geräte als jeweils individuelles Objekt beinhaltet. Die Funktion nutzt dafür die PySerial-Bibliothek (4.4.2), die die Betriebssystem-Werkzeuge für den Zugriff auf USB-Geräte nutzt und in Python bereitstellt.

Diese Liste wird nun mit einer Schleife in Zeile 5 bis 9 nach der gegebenen HWID durchsucht. Da die Variable *ports* eine Liste von Objekten beinhaltet, können die Instanzvariablen dieser Objekte *p* durchsucht werden. Neben anderen Instanzvariabeln wie etwa *device, name, serial_number* oder *manufacturer*, besitzen die Objekte die Instanzvariable *hwid*, die beispielsweise bei angeschlossenem USB-DMX-Interface folgenden String speichert:

```
USB VID:PID=04D8:FA63 SER= LOCATION=1-9
```

Somit ist es möglich mit Zeile 6 eine einfache Suche nach diesen zwei 16-Bit Zahlen zu veranlassen, die bei einem Treffer in Zeile 7 die Instanzvariable *device* in der Variable *com_port* speichert und den zuvor gespeicherten String `Error` aus Zeile 2 überschreibt. Die Instanzvariable *device* beinhaltet einen String der Form *COMX*, wobei das *X* für eine Dezimalzahl steht, die vom Betriebssystem vergeben wurde. Dieser String wird danach in Zeile 9 zurückgegeben und die Funktion abgeschlossen. Zeilen 11 und 12 zeigen die Verwendung der Funktion. Die Werte der Variablen *dmx_com* und *mavo_com* können danach als Strings zur Erzeugung der Objekte für das USB-DMX-Interface und des Gossen MAVOSPEC BASE genutzt werden.

4.6.2 DMX-Output

Die Berechnung der DMX-Werte basiert auf der *linspace()*-Funktion aus der Numpy-Bibliothek (4.4.3). Diese erzeugt ein Array basierend auf drei Variablen: *start*, *stop* und *num*. Anhand dieser

Werte berechnet Numpy ein Array mit so vielen Elementen wie in der Variable *num* definiert, wobei die Werte zwischen *start* und *stop* linear interpoliert werden.
So führt folgendes Minimalbeispiel:

```
1 import numpy as np
2 lin_array = np.linspace(start=0,stop=10,num=11,endpoint=True)
3 print(lin_array)
```

Listing 4.6: *numpy.linspace()*-Minimalbeispiel

zu diesem Output:

```
>>> [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

Diese Funktion wird nun in Zeile 1 des folgenden Codeausschnitts zur Berechnung aller DMX-Werte der Messreihe verwendet:

```
1 dmx_arr = np.linspace(dmx_min, dmx_max, steps_count, endpoint=True)
2
3 if invertdmx: dmx_arr = dmx_arr[::-1]
4
5 if not dmx_16_bool:
6     dmx_val = round(dmx_arr[x])
7     if dmx_val > 255: dmx_val = 255
8     if dmx_val < 0: dmx_val = 0
9 if dmx_16_bool:
10    msb,lsb = convert_16to8bit(round(dmx_arr[x]))
11    dmx_val = msb
12    dmx_val_lsb = lsb
```

Listing 4.7: DMX-Array

Die nachfolgenden Codezeilen kümmern sich um die Verarbeitung der Werte. Zeile 3 fragt dabei die Bool-Variable *invertdmx* ab, die, wenn sie 1 bzw. `True` gesetzt wurde, dafür sorgt, dass durch die Anweisung `[::-1]` die Reihenfolge des Arrays invertiert wird. Somit starten die DMX-Werte nicht bei dem niedrigsten Wert *dmx_min*, sondern bei dem höchsten: *dmx_max*.

Zeile 5 und 9 fragen durch die Variable *dmx_16_bool* ab, ob von den Nutzer:innen eine 16-Bit Auflösung der DMX-Werte ausgewählt wurde. Ist dies nicht der Fall, springt die Software in Zeile 6, in der das Float-Element *x* des Arrays in die Variable *dmx_val* gespeichert wird. Die Variable *x* entspricht dabei dem jeweiligen Messschritt-1. So ist dieser Wert in dem ersten Messchritt gleich 0. Dementsprechend wird mit *dmx_arr[0]* der erste Wert des Arrays in die Variable *dmx_val* gespeichert. Die zwei nachfolgenden Zeilen 7 und 8 sind präventiver

Fehlerschutz, die bei einem Rundungsfehler Werte außerhalb des 8-Bit-Wertebereichs von 0 bis 255 korrigieren.

Wurde ein 16-Bit Wertebereich von 0 bis 65535 ausgewählt, so springt die Software in Zeile 10. Hier werden nun mit Hilfe der Funktion *convert_16to8bit()* die zwei Variablen *msb* (most significant byte) und *lsb* (least significant byte) berechnet, die als Darstellung für einen 16-Bit Wert aus zwei 8-Bit Werte dienen.

```
1 def convert_16to8bit(x, n_bytes=2, order='big'):
2     msb, lsb = x.to_bytes(n_bytes, byteorder=order)
3     return (msb, lsb)
```

Listing 4.8: Funktion - 16:8-Bit-Konvertierung

Diese sehr kleine Funktion nutzt die in Python implementierte Funktion *x.to_bytes()*, die einen Wert x in *n_bytes* viele Bytes aufteilt. Dabei gibt die *byteorder* die Reihenfolge von *msb* und *lsb* an. Bei einer *byteorder = 'big'*, ist das *„most significante byte“* an vorderer Stelle, bei *'little'* dementsprechend andersrum.
So erzeugt der Befehl *convert_16to8bit(32768, n_bytes=2, order='big'))* den Output
`>>> (128, 0).`

4.6.3 MAVOSPEC BASE - Bibliothek

Die Ansteuerung des Spektrometers wurde bereits in Kapitel 3.1 angeschnitten. So basiert die Kommunikation auf dem seriellen CDC-Protokoll, das selbst auf UART basiert. Zur Kommunikation werden definierte ASCII-Befehle an das Spektrometer gesendet, welches auf diese ebenso mit ASCII-Strings antwortet. Die Antwortlängen sind dabei fest definiert und werden für die korrekte Decodierung der Antworten benötigt. Diese Befehle und Antwortlängen sind von Gossen in dem herunterladbaren *Software Development Kit* (kurz SDK) dokumentiert und ermöglichen die gesamte Steuerung des Messgerätes.
Zur Umsetzung dieser Möglichkeiten in der Software lag die Programmierung einer eigenen Python-Bibliothek für das Gossen MAVOSPEC BASE nah. Diese ermöglicht die Nutzung der Schnittstelle auch in anderen Projekten und außerdem eine deutlich vereinfachte Syntax im Hauptprogramm. Dafür wird die Bibliothek in eine weitere *.py* Datei ausgelagert und über den `import` Befehl in die Software eingebunden.

Folgend findet sich ein Beispiel zur Nutzung der Bibliothek:

```
1  import mavospec_base as mavo
2
3  try:
4      mavo.Connect(mavo_com)
5      mavo_success = True
6  except:
7      print("ERROR: Cannot connect to Mavospec Base")
8      mavo_success = False
9
10 mavo.measure()
11
12 mes_cct = mavo.CCT()
13 CCT = mes_cct[0]
14 DUV = mes_cct[1]
15 E   = mes_cct[2]
```

Listing 4.9: MAVOSPEC BASE - Bibliothek - Beispiel

In diesem Beispiel wird in Zeile 1 die Bibliothek als Alias eingebunden. Dabei entspricht der Dateiname *mavospec_base.py* der Bibliothek und das importierte Modul wird *mavo* benannt. Daraufhin wird ab Zeile 3 mit dem Befehl `try` versucht, sich mit dem Spektrometer zu verbinden, indem die Funktion `Connect()` der Bibliothek ausgeführt wird.

```
1  def Connect(comPort):
2      global serialPort
3      serialPort = serial.Serial(port=comPort, baudrate=9600, bytesize=8,
       timeout=5, stopbits=serial.STOPBITS_ONE)
```

Listing 4.10: MAVOSPEC BASE - Connect-Funktion

Diese Funktion nutzt die *pySerial*-Bibliothek (4.4.2) für den Verbindungsaufbau und verwendet dabei die UART-Einstellungen aus der Tabelle 3.2 sowie den *comPort* aus Listing 4.5.
Ist der Verbindungsaufbau erfolgreich, so wird in Listing 4.9 Zeile 5 die Bool-Variable *mavo_success* auf *True* gesetzt. Schlägt der Aufbau fehl, so wird in Zeile 8 diese Variable dementsprechend auf *False* gesetzt. Diese Variable fungiert als Flag zur Kontrolle des Verbindungsstatus und ermöglicht es der Software Nutzer:innen über den Verbindungsstatus zu informieren. Die Funktion `measure()` in Zeile 10 löst nun die Messung aus. Die Funktion `CCT()` ruft nach durchgeführter Messung die aus dem Spektrum berechneten Messwerte ab und speichert sie in der Variable *mes_CCT*.

```
def CCT():
    command = "calculateddata? 02"
    replylength = 55
    return CDCCommand(command, replylength)
```

Listing 4.11: MAVOSPEC BASE - CCT - Funktion

Dabei sind die Funktionen nach einem festen Schema aufgebaut, bei denen der ASCII-Befehl in der Variable *command* und die Antwortlänge in der Variable *replylength* definiert werden. Diese werden nun in dem `return`-Befehl der Funktion direkt in die Funktion `CDCCommand()` übergeben.

```
def CDCCommand(command, replylength,):
    c = command
    sendC = c + "\r\n"
    serialPort.write(bytes(sendC, encoding='utf8'))
    time.sleep(0.1)
    x = serialPort.read(replylength).decode('utf-8')
    return x.split()
```

Listing 4.12: MAVOSPEC BASE - CDCCommand - Funktion

Diese Funktion übernimmt das Senden der CDC-Befehle und decodiert anschließend die Antworten des MAVOSPEC BASE. Dafür wird an den übergebenen Befehl mit „\r\n“ ein *Carriage Return* sowie ein *Line Feed* angehängt, welche in Windows als Zeilenumbruch und damit als Befehlsende interpretiert werden. In der darauffolgenden Zeile wird nun der Befehl codiert und gesendet. Die Antwort wird anschließend wieder decodiert, in die Variable *x* gespeichert und an die Funktion `split()` übergeben. Diese ermöglicht es in Python einen String in ein Array aufzuteilen. In Listing 4.9 Zeile 13 bis 15 werden die Messwerte nun aus diesem Array in einzelne Variablen zur weiteren Nutzung abgespeichert.

4.6.4 Homogenitätsmessung

Zur Messung der Homogenität des Lichtkegels kann eine „Hotspot“-Messung ausgelöst werden, bei der der Roboter mit dem Messgerät auf der horizontalen Achse benutzerdefiniert viele Messpunkte anfährt. Zur korrekten Beurteilung darf das Messgerät dabei jedoch nicht die Distanz zu dem Scheinwerfer verändern. Somit entspricht die Bewegung des Messgeräts keiner linearen auf der horizontalen Achse, sondern einem gedachten Kreis mit dem Lichtaustrittspunkts als Mittelpunkt. Dabei wird außerdem das Messgerät von dem Roboter so eingedreht, dass der Sensor immer exakt auf den Scheinwerfer ausgerichtet wird.

Die mathematische Grundlage zur Berechnung des Winkels und der Koordinaten im Raum bietet dabei eine Erweiterung der allgemeinen Kreisgleichung (Şanal 2015, S. 45):

$$(x - m_1)^2 + (y - m_2)^2 = r^2 \tag{4.1}$$

Angewandt auf die folgende Skizze und umgewandelt in die Parameterform der Kreisgleichung ergibt sich der Ausgangspunkt zur Aufstellung der Formeln, die in die Software die Berechnung der x und y-Koordinaten des Roboters ermöglichen.

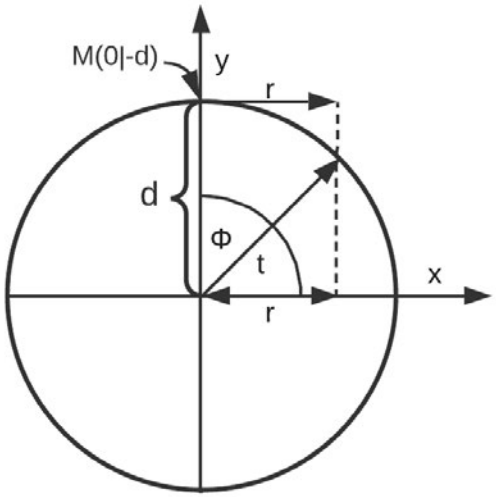

Abbildung 4.4: Grafik - Kreisgleichung

Der Mittelpunkt des Kreises $(0|0)$ ist die Position des Lichtsaustritts des Scheinwerfers. Die Distanz d ist der Abstand zu dem Messsensor und der Punkt M ist die Position des Messsensors, von dem die x und y-Koordinaten für den Roboter relativ bemessen werden. Dies ist dement-

sprechend der relative Nullpunkt des Roboters. r definiert die maximale Distanz der Messung des Lichtkegels auf der horizontalen Achse. t bezeichnet den Winkel, den die Parameterform der allgemeinen Kreisgleichung als Parameter benutzt, während Φ dem Winkel entspricht, um den der Roboter das Messgerät eindrehen muss, um das Messgerät auf den Scheinwerfer auszurichten.

Die Parameterform der erweiterten allgemeinen Kreisgleichung, mit eingesetzten Variablen aus Abbildung 4.4:

$$x = d \cdot \cos(t) + m_1 \tag{4.2}$$

$$y = d \cdot \sin(t) + m_2 \tag{4.3}$$

Nun kann Φ durch die Differenz von $\frac{\pi}{2}$ und t ausgedrückt werden:

$$\Phi = \frac{\pi}{2} - t \Leftrightarrow t = \frac{\pi}{2} - \Phi \tag{4.4}$$

Einsetzen von M und t führt zu:

$$x = d \cdot \cos(\frac{\pi}{2} - \Phi) \tag{4.5}$$

$$y = d \cdot \sin(\frac{\pi}{2} - \Phi) - d \tag{4.6}$$

Nun kann $x = r$ gesetzt werden und wie folgend die Formel nach Φ umgestellt werden:

$$r = d \cdot \cos(\frac{\pi}{2} - \Phi) \qquad | : d \tag{4.7}$$

$$\frac{r}{d} = \cos(\frac{\pi}{2} - \Phi) \qquad | : \arccos() \tag{4.8}$$

$$\arccos(\frac{r}{d}) = \frac{\pi}{2} - \Phi \qquad | - \frac{\pi}{2} \tag{4.9}$$

$$\arccos(\frac{r}{d}) - \frac{\pi}{2} = -\Phi \qquad | : (-1) \tag{4.10}$$

$$-\arccos(\frac{r}{d}) + \frac{\pi}{2} = \Phi \tag{4.11}$$

$$\Phi = \arccos(-\frac{r}{d}) - \frac{\pi}{2} \tag{4.12}$$

Mit der Formel 4.12 für das maximale Φ kann nun ein Array erstellt werden, dass Φ an der Achse spiegelt und lineare Abstände, abhängig von der gewählten Anzahl an Messpunkten, berechnet.

```
phi_array = np.linspace(-phi, phi, steps_count, endpoint=True)
```

Listing 4.13: Softwareumsetzung - Φ-Array

Durch die Berechnung des Φ-Arrays können nun jeweils pro Array-Eintrag auch die Werte für x und y berechnet werden, die von dem Roboter in dem jeweiligen Messschritt angefahren werden sollen.

Hierbei ist jedoch noch eine Übersetzung in das Koordinatensystems des HORST900 nötig.

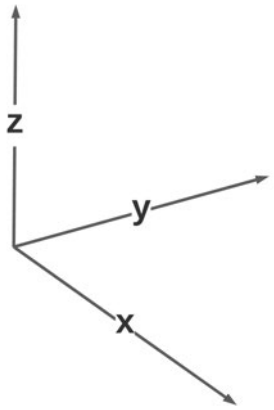

Abbildung 4.5: Koordinatensystem Roboter Horst900

Da die x und y-Achsen in der Draufsicht zu dem Koordinatensystem aus Abbildung 4.4 vertauscht sind, müssen die Formeln für x und y ebenso vertauscht werden.
Außerdem erzeugt die Formel 4.6 rein negative Werte außerhalb des gedachten Nullpunkts M. Der Roboter bewegt sich damit nicht in die Richtung des Scheinwerfers sondern spiegelverkehrt von dem Scheinwerfer weg. Dementsprechend ist für diese Formel noch eine Spiegelung an der nun x-Achse nötig.

Daraus resultieren die folgenden zu implementierenden Formeln:

$$x = -(d \cdot \sin(\frac{\pi}{2} - \Phi) - d) \tag{4.13}$$

$$y = d \cdot \cos(\frac{\pi}{2} - \Phi) \tag{4.14}$$

```
for i in (phi_array):
    x_rel = -(d * np.sin(math.radians(90 - i)) - d)
    x_array.append(np.round(x_rel, 3)

    y_rel = d * np.cos(math.radians(90 - i))
    y_array.append(np.round(y_rel, 3))
```

Listing 4.14: Softwareumsetzung - XY-Koordinaten

4.7 Benutzeroberfläche

Die Bedienung der Software funktioniert per Browser und einer klassischen HTML-Website. Mithilfe der „Flask“-Bibliothek und einem Webserver wird diese Webpage für Nutzer:innen im Webbrowser dargestellt und die Kommunikation zwischen Benutzereingaben und der Softwarelogik gehandhabt. So können Variablen von den Nutzer:innen händisch eingegeben und berechnete Werte den Nutzer:innen dargestellt werden.

In folgender Abbildung (4.6) ist die Oberfläche dargestellt und in Sektionen unterteilt:

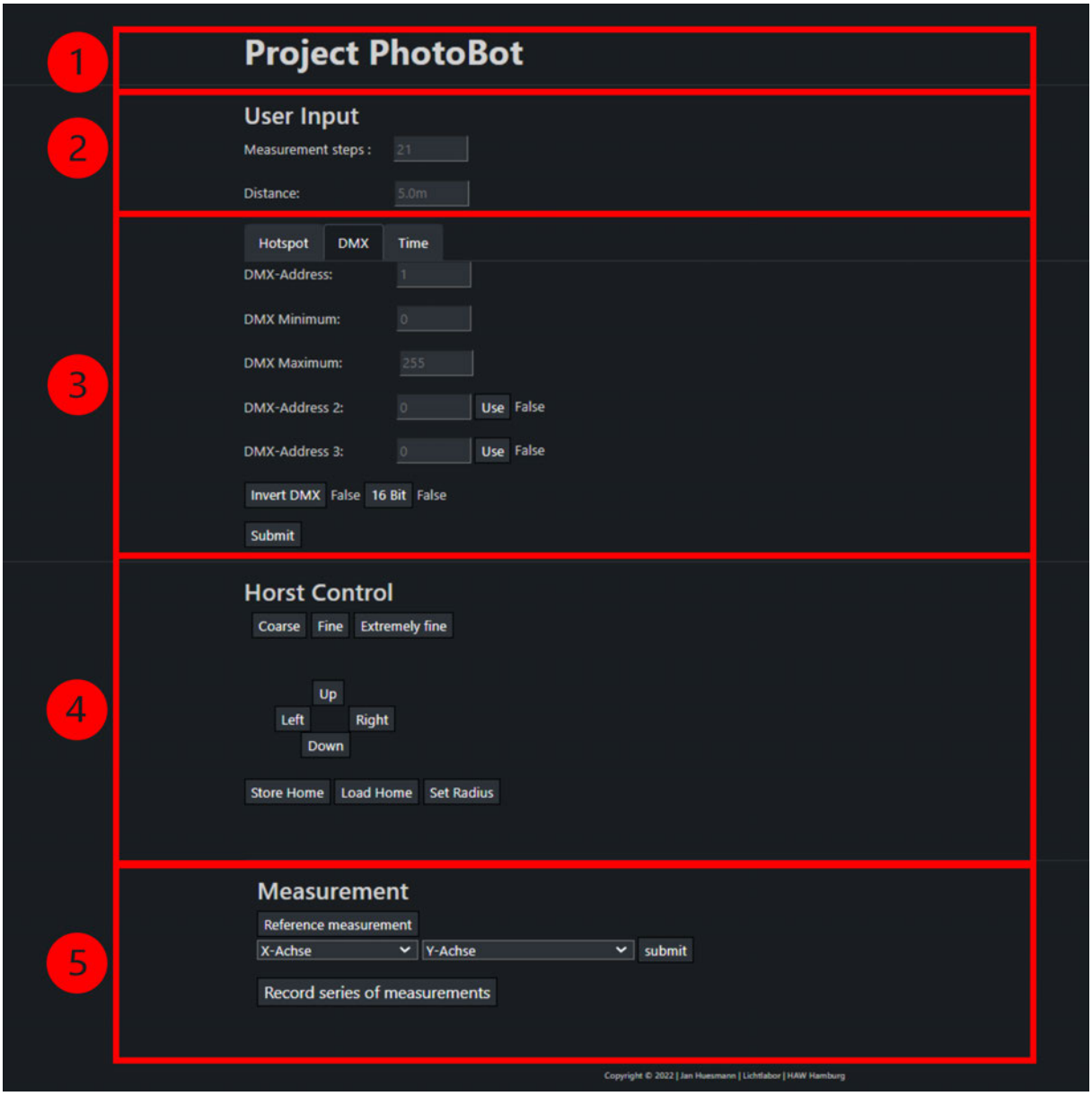

Abbildung 4.6: Screenshot - Benutzeroberfläche

Die grundlegende Farbgebung ist an moderne „Dark Modes“ angelehnt. Diese dunkle Farbgestaltung ist bei der Nutzung augenfreundlich und bietet sich bei der Nutzung in einer dunklen Umgebung zur Messung von Photometrie an. So werden Nutzer:innen nicht geblendet und Messdaten nicht durch streuendes Licht verfälscht. Der Aufbau ist bewusst simpel gehalten und ist in fünf Abschnitte zu unterteilen. Abschnitt ① ist dabei lediglich der Titel und dient mit seiner Größe als erster Orientierungspunkt für Nutzer:innen.

Direkt unter diesem Titel befindet sich in Sektion ② der Abschnitt mit den essenziellen Nutzer:innen-Eingaben. In dem Feld „Measurement steps“ wird die Anzahl der Messschritte eingestellt und in das Feld „Distance“ wird der Abstand des Lichtaustrittpunktes am Scheinwerfer zu dem Messkopf in der Einheit Metern eingetragen. Wie in allen anderen Eingabefeldern wird in diesen grau hinterlegt der aktuelle Wert der jeweiligen Variable angezeigt, bis Nutzer:innen selber einen neuen Wert in das Feld eingeben.

Abschnitt ③ ist selber noch einmal durch Tabs dreigeteilt. In diesen Untermenüs befinden sich spezifische Einstellungen zu den drei Messmethoden „Hotspot“, DMX“ und „Time“. Als Beispiel ist in Abbildung 4.6 der DMX Tab ausgewählt. Hier können etwa Einstellungen zu DMX-Adressen und dem minimalen und maximalen Wert gemacht werden. Außerdem bietet dieses Menü die Möglichkeit weitere DMX-Adressen hinzuzunehmen, um etwa RGB-Kanäle anzusteuern. „Invert DMX“ steuert die Zählrichtung der DMX-Werte. Wird diese Variable auf `True` gesetzt, so invertiert die Software die Reihenfolge der DMX-Werte und beginnt beim Maximum. Wird die DMX-Ausgabe in 16-Bit aktiviert, so wird automatisch die darauffolgende DMX-Adresse als zweiter 8-Bit Kanal genutzt und der Wertebereich ändert sich. Solche Informationen, ebenso wie beispielsweise ungültige Eingaben, werden den Nutzer:innen dabei auf zwei verschiedenen Arten mitgeteilt: *„Mouseover“ & „Alert“*.

Die Benutzeroberfläche basiert optisch auf dem freien CSS-Framework *„Bootstrap“* und dessen Javascript-Erweiterungen. Diese Javascript-Erweiterungen ermöglichen beispielsweise den in folgender Abbildung (4.7) sichtbaren „Mouseover“-Effekt:

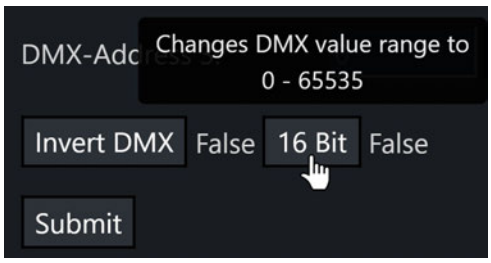

Abbildung 4.7: Screenshot - Mouseover

Diese „Mouseover“-Informationen dienen den Nutzer:innen als kleine Erläuterungen zu Funk-

tionen, ohne die Benutzeroberfläche mit Informationen zu überladen und unübersichtlich zu gestalten. Wichtige Informationen wie Fehlermeldungen und Statusmeldungen zu erfolgreichen Messungen werden mit „Alerts“ angezeigt:

Abbildung 4.8: Screenshot - Altert

Diese „Alerts“ verschieben Abschnitt ② nach unten und sind farblich hervorgehoben. Werden die Meldungen ausgeblendet, verschiebt sich die Benutzeroberfläche wieder zurück.

Abschnitt ④ dient der Steuerung des Roboters. Hier kann über die Buttons „Coarse“ (10 cm), „Fine“ (1 cm) und „Extremley Fine“ (0.1 cm) die Empfindlichkeit der darunterliegenden Richtungsbuttons eingestellt werden. Diese Richtungen bewegen das Messgerät am Roboter auf zwei Achsen in die Messposition. Über den *„Store Home“* Button wird die aktuelle Position gespeichert und als neuer Nullpunkt definiert. Alle Messungen finden nun an dieser Position statt, beziehungsweise wird bei einer „Hotspot“-Messung diese Position zur mittlere Messposition. Wird der Messkopf nun noch weiter bewegt, so kann über *„Load Home“* die Position wieder geladen werden und der Roboter bewegt sich an seinen internen Nullpunkt. *„Set Radius“* wird für die „Hotspot“-Messung benötigt. Für diesen Button wird der Roboter auf der horizontalen Achse bis an den Punkt bewegt, an dem der gewünschte maximale Wert für die Homogenitätsmessung erreicht ist. Nach Drücken des „Set Radius“-Button wird der aktuelle y-Wert als maximale Distanz gespeichert.

Im letzten Abschnitt „Measurement“ (⑤) wird die Messung durchgeführt. Hier kann über den obersten Button eine Referenzmessung ausgelöst werden, die eine zusätzliche Seite in der XLSX-Datei erzeugt und dort die Messergebnisse einer Referenzmessung im Dunkeln abspeichert. Über die beiden Dropdown-Menüs können die Messungen auf den Achsen ausgewählt werden. So ist die Auswahl der X-Achse identisch zu den Tabs in Abschnitt ③. Die Auswahl der Y-Achse umfasst die Messungen: „Beleuchtungsstärke, CCT, DUV“, „Farbkoordinaten“, „CRI“, „TM30“ und „TLCI“. So kann von den Nutzer:innen auf einfache Weise genau die Messung ausgelöst werden, die gewollt ist. Mit dem letzten Button „Record series of measurements“ wird die Messreihe gestartet und Nutzer:innen mit einem Alert über die geschätzte Messdauer informiert. Nach dem erfolgreichen Ende der Messreihe werden Nutzer:innen noch mit einem weiteren Alert über die Fertigstellung benachrichtigt.

4.8 Darstellung der Messdaten

Nach einer erfolgreichen Messung wird ein Download Fenster im Browser für die XLSX-Datei geöffnet und die Nutzer:innen können die Datei herunterladen.

Die ursprüngliche Planung der Arbeit sah eine MSQL-Datenbank auf einem externen Server vor, die die Daten aller Messungen abspeichert und mit der Python-Bibliothek „Matplotlib“ plottet. Diese Funktionalität war sogar bereits zum Großteil fertig entwickelt, bevor in den ersten Tests auffiel, dass diese Variante deutlich zu unflexibel für Dokumentationen ist, die von einem engen Standard abweichen. Da viele Nutzer:innen deutlich versierter in dem Umgang mit Tabellenkalkulation sind, ist diese Art der Messdaten-Speicherung deutlich praktikabler. Die aufgenommenen Messwerte können von anspruchsvolleren Nutzer:innen trotzdem noch in Python verarbeitet und mit Matplotlib visualisiert werden (siehe 6.2.2).

Diese Datei ist für die Ansicht und Bearbeitung mit einer Tabellenkalkulation wie beispielsweise „Microsoft Exel“ oder „LibreOffice Calc“ gedacht. Wird die Datei mit einer Tabellenkalkulation geöffnet, findet sich im ersten Tab das *Infosheet* mit grundlegenden Informationen zu der Messung. Der nachfolgende zweite Tab ist optional und speichert gegebenenfalls die Messwerte der Referenzmessung. Der dritte Tab beinhaltet die berechneten Messwerte des Spektrometers, während der vierte Tab die rohen Messwerte des Spektrums in 5 nm Schritten von 380 bis 780 nm abspeichert. Alle danach folgenden Tabs sind die automatisch generierten Plots zu den Messwerten. Dabei ist jeder Tab ein sogenanntes Diagrammblatt und beinhaltet keine Tabellen, sondern nur einen Plot.

Folgend in Abbildung 4.9 findet sich ein Beispiel, bei dem die Beleuchtungsstärke zu dem DMX-Wert des Dimmer-Kanals eines Robe Viva CMY in 30 Messschritten gemessen wurde:

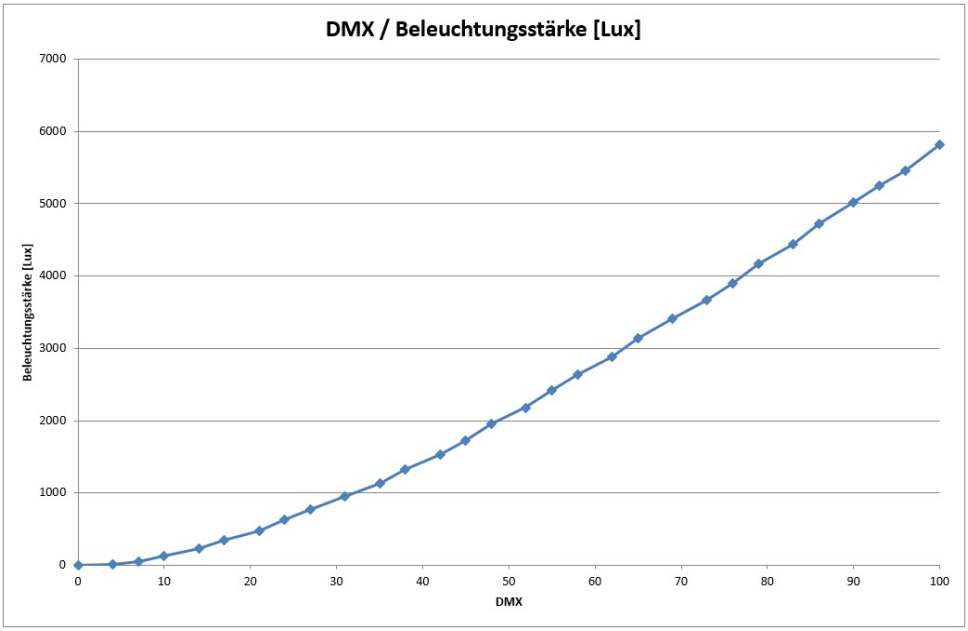

Abbildung 4.9: Plot - DMX / Beleuchtungsstärke

Die DMX-Werte sind dabei auf der X-Achse automatisch in Prozent-Werte umgewandelt worden. Dies dient zur einfacheren Verständlichkeit sowie um eine Unabhängigkeit von den Wertebereichen von 8 / 16-Bit zu erhalten. Auf der Y-Ache befindet sich die Beleuchtungsstärke in Lux, die Achsengrenzen wurden dabei automatisch ermittelt.

Obligatorisch wird bei jeder Messreihe noch das Spektrum der ersten Messung geplottet, dieser Plot findet sich immer im letzten Tab der Datei.

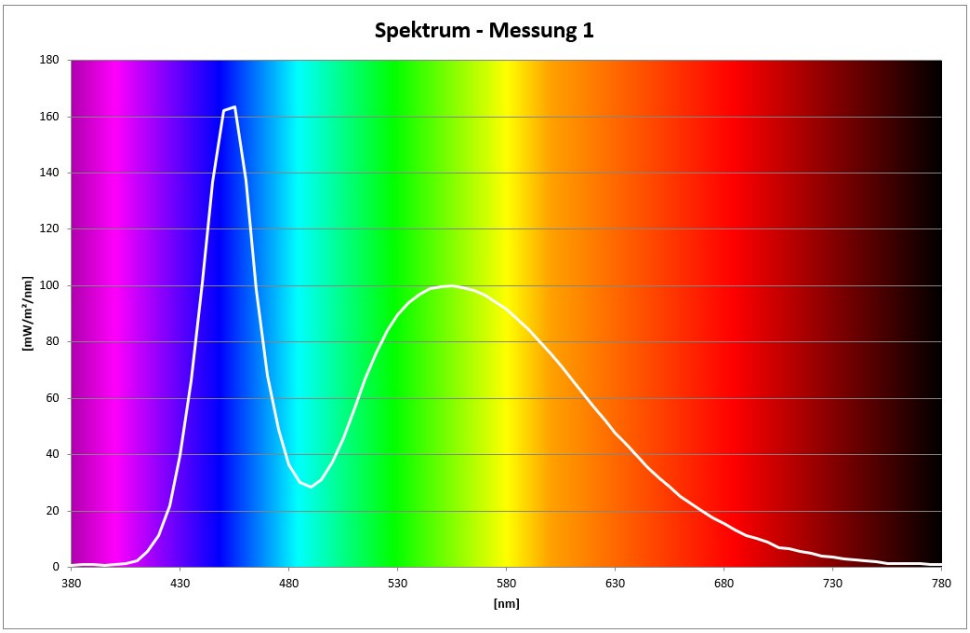

Abbildung 4.10: Plot - Spektrum

Zur Darstellung von komplexeren Messwert-Visualisierungen kann von Nutzer:innen die Python-Bibliothek „*Colour*“ in Kombination mit der Bibliothek „*Pandas*“ verwendet werden. „*Pandas*“ bietet die Möglichkeit XLSX-Dateien wieder einzulesen und die gewünschten Messwerte in Arrays zwischenzuspeichern. Mit diesen Arrays können nun Messwerte mit der „*Colours*“-Bibliothek weiterverarbeitet werden. Dabei bietet die Bibliothek neben vielen farbmetrischen Berechnungen, bei denen das Spektrum als Grundlage dient, auch die Möglichkeit Werte zu plotten.

Im Folgenden wurde beispielhaft ein Plot für den „*Tint*“-Kanal eines Robe T1 Profiles erstellt:

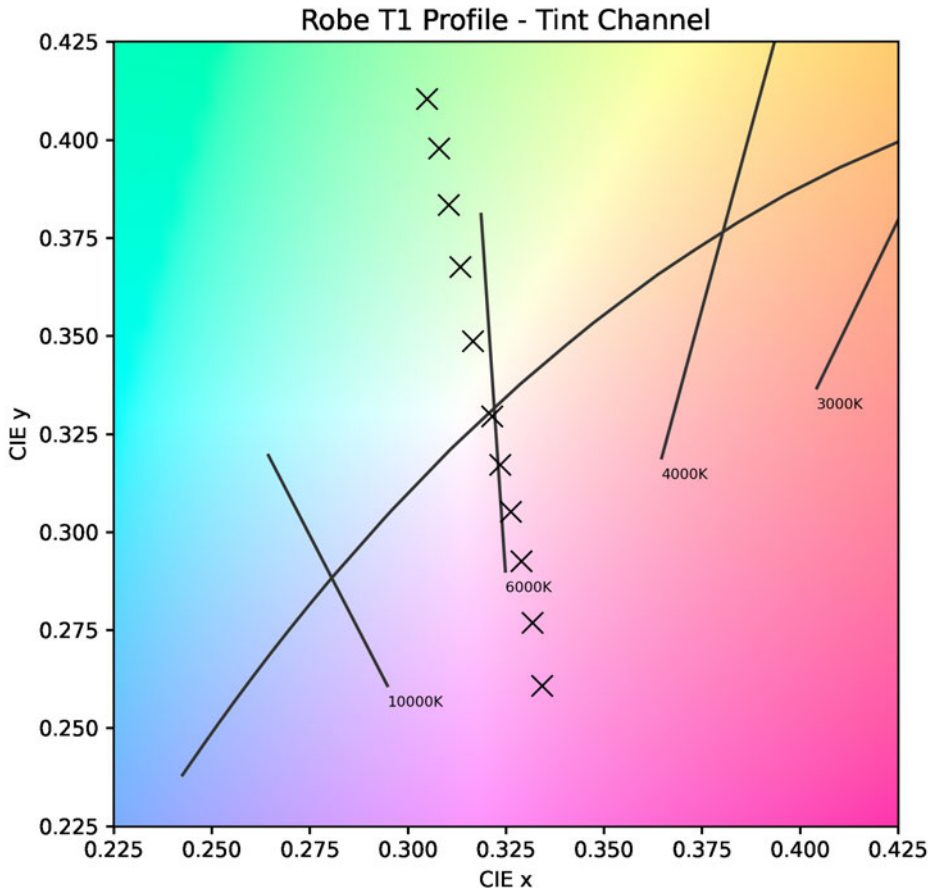

Abbildung 4.11: Plot - Robe T1 Profile - Tint / DMX

Die Visualisierung zeigt dabei einen Ausschnitt des CIE-1931-Farbraums mit dem eingezeichneten plank'schen-Kurvenzug. In diese Abbildung wurden die gemessenen x und y-Farbortkoordinaten eingezeichnet, nachdem sie aus der XLSX-Datei ausgelesen wurden.

Der Programmieraufwand ist durch die „*Colour*“-Bibliothek dabei überschaubar, bietet aber

durch die Nutzung der sehr umfangreichen *„Matplotlib“*-Bibliothek eine große Freiheit in der Gestaltung und Individualisierung der Abbildungen.

Der Python-Code zu der Abbildung 4.11 befindet sich im Anhang (6.2.2).

5 Testmessungen und Fehlerbetrachtung

In folgendem Kapitel werden die Messergebnisse der automatisierten Messung betrachtet. Dafür werden im ersten Abschnitt des Kapitels die Ergebnisse einiger Messungen dargestellt und kurz analysiert. In den danach folgenden Abschnitten werden potenzielle Fehler und Messungenaugkeiten betrachtet, die die Messungen beeinflussen können. Diese Fehler werden durch Testaufbauten und Simulationen messtechnisch erfasst und anschließend eingeschätzt.

5.1 Ergebnis-Analyse als Anwendungsbeispiel

Die beispielhafte Messung eines Scheinwerfers wurde an einem *Robe T1 Profile* durchgeführt. Dieser Scheinwerfer ist ein hochpreisiger Theater- und TV-Scheinwerfer, der mit einem Farbwiedergabeindex (CRI) von größer 95 beworben wird. Laut Herstellerangaben erreicht der Scheinwerfer außerdem einen Farbtemperaturbereich von 2700 – 8000 Kelvin, der mit einer additiven Farbmischung aus fünf LED-Farben (RGBAL) realisiert wird (ROBE Lighting 07.07.2022).

Bei dem *Robe T1 Profile* ist die Farbwiedergabequalität über einen DMX-Kanal steuerbar. Zur Überprüfung der Herstellerangaben und des Verhaltens des CRI-Kanals wurde für Abbildung 5.1 eine Messung des CRI-Wertes zum DMX-Wertebereich des Kanals vorgenommen. Die Messung fand (wie alle in dieser Arbeit besprochenen Messungen, wenn nicht anders angegeben) mit den Default-Einstellungen des Scheinwerfers bei 100% offenem Dimmer statt.

Die Abbildung 5.1 zeigt, dass der minimale Farbwiedergabeindex bei einem CRI-Wert von 78

und der maximale bei einem Wert von 93 liegt. Somit kann die Herstellerangabe des maximalen CRI-Wertes von 95 bei der Default-Farbtemperatur von 6000 K nicht bestätigt werden. Dafür zeigt der Kanal einen annähernd linearen Verlauf des CRI-Wertes, der eine einfache und schnelle Einschätzung des eingestellten Wertes durch den Lichtstellpult-Operator erlaubt.

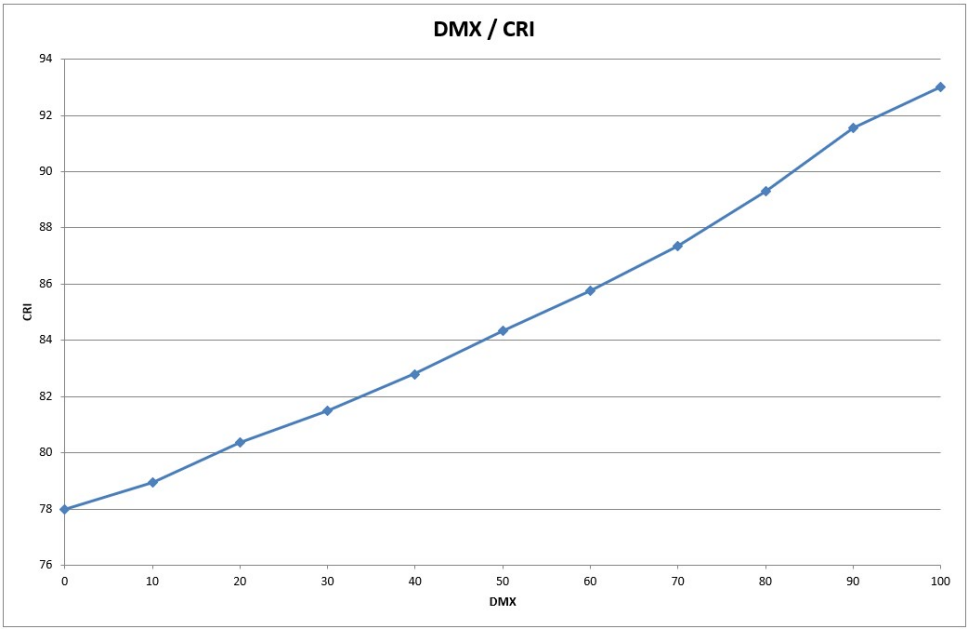

Abbildung 5.1: Plot - Robe T1 Profile - DMX / CRI

Während bei einem minimalen Wert des CRI-Kanals die fünf verschiedenen LED-Farben so angesteuert werden, dass eine maximale Helligkeit erreicht wird, werden für das Erreichen einer besseren Farbwiedergabe die fünf Farben zueinander anders abgestimmt und im Verhältnis zueinander gedimmt. Dies sorgt dementsprechend für einen geringeren Lichtstrom und eine geringere messbare Beleuchtungsstärke. Dieses Verhalten kann nun mit einer zweiten Abbildung (5.2) visualisiert werden:

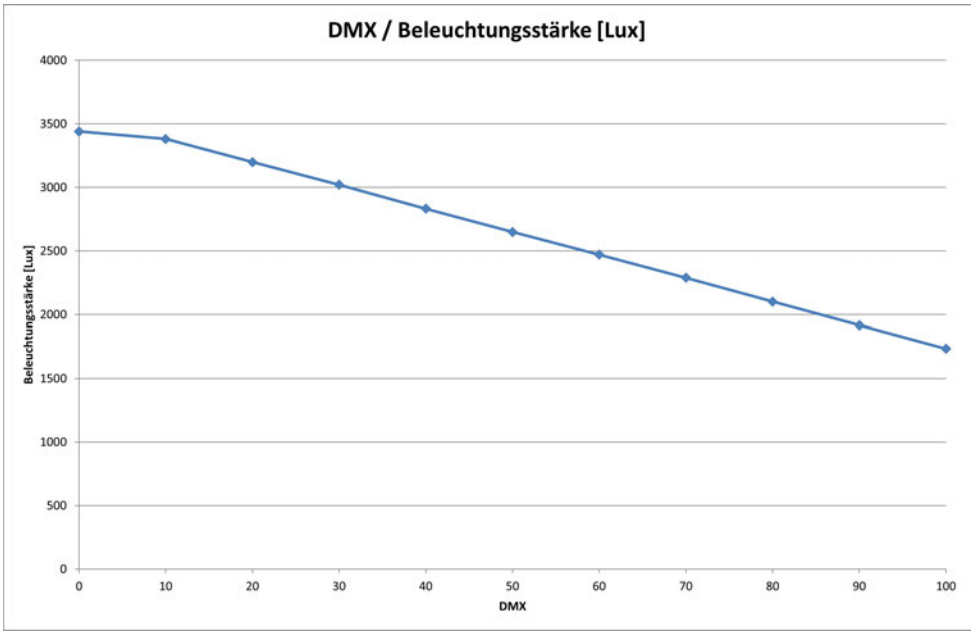

Abbildung 5.2: Plot - Robe T1 Profile - CRI / Beleuchtungsstärke

In diesem Plot wird deutlich, dass auch dieses Verhalten annähernd linear ist und dementsprechend der Kompromiss zwischen Farbwiedergabe und Beleuchtungsstärke einfach abgeschätzt werden kann. Dabei verliert der *Robe T1 Profile* ca. 50% seiner Helligkeit.

Dabei stellt sich aufgrund der Anpassung der Farben zueinander die Frage, wie sich die Farbtemperatur über den CRI-Kanal verhält. In folgender Abbildung 5.3 ist die ähnlichste Farbtemperatur (CCT) gegenüber dem CRI-Kanal dargestellt.

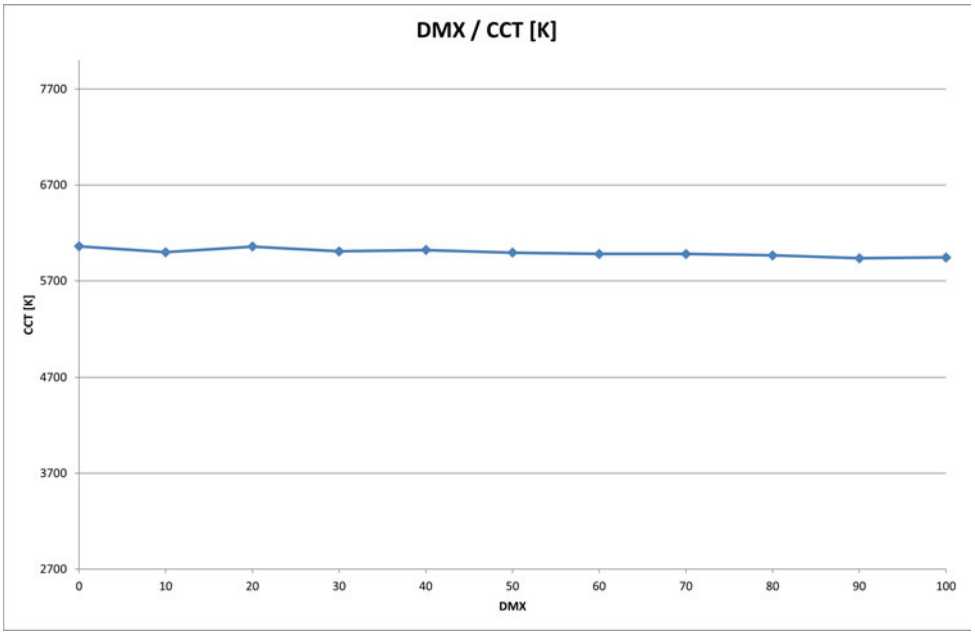

Abbildung 5.3: Plot - Robe T1 Profile - CRI / CCT

Analog zu der Abbildung 5.2 ist hier eine leichte Abweichung bei dem 10% Messpunkt erkennbar, danach bleibt die ähnlichste Farbtemperatur jedoch relativ konstant bei ca. 6000 K. Dies erlaubt beispielsweise in einer Theater-Anwendung ein langsames unsichtbares Überblenden von einem hohen CRI-Wert in eine Szene, die eine maximale Helligkeit erfordert, jedoch auf eine gute Farbwiedergabe verzichten kann.

Da der konstante CCT-Wert jedoch nicht als einziger Faktor eine unsichtbare Überblendung gewährleistet, wird in folgender Abbildung 5.4 noch der Duv-Wert betrachtet. So könnte etwa selbst bei einem konstanten CCT-Wert ein sich verändernder Grün- oder Magenta-Stich dafür sorgen, dass die Überblendung nicht live erfolgen kann oder der *Tint*-Kanal noch zusätzlich mit angepasst werden muss.

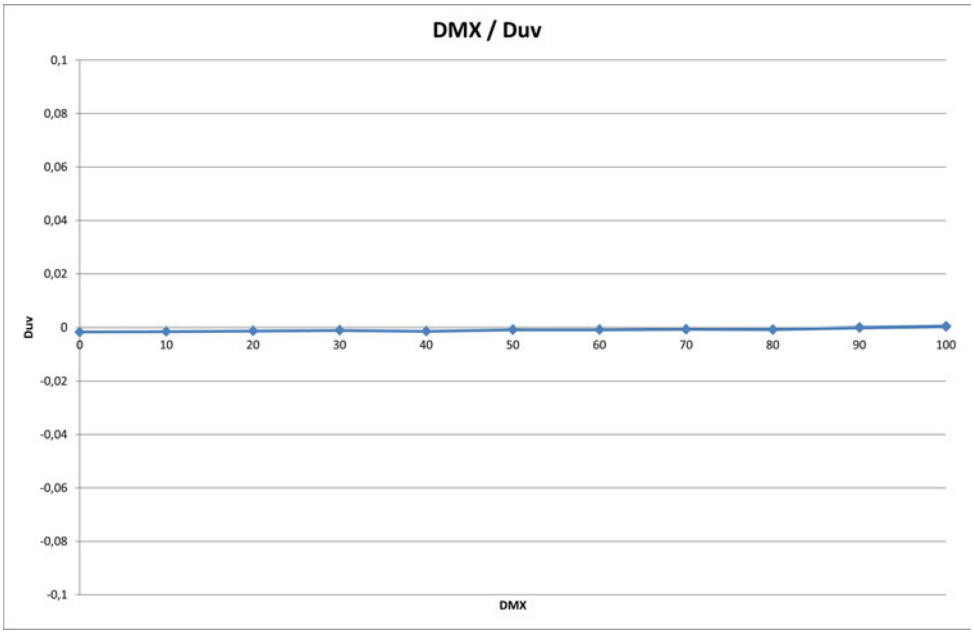

Abbildung 5.4: Plot - Robe T1 Profile - CRI / Duv

Die Abbildung zeigt, dass der CRI-Kanal den Abstand des Farborts zum plank'schen-Kurvenzug konstant verkleinert. Dabei ist der Duv-Wert zu Beginn leicht im negativen Bereich (Magenta-Anteil) und nähert sich mit steigendem DMX-Wert des CRI-Kanals Null an. Interessant ist dabei, dass der Duv Wert den Null-Wert erreicht und der erreichte Weißpunkt damit von Zuschauern möglicherweise weniger bevorzugt wird als der ursprüngliche Wert. Versuche zeigten, dass ein Weißton mit einem negativem Duv-Wert von Versuchsteilnehmern eher bevorzugt wurden als Farborte die genau auf dem Temperaturfarbzug lagen. Die Duv-Werte dafür werden von -0.002 bis -0.004 angegeben (Baer, Barfuß und Seifert 2020, S. 77). Zwar würde dieser Effekt wahrscheinlich durch die deutlich verbesserte Farbwiedergabe in einem bunten Bühnenbild aufgehoben, jedoch erlauben die genommenen Messwerte aus Abbildung 5.4 die Vermutung, dass eine leichte Anpassung des *Tint*-Kanals den Weißpunkt für Zuschauer angenehmer wirken

lässt.

Ein weiteres Beispiel ist die Messung der Homogenität des Lichtkegels des *Robe T1 Profile*. Diese „Hotspot"-Messung muss jedoch bei einer von dem Default-Wert abweichenden Zoomgröße stattfinden, da die maximale Spannweite des Roboters bei etwa 2,4 Metern liegt und somit der Durchmesser des Lichtkegels auch maximal 2,4 m groß sein darf, wenn der gesamte Lichtkegel gemessen werden soll. In der folgend abgebildeten Messung (5.5) wurde die „Beam"-Größe auf ca. 2 m Durchmesser bei 5 m Distanz begrenzt und die Roboterspannweite von 2,4 m ausgenutzt. So sind Messungen außerhalb des Lichtkegels möglich, die eine Aussage über die Helligkeit außerhalb des scharfgestellten Beams erlauben.

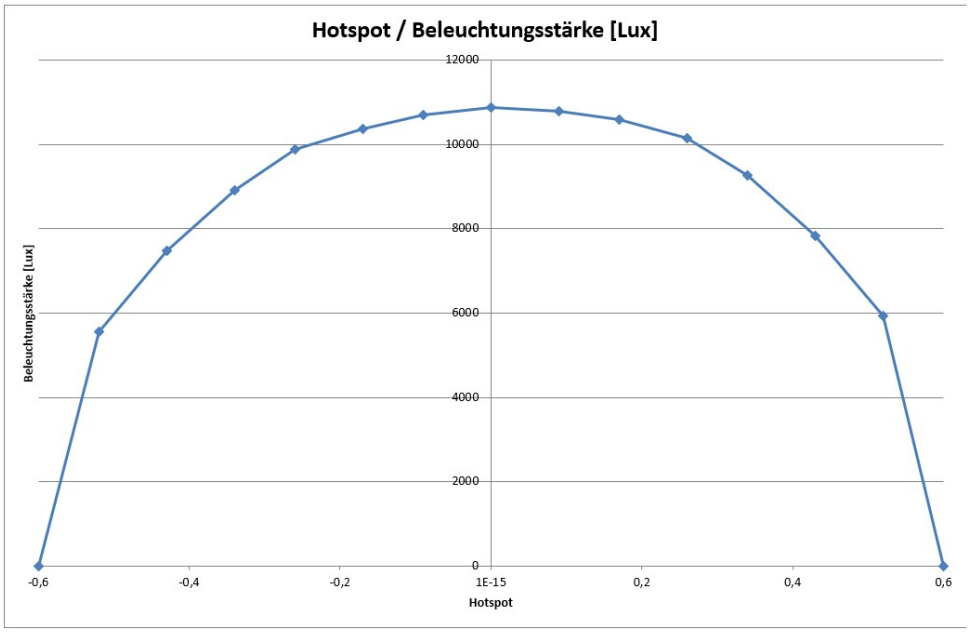

Abbildung 5.5: Plot - Robe T1 Profile - Hotspot / Beleuchtungsstärke

In dieser Abbildung wird sichtbar, dass der *Robe T1 Profile* einen deutlichen Hotspot hat. Dieser ist, auf der gemessenen horizontalen Achse, symmetrisch. Während in der Mitte des Lichtkegels 10870,79 lx gemessen werden erreicht die Beleuchtungsstärke in 0,5 m Entfernung nur noch einen Wert von 5931,18 lx und ist somit um ca. 46% geringer. Die Abschottung außerhalb des Lichtkegels funktioniert gut und es wurden bei 0,6 m jeweils lediglich ca. 7 lx gemessen.

5.2 Fehlerbetrachtung der Positionierung

Bei der Betrachtung der möglichen Fehler ist die Ausrichtung des Scheinwerfers zum Messgerät die größte Fehlerquelle. Die Position des Spektrometers wird durch Roboter und Halterung bestimmt. Die Halterung ist so konstruiert, dass sie das Messgerät fest umschließt und vor möglichem Verrutschen schützt. Dementsprechend ist die Position des Messgerätes von der Position des Roboters bestimmt. Da der Roboter auf Höhe der Messbank positioniert ist, ist die Ausrichtung des Roboters auf seiner Position besonders relevant. Die möglichen Fehler bei der Ausrichtung des Roboters auf seinen drei Hauptachsen soll im Folgenden mit Hilfe des Roboters selbst simuliert werden. Der Versuch kann keine vollständige Fehlerbetrachtung durch Positionsfehler leisten, soll jedoch das Problem verdeutlichen und als Einschätzung dienen.

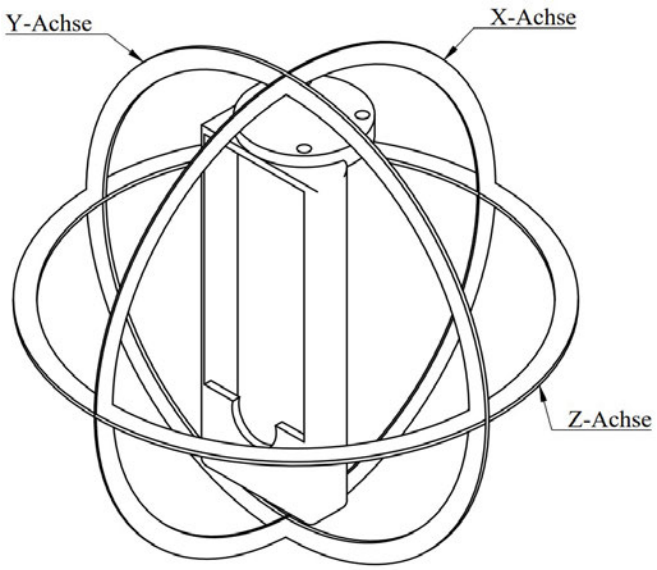

Abbildung 5.6: XYZ-Achsen des Spektrometers am Roboter

Es wurde dafür folgender Testaufbau gemacht:

Eine 1000W Stufenlinse wurde in drei Metern Entfernung vor dem Messkopf positioniert. Das Spektrometer war in der Halterung an dem Roboter angebracht und sowohl Roboter als auch Messbank wurden händisch möglichst genau ausgerichtet. Mit einer digitalen Wasserwaage wurden die Winkel bestmöglich kontrolliert, so dass sowohl der Scheinwerfer als auch der Roboter inklusive Messgerät und Halterung zueinander gerade waren. Mit einigen Testmessungen wurde nun das Spektrometer an dem Roboter an die Stelle mit der größten Helligkeit in der Mitte des Lichtkegels positioniert. Der programmierte Testzyklus umfasste nun acht Messschritte: Die erste Messung war eine Messung in der Ausgangslage, ohne eine simulierte

Schräglage. Die danach folgenden sieben Messungen simulierten durch Kippen des Messkopfes um jeweils 5° pro Achse das Kippen des gesamten Roboters durch falsche Positionierung. Der Testzyklus fuhr dabei auch alle Kombinationen der Schräglagen an.

Folgend sind die Mittelwerte der gemessenen Beleuchtungsstärken der fünf aufgenommenen Messreihen im Bezug zu den Winkeln aufgeführt:

X-Achse [°]	Y-Achse [°]	Z-Achse [°]	$\overline{E}$ [lux]
0	0	0	7321,2
0	0	-5	7251,0
0	-5	0	7189,8
0	-5	-5	7093,4
-5	0	0	7233,0
-5	0	-5	7170,2
-5	-5	0	7138,0
-5	-5	-5	7059,6

Tabelle 5.1: Messergebnisse der Fehlerbetrachtung: Positionierung Roboter

Zur Visualisierung dieser Ergebnisse folgt Abbildung (5.7). In dieser sind die prozentualen Verluste der Beleuchtungsstärke farblich dargestellt:

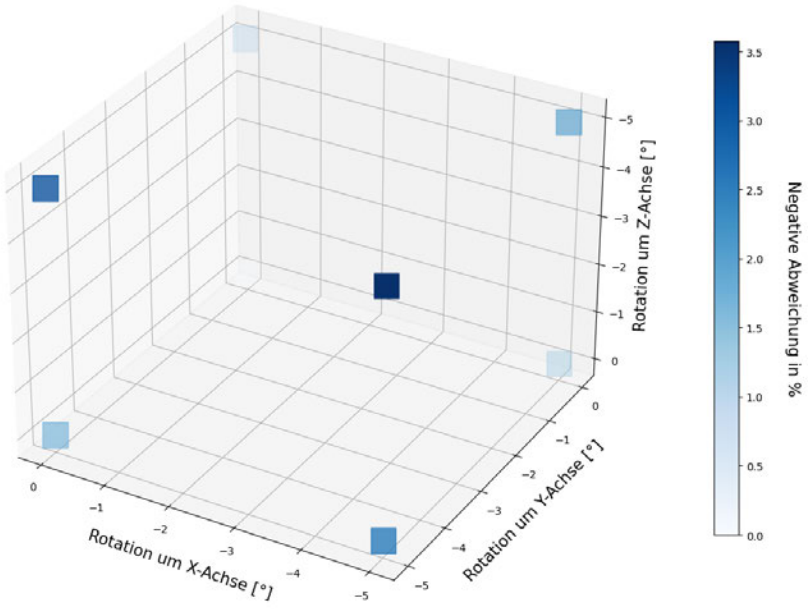

Abbildung 5.7: Prozentualer Fehler der Beleuchtungsstärke bei schiefer Positionierung

In dieser Abbildung ist der Ausgangsmesswert bei $(0\ 0\ 0)$ und einem Wert von Null transparent, die anderen Positionen sind farblich ihrer Abweichung entsprechend gekennzeichnet. Wie zu erwarten, liegt die größte Abweichung bei der Position, an der der Messkopf auf allen Achsen um fünf Grad angekippt wurde. Die Abweichung an dieser Position beträgt $-3,573\%$. Ähnlich stark fällt mit $-3,112\%$ das Kippen auf der Y- sowie Z-Achse ins Gewicht. Diese Achsen sind bei der Einrichtung der Messung dementsprechend besonders sorgfältig einzumessen.

5.3 Fehlerbetrachtung des Spektrometers

Zur Überprüfung der Fehlerangaben des Herstellers wurde ein weiterer Versuch aufgebaut. Dieses mal mit drei verschiedenen Beleuchtungsstärkenniveaus, um eine Einschätzung des Fehlers bei unterschiedlichen Beleuchtungsstärken und damit verbundenen Integrationszeiten zu erreichen.
Dabei wird folgend lediglich die Präzision, also das Streumaß der Messwerte, überprüft. Die Richtigkeit der Messwerte kann mit diesem Versuch nicht beurteilt werden und generelle Einschränkungen zu den Fehlergrößen werden im Anschluss noch behandelt.

1. 40 W / 420 lm Halogen-Glühlampe, – 1 m Entfernung zum Messgerät

2. 40 W / 420 lm Halogen-Glühlampe – 25 cm Entfernung zum Messgerät

3. 1000 W Halogen-Stufenlinse – 3 m Entfernung zum Messgerät, engster Abstrahlwinkel

Das Messgerät ist an dem Roboter angebracht und die Höhe des Messkopfes sowie die des hellsten Lichtaustritts aus dem Leuchtmittel sind mithilfe einiger Testmessungen eingemessen worden. Das Ziel der Messung ist die Bestimmung der Präzision oder auch Wiederholgenauigkeit des Gossen MAVOSPEC Base durch eine so schnell wie möglich aufgenommene Messreihe ohne Veränderliche. Dafür wurde ein Python-Skript programmiert, das automatisch 100 identische Messungen hintereinander auslöst und die Messdaten protokolliert. Aufgenommen wurden Daten zur Beleuchtungsstärke [E], zum CCT [K] sowie xy-Farbkoordinaten. Die ausgewählten Leuchtmittel sind konventionelle Leuchtmittel ohne Steuereletronik oder Kühlungen. Im Gegensatz zu LEDs ist ihr Lichtstrom daher, ebenso wie das emittierte Spektrum, über den Zeitraum der Messung konstant. Zwar nehmen auch konventionelle Leuchtmittel über den Zeitraum durch etwa Materialschwund und Verrußung an Helligkeit ab, diese ist innerhalb der verhältnismäßig kurzen Zeitspanne der Messung aber zu vernachlässigen.

Die Mittelwerte der jeweiligen Messreihe sind folgend in Tabelle 5.2 abzulesen:

	1	**2**	**3**
E [lux]	38,52	485,64	10290,8
CCT [K]	2742,02	2694,44	3085,29
x	0,4555	0,4598	0,4303
y	0,4081	0,4099	0,4031

Tabelle 5.2: Messergebnisse Spektrometerfehler - Mittelwerte

Zur Abschätzung des gemachten Fehlers innerhalb der Messreihe sind in folgender Tabelle 5.3 die Standardabweichungen zum Mittelwert der jeweiligen Messreihe zu finden:

	1	2	3
E [lux]	0,652	3,116	56,547
CCT [K]	30,682	2,137	3,018
x	0,0033	0,00019	0,00019
y	$2{,}3428 \cdot 10^{-3}$	$1{,}3150 \cdot 10^{-4}$	$9{,}3226 \cdot 10^{-5}$

Tabelle 5.3: Messergebnisse Spektrometerfehler - Standardabweichungen

Anhand dieser Werte wurde für die folgende Tabelle 5.4 die prozentuale Abweichung vom Mittelwert bei einer der Standardabweichung berechnet:

	1	2	3
E [lux]	1,692%	0,642%	0,549%
CCT [K]	1,119%	0,079%	0,098%
x	0,722%	0,041%	0,044%
y	0,547%	0,032%	0,023%

Tabelle 5.4: Messergebnisse Spektrometerfehler - Prozentuale Abweichung bei einer Standardabweichung

Bei einer Normalverteilung der aufgenommenen Werte gelten diese Abweichungen für 68,27% aller Messwerte. In der nachfolgenden Tabelle 5.5 sind die prozentualen Abweichungen von zwei mal der Standardabweichung zum Mittelwert aufgeführt, diese gelten für statistisch 95,45% aller Messwerte (Zwerenz 2009, S. 327):

	1	2	3
E [lux]	3,384%	1,283%	1,099%
CCT [K]	2,238%	0,159%	0,196%
x	1,445%	0,081%	0,088%
y	1,148%	0,064%	0,046%

Tabelle 5.5: Messergebnisse Spektrometerfehler - Prozentuale Abweichung bei zwei Standardabweichungen

Für die Beurteilung, ob überhaupt eine Normalverteilung vorliegt, wurde der Shapiro-Wilk-Test auf die aufgenommenen Datensätze angewandt und ein Grenzwert des Signifikanzniveaus von 5% festgelegt. Somit muss der errechnete p-Wert der Datensätze ≥ 0.05 sein, um als Normalverteilung bestätigt zu werden (Duller 2008, S. 121–122).

Zur genaueren Betrachtung der Verteilung und für die Einschätzung einer Aussagekräftigkeit der errechneten Signifikanz, wurden folgend außerdem noch Quantil-Quantil-Diagramme der Beleuchtungsstärke-Datensätze generiert:

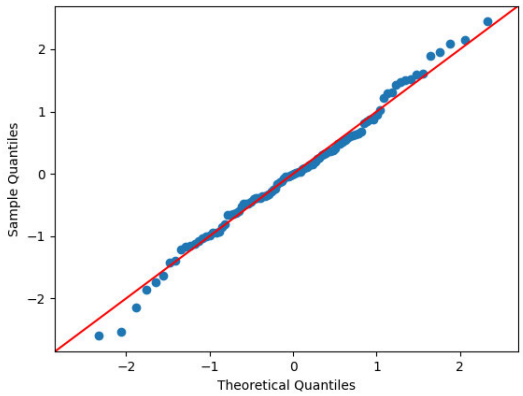

Abbildung 5.8: Q-Q-Diagramm - Messreihe 1 - p-Wert= 0,7497

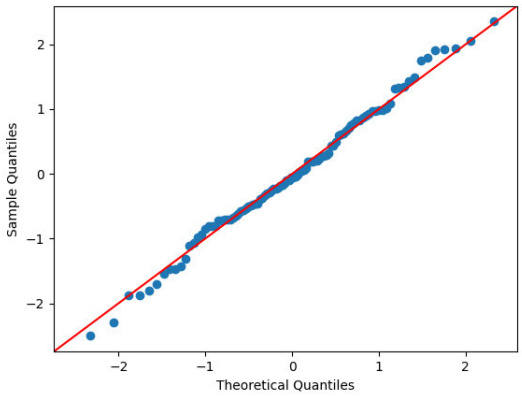

Abbildung 5.9: Q-Q-Diagramm - Messreihe 2 - p-Wert= 0,8099

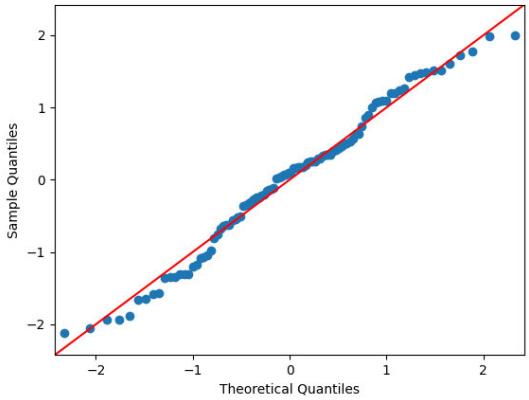

Abbildung 5.10: Q-Q-Diagramm - Messreihe 3 - p-Wert= 0,1298

Bei dieser Diagrammart werden die Messwerte anhand ihres Z-Wertes gegenüber der theoretischen Verteilung der Z-Werte einer Normalverteilung aufgetragen. Der Z-Wert gibt an, um wie viele Standardabweichungen ein Wert von dem Mittelwert abweicht.
Bei einer idealen Normalverteilung lägen die Messwerte somit alle, vom Nullpunkt ausgehend, auf einer 45°-Achse (Kohn und Öztürk 2013, S. 99–100);(Zwerenz 2009, S. 156 ff.).

Die vorliegenden Diagramme zeigen alle grundlegend eine Orientierung an der 45°-Achse. Somit kann, gemeinsam mit den errechneten p-Werten, auch auf eine grundlegende Orientierung der Datensätze an der Normalverteilung geschlossen werden. Die statistische Annahmen der prozentualen Abweichungen aus Tabelle 5.3 anhand der Standardabweichungen können somit begründet werden.

Die Aussagekräftigkeit der erhobenen Werte und des generellen Fehlers muss jedoch im folgenden noch eingeschränkt werden:
Das Gossen MAVOSPEC BASE besitzt laut Herstellerangaben ein Kosinus-korrigierten Messkopf der Klasse B nach DIN 5032-7 (Gossen 2021a, S. 35). Die Herstellerangaben für den Beleuchtungsstärkefehler unterbieten die Klassengrenzwerte der Gesamtkenngröße der DIN 5032-7 jedoch deutlich (Baer, Barfuß und Seifert 2020, S. 174). So gibt Gossen den Fehler ihrer Kosinus-Korrektur als auch des Gesamtfehlers der Beleuchtungsstärke mit 3% an. Der Klassengrenzwert für die Gesamtunsicherheit in Klasse B liegt laut DIN 5032-7 jedoch bei 10%. Die DIN 5032-7 spezifiziert neben der cos-getreuen Bewertung Fehlerquellen wie etwa V(λ)-Anpassung, Linearität oder Temperaturabhängigkeit. Gossen gibt keine aufgeschlüsselten Messunsicherheiten für die Beleuchtungsstärke an und setzt den Kosinus-Korrektur-Fehler mit dem Gesamtfehler gleich. Diese Tatsache ergibt, dass generell eine Gesamtunsicherheit der Beleuchtungsstärke von 10% und nicht 3% angenommen werden sollte. Innerhalb dieser Messunsicherheit liegen auch die erhobenen Messwerte für die Präzision aus Tabelle 5.5 für

95,45% aller Messwerte.

Weitergehend nutzt das Gossen MAVOSPEC BASE eine virtuelle, variable Belichtungszeit des verbauten CMOS-Sensors. Diese Integrationszeit liegt zwischen 10 und 3000 ms und wird bei den Messungen im Protokoll mit angegeben. Es ergibt sich jedoch besonders im Helligkeitsniveau von weniger als ca. 250 lux, dass das Messgerät bereits die maximale Integrationszeit von 3000 ms ausschöpft. Ab diesem Grenzwert ist der potenzielle Fehler bei niedrigeren Beleuchtungsstärken größer, da die Integrationszeit nicht weiter steigen und nicht mehr die gleiche Sättigung des CMOS-Sensors erreicht werden kann. Daher sollte, wenn möglich, die Messung mit mehr als 250 lux Beleuchtungsstärke und einer Integrationszeit von kleiner 3000 ms stattfinden, um den potenziellen Fehler zu minimieren.
Die verwendete Messmethode mit 230V-Halogenglühlampen liefert außerdem selber einen potenziellen Fehler. Dadurch, dass die Netzfrequenz von 50 Hz zu einem 100 Hz Flicker des Leuchtmittel führt, wird es bei den drei Messreihen Messpunkte geben, bei denen die variable Integrationszeit zufällig so gewählt wurde, dass sie bei Maxima oder Minima der Sinuskurve beginnt und auch bei selbigen endet. Dies führt ebenfalls zu einer Streuung der Werte, die im Rahmen dieser Arbeit nicht quantifizierbar ist. Typischerweise liegen die Pulsweitenmodulations (PWM)-Frequenzen von modernen LED-Scheinwerfer in der Praxis jedoch deutlich höher und werden mit steigender Frequenz unproblematischer. Der bereits angesprochene *Robe T1 Profile* bietet beispielsweise die Möglichkeiten die PWM-Frequenz auf Werte zwischen 300 und 2400 Hz festzulegen.

Eine Überprüfung der Richtigkeit und eine bessere Präzision könnten durch eine jährliche Kalibrierung durch den Hersteller erreicht werden.

6 Fazit

6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde eine Software entwickelt, die eine automatische Scheinwerfervermessung ermöglicht. Dabei wurde die Hardware des Lichtlabors der HAW Hamburg verwendet und die Software explizit auf diese zugeschnitten und entwickelt. Die wichtigsten Komponenten sind dabei der Industrieroboter HORST900 der Firma fruitcore und das Spektrometer MAVOSPEC BASE der Firma Gossen als Messgerät. Die Software wurde in der Programmiersprache Python entwickelt und ist als Webanwendung konzipiert und realisiert worden. Sie bietet dementsprechend eine Oberfläche im Browser an, mit der die nötigen Einstellungen für die Messungen vorgenommen werden können. Möglich sind dabei 3 Messszenarien: DMX, Zeit und Homogenität.

Bei der DMX-Messung werden die Werte von bis zu drei DMX-Kanäle durchlaufen und eine von den Nutzer:innen eingestellte Anzahl an Messwerten aufgenommen. So lassen sich beispielsweise Dimmerkurven beliebig genau aufzeichnen und nachvollziehen.
Bei der Zeit-Messung werden Messzeitpunkte genau definiert und die Software löst in definierten Abständen automatisch Messungen aus. So lassen sich etwa die Helligkeits- und Farbveränderungen durch Wärmeentwicklung von LED-Leuchtmitteln messen.
Bei der Homogenitätsmessung kann ein Lichtkegel in der horizontalen Achse vermessen und Helligkeits- und Farbveränderungen innerhalb des Kegels dokumentiert werden.

Die Dokumentation und Messwertdarstellung wurde als XLSX-Datei ausgeführt, die die Nutzer:innen im Browser herunterladen können. Das Tabellenkalkulationsformat ist mit Software wie Microsoft Excel oder LibreOffice kompatibel und ermöglicht so Student:innen und auch Laien einen einfachen Umgang mit Messergebnissen.

6.2 Herausforderungen

Bei der Realisierung der Software kam es, wie bei einer zeitlich eingeschränkten Entwicklung üblich, zu einigen Herausforderungen, die noch ungelöst blieben und in der Zukunft im Rahmen von weiteren Arbeiten oder Projekten gelöst werden können.

Als Erstes ist hier die Positionierung des Roboters und des Scheinwerfers zu nennen. Da sich diese Arbeit auf die Entwicklung und Umsetzung der Software fokussiert, wurde dieses Thema nur angeschnitten. Zwar wurde im Rahmen der Fehlerbetrachtung der mögliche Fehler abgeschätzt, durch eine Methodik zur genaueren Positionierung könnte dieser Fehler jedoch deutlich verringert werden. So ist der Roboter zum Zeitpunkt der Arbeit noch nicht an der Messbank im Boden verankert, so dass ein Verschieben des Roboters durch Fremdeinwirkung möglich ist. Außerdem ist die Positionierung des Scheinwerfers auf der Messbank wichtig. Ist der Scheinwerfer in allen Achsen zum Roboter nivelliert? Und ist das Einstellen eines 90-Grad Winkels in Richtung Messkopf wirklich ein 90-Grad Winkel? Außerdem ist die Positionierung des Messkopfes in der exakten Mitte des Scheinwerferkegels eine Herausforderung. Bei den im Rahmen dieser Arbeit entstandenen Messungen wurden dafür der Zoom und die Iris auf minimal eingestellt und über den Fokus auf dem Messgerät scharf gestellt. Daraufhin wurde der Messkopf über die Benutzeroberfläche in die Mitte des Lichtkegels ausgerichtet und diese Position als Home-Position abgespeichert. Diese Möglichkeiten sind aufgrund von eventuell schlechter Nivellierung von Roboter zu Scheinwerfer nicht optimal und könnten beispielsweise durch einen Laser für das Messgerät verbessert werden. Dafür könnte entweder das Spektrometer der Firma *Jeti* ersetzt werden, die diese Funktion bereits eingebaut haben oder die Entwicklung einer automatischen Nivellierung im Rahmen einer weiteren Arbeit umgesetzt werden.

Als Idee bieten sich dabei beispielsweise zwei *ESP32*-Mikrocontroller mit angeschlossenen Beschleunigungssensoren an. Diese bieten sich aufgrund der Erdanziehungskräfte als Lagesensoren an und können als elektronische „Wasserwage“ genutzt werden. So könnte mit Hilfe der Bluetooth- oder WLAN-Funktionen des ESP32 eine Kommunikation zwischen den zwei Mikrocontrollern aufgebaut werden, die eine Differenz der Nivellierung erkennen und die Werte per IP-Protokoll an die Software weiterleiten. Diese könnte mit diesen Werten nun den Roboter-Nullpunkt um ein Offset verschieben und so das Messgerät zum Scheinwerfer nivellieren. Dabei könnte außerdem die Funktion eines Lasers implementiert werden, wenn zusätzlich zum Beschleunigungssensor ein Laser an der Halterung des Messgerätes angebracht wird. Einfache 5mW Laserdioden sind bereits mit einer Platine mit den nötigen Widerständen und Anschlusspins verfügbar und können aufgrund ihrer geringen Leistungsaufnahme direkt

von einem Mikrocontroller versorgt und angesteuert werden. Das Schalten der Diode könnte über die Benutzeroberfläche gelöst werden.

Eine weitere Herausforderung ist die benötigte Zeit für eine Messung. Aufgrund der Limitierung der seriellen Schnittstelle des Gossen MAVOSPEC BASE ist eine Abfrage der Werte zeitaufwendig und liegt beispielsweise bei der Abfrage des Spektrums bei mehreren Sekunden. Da außerdem immer mehrere Befehle für die unterschiedlichen Messwerte gestellt werden müssen, summiert sich die benötigte Zeit inklusive Messdauer und Pufferzeiten für beispielsweise Roboterbewegung, in der Praxis auf eine Minimalzeit von 45 Sekunden pro Messung. Bei Messungen die zeitlich gesteuert sind, fällt diese Limitierung nur ins Gewicht, wenn eine sehr genaue Messung wie etwa alle 15 oder 30 Sekunden erreicht werden soll. Bei DMX- oder Homogenitätsmessungen wäre eine schnellere Übertragung der Daten jedoch wünschenswert, da so mehr Messungen in weniger Zeit erreicht werden könnten und außerdem Einflüsse wie etwa die Umgebungstemperatur, die sich über die Zeit ändern können, weniger potenziellen Einfluss auf das Messergebnis hätten.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingsverzeichnis

Literaturverzeichnis

Abts, Dietmar (2010). *Masterkurs Client/Server-Programmierung mit Java: Anwendungen entwickeln mit Standard-Technologien: JDBC, UDP, TCP, HTTP, XML-RPC, RMI, JMS und JAX-WS*. 3., erweiterte Auflage. Springer eBook Collection Computer Science & Engineering. Wiesbaden: Vieweg+Teubner. ISBN: 9783834897244. DOI: 10.1007/978-3-8348-9724-4.

Baer, Roland, Meike Barfuß und Dirk Seifert (2020). *Beleuchtungstechnik: Grundlagen*. 5. Auflage. Berlin: Huss-Medien GmbH. ISBN: 3341016481.

Bloch, L. (2019). *Lichttechnik*. Reprint 2019. Berlin und Boston: Oldenbourg Wissenschaftsverlag. ISBN: 9783486746747. DOI: 10.1515/9783486746747.

data2type GmbH (2022). *SpreadsheetML | Das XLSX-Format: ein ZIP-Container hats in sich*. URL: https://www.data2type.de/xml-xslt-xslfo/spreadsheetml/xlsx-format.

Duller, Christine (2008). *Einführung in die nichtparametrische Statistik mit SAS und R: Ein anwendungsorientiertes Lehr- und Arbeitsbuch*. Heidelberg: Physica-Verlag HD. ISBN: 9783790820607. URL: http://nbn-resolving.org/urn:nbn:de:bsz:31-epflicht-1618451.

fruitcore robotics GmbH (2021). *Fruitcore_2021_Produktaufnahmen_08-827x1024.jpg (827× 1024)*. URL: https://fruitcore-robotics.com/wp-content/uploads/2022/02/Fruitcore_2021_Produktaufnahmen_08-827x1024.jpg.

– (2022a). *Montageanleitung HORST900: Version 3.1*. Hrsg. von fruitcore robotics GmbH. Montageanleitung.

– (2022b). "Weiterführende technische Daten zum Robotersystem HORST900 - Version V220127". In: URL: https://fruitcore-robotics.com/wp-content/uploads/2022/02/Weiterfuehrende-technische-Daten-zum-Robotersystem-HORST900-Version-V220127.pdf.

General Python FAQ — Python 3.10.4 documentation (20.05.2022). URL: https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place.

Gossen (2021a). "Bedienungsanleitung Gossen MAVOSPEC BASE". In: URL: https://gossen-photo.de/wp-content/uploads/DL/SLMT/Mavospec_Base/Anleitung/GA_Mavospec_Base_DE.pdf.

– (2021b). *MAVOSPEC BASE SDK: USB Interface Description*. URL: https://gossen-photo.de/wp-content/uploads/DL/SLMT/Mavospec_Base/Software/MAVOSPEC_BASE_SDK.zip.

Gossen (2022). *Shop_Produkt_MavospecBase_800x800_Display1.jpg (800×800)*. URL: https://gossen-photo.de/wp-content/uploads/2017/08/Shop_Produkt_MavospecBase_800x800_Display1.jpg.

Greule, Roland (2021). *Licht und Beleuchtung im Medienbereich*. 2., aktualisierte und erweiterte Auflage. Medien. München: Hanser. ISBN: 9783446468658.

Kohn, Wolfgang und Riza Öztürk (2013). *Statistik für Ökonomen: Datenanalyse mit R und SPSS*. 2. Aufl. 2013. Springer-Lehrbuch. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 9783642373527. URL: http://nbn-resolving.org/urn:nbn:de:bsz:31-epflicht-1568657.

Microsoft (2022). *Hardware ID - Windows drivers*. URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/install/hardware-ids.

Pallets (30.05.2022). *Flask*. URL: https://palletsprojects.com/p/flask/.

ROBE Lighting (7.07.2022). *T1 Profile™ | Products | ROBE Lighting*. URL: https://www.robelighting.de/t1-profile.

Şanal, Ziya (2015). *Mathematik für Ingenieure: Grundlagen - Anwendungen in Maple*. 3., vollst. überarb. und erw. Aufl. Lehrbuch. Wiesbaden: Springer Vieweg. ISBN: 3658106417. DOI: 10.1007/978-3-658-10642-3. URL: https://link.springer.com/content/pdf/10.1007/978-3-658-10642-3.pdf.

What is WSGI? — WSGI.org (29.01.2021). URL: https://wsgi.readthedocs.io/en/latest/what.html.

Zwerenz, Karlheinz (2009). *Statistik: Einführung in die computergestützte Datenanalyse*. 4., überarb. Aufl. Managementwissen für Studium und Praxis. München: Oldenbourg. ISBN: 978-3-486-59112-5. DOI: 10.1524/9783486711035.

Anhang

6.2.1 MAVOSPEC BASE Halterung

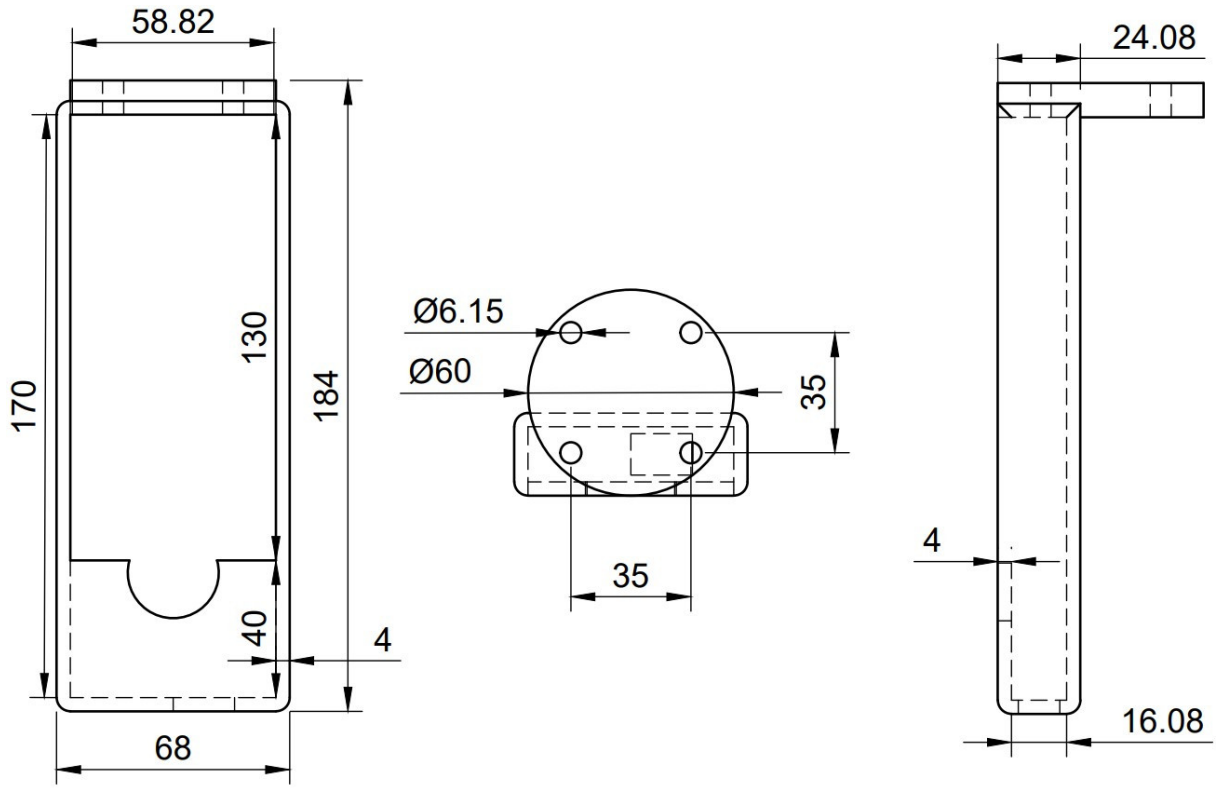

Abbildung 6.1: MAVOSPEC BASE Halterung - Zeichnung mit Bemaßungen

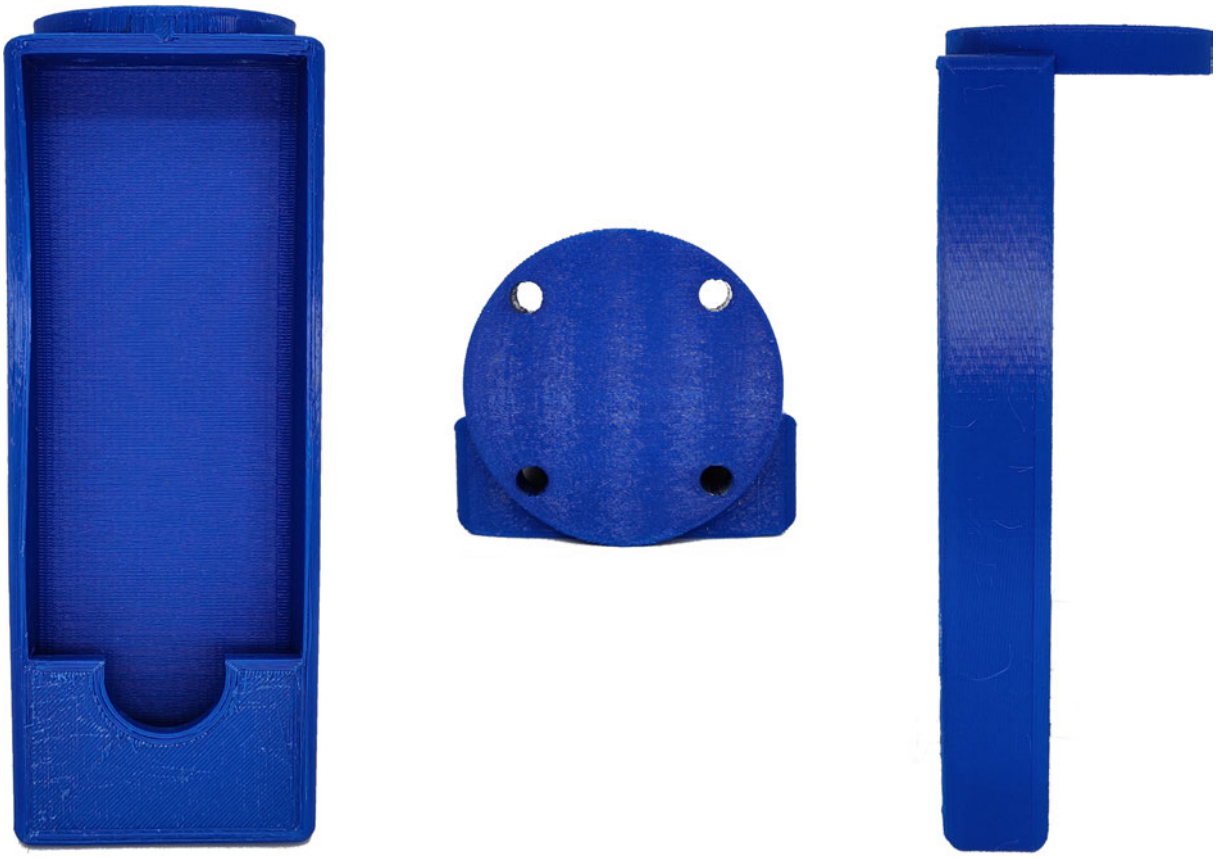

Abbildung 6.2: MAVOSPEC BASE Halterung - Fotos 3D-Druck PLA

(Interaktive 3D-Datei mit Adobe Acrobat Reader)

6.2.2 Python-Visualsierung - Beispiel - Abbildung 4.11

```
import colour
from colour.plotting import plot_chromaticity_diagram_CIE1931, render
import matplotlib.pyplot as plt
import pandas as pd

title =          "Robe T1 Profile - Tint Channel"
xlsx_filename = 'Robe T1 Profile 06_05_2022_12_27_16 Tint - DMX XY,UV.xlsx'
sheet_name=      "DMX - XY,UV"
fields =         ["X","Y"]
bounding_box =  [0.225, 0.425, 0.225, 0.425]
filename =       "Robe_T1_Profile_Tint_Plot.jpg"
x_label =        "CIE x"
y_label =        "CIE y"
dpi =            300
font =           "Arial"
font_weight =     "normal"
font_size =      11
marker =         "-x"
marker_color =   "black"
markersize =     "10"

xy_array = pd.read_excel(xlsx_filename, sheet_name=sheet_name,usecols=fields)
x_array = pd.Series.to_numpy(xy_array.X)
y_array = pd.Series.to_numpy(xy_array.Y)

plt.rcParams["figure.dpi"] = dpi

figure, axes = (
plot_chromaticity_diagram_CIE1931(standalone=False))
colour.plotting.temperature.plot_planckian_locus(axes=axes,standalone=False,method='
    CIE 1931')

for i in range(len(x_array)):
    x = x_array[i]
    y = y_array[i]
    plt.plot(x, y, marker, color=marker_color,markersize=markersize)

font = {'family': font,'weight': font_weight,'size': font_size}
plt.rc('font', **font)

render(
    standalone=True,
    filename=filename,
    dpi=dpi,
    bounding_box=bounding_box,
    x_tighten=True,
```

```
    y_tighten=True,
    tight_layout=True,
    x_label=x_label,
    y_label=y_label,
    title=title
)
```

Listing 6.1: Beispiel - Visualsierung mit Python

6.2.3 Python-Code

```
import math
import numpy as np
import threading
import time as time
from datetime import datetime, timedelta
import pause
import xlsxwriter
from dmxpy import DmxPy
import serial.tools.list_ports
from flask import Flask, render_template, request, flash, send_file

import horstfx
import mavospec_base as mavo


##_____________________________Initial Variables_____________________________

steps_count = 3
dmx_add = 1
zeit = 60
wait = 0
dmx_min = 0
dmx_max = 255
dmx_add_2 = 0
dmx_add_3 = 0

pause_until = datetime.now()

invertdmx = False
dmx_2_bool = False
dmx_3_bool = False
dmx_16_bool = False
ref_measure = False
dmx_render = False


```

```
H_x = 0.425  #initial robot position
H_y = 0.0
H_z = 0.4
H_rot = 0
H_speed = 0.7
H_step = 0.1
H_x_Home = H_x #set positions as home
H_y_Home = H_y
H_z_Home = H_z

d = 5  # Preset distance robot - spectrometer
r = 0.2  # Preset Radius Beam -------- HORST MAXIUMUM: ca. 0.6m

##_____________________________________Startup_____________________________________
app = Flask(__name__)
app.secret_key = 'lichtschwein'

import logging
log = logging.getLogger('werkzeug')
log.setLevel(logging.ERROR) #set Flasks logging level to just errors


def findComPort(pid,vid): #Function to find the ComPorts to the MAVOSPEC BASE and the
    USB-DMX-Interface
    com_port = "ERROR"

    ports = list(serial.tools.list_ports.comports())

    for p in ports:
        if pid and vid in p.hwid:
            com_port = p.device
            print("DEBUG: found serial device with VID: "+vid+" and PID: "+pid+" at "
   +com_port)
    return com_port

dmx_com  = findComPort(pid="04D8",vid="FA63")  #USB-DMX Interface PID and HID
mavo_com = findComPort(pid="1CD7",vid="4001")  #MAVOSPEC-BASE PID and HID


try: #try to connect to MAVOSPEC BASE
    mavo.Connect(mavo_com)
    mavo_success = True
except: #Error message if try fails
    print("ERROR: Cannot connect to Mavospec Base")
    mavo_success = False
try: #try to connect to USB-DMX Interface
    dmx = DmxPy.DmxPy(dmx_com, default_level=0)
```

```
82      dmx_success = True
83  except:
84      print("ERROR: Cannot connect to USB-DMX-Interface")
85      dmx_success = False
86
87  try: #try to connect to HORST900
88      #"""
89      Horst = horstfx.Horst("horstFX", "WZA3P9", "2.0.0.25:8080/xml-rpc")
90      Horst.moveHorst(H_x, H_y, H_z, H_rot, H_speed)
91      H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z = Horst.
        getCurrentRobotPosition()
92      Horst.printPosition("xyz+rz")
93      #"""
94
95      horst_success = True
96  except:
97      print("ERROR: Cannot connect to Horst")
98      horst_success = False
99
100
101 def threadFunc(): #Function to start DMX protocol in different thread than the main
        loop for continuous DMX output
102     while (dmx_render):
103         dmx.render()
104         if not dmx_render:
105             break
106
107 print("")
108 print("Open link for interface: http://127.0.0.1:5000/")
109
110
111 ##_____________________________Webpage_Functions_______________________________
112
113
114 @app.route('/user_input', methods=['POST']) #user_input route returns the variables
        that the user has set
115 def user_input():
116     if request.method == "POST":
117         global steps_count, zeit, wait, dmx_add, dmx_min, dmx_max, d, invertdmx,
        dmx_add_2,dmx_add_3,dmx_2_bool,dmx_3_bool, dmx_16_bool
118
119         print("USER INPUT VARIABLES")
120
121         if request.form.get("schritte") != "": #try to save inputs and check for
        correct input type
122             try:
123                 steps_count = int(request.form.get("schritte"))
```

```
124             except ValueError: #if ValueError occurs, the program should not stop,
    but inform about the incorrect input
125                 print("VALUE ERROR - Steps_Count")
126                 flash("Oops! Looks like you didn't enter an integer.") #flash an
    error on the webinterface
127         if request.form.get("abstand") != "":
128             try:
129                 d = float(request.form.get("abstand"))
130             except ValueError:
131                 print("VALUE ERROR - Abstand")
132                 flash("Oops! Looks like you didn't enter an integer or float.")
133         if request.form.get("zeit") != "":
134             try:
135                 if int(request.form.get("zeit")) >= 45:
136                     zeit = int(request.form.get("zeit"))
137                 else:
138                     flash("Value must be greater than 45s.")
139             except ValueError:
140                 print("VALUE ERROR - Zeit")
141                 flash("Oops! Looks like you didn't enter an integer.")
142         if request.form.get("wait") != "":
143             try:
144                 if int(request.form.get("wait")) >= 0:
145                     wait = int(request.form.get("wait"))
146                 else:
147                     flash("Value must be greater than 0min.")
148             except ValueError:
149                 print("VALUE ERROR - Zeit")
150                 flash("Oops! Looks like you didn't enter an integer.")
151
152         if request.form.get("dmx") != "":
153             try:
154                 dmx_add = int(request.form.get("dmx"))
155             except ValueError:
156                 print("VALUE ERROR - DMX")
157                 flash("Oops! Looks like you didn't enter an integer.")
158
159         if request.form.get("dmx_min") != "":
160             try:
161                 dmx_min = int(request.form.get("dmx_min"))
162             except ValueError:
163                 print("VALUE ERROR - DMX")
164                 flash("Oops! Looks like you didn't enter an integer.")
165         if request.form.get("dmx_max") != "":
166             try:
167                 dmx_max = int(request.form.get("dmx_max"))
168             except ValueError:
```

```
169                 print("VALUE ERROR - DMX")
170                 flash("Oops! Looks like you didn't enter an integer.")
171         if request.form.get("dmx_add_2") != "":
172             try:
173                 dmx_add_2 = int(request.form.get("dmx_add_2"))
174             except ValueError:
175                 print("VALUE ERROR - DMX")
176                 flash("Oops! Looks like you didn't enter an integer.")
177         if request.form.get("dmx_add_3") != "":
178             try:
179                 dmx_add_3 = int(request.form.get("dmx_add_3"))
180             except ValueError:
181                 print("VALUE ERROR - DMX")
182                 flash("Oops! Looks like you didn't enter an integer.")
183
184
185         if request.form.get('Invert DMX') == 'Invert DMX':
186             invertdmx = not invertdmx #invert the flag with button press
187             print("DEBUG: Invert DMX: "+str(invertdmx))
188
189         if request.form.get('Use Second DMX-Address') == 'Use':
190             dmx_2_bool = not dmx_2_bool
191             print("DEBUG: Use Second DMX-Address: "+str(dmx_2_bool))
192
193         if request.form.get('Use Third DMX-Address') == 'Use':
194             dmx_3_bool = not dmx_3_bool
195             print("DEBUG: Use Third DMX-Address: "+str(dmx_3_bool))
196
197         if request.form.get('dmx_16') == '16 Bit':
198             dmx_16_bool = not dmx_16_bool
199             print("DEBUG: DMX 16 Bit: "+str(dmx_16_bool))
200
201
202         print("") #print user inputs in console
203         print("Steps: " + str(steps_count))
204         print("Wait: "+str(wait))
205         print("Zeit: " + str(zeit))
206         print("DMX Adresse: " + str(dmx_add))
207         print("DMX Min: " +str(dmx_min))
208         print("DMX Max: " +str(dmx_max))
209         print("DMX Adresse 2: " + str(dmx_add_2))
210         print("DMX Adresse 3: " + str(dmx_add_3))
211             #return the variables to the webinterface
212         return render_template("index.html", steps_count=steps_count, d=d, zeit=zeit,
    wait=wait, dmx_add=dmx_add,
213                                invertdmx=invertdmx, dmx_min=dmx_min, dmx_max=dmx_max,
    dmx_add_2=dmx_add_2,dmx_add_3=dmx_add_3,
```

```
214                                 dmx_2_bool=dmx_2_bool, dmx_3_bool=dmx_3_bool,
    dmx_16_bool=dmx_16_bool)
215
216
217 @app.route("/submitted", methods=['POST']) #/submitted route to save user inputs for
    x_measure and y_measure
218 def x_y_form():
219     if request.method == "POST":
220         global x_form
221         global y_form
222
223         x_form = str(request.form.get("x_selection"))
224         y_form = str(request.form.get("y_selection"))
225         print(x_form)
226         print(y_form)
227
228         print("Estimated duration of the measurement"+str(timedelta(seconds=
    estimated_time()))) #print and flash estimated measure time
229         flash("Estimated duration of the measurement: "+str(timedelta(seconds=
    estimated_time())))
230
231         return render_template("index.html", steps_count=steps_count, d=d, zeit=zeit,
     wait=wait, dmx_add=dmx_add,
232                                 invertdmx=invertdmx, dmx_min=dmx_min, dmx_max=dmx_max,
     dmx_add_2=dmx_add_2,dmx_add_3=dmx_add_3,
233                                 dmx_2_bool=dmx_2_bool, dmx_3_bool=dmx_3_bool,
    dmx_16_bool=dmx_16_bool)
234
235
236 @app.route('/', methods=['GET', 'POST']) #default "/" route for live user inputs
237 def index():
238     if request.method == 'GET': #"GET" request, when webinterface is called
239         global mavo_success, dmx_success, horst_success
240         global invertdmx
241
242         #flash errors in the webinterface, if connection to devices failed
243         if not mavo_success:
244             flash("ERROR: Cannot connect to Mavospec Base, please connect and restart
    ")
245         if not dmx_success:
246             flash("ERROR: Cannot connect to USB-DMX-Interface, please connect and
    restart")
247         if not horst_success:
248             flash("ERROR: Cannot connect to Horst, please connect and restart")
249
250     if request.method == 'POST':
251         global H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z, H_speed,
```

```
H_rot, H_step
    global H_x_Home, H_y_Home, H_z_Home
    global r

 #HORST CONTROL
    # set different step sizes for robot controll
    if request.form.get('Coarse') == 'Coarse':
        H_step = 0.1 #0.1m
        print("DEBUG: H_step: " + str(H_step))
    elif request.form.get('Fine') == 'Fine':
        H_step = 0.01 #1cm
        print("DEBUG: H_step: " + str(H_step))
    elif request.form.get('Extremely fine') == 'Extremely fine':
        H_step = 0.001 #1mm
        print("DEBUG: H_step: " + str(H_step))

    elif request.form.get('Left') == 'Left':
        print("DEBUG: User Input Horst Links")
        try: #too fast user inputs while the robot is still processing the
previous command leads to problems, therefore try
            Horst.moveHorst(H_x, H_y - H_step, H_z, H_rot, H_speed) #call the
moveHorst function
            H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z = Horst.
getCurrentRobotPosition() #save new current position
            Horst.printPosition("xyz+rz")
        except:
            print("DEBUG: ERROR Moving Horst")
    elif request.form.get('Right') == 'Right':
        print("DEBUG: User Input Horst Rechts")
        try:
            Horst.moveHorst(H_x, H_y + H_step, H_z, H_rot, H_speed)
            H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z = Horst.
getCurrentRobotPosition()
            Horst.printPosition("xyz+rz")
        except:
            print("DEBUG: ERROR Moving Horst")
    elif request.form.get('Up') == 'Up':
        print("DEBUG: User Input Horst Hoch")
        try:
            Horst.moveHorst(H_x, H_y, H_z + H_step, H_rot, H_speed)
            H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z = Horst.
getCurrentRobotPosition()
            Horst.printPosition("xyz+rz")
        except:
            print("DEBUG: ERROR Moving Horst")
    elif request.form.get('Down') == 'Down':
        print("DEBUG: User Input Horst Runter")
```

```
293             try:
294                 Horst.moveHorst(H_x, H_y, H_z - H_step, H_rot, H_speed)
295                 H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z = Horst.
    getCurrentRobotPosition()
296                 Horst.printPosition("xyz+rz")
297             except:
298                 print("DEBUG: ERROR Moving Horst")
299 
300         elif request.form.get('Store Home') == 'Store Home': #save current position
    as new home position
301             H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z = Horst.
    getCurrentRobotPosition()
302             H_x_Home = H_x
303             H_y_Home = H_y
304             H_z_Home = H_z
305             print("Store Home: " + str(H_x_Home) + " " + str(H_y_Home) + " " + str(
    H_z_Home))
306 
307         elif request.form.get('Load Home') == 'Load Home': #load and move to the
    stored home position
308             Horst.moveHorst(H_x_Home, H_y_Home, H_z_Home, H_rot, H_speed)
309             H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z = Horst.
    getCurrentRobotPosition()
310             print("Load and move Home: " + str(H_x_Home) + " " + str(H_y_Home) + " "
    + str(H_z_Home))
311 
312         elif request.form.get('Set Radius') == 'Set Radius': #set current y-position
    as the measure radius for the beam
313             if H_y != 0: #just store radius unequal to zero
314                 r = H_y
315                 if r < 0: #invert the negative value to positiv, if robot was moved
    to the left
316                     r = r * (-1)
317                 print("Radius: " + str(r))
318 
319         elif request.form.get("Reference measurement") == "Reference measurement": #
    reference measurement in the dark
320 
321             global CCT_ref, sensorTemp, ref_measure, int_time_ref
322 
323             mavo.measure() #trigger measurement
324             CCT_ref = mavo.CCT() #call color temperature values
325             sensorTemp = mavo.sensorTemp() #call CMOS-sensor temperature
326             int_time_ref = mavo.integrationTime() #call measurement integration time
327 
328             ref_measure = True #set reference measurement flag to 1
329 
```

```
330             mavo.beep("1") #give feedback of successfull measurement
331             flash("Reference measurement successfull!") #flash successfull
    measurement on the webinterface
332
333         elif request.form.get("Messreihe") == "Messreihe": #user starts measurement
    series
334             measure_series() #call the function to handle measurement series
335
336             file_path = str("static/xlsx/"+filename) #file path for the xlsx file on
    the server side
337             print(file_path)
338             return send_file(file_path, as_attachment=True, attachment_filename=
    filename) #gives the user a prompt in
339                                                                     # the
    webinterface to download the xlsx file
340
341     return render_template("index.html", steps_count=steps_count, d=d, zeit=zeit,
    wait=wait, dmx_add=dmx_add,
342                            invertdmx=invertdmx, dmx_min=dmx_min, dmx_max=dmx_max,
    dmx_add_2=dmx_add_2,
343                            dmx_add_3=dmx_add_3,
344                            dmx_2_bool=dmx_2_bool, dmx_3_bool=dmx_3_bool, dmx_16_bool=
    dmx_16_bool)
345
346
347 def measure_series(): #function to handle the measurement series
348
349     if steps_count >= 1:
350
351         global workbook, worksheet, spectrumsheet, bold, dmx_render,ref_measure,
    pause_until, filename
352         global H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z, H_speed,
    H_rot, H_step
353
354         if not dmx_16_bool: #if 8-bit channels:
355             if not invertdmx: dmx_val = dmx_min #if DMX values are to increase from
    low to high, initial value is the minimum
356             if invertdmx : dmx_val = dmx_max #accordingly the other way round
357         if dmx_16_bool: #if 16-bit channels:
358             if not invertdmx: msb, lsb = convert_16to8bit(dmx_min) #as with 8-bit,
    but with calculating the two 8-bit values
359             if invertdmx: msb, lsb = convert_16to8bit(dmx_max)
360             dmx_val = msb #set the DMX value variable to the MSB (most significant
    byte)
361             dmx_val_lsb = lsb #set a second DMX value variable to the LSB (least
    signifacant byte)
362
```

```
363         dmx.set_channel(dmx_add, dmx_val) #set DMX channel to initial DMX value
364         if dmx_2_bool: dmx.set_channel(dmx_add_2, dmx_val) #if user set flag to 1,
    set more DMX channels to initial DMX values
365         if dmx_3_bool: dmx.set_channel(dmx_add_3, dmx_val)
366 
367         if dmx_16_bool: #if 16-bit flag is 1:
368             dmx.set_channel(dmx_add+1, dmx_val_lsb) #set the very next channel to the
     least significant byte
369             if dmx_2_bool: dmx.set_channel(dmx_add_2+1, dmx_val_lsb)
370             if dmx_3_bool: dmx.set_channel(dmx_add_3+1, dmx_val_lsb)
371 
372         dmx_render = True #enable DMX output
373 
374         th = threading.Thread(target=threadFunc) #start DMX output in second thread
375         th.start()
376 
377 
378         print("DEBUG: INITIAL EXCEL SHEET")
379         mavo.beep("1") #user feedback that initiating measurement series started
380         now = datetime.now() #store current time
381         filename = now.strftime("%d_%m_%Y_%H_%M_%S")+" "+x_form+" "+y_form+".xlsx" #
    set filename to current date and time
382                                                                                 #
    + the selected x and y axis
383 
384         workbook = xlsxwriter.Workbook("static/xlsx/"+filename) #initate the XLSX
    file
385         bold = workbook.add_format({'bold': True}) #enable bold font in the XLSX file
386         infosheet = workbook.add_worksheet("Info") #add an info sheet to the XLSX
    file
387 
388          deviceInfo = mavo.deviceInfo() #get deviceInfo from the MAVOSPEC BASE
389          fw = mavo.fw() #get MAVOSPEC BASE firmware
390          hw = mavo.hw() #get MAVOSPEC BASE hardware revision
391 
392         #write Infosheet
393         infosheet.write(0, 0, "Messgert", bold)
394         infosheet.write(0, 1, deviceInfo[1] + " " + deviceInfo[3] + " " + deviceInfo
    [4])
395         infosheet.write(1, 0, "Seriennummer", bold)
396         infosheet.write(1, 1, deviceInfo[5])
397         infosheet.write(2, 0, "Hardware", bold)
398         infosheet.write(2, 1, hw[0])
399         infosheet.write(3, 0, "Firmware", bold)
400         infosheet.write(3, 1, fw[0])
401         infosheet.write(4, 0, "Datum", bold)
402         infosheet.write(4, 1, now.strftime("%d.%m.%Y"))
```

```
403         infosheet.write(5, 0, "Uhrzeit Start", bold)
404         infosheet.write(5, 1, now.strftime("%H:%M:%S"))
405         infosheet.write(7, 0, "Abstand in m", bold)
406         infosheet.write(7, 1, d)
407 
408         infosheet.set_column("A:A", 12.67) #set the A column to the size of 12.67
409 
410         if ref_measure: #add extra sheet to the XLSX file, if a reference measurement
     was executed
411             refsheet = workbook.add_worksheet("Referenzmessung")
412             # fill sheet with values
413             refsheet.write("A1", "Beleuchtungsstrke in Lux", bold)
414             refsheet.write("A2", "CCT in K", bold)
415             refsheet.write("A3", "duv", bold)
416             refsheet.write("B1", float(CCT_ref[2]))
417             refsheet.write("B2", int(CCT_ref[0]))
418             refsheet.write("B3", float(CCT_ref[1]))
419 
420             refsheet.write("A4", "Integrationszeit in ms", bold)
421             refsheet.write("B4", int(int_time_ref[0]))
422 
423             refsheet.write("A5", "Sensor Temperatur in  C ", bold)
424             refsheet.write("B5", float(sensorTemp[0].replace(",",".")))
425 
426             refsheet.set_column('A:A', 16.22)
427 
428         #add sheets to
429         worksheet = workbook.add_worksheet(x_form + " - " + y_form)
430         spectrumsheet = workbook.add_worksheet("Spektrum")
431 
432         mavo.beep("2") #user feedback that  initiating XLSX file is done and
    measurement series will start
433 
434         if x_form == "Hotspot": #if the "Hotspot" measurement is selected, move the
    robot to its home position
435             print("DEBUG: Move Horst Home")
436             Horst.moveHorst(H_x_Home, H_y_Home, H_z_Home, H_rot, H_speed)
437             H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z = Horst.
    getCurrentRobotPosition()
438             Horst.printPosition("xyz+rz")
439             time.sleep(0.5)
440 
441 
442         current_time = datetime.now() #save current time in variable
443         print(current_time)
444         pause_until = current_time + timedelta(0, wait * 60) #calculate the pause
    before the first measurement starts,
```

```
                                                    # if the user set a wait
    time

        global x,y

        for x in range(steps_count): #loop through the amount of measurements steps

            x_measure(x_form)
            y_measure(y_form)

        x += 1



        #
    _________PLOTS__________POTS_________PLOTS__________PLOTS__________PLOTS__________PLO

        zoom = 85 #set zoom size of plot sheets

     #initiate the plot sheets in the XLSX file, depending on the selected axis
        if y_form == "E,CCT,DUV":
            E_plotsheet = workbook.add_chartsheet("Plot " + x_form + " - " + "Bel.
    st rke")
            E_plot = workbook.add_chart({'type': 'scatter', 'subtype': '
    straight_with_markers'})
            E_plot.add_series({
                'name': [x_form + " - " + y_form, 0, 1],
                'categories': [x_form + " - " + y_form, 1, 0, x, 0],
                'values': [x_form + " - " + y_form, 1, 1, x, 1], })
            E_plot.set_legend({'none': True})
            E_plot.set_x_axis({'name': x_form, 'min': x_min, 'max': x_max})
            E_plot.set_y_axis({'name': "Beleuchtungsstrke [Lux]"})
            E_plot.set_title({'name': x_form + " / " + "Beleuchtungsstrke [Lux]"})
            E_plotsheet.set_chart(E_plot)
            E_plotsheet.set_zoom(zoom)

            CCT_plotsheet = workbook.add_chartsheet("Plot " + x_form + " - " + "CCT")
            CCT_plot = workbook.add_chart({'type': 'scatter', 'subtype': '
    straight_with_markers'})
            CCT_plot.add_series({
                'name': [x_form + " - " + y_form, 0, 2],
                'categories': [x_form + " - " + y_form, 1, 0, x, 0],
                'values': [x_form + " - " + y_form, 1, 2, x, 2], })
            CCT_plot.set_legend({'none': True})
            CCT_plot.set_x_axis({'name': x_form, 'min': x_min, 'max': x_max})
            CCT_plot.set_y_axis({'name': "CCT [K]", 'min': 2700, 'max': 8000})
            CCT_plot.set_title({'name': x_form + " / " + "CCT [K]"})
```

```
            CCT_plotsheet.set_chart(CCT_plot)
            CCT_plotsheet.set_zoom(zoom)

            DUV_plotsheet = workbook.add_chartsheet("Plot " + x_form + " - " + "Duv")
            DUV_plot = workbook.add_chart({'type': 'scatter', 'subtype': '
straight_with_markers'})
            DUV_plot.add_series({
                'name': [x_form + " - " + y_form, 0, 3],
                'categories': [x_form + " - " + y_form, 1, 0, x, 0],
                'values': [x_form + " - " + y_form, 1, 3, x, 3], })
            DUV_plot.set_y_axis({})
            DUV_plot.set_legend({'none': True})
            DUV_plot.set_x_axis({'name': x_form, 'min': x_min, 'max': x_max})
            DUV_plot.set_y_axis({'name': "Duv", 'min': -0.1, 'max': 0.1})
            DUV_plot.set_title({'name': x_form + " / " + "Duv"})
            DUV_plotsheet.set_chart(DUV_plot)
            DUV_plotsheet.set_zoom(zoom)
        # elif y_form == "XY,UV":
        elif y_form == "CRI":
            plotsheet = workbook.add_chartsheet("Plot " + x_form + " - " + y_form)
            plot = workbook.add_chart({'type': 'scatter', 'subtype': '
straight_with_markers'})
            plot.add_series({
                'name': [x_form + " - " + y_form, 0, 1],
                'categories': [x_form + " - " + y_form, 1, 0, x, 0],
                'values': [x_form + " - " + y_form, 1, 1, x, 1], })
            plot.set_legend({'none': True})
            plot.set_x_axis({'name': x_form, 'min': x_min, 'max': x_max})
            plot.set_y_axis({'name': y_form})
            plot.set_title({'name': x_form + " / " + y_form})
            plotsheet.set_chart(plot)
            plotsheet.set_zoom(zoom)
        elif y_form == "TM30":
            plotsheet = workbook.add_chartsheet("Plot " + x_form + " - " + y_form)
            plot = workbook.add_chart({'type': 'scatter', 'subtype': '
straight_with_markers'})
            plot.add_series({
                'name': [x_form + " - " + y_form, 0, 1],
                'categories': [x_form + " - " + y_form, 1, 0, x, 0],
                'values': [x_form + " - " + y_form, 1, 1, x, 1], })
            plot.add_series({
                'name': [x_form + " - " + y_form, 0, 2],
                'categories': [x_form + " - " + y_form, 1, 0, x, 0],
                'values': [x_form + " - " + y_form, 1, 2, x, 2], })
            plot.set_x_axis({'name': x_form, 'min': x_min, 'max': x_max})
            plot.set_y_axis({'name': y_form})
            plot.set_title({'name': x_form + " / " + y_form})
```

```
530             plotsheet.set_chart(plot)
531             plotsheet.set_zoom(zoom)
532         elif y_form == "TLCI":
533             plotsheet = workbook.add_chartsheet("Plot " + x_form + " - " + y_form)
534             plot = workbook.add_chart({'type': 'scatter', 'subtype': '
    straight_with_markers'})
535             plot.add_series({
536                 'name': [x_form + " - " + y_form, 0, 1],
537                 'categories': [x_form + " - " + y_form, 1, 0, x, 0],
538                 'values': [x_form + " - " + y_form, 1, 1, x, 1], })
539             plot.set_legend({'none': True})
540             plot.set_x_axis({'name': x_form, 'min': x_min, 'max': x_max})
541             plot.set_y_axis({'name': y_form})
542             plot.set_title({'name': x_form + " / " + y_form})
543             plotsheet.set_chart(plot)
544             plotsheet.set_zoom(zoom)
545 
546         spectrumplotsheet = workbook.add_chartsheet("Plot Spektrum")
547         spectrumplot = workbook.add_chart({'type': 'scatter', 'subtype': 'straight'})
548         spectrumplot.add_series({
549             'categories': '=Spektrum!$B$1:$CD$1',
550             'values': '=Spektrum!$B$2:$CD$2',
551             'line': {'color': 'white'},
552         })
553         spectrumplot.set_plotarea({
554             'gradient': {
555                 'colors': ['#B000AE', '#FF00FC', '#0000FF', '#00FFFF', '#00FF00', '#
    FFFF00', '#FFA200', '#FF0000',
556                            '#000000'],
557                 'positions': [0, 5, 17, 26, 37, 50, 55, 76, 100],
558                 'angle': 0.01}
559         })
560         spectrumplot.set_legend({'none': True})
561         spectrumplot.set_x_axis({'name': "[nm]", 'min': 380, 'max': 780})
562         spectrumplot.set_y_axis({'name': "[mW/ m  /nm]"})
563         spectrumplot.set_title({'name': "Spektrum - Messung 1"})
564         spectrumplotsheet.set_chart(spectrumplot)
565         spectrumplotsheet.set_zoom(zoom)
566 
567     now = datetime.now() #store current time in variable to log the time the
    measurement series finished
568     infosheet.write(6, 0, "Uhrzeit Stop", bold)
569     infosheet.write(6, 1, now.strftime("%H:%M:%S"))
570 
571     Horst.moveHorst(H_x_Home, H_y_Home, H_z_Home, H_rot, H_speed) #move Horst home
572     H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z = Horst.
    getCurrentRobotPosition()
```

```
573     Horst.printPosition("xyz+rz")
574 
575     workbook.close() #close and save the XLSX file
576 
577     ref_measure = False #reset the reference measurement flag
578     dmx_render = False  #disable the DMX output
579 
580     mavo.beep("3") #user feedback that the measurement series is finished
581     print("DEBUG: workbook closed")
582 
583     return
584 
585 
586 # _____________________________Subfunctions_____________________________
587 
588 def x_measure(x_form):
589     global dmx_add, x_min, x_max, d, current_time, pause_until
590 
591     if x_form == "DMX": #if selected x axis is DMX initiate log sheets in XLSX file
592         worksheet.write(0, 0, "DMX in % ", bold)
593         spectrumsheet.write(0, 0, "DMX in %", bold)
594 
595         dmx_arr = np.linspace(dmx_min, dmx_max, steps_count, endpoint=True) #
    calculate an array with DMX values between
596                                                                             #the min.
     and max. DMX value with equal spacing
597         if invertdmx: dmx_arr = dmx_arr[::-1] #invert the DMX value array if flag is
    1
598 
599         if not dmx_16_bool: #if 8-bit values:
600             dmx_val = round(dmx_arr[x]) #round the float values in the array to
    integers
601             if dmx_val > 255: dmx_val = 255 #if an error occurs that the values are
    out of range, set values in range
602             if dmx_val < 0: dmx_val = 0
603         if dmx_16_bool:
604             msb,lsb = convert_16to8bit(round(dmx_arr[x])) #calculate the value from
    the array into two 8 bit values
605                                                           # for a 16-bit output on
    two DMX channels
606             dmx_val = msb
607             dmx_val_lsb = lsb
608 
609 
610         print("DEBUG: dmx_arr: " + str(dmx_arr))
611         print("DEBUG: dmx_val: " + str(dmx_val))
612 
```

```
613         x_val = round(((dmx_val-dmx_min)/(dmx_max-dmx_min))*100) # calculate
    percentage value from minimal and maximal DMX value
614         worksheet.write(x + 1, 0, x_val) # write percentage values into XLSX file
615         spectrumsheet.write(x + 1, 0, x_val)
616 
617         dmx.set_channel(dmx_add, dmx_val) # set DMX channel to calculated DMX value
618         if dmx_2_bool: dmx.set_channel(dmx_add_2, dmx_val) # if flag for more DMX
    values is 1 set these channels also
619         if dmx_3_bool: dmx.set_channel(dmx_add_3, dmx_val)
620 
621         if dmx_16_bool: # if 16-bit output is enabled:
622             dmx.set_channel(dmx_add+1, dmx_val_lsb) # set DMX value of the very next
    channel to the LSB
623             if dmx_2_bool: dmx.set_channel(dmx_add_2+1, dmx_val_lsb)
624             if dmx_3_bool: dmx.set_channel(dmx_add_3+1, dmx_val_lsb)
625         time.sleep(1) # wait 1 second for mechanical parts in the moving head to get
    in position
626 
627         x_min = 0 # set value range for the plot
628         x_max = 100
629 
630         return
631 
632     elif x_form == "Zeitmessung": #if selected x axis is a timed measurement initiate
     log sheets in XLSX file
633 
634         if x + 1 <= steps_count: # if the last measurement has not yet happened, then
    :
635             if x == 0: # if it is the first measurement, then:
636                 mavo.beep("3") # user feedback, if user wants to measure brightness
    progression over time
637                 current_time = pause_until # set the variable current_time to the
    time, which was stored in the variable
638                                            # pause_until, until which is waited
    before the next measurement
639 
640             if x > 0: # if it is NOT the first measurement
641                 pause_until = current_time + timedelta(0,zeit) # store the time to
    wait before the measurement in
642                                                                # the variable
    pause_until, which is the sum of the
643                                                                # current time and the
     time variable set by the user
644 
645                 current_time = pause_until # overwrite the current_time variable with
     the new calculated time for the next loop
646 
```

```
647         worksheet.write(0, 0, "Zeit in Minuten", bold)
648         spectrumsheet.write(0, 0, "Zeit in Minuten", bold)
649         worksheet.write(x + 1, 0, (zeit*x)/60) # log time in minutes not seconds in
    the XLSX file
650         spectrumsheet.write(x + 1, 0, (zeit*x)/60)
651 
652         x_min = 0 # set value range for plot in minutes
653         x_max = zeit*(steps_count-1)/60
654 
655         return
656 
657     elif x_form == "Hotspot": # if selected x axis is a hotspot measurement initiate
    log sheets in XLSX file
658 
659         phi = round(math.degrees(np.arccos(-r / d) - math.radians(90)), 3) #
    calculate the angle the robot needs to rotate
660                                                                             # and
    round to and round to 3 decimal places
661         print("DEBUG: phi: " + str(phi))
662         x_min = -r # set value range for the plot
663         x_max = r
664 
665         #DELETE? phi_array = []
666         x_array = []
667         y_array = []
668 
669         phi_array = np.linspace(-phi, phi, steps_count, endpoint=True) # create an
    array that mirrors the max. phi and
670                                                                 # calculates lineare
    spacings between the measurement points
671 
672         for i in (phi_array):  # calculate x and y values for every phi value in
    phi_array
673             x_rel = -(d * np.sin(math.radians(90 - i)) - d)
674             x_array.append(np.round(x_rel, 3)) # append the value to the array
675 
676             y_rel = d * np.cos(math.radians(90 - i))
677             y_array.append(np.round(y_rel, 3))
678 
679         print(phi_array)
680         print(x_array)
681         print(y_array)
682 
683         print(
684             "Move X : " + str(round(float(H_x + x_array[x]),2)) + " Move Y :" + str(
    round(float(H_y + y_array[x]),2))
685             + " Move Z :" + str(round(float(H_z),2)) + " Move RZ:" + str(round(float(
```

```
    phi_array[x]),2)))
686         # move robot to the calculated position
687         Horst.moveHorst(float(H_x + x_array[x]), float(H_y + y_array[x]), float(H_z),
     float(phi_array[x]), H_speed)
688         time.sleep(0.5)  # wait for robot to arrive
689 
690         worksheet.write(0, 0, "Abstand zu Mittelpunkt [m]", bold)  # write y value in
     XLSX file
691         spectrumsheet.write(0, 0, "Abstand zu Mittelpunkt [m]", bold)
692         worksheet.write(x + 1, 0, round(y_array[x],2))
693         spectrumsheet.write(x + 1, 0, round(y_array[x],2))
694         return
695     else:
696         print("DEBUG: no known X-Form selected")
697         return
698 
699 
700 def y_measure(y_form):
701     print("Pause until " + str(pause_until))
702     pause.until(pause_until)  # pause until the time in variable pause_until for
    timed measuring
703     print("")
704     print("Messung " + str(x + 1))
705     print("DEBUG: Now: "+str(datetime.now()))
706     print("mavo.measure()")
707     mavo.measure()  # trigger measurement
708 
709     if y_form == "E,CCT,DUV":  # if user set y-axis to "E,CCT,DUV":
710 
711         # print("DEBUG: cct y-form")
712 
713         worksheet.write(0, 1, "Beleuchtungsstrke in Lux", bold)  # name rows in
    XLSX file
714         worksheet.write(0, 2, "CCT in K", bold)
715         worksheet.write(0, 3, "duv", bold)
716         worksheet.write(0, 4, "Integrationszeit in ms", bold)
717 
718         mes_cct = mavo.CCT()  # call measurement values from MAVOSPEC
719         int_time = mavo.integrationTime()
720         CCT = mes_cct[0]  # store measurement values in own variables
721         DUV = mes_cct[1]
722         E = mes_cct[2]
723 
724         worksheet.write(x + 1, 1, float(E))  # log measurement values in XLSX file
725         worksheet.write(x + 1, 2, int(CCT))
726         worksheet.write(x + 1, 3, float(DUV))
727         worksheet.write(x + 1, 4, int(int_time[0]))
```

```
728
729     elif y_form == "XY,UV":
730         # print("DEBUG: XY y-form")
731
732         worksheet.write(0, 1, "X", bold)
733         worksheet.write(0, 2, "Y", bold)
734         worksheet.write(0, 3, "U", bold)
735         worksheet.write(0, 4, "V", bold)
736         worksheet.write(0, 5, " U  ", bold)
737         worksheet.write(0, 6, " V  ", bold)
738         worksheet.write(0, 7, "Integrationszeit in ms", bold)
739
740         mes_CC = mavo.colorCoordiantes()
741         int_time = mavo.integrationTime()
742         X = mes_CC[0]
743         Y = mes_CC[1]
744         U1 = mes_CC[2]
745         V1 = mes_CC[3]
746         U2 = mes_CC[4]
747         V2 = mes_CC[5]
748
749         worksheet.write(x + 1, 1, float(X))
750         worksheet.write(x + 1, 2, float(Y))
751         worksheet.write(x + 1, 3, float(U1))
752         worksheet.write(x + 1, 4, float(V1))
753         worksheet.write(x + 1, 5, float(U2))
754         worksheet.write(x + 1, 6, float(V2))
755         worksheet.write(x + 1, 7, int(int_time[0]))
756
757     elif y_form == "CRI":
758
759         # print("DEBUG: CRI y-form")
760
761         worksheet.write(0, 1, "Ra", bold)
762         worksheet.write(0, 2, "R1", bold)
763         worksheet.write(0, 3, "R2", bold)
764         worksheet.write(0, 4, "R3", bold)
765         worksheet.write(0, 5, "R4", bold)
766         worksheet.write(0, 6, "R5", bold)
767         worksheet.write(0, 7, "R6", bold)
768         worksheet.write(0, 8, "R7", bold)
769         worksheet.write(0, 9, "R8", bold)
770         worksheet.write(0, 10, "R9", bold)
771         worksheet.write(0, 11, "R10", bold)
772         worksheet.write(0, 12, "R11", bold)
773         worksheet.write(0, 13, "R12", bold)
774         worksheet.write(0, 14, "R13", bold)
```

```
775         worksheet.write(0, 15, "R14", bold)
776         worksheet.write(0, 16, "R15", bold)
777         worksheet.write(0, 17, "Integrationszeit in ms", bold)
778 
779         mes_CRI = mavo.CRI()
780         int_time = mavo.integrationTime()
781         Ra = mes_CRI[0]
782         R1 = mes_CRI[3]
783         R2 = mes_CRI[4]
784         R3 = mes_CRI[5]
785         R4 = mes_CRI[6]
786         R5 = mes_CRI[7]
787         R6 = mes_CRI[8]
788         R7 = mes_CRI[9]
789         R8 = mes_CRI[10]
790         R9 = mes_CRI[11]
791         R10 = mes_CRI[12]
792         R11 = mes_CRI[13]
793         R12 = mes_CRI[14]
794         R13 = mes_CRI[15]
795         R14 = mes_CRI[16]
796         R15 = mes_CRI[17]
797 
798         worksheet.write(x + 1, 1, float(Ra))
799         worksheet.write(x + 1, 2, float(R1))
800         worksheet.write(x + 1, 3, float(R2))
801         worksheet.write(x + 1, 4, float(R3))
802         worksheet.write(x + 1, 5, float(R4))
803         worksheet.write(x + 1, 6, float(R5))
804         worksheet.write(x + 1, 7, float(R6))
805         worksheet.write(x + 1, 8, float(R7))
806         worksheet.write(x + 1, 9, float(R8))
807         worksheet.write(x + 1, 10, float(R9))
808         worksheet.write(x + 1, 11, float(R10))
809         worksheet.write(x + 1, 12, float(R11))
810         worksheet.write(x + 1, 13, float(R12))
811         worksheet.write(x + 1, 14, float(R13))
812         worksheet.write(x + 1, 15, float(R14))
813         worksheet.write(x + 1, 16, float(R15))
814         worksheet.write(x + 1, 17, int(int_time[0]))
815 
816     elif y_form == "TM30":
817 
818         # print("DEBUG: tm30 y-form")
819 
820         worksheet.write(0, 1, "Rf", bold)
821         worksheet.write(0, 2, "Rg", bold)
```

```
822         worksheet.write(0, 3, "Integrationszeit in ms", bold)
823 
824         mes_tm30 = mavo.TM30()
825         int_time = mavo.integrationTime()
826         Rf = mes_tm30[0]
827         Rg = mes_tm30[1]
828 
829         worksheet.write(x + 1, 1, float(Rf))
830         worksheet.write(x + 1, 2, float(Rg))
831         worksheet.write(x + 1, 3, int(int_time[0]))
832 
833     elif y_form == "TLCI":
834         # print("DEBUG: TLCI y-form")
835 
836         worksheet.write(0, 1, "TLCI", bold)
837         worksheet.write(0, 2, "Integrationszeit in ms", bold)
838 
839         mes_tlci = mavo.TLCI()
840         int_time = mavo.integrationTime()
841         TLCI = mes_tlci[0]
842 
843         worksheet.write(x + 1, 1, int(TLCI))
844         worksheet.write(x + 1, 2, int(int_time[0]))
845 
846     else:
847         print("DEBUG: no known Y-Form selected")
848 
849     # write wavelength from 380-780nm in 5nm steps in first row of spectrumsheet
850     s = 380
851     j = 1
852     while s <= 780:  # loop until 780bn
853         spectrumsheet.write(0, j, s, bold)  # write wavelength in first row and
    always in 5nm steps in the next column
854         s += 5
855         j += 1
856 
857     spectrumsheet.write(0, j, "[nm]", bold)
858     spectrumsheet.write(1, j, "[mW/m  /nm]", bold)
859 
860     mes_spec = mavo.spectrum()  # call measured spectrum from MAVOSPEC BASE
861 
862     i = 0
863     while i <= 80:  # write the 80 measurement points in XLSX file
864         spectrumsheet.write(x + 1, i + 1, float(mes_spec[i]))
865         i += 1
866     return
867 
```

```
868
869 def convert_16to8bit(x, n_bytes=2, order='big'):  # function to convert a 16-byte
    value to two 8-bit values
870     msb, lsb = x.to_bytes(n_bytes, byteorder=order)  # using the built in to_bytes
    function from python
871     return (msb, lsb)
872
873
874 def estimated_time():   # estimate the time it will take the measurement series to
    finish
875     est_time = 0
876     est_time += 20
877     est_time += wait*60
878
879     if x_form == "Zeitmessung":
880         est_time += (steps_count-1) * zeit
881         est_time += 30
882         return est_time
883
884     if y_form == "E,CCT,DUV":
885         est_time += steps_count * 27
886         return est_time
887     elif y_form == "XY,UV":
888         est_time += steps_count * 20
889         return est_time
890     elif y_form == "CRI":
891         est_time += steps_count * 20
892         return est_time
893     elif y_form == "TM30":
894         est_time += steps_count * 20
895         return est_time
896     elif y_form == "TLCI":
897         est_time += steps_count * 20
898         return est_time
```

Listing 6.2: Python-Code

6.2.4 Gossen MAVOSPEC BASE Python-Bibliothek

```
#!/usr/bin/python
import serial
import time

def Connect(comPort):
    global serialPort
    serialPort = serial.Serial(port=comPort, baudrate=9600, bytesize=8, timeout=5,
   stopbits=serial.STOPBITS_ONE)


def Disconnect():
    serialPort.close()


def CDCCommand(command, replylength,):
    c = command
    sendC = c + "\r\n"
    serialPort.write(bytes(sendC, encoding='utf8'))  # write a string
    time.sleep(0.1)
    x = serialPort.read(replylength).decode('utf-8')
    return x.split()


# Get Sensor Information
def sensorInfo():
    command = "sensorinfo?"
    replylength = 70
    return CDCCommand(command, replylength)


# Get Device Information
def deviceInfo():
    command = "deviceinfo?"
    replylength = 135
    return CDCCommand(command, replylength)


# Get Firmware Revision
def fw():
    command = "fw?"
    replylength = 6
    return CDCCommand(command, replylength)


# Get Hardware Revision
def hw():
```

```
    command = "hw?"
    replylength = 3
    return CDCCommand(command, replylength)


# Get Calibration Data
def calibrationData():
    command = "calibrationdata?"
    replylength = 55 * 64
    return CDCCommand(command, replylength)


# Get Wavelength (Pixel)
def wavelength():
    command = "wavelength?"
    replylength = 52 * 64
    return CDCCommand(command, replylength)


# Get IntegrationTime
def integrationTime():
    command = "integrationtime?"
    replylength = 10
    return CDCCommand(command, replylength)


# Get the Status Flags
def statusFlags():
    command = "flags?"
    replylength = 70
    return CDCCommand(command, replylength)


# Get Raw spectral data
def rawSpectralData():
    command = "rawspectraldata?"
    replylength = 26 * 64
    return CDCCommand(command, replylength)


# Get battery voltage
def batteryVoltage():
    command = "batvoltage"
    replylength = 6
    return CDCCommand(command, replylength)


```

```
# Get sensor temperature
def sensorTemp():
    command = "temp?"
    replylength = 8
    return CDCCommand(command, replylength)


# Get single Ambient Sensor Conversion
def ambientSensorConversion():
    command = "ambiconversion?"
    replylength = 10
    return CDCCommand(command, replylength)


# Start a Flicker Measurement
def doFlickerMeasure():
    command = "doflickermeasure"
    replylength = 64
    return CDCCommand(command, replylength)


# Get all Ambient Sensor Data
def ambientSensorData():
    command = "ambisensordata?"
    replylength = 52 * 64
    return CDCCommand(command, replylength)


# Get calculated flicker values
def flickerDataRaw():
    command = "flickerdata?"
    replylength = 52 * 64
    return CDCCommand(command, replylength)


# Spectrum in 5nm Steps
def spectrum():
    command = "calculateddata? 01"
    replylength = 18 * 64
    return CDCCommand(command, replylength)


# CCT, duv, Evis, Ee, LamdaPeak,LambdaPeak_Val, Lambda_Dominant, Purity
def CCT():
    command = "calculateddata? 02"
    replylength = 55
    return CDCCommand(command, replylength)
```

```
140
141
142 # Color Coordinates (x/y, u/v,  u  / v  )
143 def colorCoordiantes():
144     command = "calculateddata? 03"
145     replylength = 50
146     return CDCCommand(command, replylength)
147
148
149 # CRI Values
150 def CRI():
151     command = "calculateddata? 04"
152     replylength = 3 * 64
153     return CDCCommand(command, replylength)
154
155
156 # TM30 Values
157 def TM30():
158     command = "calculateddata? 05"
159     replylength = 23 * 64
160     return CDCCommand(command, replylength)
161
162
163 # FLicker Data
164 def flickerData():
165     command = "calculateddata? 06"
166     replylength = 64
167     return CDCCommand(command, replylength)
168
169
170 # PAR Data
171 def PAR():
172     command = "calculateddata? 07"
173     replylength = 12 * 64
174     return CDCCommand(command, replylength)
175
176
177 # TLCI Data
178 def TLCI():
179     command = "calculateddata? 08"
180     replylength = 16 * 64
181     return CDCCommand(command, replylength)
182
183
184 # Get currently activated Report Items
185 def reportItems():
186     command = "reportitems?"
```

```
187     replylength = 32
188     return CDCCommand(command, replylength)
189
190
191 # Get Device Time and Date
192 def deviceTimeDate():
193     command = "datetime?"
194     replylength = 32
195     return CDCCommand(command, replylength)
196
197
198 # Get currently Displayed Function
199 def displayFunction():
200     command = "displayfunction?"
201     replylength = 2
202     return CDCCommand(command, replylength)
203
204
205 # Get current Backlight Setting
206 def backlight():
207     command = "backlight?"
208     replylength = 4
209     return CDCCommand(command, replylength)
210
211
212 ###Missing Functions
213
214 # Start Measurement
215 def measure():
216     command = "keycode M"
217     replylength = 64
218     sendC = command + "\r\n"
219     serialPort.write(bytes(sendC, encoding='utf8'))  # write a string
220     time.sleep(3)
221     x = serialPort.read(replylength).decode('utf-8')
222     return x.split()
223
224
225 ###Missing Functions
226
227 def beep(beep="1"):
228     # 1: short beep
229     # 2: long beep
230     # 3: error beep
231     # Anything else = short beep
232     command = "beep" + beep
233     replylength = 64
```

```
234     return CDCCommand(command, replylength)
235
236
237 version = '0.1'
```

Listing 6.3: Gossen MAVOSPEC BASE Python-Bibliothek

6.2.5 fruitcore HORST900 Python-Bibliothek

```
import xmlrpc.client
import numpy as np
import math

class Horst:
    def __init__(self, user="horstFX", password="WZA3P9", url="127.0.0.1:8080/xml-rpc"):
        self.URL = "@" + url
        self.USERNAME = user
        self.PASSWORD = password
        self.client = xmlrpc.client.ServerProxy("http://" + self.USERNAME + ":" + self.PASSWORD + self.URL)
        print("Initialized xmlrpc client for user '" + self.USERNAME + "'")


    def getCurrentRobotPosition(self):
        pos = self.client.HorstFX.v1.Robotcontrol.getCurrentRobotPosition()
        H_q0 = pos.get("q0")
        H_q1 = pos.get("q1")
        H_q2 = pos.get("q2")
        H_q3 = pos.get("q3")
        H_rx = pos.get("rx")
        H_ry = pos.get("ry")
        H_rz = pos.get("rz")
        H_x = pos.get("x")
        H_y = pos.get("y")
        H_z = pos.get("z")

        return H_q0, H_q1, H_q2, H_q3, H_rx, H_ry, H_rz, H_x, H_y, H_z


    def getCurrentRobotJoints(self):
        joints = self.client.HorstFX.v1.Robotcontrol.getCurrentRobotJoints()
        H_j1 = joints.get("j1")
        H_j2 = joints.get("j2")
        H_j3 = joints.get("j3")
        H_j4 = joints.get("j4")
        H_j5 = joints.get("j5")
        H_j6 = joints.get("j6")

        return H_j1, H_j2, H_j3, H_j4, H_j5, H_j6


    def moveLinear(self, H_x, H_y, H_z, H_q0, H_q1, H_q2, H_q3, H_speed):
        result = self.client.HorstFX.v1.Robotcontrol.moveLinear(H_x, H_y, H_z, H_q0, H_q1, H_q2, H_q3, H_speed)
```

```
        return result


    def moveJoint(self, H_x, H_y, H_z, H_q0, H_q1, H_q2, H_q3, H_speed):
        result = self.client.HorstFX.v1.Robotcontrol.moveJoint(H_x, H_y, H_z, H_q0, H_q1, H_q2, H_q3, H_speed)  # speed
        return result


    def moveHorst(self, H_x, H_y, H_z, H_rot, H_speed):
        H_rot -= 90  # -90 Grad Verschiebung, damit 0 Grad Messgert Richtung Scheinwerfer entspricht
        H_rot = math.radians(H_rot)
        q = get_quaternion_from_eulers(-3.14159265359, 0.0, H_rot)
        H_q0 = float(q[0])
        H_q1 = float(q[1])
        H_q2 = float(q[2])
        H_q3 = float(q[3])

        if H_z > 0.215 and -0.61 <= H_y <= 0.61: #Limits f r Horst
            result = self.client.HorstFX.v1.Robotcontrol.moveJoint(H_x, H_y, H_z, H_q0, H_q1, H_q2, H_q3, H_speed)  # speed
            return result

        else:
            print("ERROR: Value out of Range!")


    def printPosition(self,detail= "xyz+rz"):
        import app
        if detail == "xyz+rz":
            print("Horst_x: " + str(round(app.H_x,3)))
            print("Horst_y: " + str(round(app.H_y,3)))
            print("Horst_z: " + str(round(app.H_z,3)))
            print("Horst_rz: " + str(round(app.H_rz,3)))
        if detail == "xyz+rxryrz":
            print("Horst_x: " + str(app.H_x))
            print("Horst_y: " + str(app.H_y))
            print("Horst_z: " + str(app.H_z))
            print("Horst_rx: " + str(app.H_rx))
            print("Horst_ry: " + str(app.H_ry))
            print("Horst_rz: " + str(app.H_rz))
        if detail == "xyz+r+q":
            print("Horst_x: " + str(app.H_x))
            print("Horst_y: " + str(app.H_y))
            print("Horst_z: " + str(app.H_z))
            print("Horst_rx: " + str(app.H_rx))
```

```
            print("Horst_ry: " + str(app.H_ry))
            print("Horst_rz: " + str(app.H_rz))
            print("Horst_q0: " + str(app.H_q0))
            print("Horst_q1: " + str(app.H_q1))
            print("Horst_q2: " + str(app.H_q2))
            print("Horst_q3: " + str(app.H_q3))


def get_quaternion_from_eulers(yaw, pitch, roll):
    """
    https://automaticaddison.com/how-to-convert-euler-angles-to-quaternions-using-python/

    Convert an Euler angle to a quaternion.

    Input
      :param roll: The roll (rotation around x-axis) angle in radians.
      :param pitch: The pitch (rotation around y-axis) angle in radians.
      :param yaw: The yaw (rotation around z-axis) angle in radians.

    -Horst has a ZYX-orientation

    Output
      :return qx, qy, qz, qw: The orientation in quaternion [x,y,z,w] format
    """
    qx = np.sin(roll / 2) * np.cos(pitch / 2) * np.cos(yaw / 2) - np.cos(roll / 2) * np.sin(pitch / 2) * np.sin(yaw / 2)
    qy = np.cos(roll / 2) * np.sin(pitch / 2) * np.cos(yaw / 2) + np.sin(roll / 2) * np.cos(pitch / 2) * np.sin(yaw / 2)
    qz = np.cos(roll / 2) * np.cos(pitch / 2) * np.sin(yaw / 2) - np.sin(roll / 2) * np.sin(pitch / 2) * np.cos(yaw / 2)
    qw = np.cos(roll / 2) * np.cos(pitch / 2) * np.cos(yaw / 2) + np.sin(roll / 2) * np.sin(pitch / 2) * np.sin(yaw / 2)

    return [qx, qy, qz, qw]
```

Listing 6.4: fruitcore HORST900 Python-Bibliothek

Ich versichere, die vorliegende Arbeit selbständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Hamburg, 21. Juli 2022 Jan Huesmann