



Hochschule Darmstadt
- Fachbereich Elektro- und Informationstechnik -

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Engineering

im Studiengang Elektro- und Informationstechnik
Schwerpunkt Automatisierung und Informationstechnik

Thema: Entwicklung eines USB basierten LED Analyzers

Autor: Said Subhan Waizi
MatNr. 729316

Version vom: 20. August 2015

Betreuer: Christoph Thelen
Hilscher Gesellschaft für Systemautomation mbH
Rheinstraße 15
65795, Hattersheim

Betreuer Prof.: Prof. Dr.-Ing. Peter Fromm
Korreferent: Prof. Dr.-Ing. Klaus Schäfer

Inhaltsverzeichnis

Abbildungsverzeichnis	3
Tabellenverzeichnis	3
Listingverzeichnis	3
Abkürzungsverzeichnis	4
1 Hilscher Gesellschaft für Systemautomation mbH	5
2 Einleitung	6
2.1 Ausgangssituation	6
2.2 Zielsetzung	7
3 Grundlagen der Farbmessung	9
3.1 Das CIE-Normvalenzsystem	9
3.2 Die CIE-Normfarbtafel	10
3.3 Anpassung der Berechnungsalgorithmen an den TCS3472	12
4 Entwicklung der Softwarekomponenten	16
4.1 Entwicklung einer effizienten API	17
4.1.1 I/O-Handling auf dem FTDI-Chip	18
4.1.2 Die Software-I ² C Bibliothek	19
4.1.3 Die TCS3472 Bibliothek	22
4.1.4 Integration auf Geräteebene	24
4.1.5 Fehlererkennung	26
4.2 Verarbeitung der Messwerte	27
4.2.1 Der Einsatz von Lua	27
4.2.2 Einbindung der API	28
4.2.3 Die Lua Komponenten	28
4.3 Die graphische Oberfläche	29
4.3.1 Die graphische Bibliothek - wxWidgets	30
4.3.2 Die Funktionsweise der Applikation	30
4.3.3 Die Regeln des GUI Desgins	32
4.3.4 Vorstellung der GUI Applikation	35
5 Das Buildsystem	37
5.1 GitHub, CMake & Travis CI	37
5.2 Die Verwendung der Komponenten im Projekt	38
6 Fazit	40
7 Anhang	41
7.1 Farbraumtransformationen und photometrische Größen	41
7.2 Versuchsaufbau - Messung von LEDs	42
7.3 GUI - Screenshots	46
Literaturverzeichnis	48

Abbildungsverzeichnis

1	Color Controller	6
2	Gewichtungen der drei Primärfarben als Funktion der Wellenlänge	10
3	CIE-Normfarbtafel	11
4	Spektrale Empfindlichkeit des TCS3472 Farbsensors	13
5	Spektrale Empfindlichkeit des TCS3472 im XYZ- und Yxy-Farbraum . .	13
6	Angepasste spektrale Empfindlichkeit des TCS3472 im Yxy-Farbraum .	14
7	Funktionsdiagramm der Kommunikation	17
8	Beispiel I ² C Read Transfer	20
9	I ² C Umsetzung auf dem FTDI Chip	21
10	Aufnahme eines I ² C Lesebefehls mit dem Logic Analyzer	22
11	TCS3472 Pinbelegung	22
12	Sukzessives Auslesen aller Farbregister	23
13	Reihenfolge der Geräteverbindung	26
14	Abhängigkeitsdiagramm der Lua Module	27
15	Vgl. der Möglichkeiten für die Einbindung der API und der Lua Skripte	31
16	Kommunikationskomponente zwischen C/C++ und Lua Code	32
17	Anwendung der Richtlinien des UI Designs	33
18	GitHub - Eine Gesamtübersicht	40
19	Versuchsaufbau - parallele Messung mit 16 Sensoren	42
20	GUI - Hauptfenster	46
21	GUI - Testdefinition	47

Tabellenverzeichnis

1	MPSSE Befehlsatz für I/O-Operationen	18
2	Mögliche Fehler der vier Module	26
3	Aufgabe der einzelnen Komponenten im Buildsystem	39
4	Überblick über photometrische Größen	41

Listingverzeichnis

1	Sollwerttabelle für einen LED-Test	43
2	XML Ausschnitt eines erfolgreichen LED Tests	44
3	XML Ausschnitt eines fehlgeschlagenen LED Tests	45

Abkürzungsverzeichnis

IR	Infrarot	12
ams	austriamicrosystems	7
USB	Universal Serial Bus	7
LED	Leuchtdiode.	7
GUI	Graphical User Interface.	8
DLL	Dynamic Link Library	28
API	Application Programming Interface.	7
ALS	Ambient Light Sensing.	22
ADC	Analog-to-Digital-Converter	24
LWL	Lichtwellenleiter	16
VID	Vendor-ID.	25
PID	Product-ID	25
XML	Extensible Markup Language	29
MSB	Most Significant Bit	19
ACK	Acknowledge	20
CIE	Commission Internationale d'Eclairage	9
VCS	Version Control System	37
RGBC	Red, Green, Blue und Clear	22
FTDI	Future Technology Devices International	7
MPSSE	Multi-Protocol Synchronous Serial Engine	18
CDATAL	Clear Data Low Byte.	23
BDATAH	Blue Data High Byte.	23

Die in dieser Arbeit erwähnten Unternehmens-, Produkt- oder Markenbezeichnungen können Marken oder eingetragene Markenzeichen der jeweiligen Eigentümer sein. Die Wiedergabe von Marken- und/oder Warenzeichen in dieser Arbeit berechtigt nicht zu der Annahme, dass diese frei von Rechten Dritter zu betrachten seien. Alle erwähnten Marken- und/oder Warenzeichen unterliegen uneingeschränkt den länderspezifischen Schutzbestimmungen und den Besitzrechten der jeweiligen eingetragenen Eigentümer.

1 Hilscher Gesellschaft für Systemautomation mbH

Das 1986 gegründete Familienunternehmen Hilscher Gesellschaft für Systemautomation mbH beschäftigt sich mit industriellen Kommunikationssystemen. Anfangs entwickelte das Unternehmen kundenspezifische Steuerungstechnik. 1990 erweiterte Hilscher sein Produktspektrum um die damals entstehenden Technologien der Feldbusstechnik. Zurzeit sind über 200 Mitarbeiter weltweit für Hilscher tätig, davon über 150 am Standort Hattersheim am Main. Im Bereich Feldbusstechnologie beherrscht Hilscher alle industrierelevanten Kommunikationsprotokolle. Somit wird jährlich ein Umsatz von ca. 24 Millionen Euro erwirtschaftet.

2004 entwickelte Hilscher den ersten netX Chip, ein ARM basierter Netzwerkcontroller. Mit seinen speziell entwickelten Kommunikationsprozessoren bietet er eine flexible Lösung, mit der alle Feldbus- und Real-Time-Ethernet-Systeme dieser Zeit realisiert werden können. Das ist ein enormer Vorteil gegenüber der Konkurrenz, die für jedes Kommunikationssystem einen speziellen Chip benötigt. Heute umfasst die netX Familie 6 Controller, die sich hauptsächlich in der verwendeten ARM CPU und der Anzahl der realisierbaren Kommunikationskanäle unterscheiden. Zwei weitere Chips, geprägt von den Schlagwörtern *Internet of Things* und *Industry 4.0*, sind zur Zeit in der Entwicklung und werden das Produktportfolio um die Zukunftstechnologien der Automatisierungstechnik ergänzen. Durch den programmierbaren Kommunikationsteil sind die netX Controller auch für zukünftige Protokolle bestens gerüstet.

Das Portfolio von Hilscher erstreckt sich von fertigen Kommunikationslösungen inklusive Windows Konfiguration bis hin zu kundenspezifischen Entwicklungen. Weiterhin stellt Hilscher rund um die netX Familie technische Informationen und eine Entwicklungsumgebung zur Verfügung.

2 Einleitung

Die Hilscher Gesellschaft für Systemautomation mbH verfügt über eine eigene Fertigung mit angeschlossener Testabteilung. Dort werden die Platinen für die Kommunikationssysteme automatisiert bestückt, gelötet und getestet. Um dem hohen Qualitätsanspruch gerecht zu werden, kommen neben ausgiebigen Softwaretests zusätzlich moderne Testsysteme, wie zum Beispiel der Flying-Probe-Tester und das 3D Röntgensystem, zum Einsatz. Eventuell vorhandene Leuchtdioden werden bei Geräten mit hohen Stückzahlen durch einen Farbsensor überprüft. Bei Serien mit niedrigeren Stückzahlen müssen die Leuchtdioden jedoch durch das Personal der Testabteilung verifiziert werden, da in diesem Fall die kostspieligen industriellen Farbsensoren nicht wirtschaftlich sind. Dieser zusätzliche vom Testpersonal vorzunehmende Schritt ist zum einen fehleranfällig, zum anderen entspricht er nicht dem Ziel der Testoptimierung, welche von der Hilscher Gesellschaft für Systemautomation mbH angestrebt wird. Das Projekt befasst sich mit der Entwicklung einer preissensitiven Lösung, mit der eine Reihe von Leuchtdioden auf den gefertigten Platinen auf ihre Funktion getestet werden kann. So kann bereits bei Baugruppen, die in kleineren Stückzahlen gefertigt werden, ein voll automatisierter LED-Test erreicht werden.

2.1 Ausgangssituation

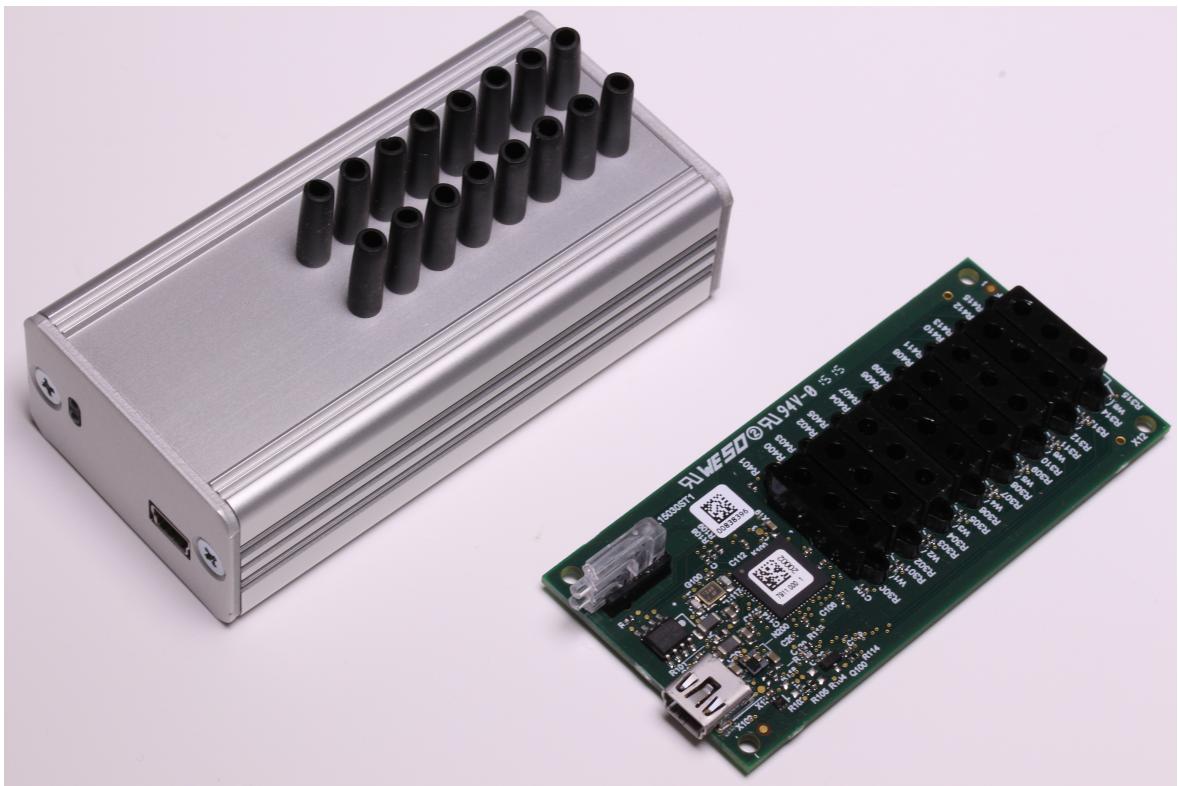


Abbildung 1: *Color Controller*

Die folgende Arbeit baut auf eine funktionstüchtige und getestete Hardware auf, die im Rahmen eines zuvor abgeleisteten Praktikums bei der Hilscher GmbH entwickelt wurde. Abbildung 1 zeigt die fertiggestellte Hardware, im Folgenden *Color Controller* genannt. Die Platine befindet sich auf der rechten Seite, das Gehäuse inklusive Platine ist auf der linken Seite der Abbildung zu sehen. Die Platine ist mit einem Universal Serial Bus (USB) Wandlerbaustein der Firma Future Technology Devices International (FTDI), einem Speicherbaustein (*EEPROM*) und 16 Farbsensoren der Firma austriamicrosystems (ams) ausgestattet. Der FTDI Chip ermöglicht eine einfache USB Anbindung der Hardware an jeden handelsüblichen PC und erlaubt so auch die Integration in die eigene Testumgebung von Hilscher. Im Zusammenspiel mit einem *EEPROM*, welches firmenspezifische Deskriptoren und die Seriennummer aufnehmen kann, können mehrere dieser Geräte verbunden und in frei konfigurierbarer Reihenfolge geöffnet werden. Die vier Farbkanäle des TCS3472 Farbsensors ermöglichen RGB- und Helligkeitsmessungen. Die Messergebnisse des Sensors werden dem Chip über eine I²C-Schnittstelle zugänglich gemacht und über USB an den PC weitergeleitet, der für die Weiterverarbeitung verantwortlich ist. Die zu testenden Leuchtdioden (LEDs) werden mit den Messkanälen des *Color Controllers* über Lichtwellenleiter verbunden. Ein Gehäuse schützt die Farbsensoren vor störendem Umgebungslicht und sorgt zusätzlich für mehr Stabilität der Hardware.

2.2 Zielsetzung

Im Rahmen der Bachelorarbeit soll die bestehende Hardwareeinheit des *Color Controllers* durch die Softwareentwicklung zu einem Gesamtprodukt erweitert werden. Zielsetzung der Arbeit ist die Fertigstellung eines Softwarepaketes, welches die Fertigung von Hilscher zum automatisierten Testen von LEDs einsetzen kann. Dafür werden zunächst die Grundlagen der Farbmessung erläutert und ein Lösungsansatz ausgearbeitet, mit dem die rohen RGBC Messdaten des Sensors in eine dominante Wellenlänge überführt werden können. Der verwendete TCS3472 Farbsensor besitzt eine feste I²C-Adresse, die weder durch Hardwareanschaltung noch durch Software verändert werden kann. Aus diesem Grund soll eine Software-I²C Bibliothek, die auf dem FTDI Chip 16 voneinander unabhängige I²C-Busse kontrollieren kann, das Grundgerüst der Ansteuerungsroutinen bilden. Über eine Application Programming Interface (API) können alle angeschlossenen *Color Controller* erkannt, geöffnet und ausgelesen werden. Auf diese API aufbauend werden zwei Programme entwickelt. Das erste Programm soll sich um die Weiterverarbeitung der Messwerte kümmern. Hier wird der Lösungsansatz für die Bereitstellung der dominanten Wellenlänge aus den RGBC Daten implementiert. Weiterhin enthält dieses Programm die Routinen, die für das Testen von Leuchtdioden benötigt werden. In diesem Rahmen wird ein grundlegendes Konzept für den LED-Test

entwickelt, mithilfe dessen die Funktionstüchtigkeit von LEDs verifiziert werden kann. Bei einem fehlgeschlagenen Test sollen verschiedene Fehlerursachen der Leuchtdioden erkannt und unterschieden werden können. Das zweite zu entwickelnde Programm, die Graphical User Interface (GUI) Applikation, zielt auf den Einsatz bei händischer Bedienung im Labor. Sie zeigt die gemessenen Farbwerte der angeschlossenen Testgeräte an, ermöglicht es, Einstellungen an den Farbsensoren vorzunehmen und realisiert die LED Stimulation an den zu testenden Baugruppen. Darüber hinaus soll die graphische Oberfläche dem Benutzer die Möglichkeit geben, einen vollständigen LED Test generieren zu können, der die Initialisierung des *Color Controllers*, die Bekanntmachung der Sollwerte der zu testenden Leuchtdioden, die Stimulation der Leuchtdioden und letztendlich die Validierung dieser beinhaltet. Der generierte Test kann anschließend in der Testumgebung von Hilscher eingebettet werden. Bei den Programmen wird besonderer Wert darauf gelegt, dass nur freie Software Komponenten verwendet werden, sodass die Lösung auf "GitHub" veröffentlicht werden kann. Um den Continuous Integration Service "*Travis CI*" zum Erstellen der Software benutzen zu können, sollen die entwickelten Programme per Cross-Platform-Build unter Linux gebaut werden können.

3 Grundlagen der Farbmessung

Mithilfe des *Color Controllers* sollen LED Parameter, wie die dominante Wellenlänge¹ und die Helligkeit gemessen werden können. Neben dem reinen Farbtest kann die Prüfung von Helligkeiten Auskunft über Fehler auf der Platine, wie z.B. Kurzschlüsse (zu hell) oder falsche Vorwiderstände (zu hell / zu dunkel), liefern. Im Idealfall erfolgen die Messungen mit solch einer Genauigkeit, dass sie mit den im Datenblatt der LED spezifizierten Werten übereinstimmen. Die Bestimmung der Sollwerte für einen LED Test könnte sich dadurch allein auf das Datenblatt stützen, was den Vorgang der Testerstellung um einiges vereinfachen würde. Der verwendete TCS3472 Farbsensor kann nativ jedoch nur Werte im RGB-Farbraum messen. Im folgenden Kapitel wird ein Lösungsansatz ausgearbeitet, der die Überführung eines Punktes aus dem RGB-Farbraum des Farbsensors in eine entsprechende dominante Wellenlänge ermöglicht. In diesem Zusammenhang erfolgt zuerst eine Einführung in die für die Berechnungen erforderliche Farbmatrik². Auf diesen Grundlagen aufbauend wird ein Berechnungsalgorithmus vorgestellt, der auf die spektralen Empfindlichkeiten des TCS3472 Farbsensors angepasst ist. Ein Überblick über die relevanten photometrischen Größen und die verwendeten Gleichungen für die Farbtransformationen sind im Anhang (Kap. 7.1) zu finden.

3.1 Das CIE-Normvalenzsystem

Das Normvalenzsystem wurde 1931 von der Commission Internationale d'Eclairage (CIE) entwickelt und stellt eine Relation zwischen der menschlichen Farbwahrnehmung und den physikalischen Ursachen des Farbreizes, der Wellenlänge, her[18]. Dieses Modell entspricht dem Sehempfinden der meisten Menschen und basiert auf umfangreichen Tests und Versuchen an einer Vielzahl von Testpersonen. Dabei sollten die Testpersonen (Beobachter) durch Ändern der Helligkeit an drei verfügbaren Lichtquellen, einen jeweils vorgegebenen Farbeindruck nachstellen. Es liegt der Ansatz zugrunde, dass sich jede vom menschlichen Auge wahrnehmbare Farbe, durch eine additive Mischung von drei geeigneten Primärfarben erzeugen ließe. Für die Erzeugung der drei Primärfarben wurde eine rote (700 nm), grüne (546.1 nm) und blaue (435.8 nm) Quecksilberdampflampe verwendet[6]. Abbildung 2 veranschaulicht das Ergebnis dieses Experiments. Hier sind die Gewichtungen der drei verwendeten Primärfarben als Funktion der Wellenlänge dargestellt. Die Empfindlichkeitskurven sind von Person zu Person gewissen Schwankungen unterworfen, als Mittelwerte sind sie jedoch als sogenannte Normalbeobachter (CIE Standard Observer) festgelegt. Der sich aus diesen drei Funktionen ergebende Farbraum wird als CIE-XYZ, bzw. XYZ-Farbraum bezeichnet.

¹Wellenlänge, die den Farbton einer LED so beschreibt, wie ihn das menschliche Auge empfindet

²Qualitative und quantitative Beschreibung von Farbe, die an das menschliche Auge angepasst ist

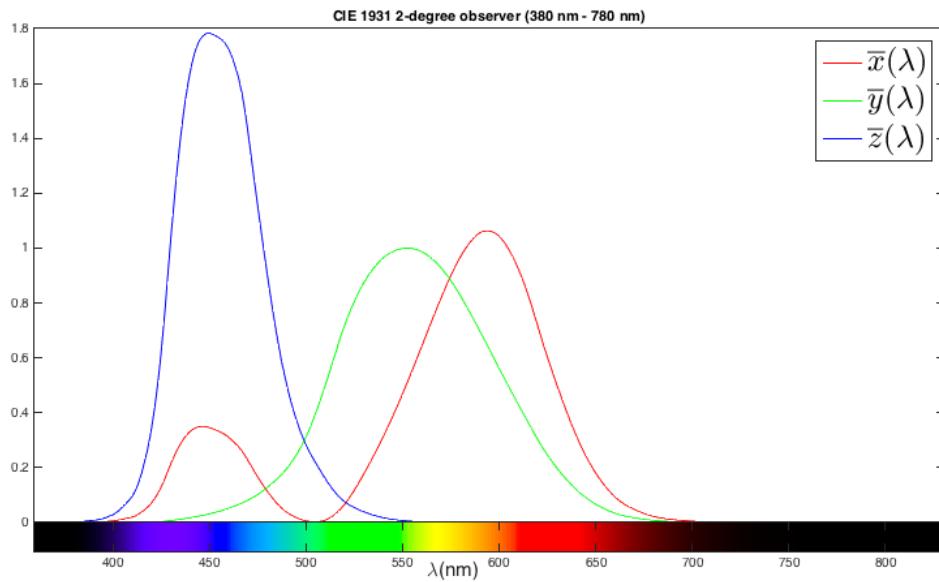


Abbildung 2: Gewichtungen der drei Primärfarben als Funktion der Wellenlänge

3.2 Die CIE-Normfarbtafel

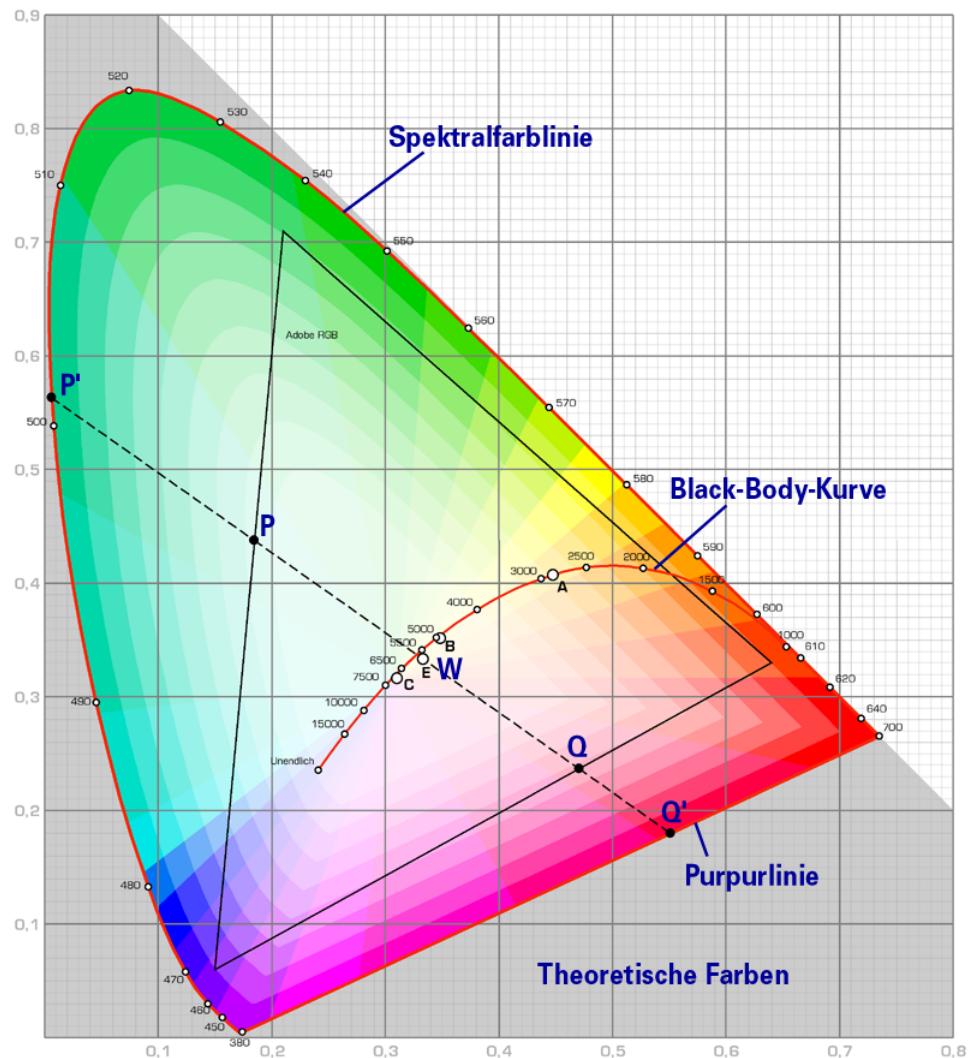
Um den in Abbildung 2 dargestellten Farbraum einerseits übersichtlich darzustellen und andererseits einen direkten Bezug zur wahrgenommenen Farbe herzustellen, wurde die zweidimensionale CIE-Normfarbtafel (Abb. 3) entwickelt. Innerhalb der hufeisenförmigen Kurve befindet sich die Menge aller vom Menschen wahrnehmbarer Farben. Da Punkte in diesem zweidimensionalen Farbraum durch ein x/y-Koordinatenpaar beschrieben werden, wird er auch CIE-Yxy oder Yxy-Farbraum genannt[4]. Dieser Farbraum kann nach Gleichungen (2)(3)(4) direkt aus dem XYZ-Farbraum abgeleitet werden. Im Folgenden werden die relevanten Elemente der CIE-Farbtafel näher erläutert.

Der Weißpunkt W befindet sich mit den Koordinaten ($x=1/3$, $y=1/3$) im mittleren Bereich der Fläche. Dieser kann je nach Beleuchtungssituation variieren und so praktisch jede Koordinate der Farbtafel einnehmen. In Bezug auf den Weißpunkt spielt die sogenannte Farbtemperatur, die sich auf der *Black-Body-Kurve* befindet, eine größere Rolle.

Die Farbtemperatur gibt an, auf welche Temperatur (Kelvin) man einen Plakschen Strahler³ erhitzten müsste, um den entsprechenden Farbeindruck auf der Black-Body-Kurve zu erhalten. Die *Black-Body-Kurve* beginnt bei niedrigen Temperaturen im rötlichen Bereich und läuft mit steigenden Temperaturen in den bläulichen Bereich über. In der LED-Messtechnik wird die Farbtemperatur für die Beschreibung von weißen LEDs verwendet[9].

³idealer schwarzer Körper

⁴Bildquelle Abb.3: <https://de.wikipedia.org/wiki/Datei:CIE-Normfarbtafel.png>

Abbildung 3: CIE-Normfarbtafel⁴

Die Spektralfarben befinden sich auf dem Rand der hufeisenförmigen Kurve und werden als die "reinen" Farben mit der höchsten Sättigung bezeichnet. Die Sättigung besitzt am Weißpunkt W den Wert Null und steigt mit zunehmendem Abstand zu W, bzw. Annäherung an die Spektralfarben an. Allen Farbpunkten innerhalb der Normfarbtafel, die auf einem Richtungsvektor liegen, der bei W beginnt und auf einen Punkt der Spektrallinie deutet, kann der gleiche Farbton zugewiesen werden. Die Punkte auf diesem Vektor unterscheiden sich lediglich in ihrer Sättigung. Der Gesamtheit der auf dem Vektor liegenden Punkte wird die dominante Wellenlänge zugeordnet, die sich am Schnittpunkt des Richtungsvektors mit der Spektrallinie ablesen lässt (vgl. [9] S.20). Als Beispiel soll hier der Vektor $\overrightarrow{WP'}$ betrachtet werden. Alle auf diesem Vektor $\overrightarrow{WP'}$ liegenden Farbpunkte besitzen den gleichen Farbton bzw. die gleiche dominante Wellenlänge, unterscheiden sich jedoch in der Sättigung. Die Aussagekraft der dominanten Wellenlänge nimmt mit Annäherung an den Weißpunkt (niedrige Sättigung) ab, bzw. mit Annäherung an die Spektralfarblinie (hohe Sättigung) zu. Folglich erscheint es lo-

gisch, dass für die Beschreibung weißer LEDs, die eine Sättigung nahe bei 0 besitzen, ein anderes Maß, wie z.B. die Farbtemperatur, verwendet wird.

Der RGB-Farbraum kann mithilfe einer $\text{RGB} \rightarrow \text{XYZ} \rightarrow \text{Yxy}$ Transformation auf der CIE-Normfarbtafel abgebildet werden. Hier besitzt er die Form eines Dreiecks, dessen Inhalt alle im RGB-Farbraum darstellbaren Farben abdeckt. Es ist ersichtlich, dass der RGB-Farbraum im Gegensatz zum CIE-Farbraum nicht in der Lage ist, alle wahrnehmbaren Farben zu beschreiben. Trotzdem kann einem Punkt des RGB-Farbraums anhand des oben beschriebenen Verfahrens eine dominante Wellenlänge auf der Spektralfarblinie zugeordnet werden. Um das zu verdeutlichen, soll der Punkt P, der im RGB-Farbraum liegt, betrachtet werden. Diesem Punkt kann die dominante Wellenlänge auf der Spektralfarblinie zugeordnet werden, die sich aus dem Schnittpunkt zwischen Verlängerung von \overrightarrow{WP} und der Spektralfarblinie ergibt. Somit besitzt der Farbpunkt P eine dominante Wellenlänge von circa 501nm.

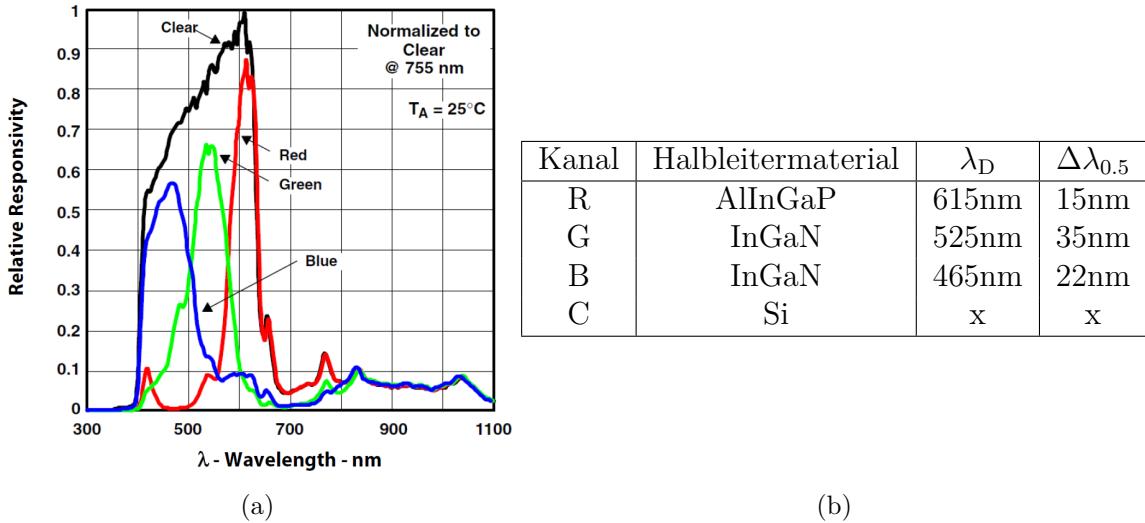
3.3 Anpassung der Berechnungsalgorithmen an den TCS3472

Um den im RGB-Farbraum gemessenen Daten des Sensors eine dominante Wellenlänge und Sättigung zuordnen zu können, muss der RGB-Farbraum des Sensors zunächst durch geeignete Farbumwandlungen in den Farbraum der Normfarbtafel transformiert werden. Die Grundlage für die Berechnungen bildet der im Datenblatt des Sensors dargestellte Verlauf der spektralen Empfindlichkeiten (Abb.4a). Die Messwerte für die Kurven wurden von ams empirisch ermittelt und sind Mittelwerte, die aus mehreren Messungen an einer Vielzahl von Sensoren gewonnen wurden. Die verwendeten Halbleitermaterialien der Photodioden sind auf der rechten Seite der Abbildung dargestellt. Sie sind maßgebend für den Verlauf der spektralen Empfindlichkeitskurven. Dabei stellt die dominante Wellenlänge λ_D ein Maß für Farbwahrnehmung dar, welche die LED im menschlichen Auge verursacht. Die spektrale Bandbreite bei halber Intensität wird durch $\Delta\lambda_{0.5}$ ausgedrückt. Um durch Infrarot (IR) Licht (ab ca. 780nm) verursachte Störungen so gering wie möglich zu halten, ist der Sensor mit einem IR-Filter ausgestattet. Die für die Zeichnung dieser Funktionen erforderlichen Messdaten wurden bei ams angefragt und in tabellarischer Form mit einer Schrittweite von 5nm zur Verfügung gestellt, sodass sie mit *Matlab*⁵ weiterverarbeitet werden konnten.

Die Messdaten werden auf den Helligkeitskanal normiert und anschließend in den XYZ-Farbraum ($\text{RGB} \rightarrow \text{XYZ}$) transformiert. Das Ergebnis der Transformation ist in Abbildung 5a zu sehen. Vergleicht man die Verläufe mit den CIE-XYZ Funktionen in Abbildung 2, lassen sich gewisse Ähnlichkeiten feststellen. Durch eine geeignete

⁵<http://de.mathworks.com/products/matlab/>

⁶Bildquelle Abb.4(links): TCS3472 Datenblatt [2] S. 10

Abbildung 4: Spektrale Empfindlichkeit des TCS3472 Farbsensors⁶

Wahl der Photodioden wurde den Sensoren künstlich der Verlauf der CIE-XYZ Kurven aufgeprägt. Die Abweichungen resultieren einerseits aus der Tatsache, dass der RGB-Farbraum im Gegensatz zum XYZ-Farbraum nicht alle wahrnehmbaren Farben darstellen kann. Zusätzlich weisen die Photodioden in den Anfangs- und Endbereichen ihrer spektralen Empfindlichkeit starke Nichtlinearitäten auf(vgl.[13] S.22/23). Dieser Sachverhalt wird besonders nach der letzten Farbraumtransformation (XYZ → Yxy) in Abbildung 5b verdeutlicht.

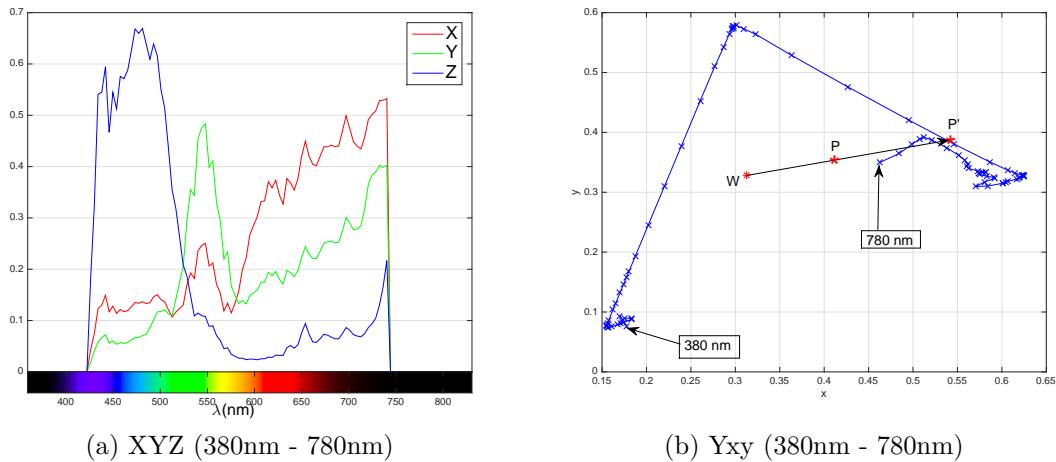


Abbildung 5: Spektrale Empfindlichkeit des TCS3472 im XYZ- und Yxy-Farbraum

Hier ist die spektrale Empfindlichkeit des Sensors und der Weißpunkt W (D65⁷) im Yxy-Farbraum dargestellt. Die Markierungen auf der Spektrallinie stellen die in 5nm Schrittweite gegebenen Wellenlängen von 380nm - 780nm dar. Die aktuelle Kurvenform (Dreieck) weist bereits sehr große Parallelen zum RGB-Farbraum auf, welcher auf der CIE-Normfarbtafel abgebildet ist. Störend wirken hier noch die Nichtlinearitäten der

⁷genormter Weißpunkt für Tageslicht mit einer Farbtemperatur von 6500K

Photodioden, die sich besonders im oberen und unteren Wellenlängenbereich zeigen. In diesem Bereich kann nach dem in Kapitel 3.2 erläuterten Ansatz keine eindeutige dominante Wellenlänge gefunden werden, da z.B. der Richtungsvektor $\overrightarrow{WP'}$ gleichzeitig mehrere Schnittpunkte mit der Spektrallinie besitzt. Die Kurve muss in den störenden Bereichen so verkürzt werden, dass der Algorithmus verwendet werden kann und trotzdem noch ein ausreichender Wertebereich für die Messung der dominanten Wellenlängen von LEDs zur Verfügung gestellt wird.

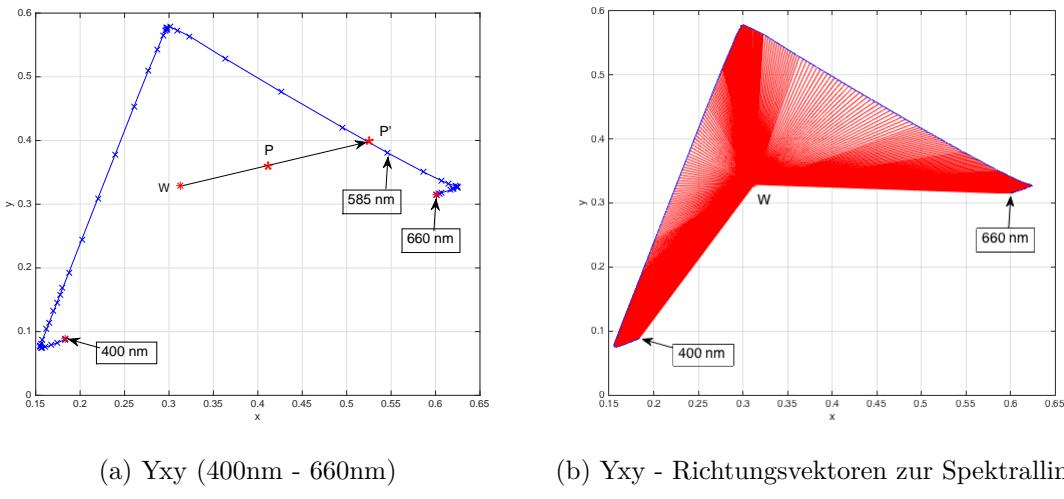


Abbildung 6: Angepasste spektrale Empfindlichkeit des TCS3472 im Yxy-Farbraum

Die verkürzte Spektrallinie ist in Abbildung 6a dargestellt. Auf dem Rand der Kurve befinden sich die Wellenlängen in einem Bereich von 400nm bis 660nm. Einem gemessenen Punkt P kann nun eindeutig eine dominante Wellenlänge auf der Spektrallinie zugeordnet werden, da sein entsprechend verlängerter Richtungsvektor nur noch einen Schnittpunkt mit der Spektrallinie besitzt. Werte unterhalb von 400nm und oberhalb von 660nm werden auf die entsprechenden Grenzen des Wertebereichs gekappt. Nach der Vorüberlegung, dass die Sättigung mit Entfernung von W, bzw. Annäherung an die Spektrallinie zunimmt (vgl.[9] S.21), beträgt sie an einem gemessenen Punkt P $Saturation = \frac{|\overrightarrow{WP}|}{|\overrightarrow{WP'}|}$. In einem letzten Schritt werden die in 5nm Schrittweite angegebenen Punkte auf der Spektrallinie so interpoliert, dass eine Granularität von weniger als 0.5nm erzielt wird. Wird ein Weißpunkt W festgelegt (hier D₆₅), kann nun ausgehend von diesem, ein Netz aus Richtungsvektoren zu den interpolierten Punkten auf der Spektrallinie aufgespannt werden (Abb. 6b). Die Richtungsvektoren dieses Netzes werden zum Zwecke einer effizienten Codeimplementation als *Look-Up-Table* zur Verfügung gestellt. Dieser arbeitet mit einem Index als *key* und gibt dementsprechend die dominante Wellenlänge als *value* zurück. Wurde ein Punkt P im Yxy-Farbraum berechnet, stellt man für diesen den Richtungsvektor \overrightarrow{WP} auf. Anschließend wird über das aufgespannte Netz der Richtungsvektoren iteriert und an jedem Element der Winkel zwischen \overrightarrow{WP} und Richtungsvektor zur Spektrallinie gebildet. Dabei wird der kleinste

Winkel gesucht und für den entsprechenden Vektor der Index aus dem Netz zurückgegeben. Der Richtungsvektor mit dem gefundenen Index entspricht der Verlängerung von \overrightarrow{WP} zur Spektrallinie.

Zusammenfassung des Algorithmus Gegeben sei ein vom TCS3472 Farbsensor gemessener Farbpunkt P_{RGBC} im RGBC-Farbraum. Für die Berechnung der dominanten Wellenlänge und Sättigung müssen folgende Schritte durchlaufen werden:

1. Normierung des gemessenen Punktes auf den C-Kanal $P_{RGBC} \rightarrow P_{RGB_n}$
2. Farbtransformation $P_{RGB_n} \rightarrow P_{XYZ}$ nach Gl.(1)
3. Farbtransformation $P_{XYZ} \rightarrow P_{Yxy}$ nach Gl.(2)(3)(4), im Folgenden nur P genannt
4. Aufstellung des Richtungsvektors \overrightarrow{WP}
5. Suche P' , der sich aus dem Schnittpunkt zwischen Verlängerung von \overrightarrow{WP} und der Spektrallinie ergibt
6. Gesuchte dominante Wellenlänge $\lambda_D(P) = \lambda_D(P')$
7. Sättigung = $\frac{|\overrightarrow{WP}|}{|\overrightarrow{WP'}|}$

Berechnung der Beleuchtungsstärke und Farbtemperatur Die Farbtemperatur wird für die Spezifizierung von weißen LEDs verwendet und wird in der Einheit Kelvin gemessen. Die Beleuchtungsstärke E_V berechnet sich aus dem Lichtstrom Φ_V , der auf eine Flächeneinheit A trifft und wird in der Einheit Lux angegeben. Sie beschreibt somit, wie stark ein Lichtempfänger, in unserem Fall der Farbsensor, beleuchtet wird. Bei LED Messungen kann die Beleuchtungsstärke als Referenz für die Helligkeit einer LED dienen, da sie eine Auskunft über den Lichtstrom gibt, der auf die aktive Sensorfläche trifft. Für die Berechnung der Farbtemperatur und der Beleuchtungsstärke stellt die Firma ams die sogenannte *Design Note DN 40*[1] zur Verfügung. In diesem Dokument werden die Berechnungen dieser beiden Parameter auf Grundlage des TCS3472 Farbsensors erläutert. Dabei werden die spektralen Empfindlichkeiten des Sensors in der Berechnung mitberücksichtigt, sodass eine gute Genauigkeit erzielt werden kann.

Fazit Der erarbeitete Lösungsansatz für die Bestimmung der dominanten Wellenlänge und Sättigung lässt effiziente und wiederholbare Messungen zu. In Kombination mit den von ams bereitgestellten technischen Dokumenten zur Berechnung der Beleuchtungsstärke und Farbtemperatur lassen sich alle für einen LED Test erforderlichen Größen mit einer sehr guten Genauigkeit messen. Dies konnte mithilfe eines *Demo Boards* der Firma Würth, auf dem sich eine Vielzahl von LEDs verschiedener Ausführung und Farbe befindet, getestet werden. Da auf diesem Board jede LED mit dem genauen Typ

gekennzeichnet ist, lässt sich im Produktkatalog der Firma das entsprechende Datenblatt und somit auch die für die LED spezifizierten Parameter finden. Der bis dato in der Testabteilung von Hilscher eingesetzte industrielle LED-Analysator[7] der Firma Feasa, welcher nicht zuletzt aufgrund seines hohen Preises durch den *Color Controller* abgelöst wird, besitzt einen Wertebereich von 450nm - 650nm. Der sich hier ergebende ähnliche Wertebereich von 400nm - 660nm deckt den Großteil der im industriellen Feld zum Einsatz kommenden LEDs ab. Die in diesem Kapitel vorgestellten Ansätze können jedoch erweitert und verbessert werden. So ist z.B. der Weißpunkt D₆₅ einer von vielen vordefinierten Weißpunkten auf der *Black-Body-Kurve*. Hier könnte eine Evaluation verschiedener Weißpunkte, vor allem in Bezug auf die Sättigungsberechnung, für bessere Messergebnisse sorgen. Dabei muss jedoch beachtet werden, dass das aktuell aufgespannte Netz der Richtungsvektoren nur für den Weißpunkt D₆₅ berechnet wurde. Bei einem Vergleich der Weißpunkte müssten diese Richtungsvektoren für den entsprechenden Weißpunkt neu definiert werden. Die Überführung einer vom Sensor gemessenen Beleuchtungsstärke in die im Datenblatt der LED angegebene Lichtstärke ist möglich, wenn der Testaufbau vollständig bekannt ist. In die Berechnung fließen sowohl die Entfernung zwischen Sensor und LED als auch der Abstrahlwinkel der Leuchtdiode ein. Weiterhin müssen Verluste, die durch eine Lichtführung im Lichtwellenleiter (LWL) entstehen können, berücksichtigt werden. Entsprechend müsste für jeden Testaufbau eine Neuanpassung der Berechnung erfolgen, da hier der Abstand zwischen Sensor und LED und die Länge der Lichtwellenleiter meist variiert. Für eine Helligkeitsreferenz, wie sie z.B. in der Testabteilung von Hilscher erwünscht ist, reicht die Beleuchtungsstärke vollkommen aus. Die Messung der im Datenblatt einer LED spezifizierten Lichtstärke könnte in der Hardwareabteilung Gebrauch finden. Hier könnte in Zukunft mithilfe entsprechender Versuchsaufbauten ein Ansatz für die Überführung von Beleuchtungsstärke in Lichtstärke ausgearbeitet werden, der an den TCS3472 Farbsensor angepasst ist.

4 Entwicklung der Softwarekomponenten

Das folgende Kapitel befasst sich mit der Entwicklung der Softwarekomponenten für die Ansteuerung der Hardware und die Verarbeitung und Visualisierung der Messergebnisse auf Seiten des PC's. Zunächst wird eine effiziente API vorgestellt, mit deren Hilfe man mehrere Geräte ansteuern und die Messwerte über USB an den PC übertragen kann. Ein in der Skriptsprache Lua geschriebenes Programm bindet diese API ein und stellt Routinen für die Farbumwandlung und Validierung von LEDs bereit. Hier wird der in Kapitel 3 erläuterte Algorithmus implementiert, der die Berechnung von dominanter Wellenlänge, Sättigung und Beleuchtungsstärke übernimmt. Die letzte Softwarekomponente umfasst die graphische Oberfläche für die Hardware. Hier sollen

die Farbwerte der angeschlossenen *Color Controller* angezeigt werden und bei Bedarf Einstellungen an den Sensoren vorgenommen werden können. Außerdem erfolgt in dieser Applikation die Stimulation der LEDs auf den zu testenden Baugruppen. Darüber hinaus soll die graphische Oberfläche dem Bediener die Möglichkeit geben, für die zu testenden Baugruppen ein Skript generieren zu können, welches in der Testumgebung von Hilscher eingesetzt werden kann. Da der entwickelte Quellcode auf *GitHub*⁸ veröffentlicht und frei zugänglich ist, wurde dieser nicht in die Bachelorarbeit aufgenommen.

4.1 Entwicklung einer effizienten API

Die API dient den später zu entwickelnden Programmen als Schnittstelle zu der angeschlossenen Hardware. Über sie sollen die am PC verbundenen *Color Controller* effizient angesteuert werden können. Hier wird die Funktionalität implementiert, die das Erkennen und Öffnen angeschlossener Geräte, das Auslesen der Farbwerte und bei Bedarf die Konfiguration der Sensoren ermöglicht. Da die API die Grundlage für die GUI-Applikation und den automatisierten LED Test darstellt, wurde hier ein besonderer Wert auf eine ausführliche Dokumentation gelegt. Diese wurde mithilfe des Dokumentationstools *Doxygen*[5] erstellt und befindet sich als 60-seitiges PDF-Dokument ebenfalls im oben aufgeführtem *GitHub* Verzeichnis.

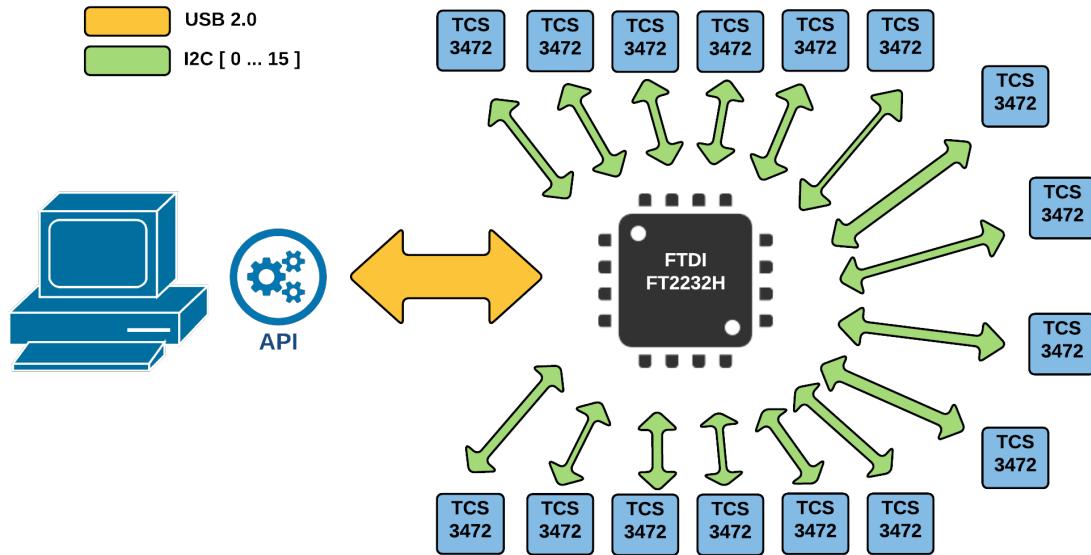


Abbildung 7: Funktionsdiagramm der Kommunikation

Die Kommunikation zwischen PC, FTDI Chip und den 16 Farbsensoren ist einfach gestaltet und wird in Abbildung 7 veranschaulicht. Über den PC werden USB-Pakete an den FTDI Chip gesendet. Diese Pakete beinhalten Befehle und Daten, die auf den ent-

⁸https://github.com/muhkuh-sys/led_analyzer

sprechenden I/O-Pins des FTDI Chips das I²C-Protokoll formen. So wird eine Kommunikation mit den TCS3472 Farbensoren ermöglicht. Sie können nun konfiguriert werden und die gemessenen Farbwerte an den FTDI-Chip leiten. Der Chip sendet die Messwerte zum Zwecke der Weiterverarbeitung über USB an den PC. Für die Realisierung dieses Kommunikationsablaufes wurden vier Module geschrieben, welche die Funktionalität jeweils kapseln. Auf unterster Ebene sorgt ein Modul für das I/O-Handling auf dem FTDI Chip. Darauf aufbauend ist im zweiten Modul die Software-I²C Funktionalität implementiert, welche die Kommunikation zu den Farbsensoren realisiert. Das dritte Modul, eine Funktionsbibliothek für die Ansteuerung des TCS3472, agiert auf Sensorebene und realisiert u.a. die Funktionen für das Auslesen der Farbwerte und die Konfiguration der Sensoren. Das letzte Modul, die API, integriert die zuvor genannten Komponenten und bildet eine Schnittstelle zur Außenwelt. Diese Module werden im Laufe der folgenden Kapitel, angefangen mit der untersten Ebene, detailliert beschrieben.

4.1.1 I/O-Handling auf dem FTDI-Chip

Der eingesetzte FT2232H USB-Wandlerbaustein der Firma FTDI verfügt über zwei getrennt voneinander ansteuerbare Kanäle. Jeder Kanal besitzt 16 I/O-Leitungen, die jeweils in *High Byte* und *Low Byte* geteilt sind. Wird der Chip in den *Multi-Protocol Synchronous Serial Engine (MPSSE)*⁹ Modus gesetzt, können die verfügbaren 32 I/O-Leitungen über spezielle Befehle konfiguriert werden. Auf Pins, denen Output Funktionalität zugewiesen wurde, können Werte geschrieben werden. Die als Input deklarierten Pins können den an ihnen anliegenden Wert zurückliefern. Die relevanten Befehle des *MPSSE* Befehlssatzes[8] wurden in Tabelle 1 zusammengefasst.

	<i>Low Byte</i>	<i>High Byte</i>
Write	0x80 <i>Wert</i> <i>Richtung</i>	0x82 <i>Wert</i> <i>Richtung</i>
Read	0x81	0x83

Tabelle 1: MPSSE Befehlsatz für I/O-Operationen

Auf diesem Befehlssatz aufbauend wurden zwei Funktionen geschrieben, die jeweils eine 32-Bit Maske für die Richtung und einen 32-Bit Wert für die Pins als Parameter erhalten. Eine 1 in der Bitmaske für die Richtung konfiguriert den entsprechenden Pin als *Output*, der Wert bestimmt den anliegenden Pegel an dem Pin, falls dieser als *Output* konfiguriert wurde. Die Funktion, die nur Ausgangspins setzt, ohne die *Input* Pins einzulesen wird im Folgenden *process_pins* bezeichnet. Zusätzlich hierzu

⁹Modus für Konfiguration und Implementierung serieller Protokolle, wie z.B. JTAG, SPI und I²C

beinhaltet die Funktion `process_pins_databack` die Befehle, um die aktuell als *Input* konfigurierten Pins einzulesen. Mit diesen Funktionen lassen sich alle Zustände im I²C-Protokoll umsetzen.

4.1.2 Die Software-I²C Bibliothek

Die Hardware des *Color Controllers* besitzt 16 TCS3472 Farbsensoren, welche ihre Messwerte über eine I²C-Schnittstelle bereitstellen. Die Sensoren besitzen jedoch eine feste I²C-Adresse, die weder durch Hardwareanschaltung noch durch Software verändert werden kann. Der Anschluss mehrerer Sensoren am selben Bus ist somit nicht möglich. Um dieses Problem zu beheben, wurde eine Software-I²C Bibliothek implementiert, die auf den 32 verfügbaren I/O-Leitungen des FTDI Chips 16 voneinander unabhängige I²C-Busse kontrollieren kann. Ziel ist hier die Bereitstellung von Funktionen, die ein oder mehr Bytes an ein Register des Sensors senden bzw. aus dem Register auslesen. Aus Effizienzgründen sollen alle Sensoren gleichzeitig angesteuert werden können. In diesem Zusammenhang erfolgt zuerst eine allgemeine Einführung in das I²C-Protokoll[12] mit anschließender Erläuterung, wie dieses auf dem Chip implementiert wurde.

Die Geräteadressierung Am Bus angeschlossene Geräte werden über ihre I²C-Adresse unterschieden und angesprochen. Das I²C-Protokoll definiert Adressen, die entweder 7 Bit oder seltener 10 Bit Länge besitzen. Da der eingesetzte I²C-Farbsensor erstere besitzt, wird im Folgenden nur auf die 7-Bit Variante eingegangen. Diese Adressierungslänge ermöglicht es, auf demselben Bus insgesamt 127 Geräte anzusteuern, vorausgesetzt sie besitzen alle verschiedene Adressen. Nach dem Senden der 7-Bit Adresse, sendet der Master das achte Bit, welches dem Slave mitteilt, ob eine Schreiboperation (*Bit7 Low*) oder Leseoperation (*Bit7 High*) ausgeführt werden soll.

Das I²C-Protokoll Will der I²C-Master, in unserem Fall der FTDI Chip, Daten an die Sensoren senden, beginnt er den Transfer mit einem der zwei besonderen Zustände, die im I²C-Protokoll definiert sind, der Startsequenz. Die zweite Sequenz wird Stopsequenz genannt und dient der Terminierung eines Datentransfers. Diese Zustände werden insofern als besonders bezeichnet, als dass sie die beiden einzigen Folgen sind, bei denen sich der Wert auf der Datenleitung (SDA) verändern darf, während sich die Taktleitung (SCL) noch im *High* Zustand befindet. Beim Transfer von Daten muss der Wert auf SDA konstant bleiben, solange auf SCL ein *High* Pegel anliegt.

Die Übertragung von Daten erfolgt im I²C-Protokoll in Sequenzen mit jeweils 8 Bit Länge, beginnend mit dem Most Significant Bit (MSB). Jedes Bit auf der SDA Leitung wird hierbei von SCL getaktet, indem SCL zuerst den *High* Pegel, anschließend den *Low*

Pegel annimmt. Nach einer Übertragung von 8 Bits vom Master an den Slave, erwartet der Master mit dem neunten Bit die Quittierung des Datenstroms. Diese Quittierung wird Acknowledge (ACK) genannt und besteht darin, dass der Slave die Datenleitung während des Taktens auf *Low* hält. Entsprechend muss bei einem umgekehrten Transfer der Master quittieren. Abbildung 8 soll die beschriebenen im I²C-Protokoll definierten Sequenzen anhand eines *Read* Transfers näher erläutern.

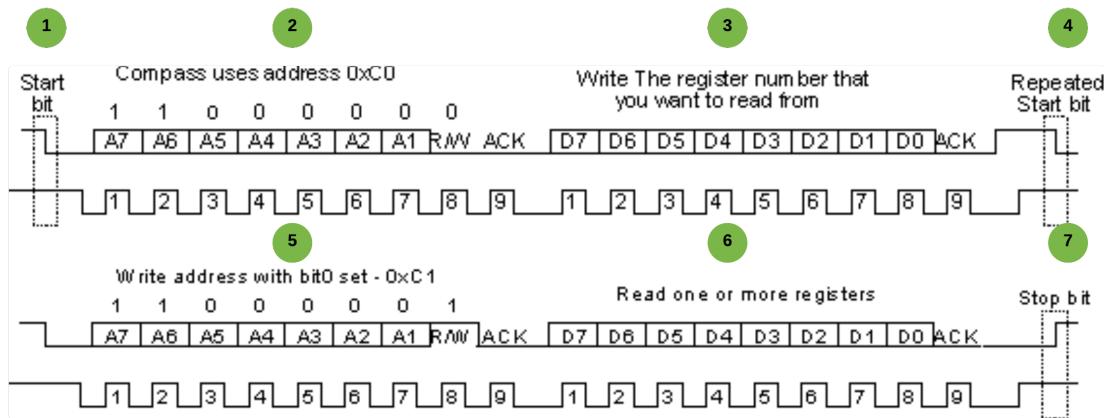


Abbildung 8: Beispiel I²C Read Transfer¹⁰

Hierbei wird von einem I²C-Slave mit der Adresse 0xC0 ein Byte gelesen. Angefangen wird der Transfer mit einer Startsequenz(1). Darauf folgt die I²C-Adresse(2) des Slaves mit dem R/W Bit auf *Low* und das Register(3), aus dem gelesen werden soll. Anschließend liegt auf dem Bus eine weitere Startsequenz an. Diese wird *Repeated Start*(4) genannt und initiiert eine Richtungsumkehrung zwischen Schreib- und Leseoperation. Sind mehrere Master am selben I²C-Bus angeschlossen, kann der *Repeated Start* zusätzlich sicherstellen, dass die begonnene Kommunikation zwischen einem Master und einem Slave nicht durch einen anderen Master unterbrochen werden kann. Auf diese Sequenz folgt noch einmal die Adresse(5), diesmal mit dem R/W Bit auf *High*, welches dem Slave mitteilt, dass der Master Daten anfordert. Diese Daten(6) legt der Slave, beginnend mit dem MSB, auf den Bus. Hat der Master alle Daten ausgelesen beendet er die Transaktion mit einer Stoppsequenz(7).

Umsetzung auf dem FTDI Chip Zugunsten des Timings werden die 32 I/O-Leitungen des Chips so aufgeteilt, dass physikalisch abwechselnd eine Daten- und eine Takteleitung vorliegt. So kann gewährleistet werden, dass Signallaufzeiten bei den beiden Leitungen eines I²C-Busses weitestgehend gleich sind. Die Funktionen auf I/O-Ebene werden verwendet, um auf den Pins des FTDI Chips die im I²C-Protokoll definierten Sequenzen, wie z.B. Start- und Stoppbit zu erzeugen. Die Taktrate wird am FTDI Chip eingestellt

¹⁰Bildquelle Abb.8 http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html

und wird erzeugt, indem die 16 dafür vorgesehenen Leitungen entsprechend getoggled werden. Werden Daten von den Sensoren erwartet, z.B. in Form eines Datenbits oder eines ACK Bits, werden die 16 Datenleitungen kurzzeitig auf *Input* geschaltet und der anliegende Wert eingelesen. Bedingt durch die USB Latenzzeit würde die Übertragung jeder dieser Sequenzen als einzelnes USB Paket eine große Zeit in Anspruch nehmen. Hier bietet es sich an, zuerst alle Daten und Befehle, die für einen I²C-Transfer benötigt werden, in einem Paket zu sammeln und dieses anschließend als ein großes USB Paket an den Chip zu übertragen. Abbildung 9 soll den Vorgang näher erläutern.

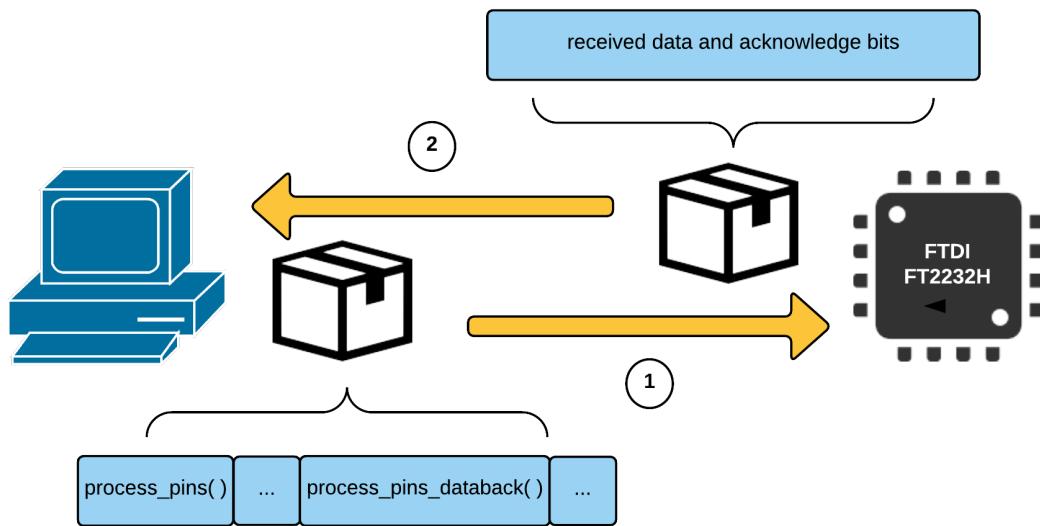


Abbildung 9: I²C Umsetzung auf dem FTDI Chip

Jeder I²C-Befehl in der implementierten Bibliothek führt intern die in der I/O-Bibliothek beschriebenen Funktionen *process_pins()* und *process_pins_databack()* aus. So werden die entsprechenden Zustände auf den Leitungen erzeugt und das I²C-Protokoll nachgebildet. Enthält das Paket alle Befehle und Daten, um z.B. eine I²C Lese- oder Schreiboperation auf dem Bus auszuführen, wird dieses Paket über die USB Leitung an den FTDI Chip übertragen. Dieser geht den Inhalt des Paketes durch und setzt anhand der enthaltenen Befehle und Daten die I/O-Pins, bzw. liest diese zu bestimmten Zeitpunkten ein. Die Daten, die vom Chip eingelesen wurden, können anschließend vom PC angefragt und den 16 verschiedenen I²C-Bussen zugeordnet werden. Abbildung 10 zeigt beispielhaft die Aufnahme eines Lesebefehls, bei dem ein Register eines angeschlossenen Farbsensors ausgelesen wurde. Dabei wurden Datenleitung und Taktleitung des Busses mithilfe eines Logic Analyzers der Firma Saleae abgegriffen und über die mitgelieferte Software *Logic*¹¹ dekodiert.

¹¹<http://support.saleae.com/hc/en-us/articles/201589175>

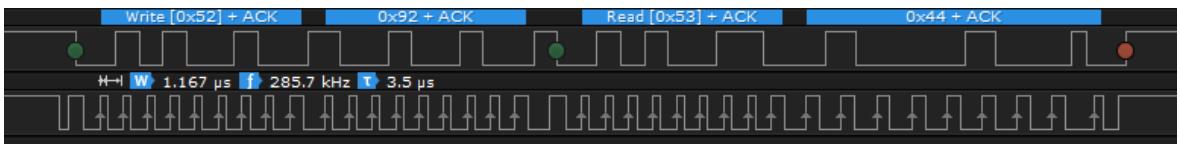


Abbildung 10: Aufnahme eines I²C Lesebefehls mit dem Logic Analyzer

Fazit Über die implementierte I²C-Bibliothek lassen sich 16 voneinander unabhängige Busse über den FTDI Chip ansteuern. Sie beinhaltet die Funktionalität, die benötigt wird, um angeschlossene Slaves parallel zu beschreiben oder auszulesen. Die Tatsache, dass manche I²C-Bausteine nach jedem Lesebefehl intern die Registeradresse erhöhen, und man auf diese Art in einem I²C-Transfer mehrere Register auf einmal auslesen kann, wurde in den Routinen mit berücksichtigt. Auf diese Weise können alle 4 Farbregister des Sensors (8 Byte) über zwei USB Pakete (Befehl hin, Daten zurück) ausgelesen werden. Da die Geschwindigkeit, mit der die Sensoren ausgelesen werden können, größtenteils durch die USB Latenzzeit vorgegeben ist, führt die Reduzierung der Paketanzahl zu einer Erhöhung der möglichen Abtastrate der Farbsensoren. Eine hohe Abtastrate ist z.B. bei der Messung der Frequenz von blinkenden LEDs dringend erforderlich. Die I²C-Bibliothek ist jedoch ungeeignet für I²C-Slaves, welche *Clock-Stretching*¹² Funktionalität besitzen. Da die verwendeten TCS3472 Farbsensoren diese Eigenschaft jedoch nicht besitzen, wurde *Clock-Stretching* bei der Implementierung der I²C-Bibliothek nicht mit berücksichtigt.

4.1.3 Die TCS3472 Bibliothek

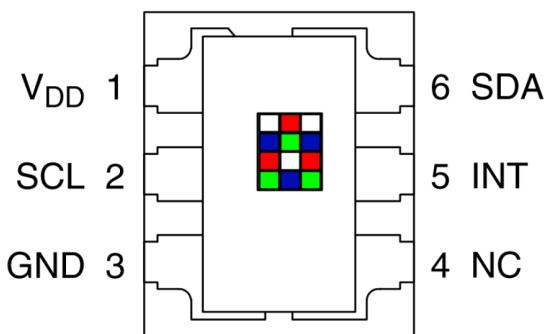


Abbildung 11: TCS3472 Pinbelegung¹³

Der TCS3472 *Color Light-To-Digital Converter with IR Filter* von ams ist ein Farbsensor, der 4 Messkanäle für Red, Green, Blue und Clear (RGBC) besitzt. Es können somit jeweils der rote, grüne und blaue Farbanteil des einfallenden Lichts gemessen werden. Der vierte Messkanal gibt Auskunft über die Helligkeit. Ein Infrarot Blockfilter ist bereits auf dem Chip integriert und ermöglicht durch Minimierung des Infrarotanteils die Durchführung von

präzisen Messungen. Durch diesen Filter ist der Sensor sehr gut geeignet für Ambient Light Sensing (ALS), das u.a. für die automatische Hintergrundbeleuchtung von Displays eingesetzt wird. Um den hohen Dynamikumfang¹⁴ von 3800000:1 zu ermögli-

¹²I²C-Slave verzögert die Übertragung, indem er die Taktleitung in der Low-Phase gegen Masse hält

¹³Bildquelle Abb.11: TCS3472_Datasheet_EN_v2.pdf - S.4

¹⁴Quotient aus größtem und kleinstem unterscheidbaren Helligkeitswert

chen, können Einstellungen bezüglich Verstärkung und Belichtungszeit vorgenommen werden. Der niedrige Stromverbrauch von $235 \mu\text{A}$ kann durch das Aktivieren von *Wait-States*¹⁵ noch weiter herabgesetzt werden. Eine I²C-Schnittstelle bietet durch die *Fast Mode* Kompatibilität die Möglichkeit mit Geschwindigkeiten von bis zu 400 kBit/s zu kommunizieren. Hier gibt es verschiedene Varianten bezüglich der I²C-Betriebsspannung, z.B. über 1.8V (TCS34727) oder V_{DD} (TCS34725). Typische Anwendungsmöglichkeiten sind u.a. Messung der Farbtemperatur, Messung von Umgebungslicht für automatische Hintergrundbeleuchtung von Displays und die Verifikation und das Sortieren von Produkten bezüglich ihrer Farbe. Den hier implementierten Funktionen liegt das Datenblatt des TCS3472 Farbsensors[2] zugrunde. Es erfolgt eine Zusammenfassung der umgesetzten TCS3472 Funktionalität.

Farbregister Der Sensor besitzt vier 16-Bit Farbregister für rot, grün, blau und die Helligkeit. Diese sind intern als acht 1-Byte Register angeordnet und befinden sich in einem zusammenhängenden Speicherbereich. Der Bereich beginnt mit dem Clear Data Low Byte (CDATAL) und endet mit dem Blue Data High Byte (BDATAH) Register. Wird beim Lesen eines Registers des Sensors ein spezielles Bit (*auto-increment protocol transaction bit*) gesetzt, inkrementiert dieser automatisch nach jedem gelesenen Byte die Registeradresse, aus der gelesen werden soll. So können mithilfe eines I²C-Lesebefehls beginnend an CDATAL und gesetztem Autoinkrementierbit alle 8 Farobytes des Sensors sukzessiv ausgelesen werden. Abbildung 12 zeigt eine mit dem Logic Analyzer aufgenommene Farbmessung. Hier wird zusätzlich zu den 8 Bytes für die Farben ein neuntes Byte, das *Status* Register ausgelesen. Auf dieses wird später näher eingegangen.



Abbildung 12: Sukzessives Auslesen aller Farbregister

Dynamikumfang Die Belichtungszeit und Verstärkung des Sensors sind maßgebend für den hohen Dynamikumfang von 3800000:1 verantwortlich. Der Dynamikumfang bezeichnet in der Optik den Quotienten aus größtem und kleinstem von Rauschen bzw. Körnung unterscheidbaren Helligkeitswert. Durch diese Eigenschaft des TCS3472 Farbsensors können auch bei stark variierenden Lichtquellen akkurate Messergebnisse bezüglich Farbe und Helligkeit erzielt werden. Die Verstärkung des Sensors kann über das *Control* Register als 1-fach, 4-fach, 16-fach oder 60-fach eingestellt werden. Die Konfiguration der Belichtungszeit erfolgt im *Atime* Register und kann in 2.4ms Schritten von 2.4ms bis 700ms eingestellt werden. Je nach eingestellter Belichtungszeit variiert der in

¹⁵Situation, in der sich ein Programm oder Prozessor schlafen legt, bis eine Zeit verstrichen ist

den Farbregistern maximal mögliche Wert. Dieser Wert berechnet sich nach der Formel $RGBCCCount_{max} = \frac{Atime_{ms} * 1024}{2.4}$ und kann, bedingt durch die 16-Bit Register, maximal 65535 betragen. $Atime_{ms}$ stellt hierbei die eingestellte Belichtungszeit in Millisekunden dar. Während helle Lichtquellen eine relativ geringe Belichtungszeit und Verstärkung erfordern, wird bei schwach leuchtenden Lichtquellen eine lange Belichtungszeit und eine hohe Verstärkung vorgesehen. Der Dynamikumfang ist für die geplante Applikation dringend erforderlich, da das Produktspektrum von LEDs sehr groß ist und sowohl sehr helle als auch schwach leuchtende LEDs zu testen sind.

Validierung der Messergebnisse Der Sensor bietet verschiedene Möglichkeiten, die Ergebnisse einer Farbmessung hinsichtlich ihrer Validität zu überprüfen. Ein *ID* Register beinhaltet eine für den TCS3472 Farbsensor spezifische Identifikationsnummer. Diese kann z.B. im Verlauf der Initialisierung des Sensors ausgelesen werden, um die Kommunikation zu dem Farbsensor zu überprüfen. Über Bit0 (*Avalid-Bit*) im *Status* Register kann herausgefunden werden, ob die Farbumwandlungen der Analog-to-Digital-Converter (ADCs) bereits abgeschlossen sind, oder ob der Sensor noch mehr Zeit benötigt. Weiterhin kann über die Kombination aus aktuell anliegenden Werten in den Farbregistern und eingestellter Belichtungszeit überprüft werden, ob der Sensor in die Sättigung gelangt ist. Der maximal mögliche Wert in den Farbregistern bei einer eingestellten Belichtungszeit von z.B. $Atime_{ms} = 48\text{ms}$ berechnet sich nach oben genannter Formel zu $RGBCCCount_{max} = 20480$. Erreicht mindestens eins der vier Farbregister diesen Wert, ist die Sättigung erreicht worden. Die zu messende Lichtquelle ist für die eingestellte Beleuchtungszeit und / oder Verstärkung zu hell und die in Kapitel 3 vorgestellten Algorithmen dürfen für die Berechnung der dominanten Wellenlänge, Sättigung und Beleuchtungsstärke nicht mehr herangezogen werden. In diesem Fall muss die Messung wiederholt werden, nachdem die Verstärkung und / oder die Belichtungszeit verringert wurde.

4.1.4 Integration auf Geräteebene

Das letzte in diesem Bezug zu entwickelnde Modul stellt die API dar. Sowohl die graphische Oberfläche als auch das zu entwickelnde Testprogramm für das automatisierte Testen der LEDs werden über diese Schnittstelle die Funktionen für die Ansteuerung des Gerätes beziehen. Sie kapselt einerseits die Funktionalität der entwickelten TCS3472 Bibliothek für das Auslesen und die Konfiguration der Sensoren und stellt andererseits Funktionen für die Geräteverbindung bereit. Die Programmierung und Initialisierung der *Color Controller* erfolgt über die "*libFTDI*". Diese stellt eine *Open Source Library* dar, welche eine Großzahl der von FTDI bereitgestellten Chips, so auch

den hier verwendeten FT2232H, unterstützt. Im Folgenden wird auf die Funktionen der Geräteverbindung näher eingegangen.

Gerätesuche und -verbindung Die Hardware des *Color Controllers* ist mit einem *EEPROM* ausgestattet. Dieser Speicherbaustein wurde im Verlauf der Inbetriebnahme der Hardware u.a. mit firmenspezifischen Deskriptoren, einer eindeutigen Seriennummer und einer VID/PID Kombination beschrieben. Die Kombination aus Vendor-ID (VID) und Product-ID (PID) dienen dem PC zur eindeutigen Identifikation eines USB-Gerätetyps. Dabei stellt die VID eine Firmenkennung dar, die PID steht für eine bestimmte USB-Gerätekasse.

Die Funktion `scan_devices()` soll nach allen angeschlossenen *Color Controller* Testgeräten (VID=0x1939, PID=0x0024) suchen. Sind Geräte dieser Art angeschlossen, wird die im *EEPROM* gespeicherte Seriennummer ausgelesen und intern in einer Liste gespeichert. Die Funktion liefert die Liste der Seriennummern und die Anzahl der gefundenen *Color Controller* zurück. Die Funktion `connect_to_devices()` ist für die Geräteverbindung und -initialisierung verantwortlich. Sie erhält die Anzahl der gefundenen Geräte und die Liste der Seriennummern als Eingangsparameter. Da der eingesetzte FT2232H zwei verschiedene Kanäle beinhaltet (Kanal A und Kanal B), werden pro gefundenem Gerät zwei Speicherbereiche initialisiert. Die Initialisierung schließt die Konfiguration des Kanals und das Setzen des Operationsmodus ein. Hier wird der beschriebene *MPSSE* Modus verwendet, der das Setzen von I/O-Pins über USB-Kommandos (s. Kap. 4.1.1) ermöglicht. Jeder Kanal wird nun anhand der VID/PID Kombination und der Seriennummer an der entsprechenden Stelle der Eingangsliste geöffnet. Die Funktion selbst liefert eine Liste mit Zeigern auf alle initialisierten Kanäle zurück. Diese kann nun von allen weiteren Funktionen des Frameworks für die Ansteuerung der Geräte verwendet werden.

Die Reihenfolge der Geräteverbindung Die Reihenfolge bei der Geräteverbindung sollte laut Anforderung deterministisch und bei Bedarf vom Benutzer frei konfigurierbar erfolgen. Falls mehrere *Color Controller* angeschlossen sind, wird durch die Deterministik gewährleistet, dass die Zuordnung der Messkanäle bei jedem Testdurchlauf gleich bleibt. Hierfür ordnet die Gerätesuche intern die Seriennummern in aufsteigender Reihenfolge an. Werden z.B. drei *Color Controller* erkannt, befindet sich das Gerät mit Seriennummer "CoCo20000" an erster, "CoCo20001" an zweiter und "CoCo20002" an letzter Stelle der zurückgegebenen Liste (vgl. Abb. 13a). Gibt der Benutzer hier keine Vorgabe, wird diese Reihenfolge auch bei der anschließenden Geräteverbindung eingehalten. Würde in einem bereits aufgebautem Testaufbau z.B. "CoCo20001" ausfallen, könnte dieses durch "CoCo20007" ersetzt werden. Bedingt durch die abweichende Seriennummer würde sich die Reihenfolge der Geräteverbindung und somit die Zuordnung

Index	Seriennummer
1	CoCo20000
2	CoCo20001
3	CoCo20002

(a)

Index	Seriennummer
1	CoCo20000
2	CoCo20002
3	CoCo20007

(b)

Abbildung 13: Reihenfolge der Geräteverbindung

der Messkanäle ändern (vgl. Abb. 13b). Eine Neuanpassung des Testaufbaus und der Testskripte wäre mit zu viel Aufwand verbunden. Um diesem möglichen Problem entgegen zu wirken, kann der Benutzer die Reihenfolge bei der Geräteverbindung auch selbst vorgeben. Hierfür stellt die API Funktionen bereit, um die Anordnung der Elemente in der Liste der Seriennummern anzupassen. Bei anschließender Geräteverbindung erfolgt die Indizierung nach der eingestellten Reihenfolge des Benutzers.

4.1.5 Fehlererkennung

Modulebene	Fehlerart	Beispiel
TCS3472	Sensorspezifisch	Farbkanäle eines Sensors sind gesättigt
Software-I ² C	I ² C-Fehler	I ² C-Slave gibt kein ACK zurück
Geräteebene	FTDI-Fehler	FTDI-Kanal lässt sich nicht öffnen
I/O-Handling	USB-Fehler	USB-Gerät kann nicht erreicht werden

Tabelle 2: Mögliche Fehler der vier Module

Auf jeder Ebene der vorgestellten Module können verschiedene Fehlerursachen zu einem Fehlverhalten des Gerätes führen. Die Ebenen mit den möglichen vorkommenden Fehlern sind in Tabelle 2 dargestellt. So können z.B. auf der untersten Ebene USB-Fehler auftreten, die zur Folge haben, dass keine USB-Pakete mehr an den FTDI Chip gesendet oder von diesem empfangen werden können. Die nächst höhere Ebene, die Geräteebene, verwendet die Funktionen der "libFTDI" Bibliothek. Hier können z.B. Fehler bei der Konfiguration und Initialisierung der FTDI-Chips auftreten. Eine Ebene darüber, in der I²C-Bibliothek, drückt sich ein Fehler z.B. darin aus, dass ein angeschlossener I²C-Slave am Bus das Empfangen eines Datenpaketes nicht richtig quittiert. In der TCS3472 Bibliothek können zusätzlich sensorspezifische Fehler auftreten. So können mehrere Sensoren die Sättigung ihrer Farbkanäle erreicht haben, da die eingestellte Kombination aus Verstärkung und Belichtungszeit nicht geeignet für die zu messende Lichtquelle war. Hier muss neben der Art des auftretenden Fehlers (Sättigung der Farbkanäle) zusätzlich die Information mitgeliefert werden, bei welchen Sensoren der Fehler auftrat. Um gezielt auf einen Fehler reagieren zu können, muss dem Benutzer die Möglichkeit gegeben werden, diese unterscheiden zu können. Dafür wurden in den vier Modulen Fehlercodes eingeführt. Diese werden im Fehlerfall an die nächst höhere

Ebene weitergegeben und gelangen auf diese Weise über die API an überlagerten Programme, wie z.B. die GUI-Applikation oder den automatisierten LED-Test. Um auf Sensorebene zusätzlich zur Fehlerart die fehlerhaften Sensoren kenntlich zu machen, werden die niederwertigsten 16-Bit des Fehlercodes für die Kennzeichnung der fehlerhaften Sensoren verwendet. Sind *Bit0*, *Bit11* und *Bit13* im Fehlercode gesetzt, deutet das auf einen Fehler an Sensor 1, 12 und 14.

4.2 Verarbeitung der Messwerte

Über die API kann sich der PC nun mit den angeschlossenen *Color Controllers* per USB verbinden, diese konfigurieren und die rohen RGBC-Werte auslesen. Dabei wird die Verarbeitung der Messdaten auf den PC ausgelagert. Da die zu testenden Baugruppen in der Testabteilung von Hilscher meist über USB an den PC angeschlossen werden, bietet es sich an, die hohe Verarbeitungsgeschwindigkeit der bereits vorhandenen PCs auch für die Verarbeitung der Messdaten des *Color Controllers* zu verwenden. Abbildung 14 zeigt die einzelnen für die Verarbeitung zuständigen Module, die im Verlauf dieses Kapitels vorgestellt werden.

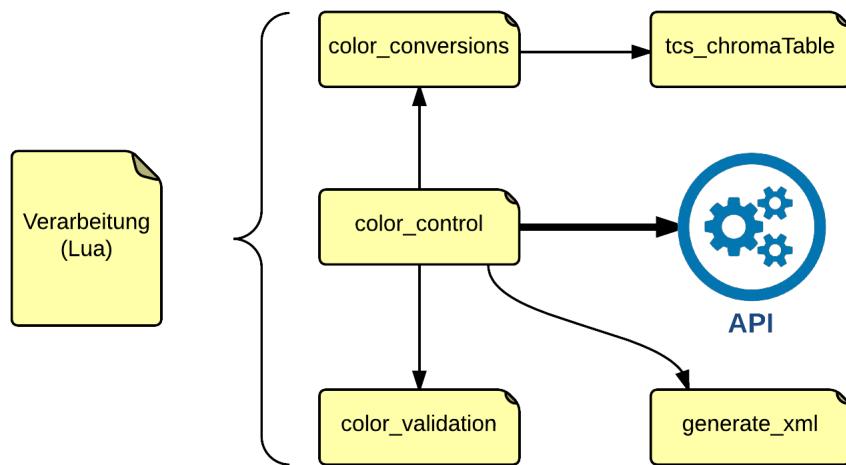


Abbildung 14: Abhängigkeitsdiagramm der Lua Module

4.2.1 Der Einsatz von Lua

Die Programme für die Verarbeitung der Messdaten sind in der Skriptsprache Lua¹⁶ geschrieben. Die großen Vorteile dieser Skriptsprache liegen bei der geringen Größe des Interpreters (120 kByte), der leichten Erweiterbarkeit und der hohen Ausführungs geschwindigkeit im Vergleich zu anderen Skriptsprachen. So hat sich der Einsatz dieser

¹⁶<http://www.lua.org/>, Lua portugisisch für Mond

Skriptsprache bereits bei bekannten Anwendungen, wie "*Adobe Photoshop Lightroom*" oder "*World of Warcraft*" etabliert. Über die C-Bibliothek kann der Lua-Interpreter relativ einfach in bestehende Programme integriert werden. Reicht der Basisumfang des Interpreters für die gewünschte Anwendung nicht aus, kann er durch externe Bibliotheken einfach erweitert werden[10]. Die Testskripte, die in der Testabteilung von Hilscher zum Einsatz kommen, sind größtenteils in Lua geschrieben. Die Testskripte und Testsoftware des *Color Controllers* soll später auch in diese auf Lua basierende Umgebung integriert werden können. Durch den Einsatz dieser Skriptsprache in der Software des *Color Controllers* soll somit die Integration in die Testumgebung von Hilscher stark vereinfacht werden.

4.2.2 Einbindung der API

Die Einbindung der in Kapitel 4.1 vorgestellten API findet im Modul *color_control.lua* statt. Um diese in C geschriebene API von Lua aus verwenden zu können, wird das Software Tool *Swig*¹⁷ verwendet. Dieses Tool ermöglicht die Anbindung von in C und C++ geschriebener Softwarekomponenten zu einer Vielzahl höherer Programmiersprachen, wie z.B. Javascript, Python, Ruby und Lua. Das Tool wird dazu verwendet, um die in C geschriebene API der Skriptsprache Lua in Form einer Dynamic Link Library (DLL) zugänglich zu machen. Diese DLL kann anschließend problemlos in Lua eingebunden und verwendet werden. Das Modul *color_control.lua* kapselt somit die Ansteuerungsroutinen der API und macht diese für alle weiteren Lua Komponenten zugänglich.

4.2.3 Die Lua Komponenten

Das Schnittstellenmodul *color_control.lua* bindet weitere Lua Module ein. Diese sind in Abb. 14 dargestellt und übernehmen jeweils Funktionen für die Verarbeitung der Messwerte.

Farbumwandlungen Die Umwandlung der rohen RGBC Messdaten wurde im Modul "color_conversions.lua" implementiert. Hier sind die Ideen und Algorithmen aus Kapitel 3 in Lua Code umgesetzt worden. Eine mächtige Eigenschaft von Lua stellen die sogenannten *Tables* dar. *Tables* sind assoziative Arrays, die sich nicht nur mit Zahlen, sondern auch mit diversen Datentypen von Lua ansprechen lassen. Diese werden in "color_conversions.lua" zum Speichern von Werten in verschiedenen Farträumen verwendet. Das hier eingebundene Modul *tcs_chromaTable.lua* speichert die Messdaten der interpolierten spektralen Empfindlichkeit des TCS3472 Farbsensors und das aufge-

¹⁷<http://www.swig.org/>

spannte Netz der Richtungsvektoren in einer solchen Tabelle. Neben den Farbräumen, die als Zwischenziele für die Berechnung der dominanten Wellenlänge, Sättigung und Beleuchtungsstärke benötigt werden ($RGB \rightarrow XYZ \rightarrow Yxy$), beinhaltet dieses Modul auch die Berechnungsfunktionen für weitere Farbräume, wie z.B. dem HSV¹⁸ oder dem L*a*b-Farbraum¹⁹. Diese waren zwar nicht Teil der Anforderungen, können so jedoch, falls in Zukunft erwünscht, leicht einbezogen werden.

Validierung der LEDs Das Modul *color_validation.lua* wird zum Testen der LEDs eingesetzt. Dieses Modul arbeitet auf Basis einer Sollwerttabelle, welche für jede zu testende LED die dominante Wellenlänge, Sättigung und Beleuchtungsstärke und zusätzlich Toleranzen für jeden dieser Parameter beinhaltet. Bei einem Test werden die aktuellen Messwerte (Istwerte) mit den für die Testbaugruppe gewünschten Sollwerten verglichen und ein detailliertes Testresultat zurückgegeben. In Kombination mit dem Modul "*generate_xml.lua*" kann das Testresultat in Extensible Markup Language (XML) Format ausgegeben, bzw. gespeichert werden. Die Wahl eines solchen Formats lässt das Testergebnis einerseits übersichtlich darstellen, andererseits auch relativ einfach mithilfe sogenannter XML-Parser weiterverarbeiten. Kapitel 7.2 im Anhang beschreibt einen kompletten Versuchsaufbau, bei dem eine Reihe von LEDs auf einem Experimentierboard mithilfe des *color_validation.lua* Moduls auf ihre Funktion getestet werden. Die zugehörige Sollwerttabelle für diesen Versuchsaufbau ist unter Listing 1 zu finden. Listings 2 und 3 zeigen beispielhaft die Ausschnitte des XML Testresultats bei einem erfolgreichen und bei einem fehlgeschlagenen LED Test.

4.3 Die graphische Oberfläche

Die letzte größere Softwarekomponente des Projektes stellt die graphische Oberfläche dar. Über diese soll sich der Benutzer mit allen angeschlossenen Testgeräten verbinden und die Farbwerte auslesen können. Um die Zuordnung zwischen gemessener dominanter Wellenlänge und der vom Menschen wahrgenommenen Farbe zu vereinfachen, wäre hier eine Visualisierung der Messwerte von Vorteil. Die Sensoren des Testgeräts sollen bei Bedarf bezüglich der Belichtungszeit und Verstärkung einfach konfiguriert werden können. Eine Hauptkomponente der graphischen Oberfläche bildet die automatisierte Testerstellung. Dem Benutzer soll hier ein Werkzeug gegeben werden, um auch komplexe LED Tests für die zu testenden Baugruppen generieren zu können. Das generierte Testskript soll so die bestehenden Testroutinen einer Baugruppe um den automatisierten LED Test erweitern. Im Verlauf dieses Kapitels werden die verwendete graphische

¹⁸<http://www.wisotop.de/hsv-und-hsl-farbmodell.shtml>

¹⁹<http://www.wisotop.de/lab-farbmodell.shtml>

Bibliothek, Ansätze der GUI Programmierung und die Funktionsweise der Applikation grundlegend vorgestellt.

4.3.1 Die graphische Bibliothek - *wxWidgets*

*wxWidgets - Cross-Platform GUI Library*²⁰ ist eine C++ Bibliothek, welche dem Benutzer ermöglicht, ausgereifte Applikationen für Windows, Mac OS X, Linux und viele weitere Plattformen zu entwickeln. Da hierbei die Codebasis von *wxWidgets* für jede Plattform dieselbe ist, kann z.B. eine für Windows geschriebene Applikation mit nur wenigen Anpassungen auch für das Linux Betriebssystem kompiliert werden. Die Eigenschaft, dass auf jeder Plattform die nativen GUI Komponenten des Systems verwendet werden, gibt den mit *wxWidgets* entwickelten Programmen das von der entsprechenden Plattform gewohnte Aussehen. Darüber hinaus bietet sie einfache Anbindungsmöglichkeiten für Skriptsprachen, wie Python, Ruby, Perl und auch Lua. Letztere nennt sich *wxLua*²¹ und wird z.B. auch bei der graphischen Oberfläche der Testumgebung von Hilscher eingesetzt. Für den Einsatz dieser Bibliothek spricht nicht zuletzt die bereits vorhandene Kompetenz bei Hilscher. Legt man der Auswahl des passenden GUI Toolkits einige Vergleichskriterien, wie z.B. die kommerziell freie Nutzbarkeit, die Dokumentation des Codes oder den Funktionsumfang zugrunde, behauptet sich *wxWidgets* als eine der ausgereiftesten GUI Lösungen[19].

4.3.2 Die Funktionsweise der Applikation

Die graphische Oberfläche muss einerseits die API für die Geräteverbindung und das Auslesen der Messwerte, andererseits die Lua Skripte für die Messwertverarbeitung einbeziehen können. Es gibt mehrere Möglichkeiten, diese in die GUI Applikation aufzunehmen. Im Folgenden werden die Optionen verglichen und es wird eine entsprechende Auswahl getroffen.

Die erste Möglichkeit ist in Abbildung 15a dargestellt. Dabei holt sich die in C++ geschriebene GUI Applikation die rohen Messwerte über die in C implementierte API. Indem die Applikation einen Lua Interpreter einbindet, kann sie zusätzlich mit dem Lua Skript für die Verarbeitung der Messwerte kommunizieren. Vereinfacht ausgedrückt, übergibt die Applikation an das Skript rohe Messwerte und holt sich die Werte aus den Zielfarträumen zurück. Der Vorteil liegt hier offensichtlich bei der relativ problemlosen Anbindung von C Code (API) in C++ Code (GUI). Der Nachteil liegt darin, dass sowohl die GUI Applikation als auch das generierte Testfile Daten zwischen zwei externen Modulen austauschen müssen.

²⁰<https://www.wxwidgets.org/>

²¹<http://wxlua.sourceforge.net/>

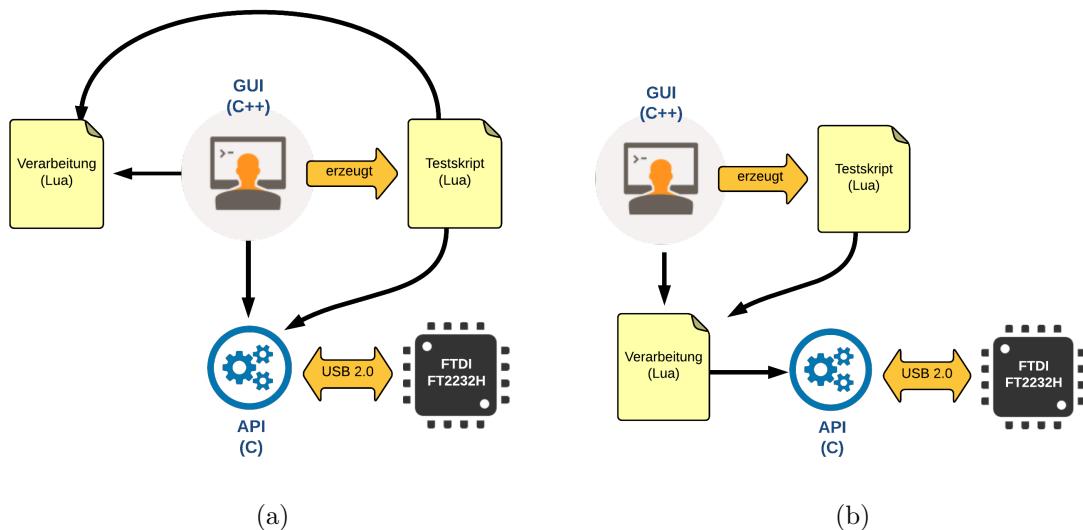


Abbildung 15: Vgl. der Möglichkeiten für die Einbindung der API und der Lua Skripte

Um den Austausch von Daten zwischen GUI und *Color Controller* auf eine einheitliche Schnittstelle zu beschränken, wurde die in Abbildung 15b dargestellte Variante implementiert. Dabei kommuniziert die Applikation lediglich mit dem Lua Skript, welches wiederum intern die Funktionen der API kapselt. Sowohl Applikation als auch das später in der Testumgebung eingesetzte Testskript kommunizieren folglich über eine einheitliche Schnittstelle. Dieser Weg ist zwar mit Mehraufwand verbunden, jedoch werden hier potentielle Fehler vermieden, die durch eine Kommunikation zu zwei verschiedenen Schnittstellen hervorgerufen werden können.

Kommunikation zwischen Lua und C/C++ Beide zuvor vorgestellten Varianten arbeiten auf GUI Seite mit C++ Code und kommunizieren zum Zwecke der Messwertverarbeitung (Abb. 15a) oder der Messwertverarbeitung + Geräteverbindung (Abb. 15b) mit Lua Code. Dafür gibt es in Lua eine sogenannte "C API"²², die im Folgenden "LuaC API" genannt wird, um eine Verwechslung mit der bereits vorgestellten API des *Color Controllers* zu vermeiden. Die "LuaC API" stellt eine Ansammlung von Funktionen bereit, die eine Interaktion zwischen C/C++ Code und Lua Code ermöglichen. In unserem Fall, übernimmt der C++ Code die Kontrolle über die Interaktion und wird deshalb auch *application code* genannt, während der verwendete Lua Code auch als *library code* bezeichnet wird.

Abbildung 16 zeigt die Komponente, die den Datenaustausch zwischen diesen zwei Umgebungen ermöglicht. Sie wird als virtueller *Stack* bezeichnet und ist Teil des Lua Interpreters. Die Verbindung von C/C++ und Lua erfordert gleichzeitig die Vereinigung zweier widersprüchlicher Ansätze dieser Programmiersprachen. Während C und C++ explizite Speicherdeallokation vom Programmierer fordern, übernimmt die Lua Umge-

²²<http://www.lua.org/pil/24.html>

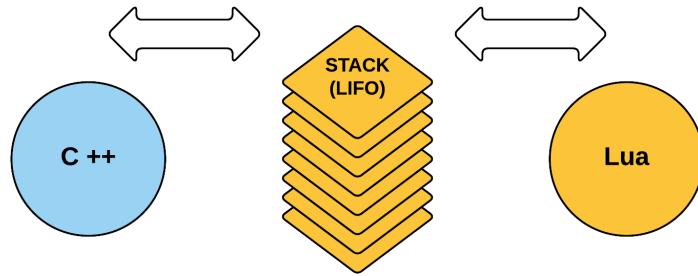


Abbildung 16: Kommunikationskomponente zwischen C/C++ und Lua Code

bung die Speicherbereinigung (*garbage collection*) selbst. Der zweite große Unterschied liegt bei der verwendeten Typisierungsart. Die Typisierung von Variablen erfolgt in C und C++ statisch, d.h., dass der Datentyp von Variablen bereits während der Kompilierung festgelegt wird. Variablen in Lua unterliegen der dynamischen Typisierung. Hier werden sie erst zur Programmalaufzeit festgelegt. Der *Stack* hilft dem Benutzer, die unterschiedlichen Ansätze dieser beiden Programmiersprachen zu vereinen. Alle Funktionen der "LuaC API" arbeiten mit Werten, die auf dem Stack gespeichert sind. Bei der Verwendung der "LuaC API" unterliegt die Speicherfreigabe auf dem *Stack* allein der Verantwortung des Programmierers. Außerhalb des *Stacks* kann sich Lua weiterhin um die *garbage collection* kümmern. Auf C/C++ Seite wird das Typisierungsproblem gelöst, indem bei jeder Anfrage einer Variablen vom *Stack*, explizit der Datentyp angegeben wird, der bei der Variablen erwartet wird.

4.3.3 Die Regeln des GUI Desgins

Es wurden einige Regeln und sogar eine umfangreiche ISO-Norm (EN ISO 9241) für das Erstellen "guter" Benutzeroberflächen definiert. Bei der Entwicklung der graphischen Oberfläche der *Color Controller* Software wurde versucht, diese Regeln weitestgehend einzuhalten. Zusätzlich gab es während der Entwicklung der Applikation Feedback und Verbesserungsvorschläge aus der *User Interface* Abteilung von Hilscher. Allgemein sollen die Richtlinien dazu beitragen, dass der Benutzer, in unserem Fall der Mitarbeiter der Testabteilung, die Software, mit welcher er arbeitet, möglichst effizient bedienen kann. Im Folgenden sind die "acht goldenen Regeln des UI-Designs" aus dem Buch "*Desigining the User Interface*" von Ben Schneidermann[15] aufgelistet, nach denen auch die graphische Oberfläche des *Color Controllers* entwickelt wurde. Obwohl die erste Auflage des Buchs im Jahre 1987 erschienen ist, hat der Inhalt bis heute Bestand. Abbildung 17 stellt dar, wie diese Regeln in Bezug auf die Applikation umgesetzt wurden.

1. Einhaltung der Konsistenz

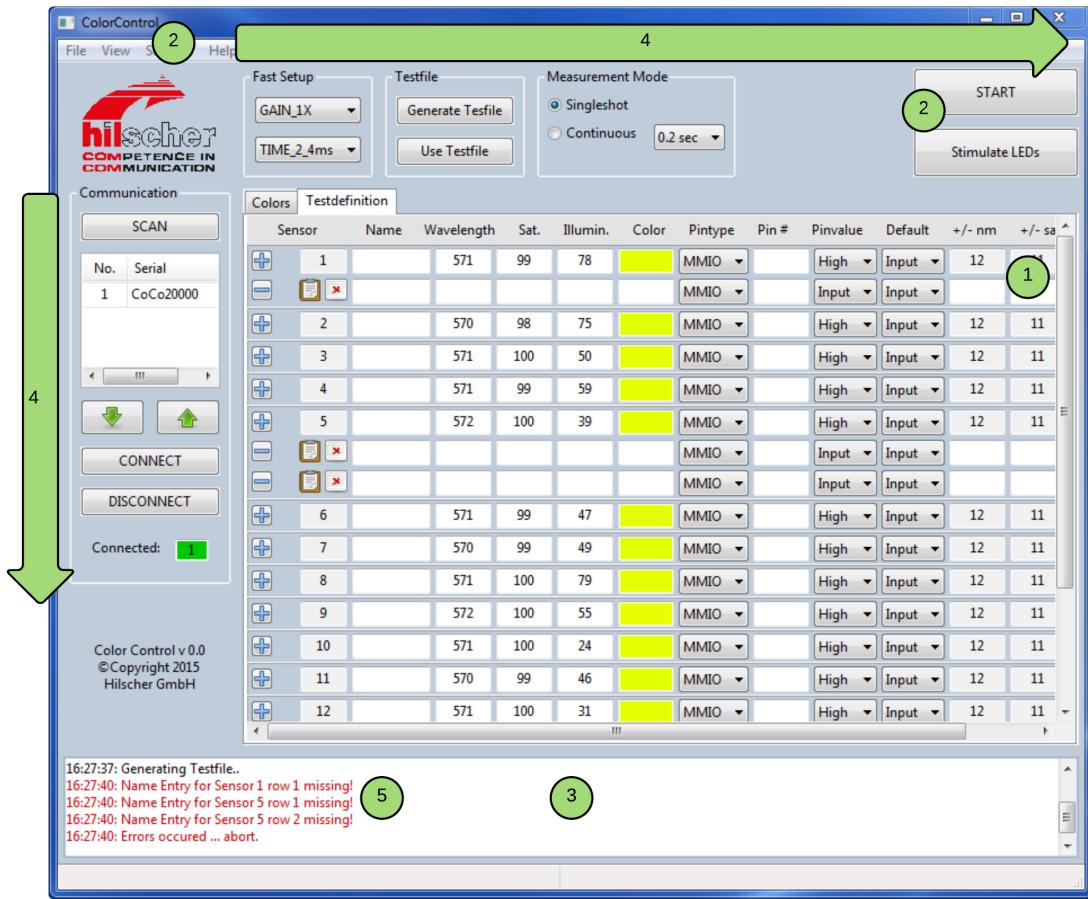


Abbildung 17: Anwendung der Richtlinien des UI Designs

Innerhalb der Anwendung sollen ähnliche Aktionen, Inhalte und Meldungen immer gleich gestaltet und gleich zu bedienen sein. So muss der Benutzer nicht fortlaufend neue Bedienkonzepte erlernen, sondern, sobald einmal eingeprägt, das Bedienkonzept für ähnliche Aufgaben weiterverwenden können. So sind z.B. die Log-Meldungen oder die Anordnung der Spalten unter dem Reiter Testdefinition oder Colors gleich gestaltet.

2. Bereitstellung von Shortcuts für erfahrene Benutzer

Benutzern, die häufiger mit der Software interagieren, sollte die Möglichkeit gegeben werden, den Interaktionsprozess unter Zuhilfenahme von Tastenkombinationen (Shortcuts) oder Makros zu verkürzen. Das ermöglicht, häufig benutzte Aktionen schneller auszuführen und erspart dem erfahrenen Bediener wertvolle Zeit. Dafür wurde jede Aktion, die der Benutzer auf der graphischen Oberfläche initiieren kann, mit einem Shortcut versehen. Vertraute Kombinationen, z.B. für das Speichern (**Strg**+**S**) und Öffnen (**Strg**+**O**) von Dateien wurden beibehalten.

3. Rückgabe von informativem Feedback

Eine weitere außerordentlich wichtige Eigenschaft für den Benutzer stellen klare Feedbacks dar. Als Rückmeldung auf Benutzereingaben, kann man dem Benutzer so mitteilen, ob die von ihm durchgeführte Aktion den gewünschten Effekt hat, oder ob es Probleme bei der Ausführung gab. Dabei sollte vor allem die Art und Auffälligkeit des Feedbacks berücksichtigt werden. So können häufig ausgeführte, unkritische Befehle ein dezenteres Feedback erhalten, wohingegen kritische Aktionen und Fehler ein deutlich erkennbares Feedback zurückgeben sollten. Dafür wurde in der Applikation eine Logbox implementiert. In dieser wird dem Benutzer auf jede abgeschlossene Aktion ein Feedback gegeben. Im Fehlerfall wird die Rückmeldung in rot dargestellt.

4. Design von Dialogen zur Verdeutlichung der Abgeschlossenheit

Aktionssequenzen sollten in Gruppen organisiert sein, die zumindest einen Anfang und ein Ende aufweisen. Dadurch kann der Benutzer bereits anhand der Anordnung der Felder durch eine Aufgabe geführt werden. Eine Navigation quer durch das Fenster oder sogar zwischen verschiedenen Fenstern, um eine zusammenhängende Aktionssequenz zu erledigen, ist sowohl zeitlich als auch hinsichtlich der Benutzerfreundlichkeit ineffizient. In Kombination mit dem Feedback nach einer abgeschlossenen Aktionssequenz wird ihm ein Gefühl des ordnungsmäßigen Abschlusses und somit der Entlastung mitgeteilt. Zusammengehörende Funktionen, wie z.B. die Geräteverbindung, wurden so angeordnet, dass sie den Benutzer durch die Aktionssequenz führen.

5. Bereitstellung einer einfachen Fehlerbehandlung

Ein System sollte so entwickelt werden, dass es keine vom Benutzer verursachten Fehler geben kann. Da das in vielen Fällen jedoch nicht möglich ist, sollte zumindest die Auswirkung des Fehlers auf das System so gering wie möglich gehalten werden. Tritt ein Fehler auf, soll das System den Fehler erkennen, den Benutzer darauf hinweisen und ihm durch Feedback die Möglichkeit zur Korrektur geben. Füllt der Benutzer z.B. nicht alle Felder aus, die für das Generieren eines Tests erforderlich sind, wird er entsprechend darauf hingewiesen.

6. Bereitstellung umkehrbarer Aktionen

Die Funktionalität umkehrbarer Funktionen ermutigt den Benutzer verschiedene Elemente der GUI auszuprobieren. Er weiß, dass ausgeführte Aktionen bei Bedarf rückgängig gemacht werden können. Aus Zeitgründen wurde diese Funktionalität in der aktuellen Version der Applikation nicht implementiert, wird jedoch in der Entwicklung der nächsten Version mit berücksichtigt.

7. Unterstützung interner und lokaler Kontrolle

Benutzer sollten in allen Bereichen der graphischen Oberfläche das Gefühl haben, dass sie über die Kontrolle des Systems verfügen. Der Benutzer soll Aktionen initiieren können, statt auf diese reagieren zu müssen.

8. Entlastung des Kurzzeitgedächtnisses

Um die Belastung des Kurzzeitgedächtnisses so gering wie möglich zu halten, sollten Darstellungsfenster nicht überfüllt sein. Weiterhin sollte die Verwendung mehrerer Fenster und die Notwendigkeit des Hin- und Hernavigierens durch diese vermieden werden. Hier wurde nach dem Konzept gearbeitet, so wenige Elemente wie möglich und so viele wie nötig zu verwenden. Eine unnötige Aufteilung auf mehrere Fenster wurde vermieden.

4.3.4 Vorstellung der GUI Applikation

Abbildung 20 im Anhang zeigt das Hauptfenster der graphischen Oberfläche. In der Gruppierung *Communication*(1) kann man die angeschlossenen *Color Controller* detektieren, die Reihenfolge der Geräteverbindung vorgeben und sich mit den Geräten verbinden. Die erfolgreiche Verbindung und die Anzahl der verbundenen Geräte wird im *Connected* Feld angezeigt. Anschließend kann man die gewünschten Sensoreinstellungen(2) für Verstärkung und Belichtungszeit vornehmen. Will man eine Messung starten(5), kann davor ausgewählt werden(4), ob nur eine Messung oder kontinuierliche Messungen in einstellbaren Intervallen gestartet werden sollen. Die Ergebnisse der Messung werden unter dem Reiter *Colors*(6) angezeigt. Hier ist die dominante Wellenlänge, die Sättigung und die Beleuchtungsstärke zu sehen. Um die Zuordnung zwischen dominanter Wellenlänge und Farbe der LED zu vereinfachen, wird die gemessene dominante Wellenlänge im Feld *Color* durch die entsprechende Farbe visualisiert. Über die Spalte *Clear Level* wird dem Benutzer mitgeteilt, wie weit der jeweilige Sensor von der Sättigung entfernt ist. Bei Näherung an das Maximum bietet sich eine Rücknahme der Verstärkung und/oder der Belichtungszeit an. Während man unter *Fast Setup*(2) die Konfiguration aller angeschlossenen Geräte vornimmt, kann man unter den Tabelleneinträgen *Gain* und *Integration Time* die Einstellungen an einzelnen Sensoren anpassen. Über den letzten Spalteneintrag *Status* erhält der Benutzer eine Information über den Betriebszustand der Geräte bzw. der Sensoren. Die in Kapitel 4.1.5 erläuterten Fehlercodes der API werden in entsprechende Strings umgewandelt und hier angezeigt. In der Logbox(7) erhält der Benutzer Rückmeldungen nach abgeschlossenen Aktionen.

Abbildung 21 im Anhang zeigt eine Aufnahme des Applikationsfensters mit dem aktiven Reiter *Testdefinition*. Unter diesem Reiter kann der Benutzer die Sollwerte für einen Test vorgeben und optional die LEDs an den angeschlossenen *netX* Controllern der Firma Hilscher ansteuern(5). Die Verbindung dieser zwei Aktionen in einer Oberfläche soll den Vorgang der Testerstellung vereinfachen, da der Benutzer hier nicht auf eine weitere Software für die LED-Ansteuerung auf der *netX* Ebene angewiesen ist. Die erste Zeile jedes Sensors stellt die aktuellen Messwerte dar. Es können einem Sen-

sor beliebig viele Testreihen zugewiesen werden. Hierbei wurde besonderer Wert darauf gelegt, dass die Anzahl der Testreihen pro Sensor variabel gestaltet ist. Auf diese Weise kann der Test an einfache, Duo- oder sogar RGB-LEDs angepasst werden, bei denen sowohl jede Farbe einzeln, als auch Farbkombinationen getestet werden. In Abbildung 21 sind jedem Sensor zwei Testreihen zugeordnet. Im ersten Testzustand wird geprüft, ob die angeschlossenen LEDs eingeschaltet werden können und die gewünschte dominante Wellenlänge, Sättigung und Beleuchtungsstärke aufzeigen. Anschließend wird im zweiten Testzustand getestet, ob die LEDs ausgeschaltet werden können. Sind alle gewünschten Testzustände für eine zu prüfende Baugruppe definiert worden, kann das Testskript schließlich generiert werden(3). Dabei wird intern geprüft, ob alle für einen Test erforderlichen Einträge existieren und eine entsprechende Rückmeldung gegeben (s.Abb.17). Damit Fehler, wie z.B. falsche Sollwerte oder unpassende Einstellungen am Sensor nicht erst in der endgültigen Testumgebung auffallen, wird dem Benutzer bereits in der graphischen Oberfläche die Möglichkeit gegeben, einen generierten Test auszuführen und das Testresultat einzusehen(3). Der aktuelle Zustand der Testerstellung kann jederzeit gespeichert und z.B. für eine spätere Fortsetzung geöffnet werden. Diese Option reduziert den Aufwand bei der Testerstellung für ähnliche Baugruppen, da auf diese Weise ein fertiggestellter Test geöffnet und an die jeweilige Baugruppe angepasst werden kann.

5 Das Buildsystem

Das Projekt wurde in der Abteilung *netX Tools* betreut, welche unter anderem eine Open Source Infrastruktur zum Testen von Produkten auf netX Basis bereitstellt. Um die Software auf *Github* veröffentlichen zu können, wurde besonderer Wert darauf gelegt, dass nur freie Softwarekomponenten verwendet werden. Zusätzlich sollten die entwickelten Komponenten per *Cross-Platform-Build* unter Linux gebaut werden können, sodass der Continuous Integration Service *Travis CI* zum Erstellen der Software genutzt werden kann. Dabei bieten *Github* und *Travis CI* eine fertige und zuverlässige Infrastruktur, die eine einfache Integration in das Projekt ermöglichen. Im folgenden Kapitel erfolgt zunächst eine generelle Einführung in die einzelnen Komponenten des Buildsystems. Anschließend soll die Integration der Komponenten in die Software des *Color Controllers* erläutert werden.

5.1 GitHub, CMake & Travis CI

GitHub ist ein webbasierter Git Hosting-Dienst für Software-Entwicklungsprojekte und verwendet intern das Version Control System (VCS) *Git*. Der Einsatz eines VCS soll die Entwicklung von Softwareprojekten protokollieren und die Zusammenarbeit im Team vereinfachen. Hier werden Funktionen zur Verfügung gestellt, um Veränderungen an der Software nachzuverfolgen und bei Bedarf, z.B. bei auftretenden Fehlern, rückgängig machen zu können. Weiterhin wird durch die Verwendung solcher Systeme gleichzeitig eine Datensicherung der Software erreicht. Aus größeren Softwareprojekten, an denen eine Vielzahl von Entwicklern parallel arbeiten, ist der Einsatz eines VCS nicht mehr wegzudenken. Hier wird mit dem Ansatz gearbeitet, dass jeder Entwickler an seinem eigenen Entwicklungszweig arbeitet, und die Ergebnisse anschließend zu einem kompletten System zusammengefasst werden[14]. Neben den Funktionen eines klassischen VCS bietet *GitHub* jedoch zusätzlich sogenannte *Community* Funktionalitäten, welche gleichgesinnte Entwickler bei der Entwicklung gemeinsamer Projekte unterstützen sollen. Im Jahr 2011 war *GitHub* bei Open Source Software, gemessen an der Anzahl der *Commits*, der populärste Dienst seiner Art[17].

CMake stellt eine erweiterbare Open Source Software dar, welche den Build Prozess system- und compilerunabhängig verwaltet. Dabei ist *CMake*²³ so ausgelegt, dass es leicht in die native Build Umgebung integriert werden kann. Hierfür werden in den entsprechenden Source Verzeichnissen lediglich sogenannte Konfigurationsfiles (*CMakeLists.txt*) benötigt. Anhand dieser Konfigurationsfiles stellt *CMake* die native Um-

²³<http://www.cmake.org>

gebung zur Verfügung, um Source Code zu kompilieren, Bibliotheken und Wrapper²⁴ zu generieren, ausführbare Programme zu erstellen und dem Installationsprozess die gewünschte Verzeichnisstruktur zu geben. Die Struktur der Konfigurationsdateien ist meist hierarchisch gegliedert und beginnt mit dem Konfigurationsfile im Hauptverzeichnis, welches intern die Konfigurationsfiles der Unterverzeichnisse ausführt[3].

Travis CI ist ein Open Source *Continuous Integration Service*, der für das Bauen und Testen von auf *GitHub* veröffentlichten Projekten verwendet wird. Um *Travis CI* in die eigene Software zu integrieren, wird im Hauptverzeichnis des *GitHub* Repositories ein spezielles Konfigurationsfile (*.travis.yml*) benötigt. Anhand dieses Konfigurationsfiles wird bei jedem Einchecken in das Repository eine virtuelle Maschine gestartet, die für das Projekt benötigten Module bezogen und anschließend der Build Prozess gestartet. Dieser kann das Übersetzen und Linken der Anwendungsteile, die Durchführung von Tests an den Softwarekomponenten oder das Hochladen der erfolgreich gebauten Komponenten übernehmen. Dabei beinhaltet die virtuelle Maschine nur das obligatorische Betriebssystem, alle weiteren benötigten Module müssen manuell mithilfe des Konfigurationsfiles bezogen werden. Das schafft die Grundvoraussetzung für eine wohl definierte Build Umgebung, bei welcher der Entwickler vollkommene Kontrolle über bereits vorhandene und zu installierende Pakete besitzt. Nach einem abgeschlossenen Vorgang kann der Entwickler optional über den Status des Builds informiert werden[16].

5.2 Die Verwendung der Komponenten im Projekt

Der gesamte entwickelte Quellcode des Projektes befindet sich auf einem *GitHub* Repository der Firma Hilscher. Das Hauptverzeichnis enthält die benötigten Konfigurationsfiles für *CMake* und *Travis CI*. Sobald auf diesem Repository eingecheckt wird, initiiert *Travis CI* eine virtuelle Maschine (64-Bit Ubuntu 12.04 LTS Server Edition). Aufgabe der virtuellen Maschine ist es, die wohl definierte Build Umgebung für den kommenden Build Prozess zur Verfügung zu stellen. Über die *.travis.yml* Datei werden in einem *before_install* Schritt die Pakete, die für das erfolgreiche Erstellen der Softwarekomponenten benötigt werden, bekannt gemacht und installiert. Diese sind u.a. *Swig* für die Lua Anbindung, *Doxygen* für die Dokumentation, *CMake* für den Build Prozess und Python für das Hochladen der gebauten Komponenten auf eine Filehosting Seite. Ist der *before_install* Prozess abgeschlossen, wird mithilfe eines Shellskripts der Build Vorgang in die Wege geleitet. Dabei generiert *CMake* die Makefiles für eine 32-Bit und eine 64-Bit Version des Projekts. Nach erfolgreicher Kompilierung des kompletten Source Codes stehen die API (led_analyzer.dll) mit generierter Codedokumentation

²⁴Softwarecode, der ein anderes Stück Softwarecode umgibt, um z.B. die Anbindung zwischen verschiedenen Programmiersprachen zu ermöglichen

und die GUI (ColorControl.exe) zur Verfügung. In einem letzten Installationsschritt legt *CMake* die gewünschten Verzeichnisstrukturen an, kopiert die für das Projekt relevanten Dateien in diese hinein und erzeugt daraus ein ZIP-Archiv. Dieses wird von *Travis CI* anschließend in einem *after_success* Schritt auf einer Filehosting Seite zum Download bereitgestellt²⁵.

 GitHub	 Travis CI	 CMake
<ul style="list-style-type: none"> • Versionsverwaltung • Datensicherung 	<ul style="list-style-type: none"> • Initiierung der virtuellen Maschine • Installation der benötigten Pakete • Anstarten des Buildvorgangs (CMake) • Hochladen des ZIP-Archivs auf eine Filehosting Seite 	<ul style="list-style-type: none"> • Generierung der Makefiles (32/64-Bit) • Kompilierung des Source Codes • Installation der gebauten Dateien in die entsprechenden Verzeichnisstrukturen

Tabelle 3: Aufgabe der einzelnen Komponenten im Buildsystem²⁶

²⁵https://dl.bintray.com/muhkuh/Muhkuh/org/muhkuh/tools/led_analyzer/

²⁶Bildquellen Tab.3 (v.l.n.r.): <https://cdn4.iconfinder.com/data/icons/iconsimple-logotypes/512/github-512.png>, <https://travis-ci.com/img/travis-mascot-200px.png>, http://s3.amazonaws.com/cloud.ohloh.net/attachments/920/CMake-logo-triangle-high-res_med.png

6 Fazit

Mit der Fertigstellung der Softwarekomponenten und der Bereitstellung einer wohl definierten Build Umgebung wurde die Softwareentwicklung des *Color Controllers* erfolgreich abgeschlossen. Die Firma Hilscher ist nun in der Lage, ein eigenes kostengünstiges System herzustellen, welches auf die Anforderungen der Testabteilung abgestimmt ist und hier effizient für das automatisierte Testen von LEDs eingesetzt werden kann. Aufgrund der niedrigen Herstellungskosten, erweist sich der Einsatz des *Color Controllers* bereits bei Baugruppen, die nur in geringeren Stückzahlen gefertigt werden, als wirtschaftlich. In der Testabteilung wird zur Zeit ein Testadapter für eine neue Kundenbaugruppe entwickelt. Dabei wird der automatisierte LED Test mithilfe des *Color Controllers*, statt des bis dato teuer eingekauften Testsystems der Firma Feasa, erfolgen. Der geplante Einsatz des *Color Controllers* kann bereits als Bestätigung für eine erfolgreiche Arbeit angesehen werden. Im Laufe des Projektes konnten außerordentlich wertvolle Erfahrungen hinsichtlich der Entwicklung eines gesamten aufeinander abgestimmten Systems gewonnen werden. Beginnend mit dem Sammeln der Anforderungen, über die Konzeptionierung und Entwicklung der Hardware bis hin zur Implementierung einer darauf abgestimmten Software, wurden verschiedene Bereiche der Produktentwicklung kennen gelernt. Weiterhin gewährte die Zusammenarbeit mit der Hardwareabteilung, Testabteilung, *User Interface* und *netX Tools*, welche durch die Vielfalt dieses Projekts zustande gekommen ist, interessante Einblicke in ihre jeweiligen Arbeitsweisen und Methoden. Mit 13000 Zeilen selbstgeschriebenem Code und unter Verwendung dreier verschiedener Programmiersprachen ist ein umfangreiches System entstanden, welches die Firma Hilscher Gesellschaft für Systemautomation mbH näher an ihr Ziel der Test- und Qualitätsoptimierung bringt.



Abbildung 18: GitHub - Eine Gesamtübersicht

7 Anhang

7.1 Farbraumtransformationen und photometrische Größen

RGB → Yxy

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (1)$$

XYZ → Yxy

$$Y = Y \quad (2)$$

$$x = \frac{X}{X + Y + Z} \quad (3)$$

$$y = \frac{Y}{X + Y + Z} \quad (4)$$

Photometrische Größen

Größe	Symbol	SI-Einheit	Formel
Lichtstrom	Φ_V	Lumen (lm)	$\Phi_V = I_V * \Omega$ Ω : Raumwinkel, in dem die Lichtquelle strahlt
Lichtstärke	I_V	Candela (cd)	$I_V = \frac{\Phi_V}{\Omega}$
Beleuchtungsstärke	E_V	Lux (lx)	$E_V = \frac{\Phi_V}{A}$ A : Beleuchtete Fläche
Leuchtdichte	L_V	$\frac{cd}{m^2}$	$L_V = \frac{I_V}{A}$

Tabelle 4: Überblick über photometrische Größen

7.2 Versuchsaufbau - Messung von LEDs

Abbildung 19 zeigt einen Versuchsaufbau, bei dem mehrere LEDs auf ihre Funktion getestet werden. Hierbei sind alle 16 verfügbaren Messkanäle des *Color Controllers* über LWL mit den Duo-LEDs einer Experimentierplatine verbunden. Auf dem Experimentierboard werden zunächst die grünen LEDs angesteuert. Diese sind im Datenblatt als *Green*[11] aufgeführt. Anschließend wird der unter Kapitel 4.2 beschriebene LED Test gestartet, der mit der Sollwerttabelle aus Listing 1 arbeitet. Der Test vergleicht die aktuellen Messwerte für dominante Wellenlänge, Sättigung und Beleuchtungsstärke mit den entsprechenden Werten der Sollwerttabelle und gibt das Testergebnis in XML-Format aus. Ein Ausschnitt des erfolgreichen Testresultats ist in Listing 2 zu finden. Als nächstes soll ein Fehler simuliert werden, indem auf dem Experimentierboard die erste und letzte Leuchtdiode ausgeschaltet werden. Bei einem wiederholten Start des LED Tests werden die fehlerhaften LEDs erkannt und im Testergebnis entsprechend gekennzeichnet. Der Ausschnitt aus dem Testergebnis des fehlgeschlagenen Tests ist unter Listing 3 zu finden.

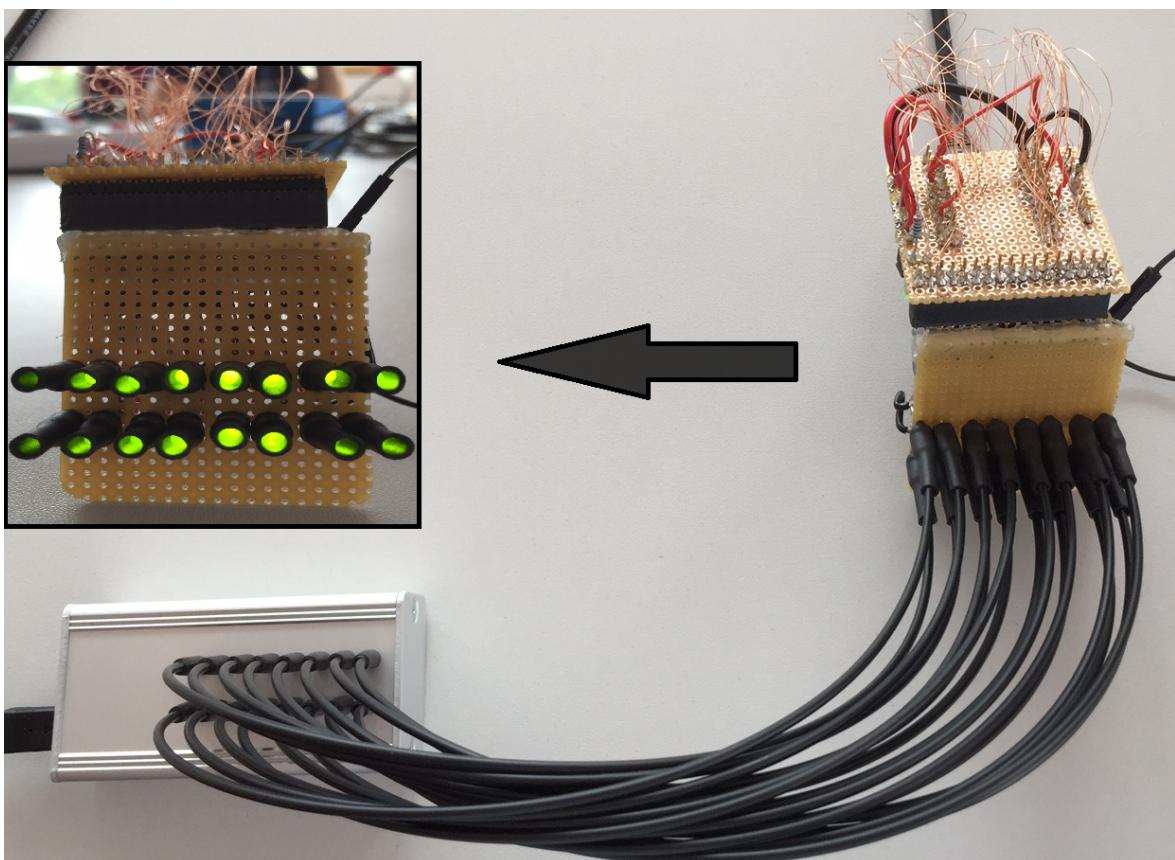


Abbildung 19: Versuchsaufbau - parallele Messung mit 16 Sensoren

```

1 local tTestSet0 = {
2
3 [ 0] = {
4     [ 1] = {name = "g1", nm = 570, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
5         tol_lux = 50},
6     [ 2] = {name = "g2", nm = 570, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
7         tol_lux = 50},
8     [ 3] = {name = "g3", nm = 571, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
9         tol_lux = 50},
10    [ 4] = {name = "g4", nm = 571, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
11        tol_lux = 50},
12    [ 5] = {name = "g5", nm = 571, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
13        tol_lux = 50},
14    [ 6] = {name = "g6", nm = 571, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
15        tol_lux = 50},
16    [ 7] = {name = "g7", nm = 570, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
17        tol_lux = 50},
18    [ 8] = {name = "g8", nm = 571, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
19        tol_lux = 50},
20    [ 9] = {name = "g9", nm = 572, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
21        tol_lux = 50},
22    [10] = {name = "g10", nm = 571, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
23        tol_lux = 50},
24    [11] = {name = "g11", nm = 570, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
25        tol_lux = 50},
26    [12] = {name = "g12", nm = 571, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
27        tol_lux = 50},
28    [13] = {name = "g13", nm = 571, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
29        tol_lux = 50},
30    [14] = {name = "g14", nm = 572, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
31        tol_lux = 50},
32    [15] = {name = "g15", nm = 571, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
33        tol_lux = 50},
34    [16] = {name = "g16", nm = 571, tol_nm = 10, sat = 90, tol_sat = 20, lux = 60,
35        tol_lux = 50}
36 }
37
38 }

```

Listing 1: Sollwerttabelle für einen LED-Test

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <LuaReport>
3   <Version major="1" minor="0"/>
4   <Result description="All readings ok!" result="pass"/>
5   <Traces>
6     <Trace minThreshold="565" maxThreshold="575" unit="nm" value="571" message="# 1
      - Wavelength is in range" success="true"/>
7     <Trace minThreshold=" 70" maxThreshold="100" unit "%" value=" 97" message="# 1 -
      Saturation is in range" success="true"/>
8     <Trace minThreshold=" 10" maxThreshold="110" unit="lux" value=" 78" message="# 1
      - Illumination is in range" success="true"/>
9     <Trace minThreshold="565" maxThreshold="575" unit="nm" value="569" message="# 2
      - Wavelength is in range" success="true"/>
10    <Trace minThreshold=" 70" maxThreshold="100" unit "%" value=" 96" message="# 2 -
      Saturation is in range" success="true"/>
11    <Trace minThreshold=" 10" maxThreshold="110" unit="lux" value=" 77" message="# 2
      - Illumination is in range" success="true"/>
12    . . .
13    . . .
14    <Trace minThreshold="566" maxThreshold="576" unit="nm" value="568" message="#15
      - Wavelength is in range" success="true"/>
15    <Trace minThreshold=" 70" maxThreshold="100" unit "%" value=" 85" message="#15 -
      Saturation is in range" success="true"/>
16    <Trace minThreshold=" 10" maxThreshold="110" unit="lux" value=" 22" message="#15
      - Illumination is in range" success="true"/>
17    <Trace minThreshold="566" maxThreshold="576" unit="nm" value="572" message="#16
      - Wavelength is in range" success="true"/>
18    <Trace minThreshold=" 70" maxThreshold="100" unit "%" value="100" message="#16 -
      Saturation is in range" success="true"/>
19    <Trace minThreshold=" 10" maxThreshold="110" unit="lux" value=" 67" message="#16
      - Illumination is in range" success="true"/>
20  </Traces>
21 </LuaReport>
```

Listing 2: XML Ausschnitt eines erfolgreichen LED Tests

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <LuaReport>
3   <Version major="1" minor="0"/>
4   <Result description="Errors with LED(s): 1,16" result="interrupt"/>
5   <Traces>
6     <Trace minThreshold="565" maxThreshold="575" unit="nm" value=" 0" message="# 1 -
      Wavelength is not in range" success="false"/>
7     <Trace minThreshold=" 70" maxThreshold="100" unit "%" value=" 0" message="# 1 -
      Saturation is not in range" success="false"/>
8     <Trace minThreshold="565" maxThreshold="575" unit="nm" value="569" message="# 2 -
      - Wavelength is in range" success="true"/>
9     <Trace minThreshold=" 70" maxThreshold="100" unit "%" value=" 96" message="# 2 -
      Saturation is in range" success="true"/>
10    <Trace minThreshold=" 10" maxThreshold="110" unit="lux" value=" 77" message="# 2 -
      - Illumination is in range" success="true"/>
11    . . .
12    . . .
13    <Trace minThreshold="566" maxThreshold="576" unit="nm" value="568" message="#15 -
      - Wavelength is in range" success="true"/>
14    <Trace minThreshold=" 70" maxThreshold="100" unit "%" value=" 85" message="#15 -
      Saturation is in range" success="true"/>
15    <Trace minThreshold=" 10" maxThreshold="110" unit="lux" value=" 22" message="#15 -
      - Illumination is in range" success="true"/>
16    <Trace minThreshold="566" maxThreshold="576" unit="nm" value=" 0" message="#16 -
      Wavelength is not in range" success="false"/>
17    <Trace minThreshold=" 70" maxThreshold="100" unit "%" value=" 0" message="#16 -
      Saturation is not in range" success="false"/>
18  </Traces>
19 </LuaReport>
```

Listing 3: XML Ausschnitt eines fehlgeschlagenen LED Tests

7.3 GUI - Screenshots

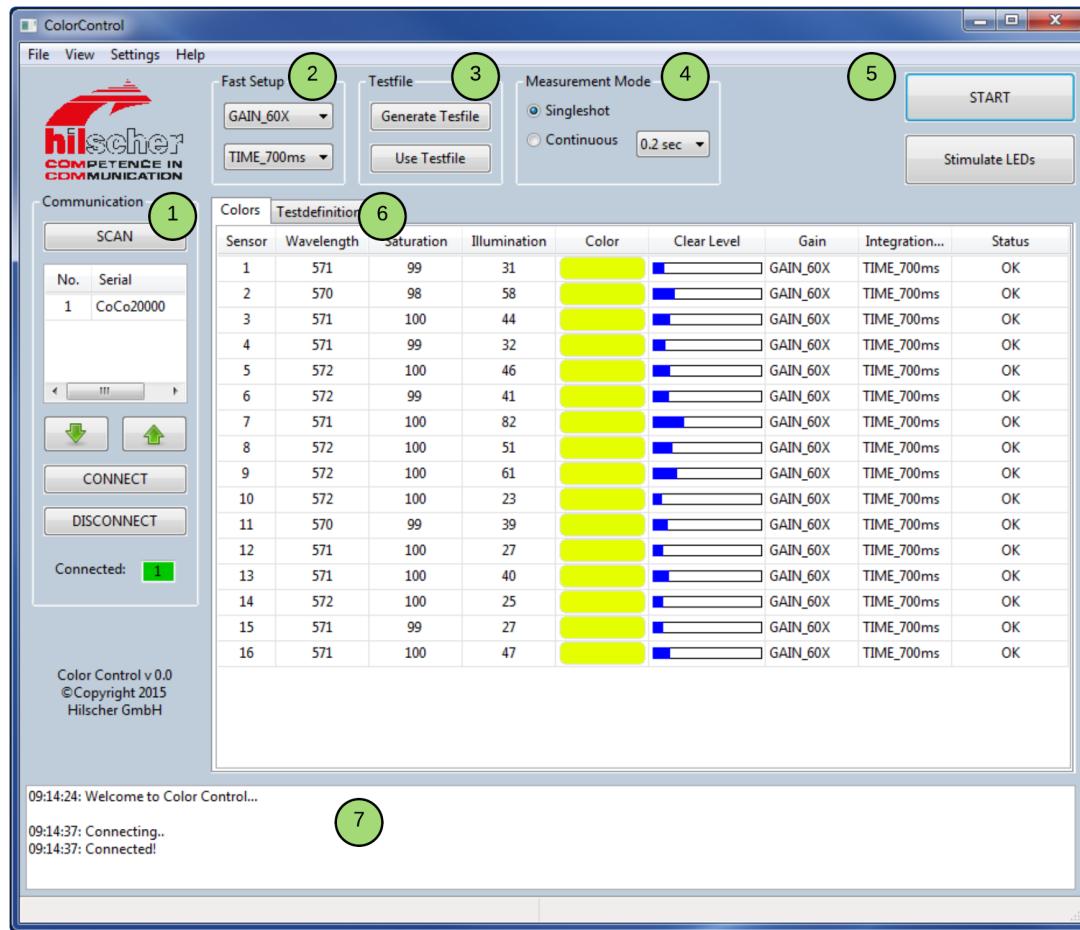


Abbildung 20: GUI - Hauptfenster

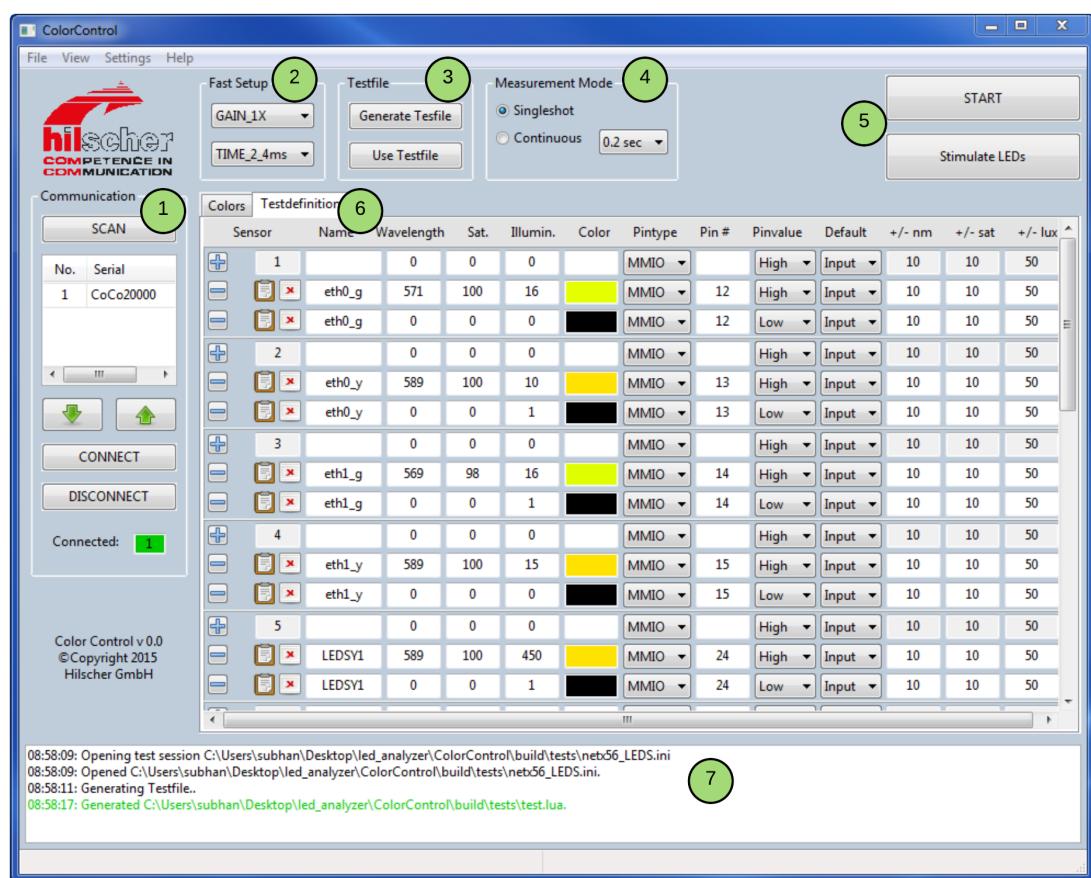


Abbildung 21: GUI - Testdefinition

Literaturverzeichnis

- [1] AMS (Hrsg.): *Lux and CCT Calculations using ams Color Sensors*. ams, <http://ams.com/eng/Products/Light-Sensors/Color-Sensor/TCS34725>. – Vers. 1.0
- [2] AMS (Hrsg.): *TCS3472_Datasheet_EN_v2.pdf*. ams, 04 2013. <http://ams.com/eng/Products/Light-Sensors/Color-Sensor/TCS34725>. – Vers. 1.0
- [3] CMAKE: Reference Documentation. <http://www.cmake.org/documentation/>
- [4] COMMISSION INTERNATIONALE DE L'ECLAIRAGE (Hrsg.): "Colorimetry". Commission Internationale de l'Eclairage. – CIE 15.2-1986
- [5] DOXYGEN: Doxygen Documentation Overview. <http://www.stack.nl/~dimitri/doxygen/manual/index.html>
- [6] FALK, David ; BRILL, Dieter ; STORK, David: *Seeing the Light*. John Wiley and Sons, 1985
- [7] FEASA ENTERPRISES LIMITED (Hrsg.): *Feasa LED Analyser*. Feasa Enterprises Limited, 09 2013. http://www.feasa.ie/PDFs/Feasa_LED_Analysers.pdf. – Rev. 7
- [8] FTDI (Hrsg.): *AN2232C-01_MPSSE_Cmnd.pdf*. FTDI, 10 2006. http://www.ftdichip.com/Support/Documents/AppNotes/AN2232C-01_MPSSE_Cmnd.pdf. – Vers. 2.2
- [9] INSTRUMENT SYSTEMS GMBH (Hrsg.): *LED-Messtechnik*. Instrument Systems GmbH, http://www.instrumentsystems.de/fileadmin/editors/downloads/Products/LED_Handbuch.pdf. – Vers. 1.0
- [10] KÜHNE, Matthias: *Lua - eine Skriptsprache*. TU Dresden, http://st.inf.tu-dresden.de/files/teaching/ss09/PS09/kuehne_ausarbeitung.pdf
- [11] KINGBRIGHT (Hrsg.): *L-59EGW-CA.pdf*. Kingbright, <http://www.farnell.com/datasheets/1446139.pdf>. – Rev. 12
- [12] NXP (Hrsg.): *I2C-bus specification and user manual*. NXP, 04 2014. http://www.nxp.com/documents/user_manual/UM10204.pdf. – Rev. 6
- [13] SAITO, Terubumi ; DALLA BETTA, Prof. Gian F.: *Spectral Properties of Semiconductor Photodiodes*. InTech, 2011. <http://www.intechopen.com/books/advances-in-photodiodes/spectral-properties-of-semiconductor-photodiodes>. – Vers. 1.0
- [14] SCHÜTZE, Jan: *Verteilte Versionsverwaltung*. FH Wedel, <http://www.fh-wedel.de/~si/seminare/ws08/Ausarbeitung/06.vvw/vvw1.htm>
- [15] SHNEIDERMAN, Ben ; PLAISANT, Catherine: *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison Wesley, 2004. – 4th edition
- [16] TRAVISCI: Reference Documentation. <http://docs.travis-ci.com/>
- [17] WIKIPEDIA: GitHub. <https://de.wikipedia.org/wiki/GitHub>
- [18] WIKIPEDIA: CIE-Normvalenzsystem. (2015), 06. <https://de.wikipedia.org/wiki/CIE-Normvalenzsystem>

- [19] wxWIKI: wxWidgets Compared To Other Toolkits. (2012), 06. https://wiki.wxwidgets.org/WxWidgets_Compared_To_Other_Toolkits

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Bachelorarbeit

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Soweit ich auf fremde Materialien, Texte oder Gedankengänge zurückgegriffen habe, enthalten meine Ausführungen vollständige und eindeutige Verweise auf die Urheber und Quellen. Alle weiteren Inhalte der vorgelegten Arbeit stammen von mir im urheberrechtlichen Sinn, soweit keine Verweise und Zitate erfolgen. Mir ist bekannt, dass ein Täuschungsversuch vorliegt, wenn die vorstehende Erklärung sich als unrichtig erweist.

Unterschrift :

Ort, Datum :

