Bachelorarbeit
im Studiengang Audiovisuelle Medien

Methode zur Farbkorrektur bei der Verwendung von LED-Walls mit Kameras

Mit besonderem Fokus auf blickwinkelabhängige Farbkorrektur

vorgelegt von
Niko Pallas
Martrikelnummer: 35938

an der Hochschule der Medien Stuttgart am
01. März 2022
zur Erlangung des akademischen Grades eines
Bachelor of Engineering

Erstprüfer: Prof. Dr. Jan Fröhlich
Zweitprüfer: Patrick Bülles

Ehrenwörtliche Erklärung

Hiermit versichere ich, *Niko Pallas*, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel:

Methode zur Farbkorrektur bei der Verwendung von LED-Walls mit Kameras

Mit besonderem Fokus auf blickwinkelabhängige Farbkorrektur

selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

München, den *28. Februar 2022*

Niko Pallas

Kurzfassung

Bei *LED Virtual Production* werden LED-Walls mit Kameras abgefilmt. In Echtzeit gerenderte virtuelle Sets werden mit Hilfe von Kamera-Tracking für die Kamera perspektivisch korrekt auf die LED-Walls projektiert. Objekte oder Protagonisten vor der LED-Wall scheinen sich so in dem virtuellen Set zu befinden.

Nicht nur in diesem Szenario, sondern in jedem anderen, in dem LED-Walls mit Kameras abgefilmt werden, entsteht ein Problem: Das angezeigte Bild stimmt farblich nicht mit dem überein, welches die Kamera aufgenommen hat. Außerdem verändert sich das aufgenommene Bild drastisch je nach Blickwinkel auf die LED-Wall.

Verschiedene Produktionen lösen diese Probleme anders. Bis jetzt sind wenige Methoden öffentlich bekannt, um diese farbliche Diskrepanz zu korrigieren und keine, die das Problem der Blickwinkelabhängigkeit angeht.

Diese Arbeit versucht, die bis dato veröffentlichte Methode zur Korrektur nachvollziehbar zu machen und um ein Modul zur blickwinkelabhängige Korrektur zu erweitern. In einem Testaufbau wird das blickwinkelabhängige Verhalten eines LED-Panels gemessen und die erstellte Methode evaluiert.

Schlagwörter: XR, Virtuelle Produktionen, In-Camera-Kalibrierung, In-Camera-VFX, XR-Studio

Abstract

With *LED Virtual Production*, LED walls are filmed with cameras. Virtual sets rendered in real time are projected onto the LED walls with correct perspective using camera tracking. Objects or protagonists that are in front of them seem to be in the virtual set.
But not only in this scenario, but in any other where LED walls are filmed with cameras, a problem arises. The image to be displayed does not match the colorvalue of what the camera captures. In addition, the recorded image changes drastically depending on the angle from which the LED wall is viewed.
Various productions solve these problems differently, until now only few methods are publicly available to correct this color discrepancy and none that address the viewing angle dependency problem.
This work tries to make the so far published method for correction comprehensible and to extend it with a module for viewing angle dependent correction. In a test setup, the viewing angle dependent behavior of an LED panel is measured and the created method is evaluated here.

Keywords: XR, virtual production, in-camera-calibration, in-cam-vfx, xr-studio

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

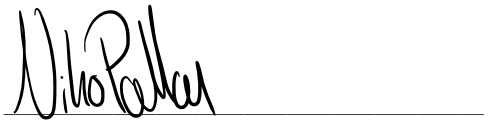
Tabellenverzeichnis

Vorwort

Bei XR-Produktionen mit dem *disguise Inc. XR-Workflow*, durfte ich in den letzten eineinhalb Jahren diesen Workflow für einige Live-Sendungen anleiten und verantworten. Herausfordernd an diesem Workflow waren unter anderem die Farbabweichungen generell zwischen dem gerenderten Bild im Medienserver und dem Kamerabild, das von der LED-Wall abgefilmte Bild. Besonders sichtbar waren zusätzlich Veränderungen des abgefilmten Bilds je nach Blickwinkel. Die bisherige Lösung ist das Farbeinmessen im Standard-Blickwinkel und dynamisch, mit einem Controller, Farbkorrektur-Parameter wie Helligkeit und Farbton pro LED-Wall je Blickwinkel händisch anzupassen. Zu beobachten ist dabei auch, dass die generelle Farbeinmessung in der Praxis oft artefaktbehaftet bzw. schwer nachvollziehbar ist.

Die vorliegende Bachelorarbeit befasst sich deshalb mit der Frage, inwiefern LED-Walls bei der Verwendung mit Kameras so farbkorrigiert werden können, dass das aufgenommene Bild über verschiedene Blickwinkel hinweg dem entspricht, das dem Computer bzw. Medienserver vorliegt. Dabei wurde die aktuell veröffentlichte Methode von James u. a. 2021 nachvollzogen und erweitert, sodass diese Frage evaluiert werden kann und die Methode für zukünftige Arbeiten quelloffen verfügbar ist.
Außerdem wurde das Verhalten der Strahlung eines LED-Panels für verschiedene Blickwinkel mit einem Spektrometer untersucht.

München, den 28. Februar 2022



Niko Pallas

1 Einleitung

Eine LED-Wall mit einer Kamera abzufilmen war noch vor einigen Jahren fast undenkbar. Im Zusammenhang mit Virtual-Production oder In-Camera-VFX wird das tagtäglich in immer mehr Studios tatsächlich gemacht. Dabei wird häufig von einer Revolution des Films gesprochen. Mancher spricht sogar von der Ablösung des Green-Screens. (Slawsky 2020) Nicht nur in diesem Zusammenhang spielen abgefilmte LED-Walls eine Rolle. Auch für klassische Hintergrund-Bespielung in Fernsehstudios werden LED-Walls genutzt, um Bilder wiederzugeben, die mit einer Kamera abgefilmt werden. (FKT 2021)

1.1 Motivation

Fundamental für diese Anwendungen ist ein gesamtheitlich geplantes Farbmanagement. Ein virtuelles Set soll schließlich durch die Kamera so aussehen, wie es im Vorhinein erstellt wurde. Eine Grafik, die im einem Fernsehstudio eingeblendet wird, soll genauso aussehen, wie zuvor gestaltet. James u. a. 2021 beschreiben hierfür eine Methode, um das Bild der LED-Wall so zu kompensieren, dass es durch die Kamera wieder wie das ursprüngliche Bild wirkt. Auch das Unternehmen disguise hat ein propietäres System hierfür entwickelt. Für deren *XR-Workflow* ist, auf Grund der virtuellen *Setextension*, ein Farbmanagement essentiell, um die Illusion aufrecht zu erhalten. Am Übergang zwischen LED-Wall und der virtuellen *Setextension* dürfen keine sichtbaren farblichen Unterschiede erkennbar sein.
Ein noch nicht gelöstes Problem stellt der Blickwinkel dar. So z.B. bringt eine Korrektur für den senkrechten Blickwinkel nur für diesen einen Blickwinkel den gewünschten Effekt. Je nach Blickwinkel verändert sich der Color-shift und der Helligkeitsabfall. (Seymour 2020) Insbesondere bei *Cube*-Aufbauten von LED-Walls ist dieser Effekt stark in Bereichen zwischen den LED-Walls zu beobachten, da diese senkrecht zueinander stehen. In Kombination mit einer virtuellen *Setextension* ist dieser blickwinkelabhängige Effekt eine enorme Einschränkung für die Kamerapositionen und -bewegungen.

1.2 Ziel der Arbeit

Auf Grund der angesprochenen Problematik stellt diese Arbeit eine Methode vor, mit der es möglich ist, die Korrektur einer LED-Wall für die Verwendung mit einer Kamera blickwinkelabhängig durchzuführen. Die Methoden zur Korrektur für LED-Walls von James u. a. 2021 und disguise sind nicht ausreichend nachvollziehbar, um eine Erweiterung darauf aufzubauen.
Deshalb wird in dieser Arbeit eine Methode erläutert, welche die Methode von James u. a. 2021 nachvollzieht, quelloffen aufbereitet und um eine blickwinkelabhängige Korrektur erweitert. Um die physikalischen Auswirkungen des Blickwinkels auf die emittierte Strahlung einer LED-Wall genauer beurteilen zu können, wird ein LED-Panel über mehrere Blickwinkel mit einem Spektrometer vermessen. Hieraus wird ein besseres Verständnis über das blickwinkelabhängige Verhalten erlangt, erlaubt Schlussfolgerungen auf die Auswirkung des Blickwinkels auf das Kamerabild und die Machbarkeit einer blickwinkelabhängigen Korrektur. Die Funktionen der Einzelkomponenten der vorgeschlagene Methode werden in einem Versuchsaufbau verifiziert und evaluiert.
Ziel ist es, eine Schlussfolgerung auf die Machbarkeit einer blickwinkelabhängigen Korrektur zu ziehen und einen Vorschlag für die Umsetzung zu unterbreiten.

1.3 Struktur der Arbeit

Im ersten Schritt wird die Methode von James u. a. 2021 und dessen Grundlagen nachvollzogen. Außerdem wird der Frage nachgegangen, aus welchen Gründen es zu der Diskrepanz zwischen dem zur LED-Wall gesendeten Bild und dem aufgenommenen Kamerabild kommt. Die Gründe für die blickwinkelabhängige Veränderung der Strahlung der LED-Wall werden in einer Literaturrecherche untersucht.
In Kapitel 3 wird die erarbeitete Methode nachvollziehbar aufbereitet. Der Einfluss des Blickwinkels auf die emittierte Strahlung wird in Kapitel 4 untersucht und interpretiert. In Kapitel 5 wird die entwickelte Methode im Detail untersucht und evaluiert. In der Diskussion in Kapitel 6 werden die Erkenntnisse zusammengefasst und in den Gesamtkontext der Arbeit eingebettet.

2 Grundlagen

Eine LED-Wall soll bei der Verwendung mit Kameras so über mehrere Blickwinkel farblich korrigiert werden, dass das durch die Kamera aufgenommene Bild dem Originalbild entspricht. Der Arbeit liegen bereits veröffentlichte Konzepte und Technologien zu Grunde. Die Grundlagen sollen einen Überblick geben, mit dem Fokus auf die Bereiche, die für eine konkrete Anwendung in dieser Arbeit relevant sind.

2.1 Zusammenspiel zwischen Kamera und Displays

Eine digitale Kamera ist dafür optimiert, ein abstraktes Abbild der Farbe bzw. des sichtbaren Lichts einer Szene zu erstellen. Dabei wird in den meisten Kameras von einer realen Verteilung der Strahldichte $L_{e,\lambda}$ einer Szene auf drei digitale Werte abgebildet. Ein Display ist als Gegenstück zu verstehen. Aus einem abstrakten Abbild wird eine Verteilung der Strahldichte $L_{e,\lambda}$. Beide Systeme sind in ihrem Design für die Wahrnehmung des Menschen optimiert. (G. Sharma und Trussell 1997) Dieser Sachverhalt begründet die initiale Problemstellung. Im betrachteten Workflow wird die Kamera nicht nur verwendet, um eine realen Szene, sondern auch um ein Display aufzunehmen. In der Aufnahme soll das Display nicht von der realen Szene unterscheidbar sein.

2.1.1 Digitale Farben und Farbwahrnehmung

Der Mensch kann im Bereich von 380 bis 780nm elektromagnetische Strahlung mit Hilfe des Auges wahrnehmen. Die verteilte Strahldichte $I_{e,\lambda}$ über diesen Bereich ergibt in der menschlichen Wahrnehmung einen Farbeindruck. (Schmidt 2013, S. 16f) Durch die additive Mischung von drei Primärfarben, können nach Graßmann 1853 Farbeindrücke, also spektral anders verteilte Strahldichten, dargestellt werden. In Experimenten, unter anderem von Maxwell 1857, konnte dies bewiesen werden. In mehreren Versuchen wurden drei *color matching functions CMFs* ermittelt. Diese *CMFs* bilden von einer spektralen Verteilung auf jeweils einen Wert ab. Diese Funktionen sollen möglichst der Ansprache pro Wellenlänge der drei Arten von Zapfen im Auge entsprechen. Von der *Commission Internationale de l'Éclairage* International Commission on Illumination *CIE* wurden diese

CMFs als Grundlage dafür verwendet, digitale Farben als numerische Werte mit drei Koordinaten *tristimulus values* zu behandeln. Aus diesen *CMFs* ergaben sich allerdings auch negative Werte, die mit den Versuchen zur Aufstellung dieser *CMFs* zusammenhängen. Daher wurden die *CIE XYZ CFMs* entwickelt. Diese imaginären Primaries *XYZ* können nicht physisch realisiert werden, jedoch alle sichtbaren Farben in drei Werten abbilden. (G. Sharma und Trussell 1997)
Mit Hilfe der *tristimulus* Werte XYZ lassen sich Farbwerte im dreidimensionalen Raum darstellen. Um nur einen Farbwert, ohne Information zu dessen Intensität, numerisch darzustellen, wurde der *CIE xyY*-Farbraum definiert. Von G. Sharma und Trussell 1997 wird dieser auch als *CIE xyz* beschrieben. Daraus ergibt sich die CIE Normfarbtafel, ein zweidimensionales Diagramm, auf dem nur die xy-Koordinaten zu sehen sind. Hier wird der *CIE XYZ*-Farbraum auf einen zweidimensionalen Raum projiziert. So kann durch einen zweidimensionalen Vektor *x,y* ein Farbwert definiert werden. (Hasche und Ingwer 2016, S. 22ff) Da der *CIE xyY*-Farbraum bzw. der *CIE XYZ*-Farbraum den Abstand zweier Farbwerte nicht proportional zur menschlichen Wahrnehmung darstellt, wurde 1976 der *CIE 1976 L*u*v**-Farbraum definiert. Der euklidische Abstand zwischen zwei Vektoren in diesem Farbraum liefert eine Farbunterschiedsformel zur Bewertung von Farbunterschieden in wahrnehmungsrelevanten Einheiten. Wenn der Abstand dagegen im *CIE XYZ*-Farbraum berechnet wird, ist das nicht der Fall. Mit den beiden Koordinaten *u'v'*, die sich für die Berechnung von *CIE 1976 L*u*v** ergeben, lässt sich ebenso ein zweidimensionaler Raum darstellen. Die hier sichtbaren Abstände zwischen zwei Farbwerten, entsprechen gleichmäßig proportional dem wahrgenommenen Unterschied. (G. Sharma und Trussell 1997)

Die numerischen Werte dieser Farben können jeweils im linearen Arbeitsraum vorliegen oder Gamma-kodiert. Dieser Sachverhalt hat seinen Ursprung in der Verwendung von Röhrenbildschirmen, die baubedingt eine sogenannte *electro-optical conversion function* OECF innehalten. Das bedeutet, dass ein numerischer Farbwert von z.B. *0.5, 0.5, 0.5* nicht der Hälfte der maximal emittierbaren Lichtmenge entspricht. Um dennoch einen linearen Zusammenhang zwischen den numerischen Werten und der Strahldichte zu haben, wurde das Bild mit einer EOCF (electronic-optical conversion function) kodiert. Die Funktionen EOCF und OECF heben sich gegenseitig auf, sodass z.B. *0.5* als Wert auch *0.5*-fache Lichtmenge bedeutet.

Digitale Farbwerte mit einer EOCF zu kodieren hat noch einen zweiten Grund. Nach dem Weber-Fechnerschen Gesetz nehmen Menschen Unterschiede nicht im mathematischen Sinne linear war. Stattdessen entspricht eine logarithmische Progression im mathematischen Sinne einer wahrgenommenen linearen Progression. Deshalb ist es nicht sinnvoll in digitalen Bilddaten einen numerisch gleichen Abstand zwischen zwei großen Werten und zwei kleinen Werten über eine Quantisierung zu erhalten. Die Werte werden je nach Einstellungen in verschiedenen Bit-Tiefen quantisiert. Das bedeutet, dass die Bildwerte zwischen *0.0* und *1.0* bei 8bit in 2^8 verschiedenen Werten gespeichert werden können. Deshalb sollen durch eine EOCF die Unterschiede in *dunklen* Bereichen eher erhalten werden als in *hellen*. Am Ende der Bildverarbeitung sollen auf dem darstellenden Medium die linearen Helligkeitsdaten so angezeigt werden, dass ein Mensch mit logarithmischer Wahrnehmung diese korrekt interpretieren kann. (Hasche und Ingwer 2016, S. 42ff) Daraus ist zu folgern, dass Bilddaten entweder im linearen Arbeitsraum oder Gamma-kodiert vorliegen. Für eine Weiterverarbeitung und Manipulation der Daten ist diese Information essentiell.

2.1.2 Farbaufnahme einer Kamera

Um ein Bild bzw. eine Strahldichte-Verteilung $L_{e,\lambda}$ einer Szene digital verarbeiten zu können, muss ein Bild abgetastet und quantisiert werden. Eine Kamera soll aus dem einfallenden Licht pro Pixel einen digitalen Farbwert aus drei Werten weitergeben. Durch ein Objektiv fällt Licht auf den Sensor. Dieser setzt Lichtintensität in ein elektrisches Signal um. In dem hier betrachteten Anwendungsbereich kommen hauptsächlich zwei Sensortypen zum Einsatz: *Charge Coupled Device* (CCD) und *Complementary Metal-Oxide-Semiconductor* (CMOS) Sensoren

Der CMOS-Sensor lässt die Spannungswerte der einzelnen Pixel direkt auslesen, wohingegen der CCD-Sensor eine Art Schieberegister ist und die Ladungen von Zelle zu Zelle weiterreicht. Grundlegend geben beide Arten ein elektrisches Signal weiter, das verstärkt wird und von einem *Analog-Digital-Wandler* (AD-Wandler) in ein digitales Signal gewandelt, also abgetastet wird. (Schmidt 2013, S. 354ff)

Um mit diesen Sensoren ein Farbbild mit drei Werten darzustellen, wird hauptsächlich zwischen zwei Bautypen unterschieden: Der erste Typ ist ein Bayer Pattern bzw. ein Mosaikfilter. Bei diesem Typ befindet sich vor jedem Sensorpixel ein Farbfilter für die wellenlängenabhängige Transmission einzelner Lichtanteile der Farben Rot, Grün oder Blau.

Das heißt für jeden Pixel liegt nur die Lichtintensität für einen bestimmten Bereich vor. Im *Demosaicing* wird je nach Algorithmus der Farbwert der jeweils anderen Kanäle aus den umliegenden Informationen interpoliert. Bei einem 3-Chip-Aufbau werden drei Sensoren verbaut. Das eintreffende Licht wird durch ein Strahlteilerprisma, bestehend aus teildurchlässigen Spiegeln, in drei Lichtanteile Rot, Grün und Blau aufgetrennt. Hier ist das Ergebnis nach der AD-Wandlung ein digitales Bild mit voller Farbauflösung pro Pixel. (Schmidt 2013, S. 378ff) Besonders bedeutsam für diese Arbeit ist der Sachverhalt, dass die Werte für Rot, Grün und Blau durch das Filtern des eintreffenden Lichts entstehen und die Charakteristiken dieser Filter den Farbwert maßgeblich beeinflussen.

2.1.3 Farbwiedergabe eines Displays

Eine Kamera kann vereinfacht als Farbaufnahme-System bezeichnet werden. Ein Display hingegen ist vereinfacht das Gegenstück hierzu: Ein Farbwiedergabe-System.
Durch additive Farbmischung ist es möglich, mit der Regelung drei richtig gewählter Primaries (drei Lichtquellen mit unterschiedlicher Strahldichten-Verteilung $L_{e,\lambda}$) einen Großteil der sichtbaren Farben für die menschliche Wahrnehmung darzustellen. Aus der Strahldichteverteilung $L_{e,\lambda}$ dieser Primärfarben lässt sich ein Farbraum der darstellbaren Farben ableiten. Die Primärfarben abgebildet auf den XYZ-Farbraum, schließen einen Raum ein, der den Raum aller darstellbaren Farben mit diesem Display definiert. Das ist der sogenannte Gamut. (G. Sharma und Trussell 1997)
Es ist zu beachten, dass die RGB-Farbwerte, die einem Display gereicht werden, vor der Anzeige weiter verarbeitet werden. Meist hat das Display ein *ICC-Profil*, das die Transformation des Gamuts und eine Formel zur Gamma-Korrektur innehält. (A. Sharma 2004) Auf die Transformation des Gamuts wird im späteren Verlauf in Abschnitt 2.5 genauer eingegangen.
Es zeigt sich, dass beim Abfilmen eines Displays eine Diskrepanz zwischen der Charakteristik der Farbfilter in der Kamera und der Charakteristik der Primaries des abzufilmenden Displays zu erwarten ist. Das führt zu unterschiedlichen Farbwerten zwischen der darzustellenden und der aufgenommenen Farbe.

2.2 Grundlagen LED-Walls

In dieser Arbeit liegt der Fokus auf LED-Matrix-Displays, sogenannten LED-Walls. Mit den Begriffen *LED-Wand* und *videowall* wird in der Literatur häufig dasselbe bezeichnet. Als LED-Wall wird in dieser Arbeit ein Display verstanden, das bestehend aus mehreren LED-Cabinets flexibel in Höhe und Breite errichtet werden kann. Jeder Pixel dieses Displays besteht aus drei Subpixeln, die jeweils von einer selbstleuchtenden LED (Rot, Grün und Blau) repräsentiert werden. Der Pixelpitch beschreibt den Abstand zwischen den Mittelpunkten zweier Pixel. (Kintrup 2022) Die hier erläuterten Grundlagen und die Methode dieser Arbeit konzentrieren sich hauptsächlich auf die LED-Wall-Technologie, die momentan in Kombination mit Kameras genutzt wird. Die dafür genutzten LED-Walls haben meist einen Pixelpitch zwischen 2.6 und 2.9mm nach Kintrup 2022, S. 4f und bis zu maximal 3.2mm nach der Empfehlung von Netflix Inc. 2021. Das für diese Arbeit verfügbare LED-Cabinet hatte einen Pixelpitch von 3.6mm. (ICT AG o. D.[b]) Die hier erläuterten Bestandteile einer solchen LED-Wall sind folgende:

- LED/ 3in1 SMD LED
- LED-Modul
- LED-Cabinet
- LED-Prozessor

2.2.1 Einzelne LED und 3in1 SMD

Funktionsweise einer Leuchtdiode

Eine Leuchtdiode, kurz LED (light emitting diode), ist ein Halbleiterbauelement. Ebenso, wie eine herkömmliche Diode (indirekter Halbleiter) ist ein Stromfluss nur in Durchlassrichtung möglich. Eine Diode besteht aus drei Schichten, die n-dotierte, die p-dotierte Schicht und die dazwischenliegende Rekombinationsschicht, siehe Abbildung 2.1.

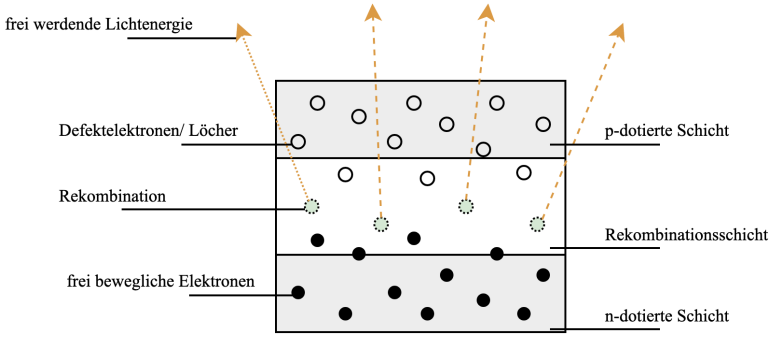


Abbildung 2.1: Schema Funktionsweise LED

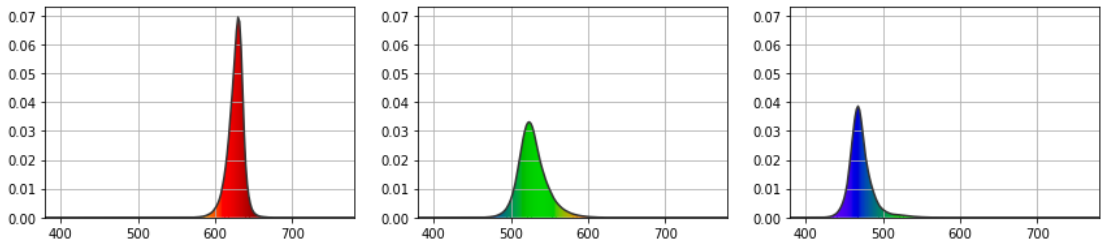
Durch das Anlegen einer Spannung in Durchlassrichtung bewegen sich die freiliegenden Elektronen der n-dotierten Schicht in Richtung der Rekombinationsschicht. Nach Überqueren der Grenzfläche rekombinieren sich diese Elektronen mit den sogenannten Löchern der p-dotierten Schicht. In diesem Prozess wechseln die Elektronen vom Leitungsband (in der n-dotierten Schicht) in das energetisch günstigere Valenzband (in der p-dotierten Schicht). Bei indirekten Halbleitern, z.B. Siliziumdioden, wird diese frei werdende Energie hauptsächlich in Wärme umgesetzt. Während bei direkten Halbleitern, z.B. Gallium-Arsenid, die Energie in Form von Licht freigesetzt wird. Die Wellenlänge hängt mit der Größe der Bandlücke zusammen. (Stiny 2015, S. 22-26, 130–143)
Für die Anwendung in LEDs kommen deshalb nur direkte Halbleiter in Frage. Durch die Wahl der Halbleiterverbindung und deren Verhältnisse wird Licht verschiedener Wellenlängen freigesetzt. Grüne und blaue LEDs bestehen meist aus Galliumnitrid und Indiumnitrid, je nach Verhältnis. Rote LEDs bestehen dagegen in der Regel aus Aluminiumindiumgalliumphosphid-Verbindungen. (Schwirten und Chen 2015).

Binning der LEDs

Da in der Herstellung der einzelnen LEDs leichte Unterschiede zwischen den einzelnen LEDs entstehen, werden diese im *Binning*-Prozess in ihrer Strahldichte L_e pro Wellenlänge λ geprüft. Je nach Qualitätsparametern des Herstellers, werden die einzelnen LEDs nach ihrer maximaler Intensität und Farbraumkoordinaten x,y sortiert, in sogenannte *Bins*. Je nach Qualitätsstandard werden viele oder wenige ähnliche *Bins* genutzt um RGB-SMD-LEDs für ein Videomodul herzustellen. Die genutzten Bins werden dann möglichst gleich verteilt über das LED-Modul angeordnet. (Thielemans 2016, S. 1740ff)

RGB-SMD-LED - Der einzelne Pixel einer LED-Wall

Ein großer Bereich, der für den Menschen sichtbaren Farben, lässt sich durch drei Primärfarben additiv darstellen. Aus der spektralen Verteilung dieser einzelnen Primärfarben lässt sich der Gamut des darstellenden Mediums ableiten. Dieser beschreibt den Raum aller mit diesen Primärfarben darstellbaren Farben. (G. Sharma und Trussell 1997)



(a) Rot (b) Grün (c) Blau

Abbildung 2.2: Strahldichteverteilung $L_{e,\lambda}$ in $W \cdot sr^{-1} \cdot m^{-2}$ und nm pro Subpixel bzw. LED (Rot, Grün, Blau) bei einem *inspireLED*-Panel

RGB-SMD-LEDs finden häufig Anwendung in der Herstellung von LED-Walls. In jedem dieser *Surface-Mounted-Device*-Bauelemente (SMD) befinden sich drei LEDs. In der RGB-Variante, die hier Verwendung findet, sind drei LEDs fest verbaut. Sie emittieren jeweils hauptsächlich Licht im rotem, grünen und blauem Bereich des sichtbaren Lichts (siehe Abbildung 2.2). Grundlegend soll in der Verwendung mit LED-Walls eine RGB-SMD-LED einen Bildpixel repräsentieren.

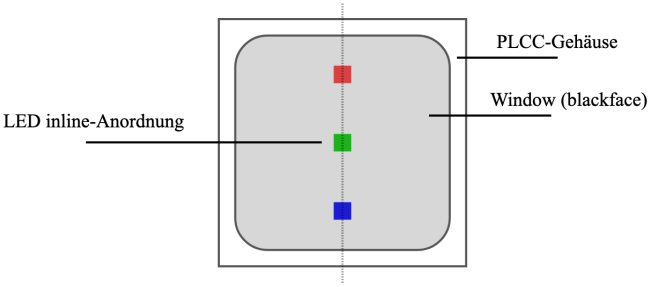


Abbildung 2.3: Aufbau von SMD-RGB-LED und Inline-Anordnung der LEDs

Die einzelnen LEDs werden im *Packaging* gemeinsam in ein *plastic leaded chip carier*-Gehäuse (PLCC) verarbeitet, sodass sie elektronisch angesteuert werden können und anschließend mit *Epoxidharz* vergossen.(Schwirten und Chen 2015)
Die Anordnung der LEDs in einem SMD-Element hat große Auswirkungen auf die Farb-

verschiebung je Blickwinkel. Die von Kintrup 2022 genutzten und erwähnten LED-Wände sind alle in der *in-line* Anordnung. Das bedeutet, dass die einzelnen LEDs vertikal übereinander angeordnet sind, siehe Abbildung 2.3. Das hat zur Folge, dass der Einblickwinkel auf der horizontalen Blickachse verbessert wird. (Schwirten und Chen 2015)
Für die hier besprochene Anwendung werden häufig *blackface*-RGB-SMD-LEDs, genutzt. Das heißt, dass das genutzte *Epoxidharz* verdunkelt ist. Dies verringert die maximale Strahldichte, aber verringert ebenso das reflektierte Licht, sodass der Schwarzwert und das Reflexionsverhalten einer LED-Wall verbessert wird. (Schwirten und Chen 2015)

2.2.2 LED-Modul, -Cabinet

Die einzelnen RGB-SMD-LEDs werden im Herstellungsprozess zu LED-Modulen zusammengefasst. Hierfür werden die einzelnen RGB-SMD-LEDs auf eine Leiterplatine *(printed circuit board - PCB)* gesteckt bzw. gelötet. (Schwirten und Chen 2015)
In den Zwischenräumen der RGB-SMD-LEDs befindet sich der *Shader*. Das ist ein Kunststoffgitter, das zwischen den einzelnen Pixeln sitzt und so Reflexionen des PCBs durch Auflicht vermindert und den Schwarzwert einer LED-Wall verbessert. Mehrere dieser LED-Module beschreiben ein LED-Panel, das in dem Gehäuse bzw. dem LED-Cabinet verbaut ist. Darin befinden sich die Stromversorgung und Anschlüsse zur Übertragung des Videosignals und die entsprechende Verteilung zu den einzelnen Modulen des LED-Panels. (Kintrup 2022)
Mit Hilfe anderer Methoden der RGB-SMD-LED Herstellung in Kombination mit der Modulherstellung lassen sich noch kleinere Pixelpitch-Werte realisieren, bis zu 0.75mm bei Prototypen. (Schwirten und Chen 2015)

2.2.3 LED-Controller / Ansteuerung von LED-Walls

Wie kommt nun das Videosignal an die einzelne LED?
Im ersten Schritt erhält der LED-Controller ein Videosignal. Im Aufbau dieser Arbeit ist das ein Videosignal via HDMI. Mit dem LED-Controller sind die LED-Cabinets meist seriell verbunden. Die maximale Anzahl der anschließbaren LED-Cabinets pro LED-Controller ist begrenzt. Die Begrenzung ergibt sich aus der Bandbreite der Anschlüsse und der Rechenleistung des LED-Controllers. (Kintrup 2022) Er sendet und empfängt Steuersignale der einzelnen LED-Cabinets, synchronisiert sie und wandelt das Videosignal in ein pro-

prietäres Protokoll, sodass die LED-Cabinets im richtigen Takt, an der richtigen Stelle im seriellen Signal den für sie relevanten Bereich erhalten. (Hyun, Kang und Kim 2016)
Über die Steuersoftware des LED-Controllers kann die gesamte LED-Wall konfiguriert werden. Relevante Einstellungen sind hierbei die Information zur Verkabelung der einzelnen LED-Cabinets, die Kalibrierungs-Daten des Herstellers und Einstellungen der Interpretation des eingehenden Videosignals, wie Gamma und Gamut. All diese Parameter werden seriell für jedes einzelne LED-Cabinet vom LED-Controller weitergereicht. Das LED-Cabinet bzw. die *Receivingcard* im Cabinet greift sich den für das LED-Panel relevanten Signalausschnitt ab und verarbeitet diesen weiter. Die Kalibrierungs-Daten werden in die *Receivingcards* jedes einzelnen LED-Panels durch das Konfigurieren des LED-Controllers eingelesen. In der Regel sind die Kalibrierungs-Daten allgemeingültig für jedes LED-Panel einer Baureihe nutzbar. Nach dem Einlesen sollte das LED-Panel konfiguriert sein, dass die einzelnen RGB-SMD-LEDs so angesteuert werden, dass ein homogenes Bild pro LED-Panel und über die ganze LED-Wall hinweg entsteht. Das Videosignal wird hier weiterverarbeitet und jeweils richtig ausgeschnitten an die Treiberchips der jeweiligen LED-Module weitergereicht. Der Treiberchip des LED-Moduls verändert durch Pulsweiten-Modulation die emittierte Lichtstrahlung der einzelnen LEDs. (Kintrup 2022) Da das Verhältnis von Stromstärke I zu Strahlungsenergie Q_e bei einer LED nicht linear verläuft, kann mit Hilfe von Pulsweiten-Modulation ein lineares Verhalten erwirkt werden. (Janglin Chen 2016, S, 12)

2.3 Ursachen der Veränderung der Strahldichteverteilung $L_{e,\lambda}$ einer LED-Wall

Dass es eine Diskrepanz zwischen dem auf der LED-Wall ausgegebenen Bild und dem aufgenommenen Bild der Kamera geben muss, ergibt sich aus den Erläuterungen der vorherigen Abschnitte. Besonders die Diskrepanz zwischen den Wellenlängen-Bändern der LEDs und die der Farbfilter des Sensors sei an dieser Stelle nochmals erwähnt. Bei der Verwendung von LED-Walls sind noch weitere Parameter zu beachten, die Einfluss auf die Strahldichte L_e generell und deren Verteilung $L_{e,\lambda}$ haben.

2.3.1 Alterung

Auf Grund der thermischen Belastung über die Lebensdauer hinweg, verliert eine LED über die Zeit an Strahlstärke. (Stiny 2015, S. 138) Hierdurch wird bei gleichem Strom eine geringere Strahlstärke erzeugt, folglich sinkt die Effizienz und die generelle Strahldichte L_e Eine LED auf *InGaN*-Basis *(Grüne- und Blaue-LED)* weist eine schnellere Alterung auf als eine LED auf *InAlGaP*-Basis *(Rote-LED)*. Hierdurch kann es über die Zeit hinweg zu einem Rotstich kommen. (Schwirten und Chen 2015) Das kann allerdings durch erneutes Kalibrieren und Einmessen der Wall, bis zu einem gewissen Grad, ausgeglichen werden. (NovaStar 2020)

Auch das Epoxidharz des Gehäuses altert. Unter anderem zeigt sich das in Form von Trübung und verringert die effektive Strahlstärke der LEDs. Außerdem treten öfter *Microcracks* auf, das sind kleine Risse im Epoxidharz, also im *Window* der RGB-SMD-LED. Sie können ebenso eine Auswirkung auf das austretende Licht haben, besonders auch auf verschiedene Einblickwinkel durch Brechung und Reflexion innerhalb des Windows. (Schwirten und Chen 2015)

Alterungsveränderungen dieser Art sind allerdings auf einen Zeitraum von mehreren Jahren zu betrachten. (Stiny 2015)

2.3.2 Temperatur

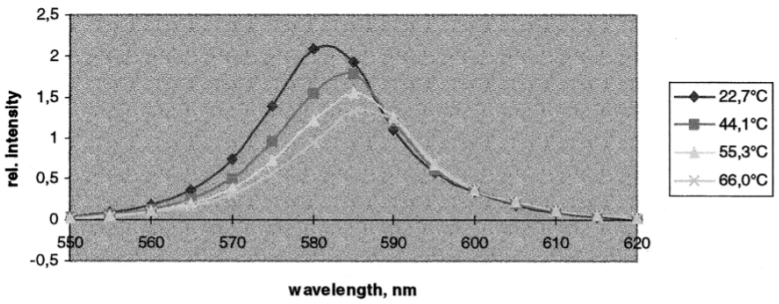


Abbildung 2.4: Veränderung der Strahldichteverteilung einer Grünen LED je nach Temperatur, Abbildung: (Schanda, Muray und Kranicz 2015)

Durch den Betriebstemperaturunterschied können ebenso Unterschiede in der Strahldichteverteilung $L_{e,\lambda}$ auftreten. (Schanda, Muray und Kranicz 2015) Die Leitfähigkeit eines Halbleiters verändert sich je nach Temperatur durch die Veränderung der Ladungsträgerdichte. Diese hat eine unmittelbare Auswirkung auf die emittierte Strahlstärke. (Stiny 2015, S. 36f)

Bestimmte LED-Controller in Kombination mit den entsprechenden *Receivingcards*, z.B. von *Megapixel Visual Reality*, können temperaturbedingte Effekte kompensieren. Hierfür müssen verschiedene Konfigurationen für die jeweilige Betriebstemperatur vorhanden sein und die Receivingcard muss die Temperatur messen und dem LED-Controller mitteilen. (Hochman und Harrison 2021)
Für diese Arbeit soll der Alterungseffekt jedoch vernachlässigbar sein, da Messung und Verifizierung unmittelbar nacheinander erfolgen sollen.

2.3.3 Einblickwinkel

(a) Einblickwinkel 0° (b) Einblickwinkel −70°

Abbildung 2.5: Angezeigter Farbwert (0.8 0.8 0.8) auf *inspireLED*-Panel bei 0° *(senkrecht)* und −70° Einblickwinkel Aufnahmen in *S-Log 2*

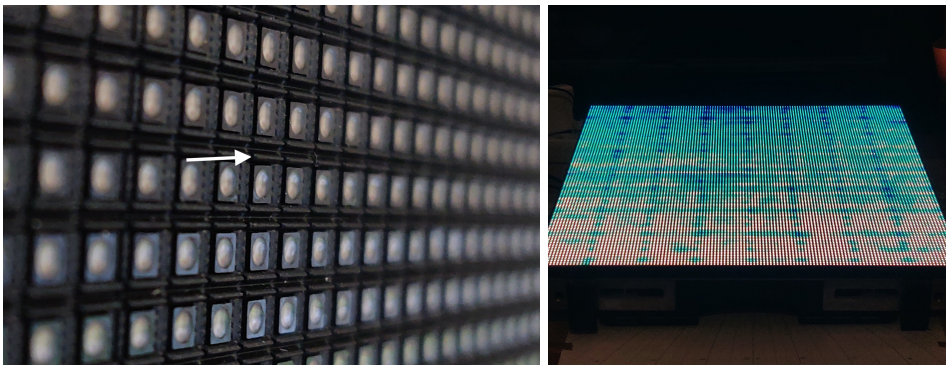
Der Einblickwinkel auf eine LED-Wall hat einen großen Einfluss auf das von einer Kamera aufgenommene Bild bzw. auf die Strahldichteverteilung $L_{e,\lambda}$. Beispielhaft ist in Abbildung 2.5 der Unterschied in *S-Log 2* sichtbar, den der Einblickwinkel macht. Dieser Unterschied hat verschiedene Ursachen, die teils additiv zueinander wirken.

Anordnung der LEDs

In manchen LED-Panels, sind die einzelnen LEDs nicht in der Inline-Anordnung, wie in Abbildung 2.3 zu sehen. Stattdessen sind sie in einer Dreiecks-Anordnung, aus der Überlegung heraus alle Subpixel mit dem gleichen Abstand zu verbauen. So kann es je nach Blickwinkel passieren, dass z.B. die rote LED die blaue LED teils oder sogar ganz verdeckt. Das hat zur Folge, dass das sichtbare Bild deutlich röter erscheint. (Hansen 2010)
Für die hier besprochene Anwendung finden sich allerdings, siehe Kintrup 2022, hauptsächlich LED-Walls mit inline-Anordnung. Wie Schwirten und Chen 2015 verdeutlichen, verringert diese Anordnung blickwinkelabhängige Effekte, besonders im Horizontalen.

Shader und Louver

Um den Schwarzwert eines LED-Moduls zu verbessern, wird der Shader angebracht. Dieser füllt den Raum zwischen den einzelnen RGB-SMD-LEDs aus. Hierdurch wird in den Zwischenräumen so wenig Licht wie möglich reflektiert, sodass der Schwarzwert des Moduls verbessert wird. Der Shader und die allgemeine Beschaffenheit des SMD-Gehäuses können, je nach Bauart, blickwinkelabhängig Bereiche der RGB-SMD-LED verdecken und damit ähnliche Verschiebungen zur Folge haben, wie eine andere Anordnung. (Schwirten und Chen 2015) Es können dezidiert Shader genutzt werden, die diese Effekte möglichst vermeiden.



(a) Louver, einer *inspireLED*

(b) Auswirkung von Louvern auf den Einblickwinkel vertikal

Abbildung 2.6: Auswirkungen von Louvern auf den Einblickwinkel

Eine besondere Bauart bzw. Erweiterung eines Shaders sind sogenannte Louver. Das sind, wie in Abbildung 2.6 (a) zu sehen, kleine über die SMD-Ebene herausragende Abhebungen. Louver werden hauptsätzlich verwendet um von oben kommendes Auflicht zu vermeiden, damit dieses nicht im *Window*, also dem Epoxidharz der einzelnen RGB-SMD-LEDs reflektiert. Das hat allerdings auch einen signifikant schlechteren Einblickwinkel im Vertikalen zur Folge, siehe Abbildung 2.6 (b). (Hansen 2010) Die roten und teilweise grünen LEDs werden, sichtbar durch den Blaustich, von den Louvern verdeckt. Ähnliche aber leichtere Effekte können auftreten, wenn der Shader nicht auf Blickwinkelabhängigkeit optimiert ist und übersteht.

LED-Walls mit Louvern werden hauptsächlich im Outdoor-Bereich verwendet. (Schwirten und Chen 2015)

Bauweise von RGB-SMD-LED

Wie in Abbildung 2.1 *(Funktionsweise der einzelnen LED)* zu sehen, müssen die Photonen zuerst den Halbleiter, ein optisch dichteres Medium, durchqueren. Ein GaN-Halbleiter hat beispielsweise einen Brechungsindex von 2.4. Dies führt nach Snells-Gesetz zu einer bedingten Blickwinkelabhängigkeit der Strahldichte. (Li und Zhang 2019)
Zu erwähnen ist ebenso das Epoxidharz. Dieses optisch dichtere Medium müssen die Photonen durchqueren, bevor sie von einer Kamera aufgenommen werden können. Das Epoxidharz erleidet, wie oben erwähnt, alterungsbedingt eine Trübung und es entstehen Microcracks. Auch von Bedeutung für den Einblickwinkel ist die Beschaffenheit der Fläche des Epoxidharzes *(Window)*, aus der die Photonen austreten. Dieses spielt eine wesentliche Rolle für blickwinkelabhängige Veränderung der Strahldichteverteilung $L_{e,\lambda}$. (Schwirten und Chen 2015) Der Einfluss einer Linse bzw. das Nichtverwenden dieser auf den Einblickwinkel wird von Stiny 2015, S. 138f genauer erläutert. So sei eine LED ohne Linse näherungsweise beschreibbar als Lambertstrahler. Die Lichtstärke I nehme mit dem Kosinus des Abstrahlwinkels ab. Eine Linse in Form einer Halbkugel dagegen hätte keine abstrahlwinkelabhängigen Veränderungen der Lichtstärke I.
Ebenso sind üblicherweise die elektronischen Bauteile auf den LEDs angebracht, sodass diese je nach Blickwinkel die Strahldichte verändern könnten. Die Flip-Chip Technologie bietet hier einen Lösungsansatz, ist allerdings deutlich teurer in der Herstellung. (Prak 2021)

Lösungsansätze

Zusammenfassend lässt sich zur blickwinkelabhängigen Veränderung der Strahldichtenverteilung $L_{e,\lambda}$ sagen, dass hier viele Effekte in Kombination wirken. Einige Konzepte zum Verringern gewisser Ursachen wurden erwähnt, unter anderem die veränderte Anordnung der einzelnen LEDs in einem RGB-SMD-LED Element oder die Möglichkeit einen Shader zu nutzen, der eine möglichst geringe Verdeckung je nach Blickwinkel verursacht und generell auf Louver im Shader zu verzichten. Diese Ansätze haben alle eine Wirkung, verringern die Blickwinkelabhängigkeit aber nicht vollständig.
Gegenwirken lässt sich durch einen gebogenen Aufbau *curved* der LED-Wall, sodass die blickwinkelabhängige Veränderung graduell über mehrere LED-Panels sichtbar wird. Das

ist allerdings häufig nicht möglich auf Grund von Platzmangel, Statik oder dem finanziellem Rahmen.
Außerdem betont der LED-Controller und Receivingcard Hersteller Megapixel VR in einem Whitepaper nochmals die Bedeutung dieser blickwinkelabhängigen Effekte und dass er diese mit seiner noch unveröffentlichten Software in Echtzeit korrigieren könne. (Hochman und Harrison 2021, S.10) Das ginge folglich für einen Betrachtungswinkel pro Zeitintervall. Zum Zeitpunkt des Verfassens dieser Arbeit ist hierzu nichts Weiteres bekannt.

2.4 Spektrometer als Messgerät

Im Rahmen dieser Arbeit wurde das verwendete LED-Panel mit einem Spektrometer über verschiedene Blickwinkel vermessen, um konkrete Aussagen über die Veränderung der Strahldichteverteilung $L_{e,\lambda}$ zu treffen.
Ein Spektrometer tastet grundlegend die spektralen Verteilung ab und ist die direkteste und vollständigste Methode zur Erfassung von Farbinformationen, da hier die Strahldichtenverteilung $L_{e,\lambda}$ in abgetasteter Form erhalten bleibt. Das Licht wird durch ein Objektiv auf ein dispersives Element gebündelt, welches es in seine spektralen Anteile zerlegt. Diese spektralen Anteile werden abgetastet und mit einem oder mehreren Sensoren aufgezeichnet. Meist wird ein Beugungsgitter als dispersives Element verwendet, da es eine nahezu lineare Beziehung zwischen Wellenlänge und Verschiebung in der Ebene der Sensoren herstellt. Bei einem optischen Prisma als dispersives Element ist diese Beziehung stark nichtlinear. (G. Sharma und Trussell 1997) Häufig wird ein CCD-Sensor genutzt, um die zerlegten Wellenlängen auszulesen. Die Werte des Bilds sind relative Werte des Spektrums, da die Pixel den entsprechenden Wellenlängen noch zugeordnet werden müssen. Um ein absolutes Spektrum auszugeben, muss das Spektrometer kalibriert werden, indem Lichtquellen mit bekannten Spektren ausgemessen werden. (Löffler-Mang 2011) Das in dieser Arbeit verwendete Spektrometer *PR-670* nimmt zwischen 380 und 780nm die Strahldichte L_e pro Wellenlängenabschnitt λ_0, λ_1 auf. Das Integral aus dieser Funktion bildet die gesamte Strahldichte L_e. Multipliziert mit der V-lambda Kurve ergibt das Integral die Leuchtdichte L_V. Ebenso können mit den oben beschriebenen *CTFs* die Werte für XYZ und alle daraus folgenden digitalen Farbwerte berechnet werden. (Photo Research 2019)

2.5 Farbmanagement: LED-Walls und Kamera

Die RGB Werte, die eine Kamera roh aufnimmt, sind immer in einem geräteabhänigen Farbraum verortet. Dieser hängt unmittelbar mit den Charakteristiken der Farbfilter zusammen. (Westland, Ripamonti und Cheung 2012b) Das hat zur Folge, dass die drei numerischen Werte der Farbinformation und der tatsächliche Farbwert nicht so zusammenhängen, dass sie auf einem Anzeigegerät für einen identischen Farbeindruck sorgen. Aus diesem Grund müssen die Werte aus dem geräteabhängigen Farbraum in einen geräteunabhängigen Farbraum mit festgelegten Primaries, z.B. *CIE XYZ* oder *ITU-R BT.709-3 bzw. Rec. 709*, transformiert werden. Es muss ein Zusammenhang zwischen den Werten des geräteabhängigen und den dazugehörigen Werten eines geräteunabhängigen Farbraums hergestellt werden. Dieser Zusammenhang ergibt die Charakterisierung der Kamera und deren Anwendung ist die Farbraumtransformation. (Cheung u. a. 2004)
Bei in der Kamera vorverarbeiteten Video- bzw. Bilddaten, transformiert die Kamera selbst das Bild in einen geräteunabhängigen Farbraum, meist *Rec.709 oder sRGB.* (Westland, Ripamonti und Cheung 2012a)
Zu beachten ist, dass das Display selbst auch eine Charakterisierung hat. Der geräteabhängige Farbraum des Displays mit RGB *(Werte der Intensität pro Primary)* muss bei der Ausgabe eines Bildes ebenso beachtet werden. In diesem Fall muss die Transformation von einem geräteunabhängigen Farbraum hin zum geräteabhängigen bekannt sein. Diese Transformation ist unter anderem im *ICC*-Profil hinterlegt (Westland, Ripamonti und Cheung 2012c) oder bei einer LED-Wall in den Konfigurationsdaten für die einzelnen Panels. (Kintrup 2022)

2.5.1 Farbraum-Transformation

Nach Green und MacDonald 2011 gibt es grundlegend drei Methoden, um Charakterisierungs-Abbildungen bzw. Farbraumtransformationen umzusetzen: physikalische Modelle, Look-Up-Tabellen und numerische Methoden. In den meisten praktischen Anwendungen werden für die Charakterisierung von Kamerasystemen entweder Look-Up-Tabellen *(LUTs)* oder numerische Methoden verwendet. (Cheung u. a. 2004) In dieser Arbeit soll hauptsächlich die numerische Methode zum Einsatz kommen, also die Nutzung einer Korrektur-Matrix.

Verwendung von LUTs

Bei der Verwendung von Look-up-Tabellen wird eine gewisse Anzahl an Kamera-RGB-Werten und den dazugehörigen entsprechenden Werten eines geräteunabhängigen Farbraums, z.B. *CIE XYZ*, ermittelt. Diese Sammlung der Wertepaare bildet die Definition einer Abbildung Form in einer Tabelle - der Look-Up-Table *LUT*. Die Werte für RGB, die nicht exakt in dieser Tabelle vorhanden sind, müssen nach einem definierten Algorithmus interpoliert werden. (Cheung u. a. 2004)
Es gibt zwei Arten von LUTs. Die erste ist die 1D-LUT. Diese bildet von einem Eingabewert auf einen Ausgabewert ab, $\mathbb{R} \to \mathbb{R}$. Damit sind Funktionen wie eine Helligkeitskorrektur-Funktion, Gamma-Funktion oder auch komplexere Kontrastfunktionen abbildbar. Jedoch keine Farbraumtransformation bei dem jeder Wert von z.B. XYZ von allen Werten RGB in verschieden Verhältnissen abhängt. Damit ist jeder Kanal einzeln modifizierbar. Die zweite Art ist die 3D-LUT. Diese bildet von drei Eingabewerten auf drei Ausgabewerte ab, $\mathbb{R}^3 \to \mathbb{R}^3$. Die Mehrheit der Farb-Operationen lässt sich mit dieser Methode abbilden, vor allem auch Farbraumtransformationen. (Selan 2012)

Verwendung einer Korrektur-Matrix

Unter der Bedingung, dass beide Werte C_{cam}, C im linearen Arbeitsraum vorliegen, gilt für eine Abbildung von $C \to C_{cam}$ mit $C_{cam}, C \in \mathbb{R}^3$ Folgendes: (Cheung u. a. 2004)

$$C = M \cdot C_{cam}$$

da sich der Zusammenhang mit $C = (X, Y, Z)$ und $C_{cam} = (R, G, B)$ auch folgend beschreiben lässt:

$$X = m_{1,1} \cdot R + m_{1,2} \cdot G + m_{1,3} \cdot B$$
$$Y = m_{2,1} \cdot R + m_{2,2} \cdot G + m_{2,3} \cdot B$$
$$Z = m_{3,1} \cdot R + m_{3,2} \cdot G + m_{3,3} \cdot B$$

Hieraus ergibt sich: Es müssen für mindestens drei Wertepaare die Werte in beiden Farbräumen bekannt sein, um die Gleichung deterministisch lösen zu können. Falls mehr als drei Wertepaare bekannt sind, werden für die Matrix M die Werte genutzt, die für alle

Wertepaare den kleinsten-quadrierten-summierten Fehler *least squares* ergeben.
Sofern die Werte im linearen Arbeitsraum vorliegen, kann die Transformation zwischen zwei Farbräume allgemeingültig mit Hilfe einer Matrixmultiplikation ausgedrückt werden. Die Inverse der Matrix M, M^{-1} ist folglich für die inverse Transformation nutzbar: $C_{cam} = M^{-1} \cdot C$ (Westland, Ripamonti und Cheung 2012a)

2.5.2 Bekannte Methoden: In-Camera-Kalibrierung der LED-Wall

Mit den oben genannten Methoden lässt sich ein Kamerabild so charakterisieren und transformieren, dass die Testwerte den Referenzwerten entsprechen. Folglich wird ein Bild so korrigiert, dass aufgenommene Farbwerte den tatsächlichen Farbwerten entsprechen.
In Kombination mit einer abgefilmten LED-Wall jedoch muss das aufgenommene Bild einer Kamera in den Bereichen anders korrigiert werden, in denen die abgefilmte LED-Wall zu sehen ist, als in den Bereichen, in denen realen Objekte zu sehen sind. Aus diesem Grund soll das an die LED-Wall zur Ausgabe gegebene Bild so modifiziert bzw. kompensiert werden, dass es durch die Kamera dem Originalbild entspricht. (James u. a. 2021)
In folgendem Abschnitt werden zwei Methoden betrachtet, die diese Transformation des auszugebenden Bilds durchführen. Zum Zeitpunk des Verfassens dieser Arbeit sind weder weitere Methoden, noch zusätzliche Ausführungen der erläuterten Methoden bekannt.

James u. a. 2021 und Buchholz und Raisch 2021, Matrixbasiert

Grundlegend beschrieben James u. a. 2021 einen geschlossenen Kreis zwischen dem Eingabebild der Kamera und dem Ausgabebild des Computers. In ihrer Methode werden das auszugebende Bild und das eingehende Bild in den linearen Arbeitsraum und den gleichen Farbraum transformiert. Dafür wird beim Eingabebild die Gammakodierung der Kamera invertiert und das Bild in den genutzten Farbraum transformiert.
Für eine große Anzahl an Referenzdaten, die auf der LED-Wall angezeigt werden, werden die entsprechenden Testdaten aus der Kamera zwischengespeichert. Aus diesen Messungen, also den Wertepaaren errechnen sie eine Korrektur-Operation. Diese wird beschrieben als 3×3-Matrix und drei 1D-LUTs zusätzlich zur erforderlichen Gamma-Funktion für das Display. Eventuell auftretende Artefakte und Fehler werden in einer *smooth 3D-LUT* korrigiert, die im letzten Schritt angewendet wird. (James u. a. 2021) Genauere Informationen

zur Errechnung dieser Operationen sind nicht gegeben.
Einen ähnlichen Ansatz wählen Buchholz und Raisch 2021. Hier wird ein *color-checker*, ein Testpattern mit bekannten Werten in geräteunabhängigen Farbräumen auf der LED-Wall ausgegeben und aufgenommen. Dieses aufgenommene Bild behandeln sie wie ein Kamerabild, welches korrigiert werden muss. Mit Hilfe des Gizmos *MMColorTarget* wurde in *(Nuke, 2021)* eine Korrekturmatrix berechnet und in der Ausgabe-Transformation des Displays im Zuspielprogramm angewendet. (Buchholz und Raisch 2021)

Methode: Disguise Systems Limited, LUT basiert

Eine andere Herangehensweise wählt disguise 2021. In der proprietären Software *d3 r20.0.3* ist für die Anwendung in XR-Produktionen eine Farbkorrektur-Messung implementiert. Hierbei werden n Farbsamples über die einzelnen LED-Walls ausgespielt und durch die Kamera aufgenommen. Aus diesen Messungen wird eine 3D-LUT pro LED-Wall errechnet. Diese LUT wird je LED-Wall als Transformation im Ausgang an die jeweilige LED-Wall angewendet. (disguise 2021) Genaue Informationen zur Funktionsweise sind hier ebenfalls nicht öffentlich verfügbar.

3 Methode zur Farbkorrektur einer LED-Wall

Aus dem vorherigen Kapitel zeigt sich, dass die Farbkorrektur eines anzuzeigenden Bilds für die LED-Wall ein notwendiger Bestandteil In-Camera-VFX Workflows ist. Sie ist ebenso essentiell in jedem weiteren Szenario, in dem eine LED-Wall durch eine Kamera aufgenommen wird. Des Weiteren zeigt sich, dass zum Zeitpunkt der Erstellung dieser Arbeit keine Methode publiziert war, die sich in diesem Zusammenhang mit dem Problem der blickwinkelabhängigen Veränderung der Strahldichte-Verteilung L_e pro Wellenlänge λ einer LED-Wall befasst.

3.1 Ziel der Methode

Die folgend ausformulierte Methode versucht den von (James u. a. 2021) vorgeschlagenen Workflow quelloffen nachzubilden. Die benötigten Bestandteile und Konzepte sollen nachvollziehbar beschrieben werden und durch in Python 3.9.7 2021 geschriebenen Code in die Praxis implementierbar sein. Des Weiteren wird der nachvollzogene Workflow um einen Vorschlag zur blickwinkelabhängigen Interpolation zwischen eingemessenen Korrekturen erweitert.

3.2 Übersicht: Workflow LED-Wall In-Camera Kalibrierung

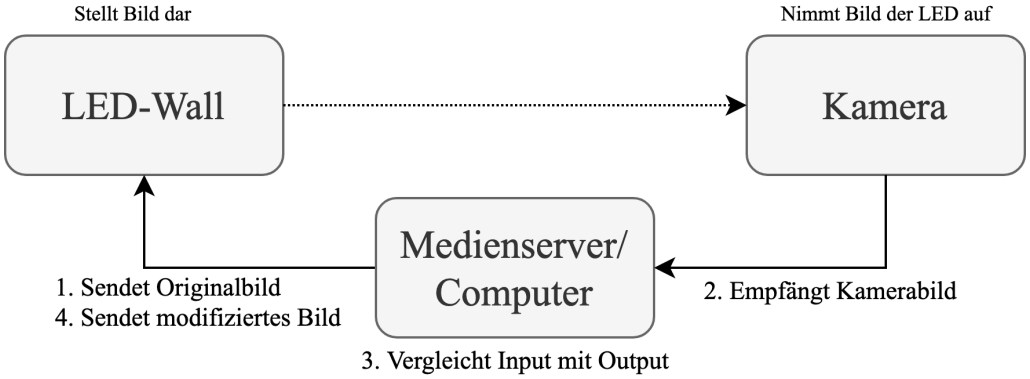


Abbildung 3.1: Workflow der In-Camera Kalibrierung

Elementare Bestandteile dieses Workflows sind hier vier Schritte:

Erstens: Der Medienserver bzw. Computer sendet das Originalbild $I_{original}$ dem darstellenden Medium, einer LED-Wall, die das Originalbild ausspielt.

Zweitens: Die Kamera reicht als aufnehmendes Medium das Kamerabild I_{cam} weiter an den Computer.

Drittens: Der Computer vergleicht den Input I_{cam} mit dem gesendeten Originalbild $I_{original}$. Aus den gesammelten Werten für I_{cam} ergibt sich die Matrix M_T der Testsamples und für $I_{original}$, die Matrix M_R der Referenzsamples. Hieraus wird eine, näherungsweise, allgemeingültige Korrekturmatrix M_{CC} für genau den Aufbau bzw. die Umgebung berechnet, in der diese Messungen stattgefunden hat.

Viertens: Das darzustellende Bild $I_{original}$ wird korrigiert bzw. kompensiert zu I_{mod}. An das darstellende Medium wird nun das modifiziertes Bild I_{mod} gesendet. Das neu aufgenommene Bild $I_{cam,neu}$ stimmt nun näherungsweise mit dem Originalbild $I_{original}$ überein. Warum in diesem Fall das darzustellende Bild korrigiert wird und nicht das aufgenommene Bild der Kamera, hat folgenden Grund: In den meisten Anwendungsfällen wird nicht nur LED-Walls abgefilmt, sondern auch davor positionierte Objekte, würde man Teile des Kamerabilds fälschlicherweise gleich farbkorrigieren, wie die LED-Wall. So würde der Bereich des Kamerabilds mit LED-Wall nahezu dem originalen Bild entsprechen, aber zum Beispiel Hauttöne davorstehender Schauspieler verfälscht werden. (James u. a. 2021)

Diese Schritte werden für mehrere Blickwinkel α wiederholt, sodass die Korrektur des Outputs aus den gesammelten Korrekturen je nach Blickwinkel α interpoliert werden kann.

3.2.1 Bestandteile des Workflows

Für ein besseres Verständnis der Korrekturoperation innerhalb des Medienservers bzw. Computers, sollten die Bestandteile des Gesamtsystems und deren Einfluss auf die Bilddaten bekannt sein.
Im Folgenden werden die drei Haupt-Hardware-Komponenten kurz umrissen.

1. Bestandteil: LED-Controller, LED-Wall

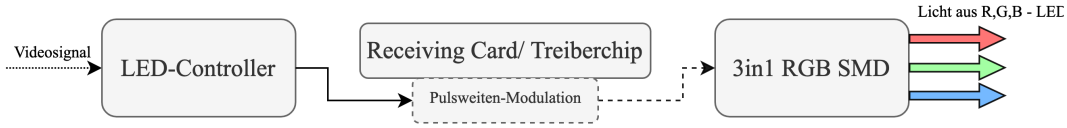


Abbildung 3.2: Schematische Darstellung einer LED-Wall: Videosignal zu Lichtstrahlen

Der LED-Controller nimmt die anliegenden Bilddaten, in diesem Fall vom Medienserver/ Computer, an und reicht diese über ein proprietäres Protokoll an die LED-Cabinets bzw. deren Receiving Card weiter und erreicht die Treiberchips der einzelnen LED-Module. Innerhalb dieses Prozesses werden die Bilddaten entsprechend des eingestellten Farbraums bzw. Gammas dekodiert, sodass die einzelnen RGB-SMD-LEDs durch Pulsweiten-Modulation angesteuert werden. (Kintrup 2022)
Für die Anwendbarkeit einer Matrixkorrektur nach (Wolf 2003) bietet die lineare Darstellung der Bildwerte durch die Pulsweiten-Modulation der einzelnen LEDs die Grundlage. Erheblich für die spätere Farbkorrektur ist die Wandlung in den betriebseigenen Farbraum, bedingt durch die Einstellungen des LED-Controllers und die Bauart der RGB-SMD-LED. Diese Leuchtdioden emittieren auf Grund ihrer Bauart nur in einem schmalen Wellenlängen-Band Strahlenergie Q. Eine genaue Ausführung dazu findet sich in Kapitel 4. Diese Bänder variieren je nach eingesetzten Komponenten und entsprechen so gut wie nie den Wellenlängen-Bändern der Farbfilter in der Kamera.

2. Bestandteil: Kamera

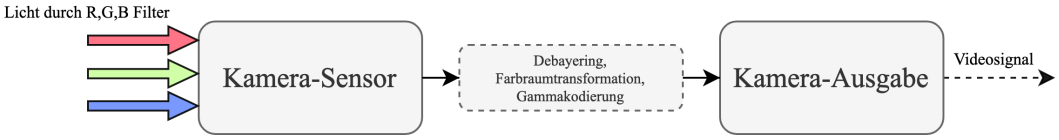


Abbildung 3.3: Schematische Darstellung einer Kamera: Lichtstrahlung zu Videosignal

Grundlegend wandelt die Kamera die Strahldichte L_e pro Farbfilterband in ein elektrisches Signal (Schmidt 2013, S. 354f). Aus der Spektralen-Verteilung der Lichtstrahlen werden durch Farbfilter die Rot-, Grün- und Blau-Anteile des dargestellten Bilds der LED-Wand aufgenommen und gewandelt. Je nach verwendeter Kamera kann die Charakteristik der Filter variieren. (Schmidt 2013, S. 379f)
Dieser Sachverhalt und der betriebsbedingte Farbraum der LED-Wall führen in Kombination zu dem beobachteten Unterschied zwischen dem originalem Bild $I_{original}$ und dem aufgenommenen Bild I_{cam}. Das linear mit der Strahldichte zusammenhängende elektrische Signal pro Farbkanal wird bevor es ausgegeben wird weiter verarbeitet. (Schmidt 2013, S.405ff)
Für den Korrekturalgorithmus (siehe Abschnitt 3.3.3) besonders relevant ist die Gamma-Kodierung und der angegebene Farbraum.

3. Bestandteil: Zentraler Medienserver/ Computer

Abbildung 3.4: Schematische Darstellung des Computers-Abschnitts, der Methode zur Farbkorrektur

Der Medienserver bzw. Computer ist in diesem Workflow das Herzstück, wie von (James u. a. 2021) vorgeschlagen. Essentiell ist, dass die Bilddaten am Output $I_{original,gamma}$, $I_{mod,gamma}$ und Input $I_{cam,gamma}$ Gamma-kodiert vorliegen. Um eine blickwinkelabhängige Korrektur, zu realisieren, muss zusätzlich der Blickwinkel als Input-Parameter vorliegen.
Im Medienserver bzw. Computer müssen die Output- und Input-Bilder im linearen Ar-

beitsraum vorliegen. Hierdurch kann der Unterschied für die zu korrigierenden Blickwinkel im ersten Schritt aufgenommen und im zweiten die Korrektur berechnet werden. Im dritten Schritt wird dann diese Korrektur je nach Blickwinkel α interpoliert und angewendet. Hier wird das modifizierte Bild I_{mod} für den Blickwinkel α berechnet und dargestellt. sodass der Input I_{cam} dem Originalbild $I_{original}$ entspricht.

3.3 Komponenten einer blickwinkelabhängigen Farbkorrektur

Im Folgenden wird der Workflow-Abschnitt zwischen Computer-Input und -Output genauer betrachtet. Der Algorithmus soll die von James u. a. 2021 genutzte Methode in Grundzügen nachbilden, sie im Detail nachvollziehbar machen und um ein Modell zur blickwinkelabhängigen Korrektur erweitern. Schlussendlich soll jedes beliebige Originalbild, mithilfe des Algorithmus' so auf der LED-Wall dargestellt werden, dass das aufgenommene Bild der Kamera dem Originalbild entspricht, siehe Abbildung 3.5.

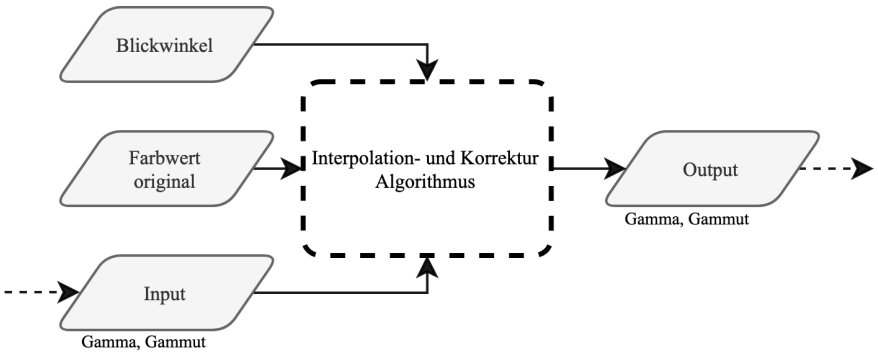


Abbildung 3.5: Schematische Darstellung des Algorithmus' zur blickwinkelabhängigen Farbkorrektur des Outputs

Wie in *Workflow In-Camera Kalibrierung* dargestellt (Abbildung 3.1), ist der Korrektur-Algorithmus zwischen Input (aufgenommenes Bild der Kamera) und Output (gesendetes Bild an die LED-Wall) zu verorten. Die Eingabeparameter bzw. Ausgabewerte des Algorithmus' sind:

- *Angle* Blickwinkel α
- *Color original* Originalbild $I_{original}$

- *Input* aufgenommenes Bild der Kamera I_{cam}
- *Output* Ausgabe-Bild an die LED-Wall I_{out}

Grundlegend lässt sich der Algorithmus in zwei zeitlich voneinander getrennte Teilbereiche aufteilen. Zu Beginn muss mithilfe des Unterschieds zwischen den dargestellten Originalbildern $I_{original}$ und den aufgenommenen Bildern I_{cam} eine mögliche Korrektur berechnet werden. Im zweiten Schritt wird diese Korrektur auf das darzustellende Bild angewendet. Die LED-Wall stellt das Bild $I_{out} = I_{mod}$ dar. Dieses wird durch die Kamera aufgenommen I_{cam} und mit dem Originalbild $I_{original}$ verglichen.

Für die Implementierung dieser Methode werden keine Testpattern mit unterschiedlichen Farbwerten verwendet. Stattdessen wird pro Messung ein Farbwert $c \in \mathbb{R}^3$ vollflächig auf jedem Pixel der LED-Wall ausgegeben. Folglich gilt näherungsweise für die Berechnungen:

$$I_{original}, I_{cam}, I_{out}, I_{mod} \in \mathbb{R}^3$$

Da der Datentyp *float*, besonders aber der Input und Output des Systems in ihrer Bit-Tiefe beschränkt sind, gilt das oben Genannte näherungsweise.

3.3.1 Farbpatch-Ausgabe

Abbildung 3.6: Schematische Darstellung der Verarbeitung von Output-Bilddaten

Um einen Farbpatch auszugeben, wird der im linearen Arbeitsraum vorliegende dreidimensionale Vektor I_{out} Gamma-kodiert und in ein TIFF-Bildformat gebracht. Auf die Gammakodierung der Bilddaten wird später in Abschnitt 3.3.3 *Linearisierung* genauer eingegangen.
In diesem Algorithmus wird aus dem dreidimensionalen Vektor $I_{out,gamma}$ eine 16bit tiefe TIFF-Datei, mit Werten zwischen 0 und 65.535, in Höhe und Breite der LED-Wall gespeichert (Implementierung siehe *Algorithmus 3*).
Diese Bilddatei ist das Gamma-kodierte Videosignals, welches an die LED-Wall gesendet wird.

3.3.2 Kamera-Bilddaten

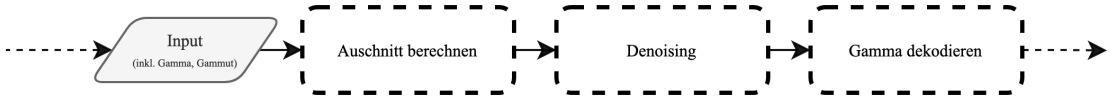


Abbildung 3.7: Schematische Darstellung der Vorverarbeitung der Input-Bilddaten

In diesem Abschnitt soll genauer auf die Verarbeitung der Bilddaten der Kamera eingegangen werden. Wie muss ein Bild I_{cam} vorverarbeitet sein, dass es für die Korrekturberechnung, genutzt werden kann? Die genaue Art und Weise der Korrekturberechnung wird später in Abschnitt 3.3.4 beschrieben,
Die relevanten Parameter sind der Input der Kamera I_{cam} und die dazugehörigen Informationen zu Gamma-Kodierung und Farbraum.

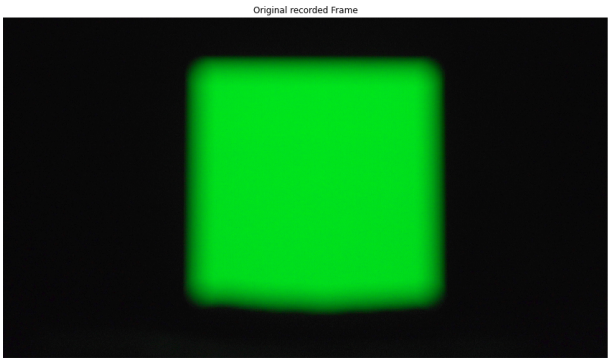
Relevanten Bereich ausschneiden

Aus dem eingehenden Videosignal wird der relevante Teil aus der Bildmitte herausgeschnitten und für den weiteren Prozess weitergereicht. Das hat mehrere Gründe. Zum einen mag es sein, dass der relevante Bereich für eine Messung generell kleiner ist als die Bildgröße. Zum Beispiel, wenn nur ein LED-Panel für die Messung genutzt wird und ein gewisser Abstand bestehen und eine spezielle Brennweite für artefaktfreie Aufnahmen eingestellt werden muss.
Zum Anderen entstehen in den Randbereichen eines Bildes einer Kamera mit Objektiv zunehmend Effekte wie Vignettierung (Lee u. a. 2017), aber auch sphärische Aberration sowie chromatische Aberration (Hasche und Ingwer 2016). Aus diesen Gründen ist es ebenfalls sinnvoll, einen möglichst nah am Bildmittelpunkt liegenden Bereich auszuschneiden, der jedoch noch genug Bildinformaion für ein Denoising enthält (siehe Abschnitt 3.3.2 *Denoising*).

Dieses Verfahren wird wie folgt implementiert. Das eingehende Videosignal liegt als vier- oder dreidimensionales *ndarray* (numpy 2021) vor. Der in *Algorithmus 4* beschriebenen Funktion wird das Video gereicht, sowie die Höhe und Breite des gewünschten Ausschnitts. Aus dem Video wird von der Bildmitte heraus ein Ausschnitt in gewählter Höhe und Breite ausgestanzt. Rückgabewert ist dieser Ausschnitt in Form eines vierdimensionalen *ndarray* (numpy 2021).

Denoising Methode

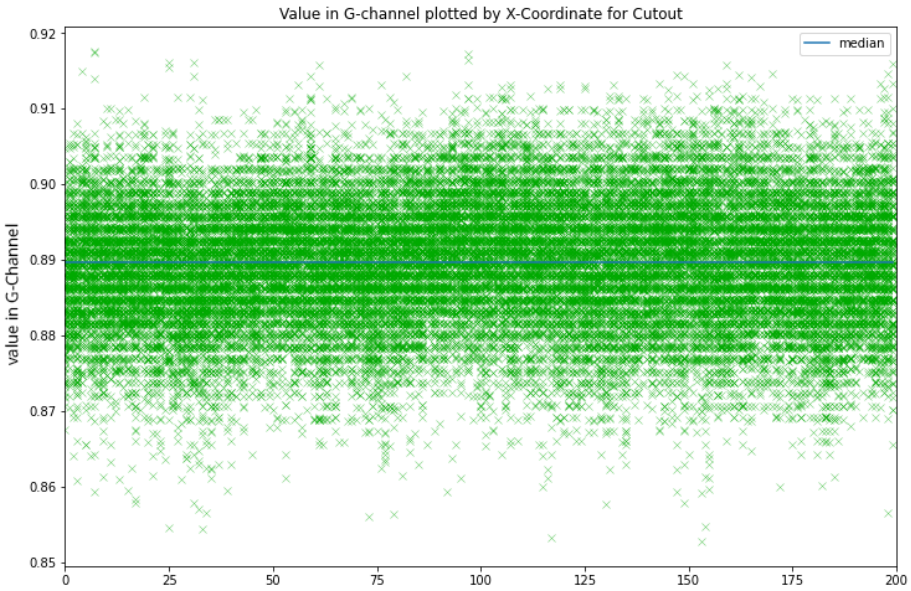
Es sei näherungsweise $I_{cam} \in \mathbb{R}^3$ für die mathematische Grundlage der Verarbeitung.
Das Video der Kamera enthält, abgesehen von Vignettierungs-Effekten, ein temporales und spatiales Rauschen um den relevanten Wert I_{cam}. (Schmidt 2013, S. 373f)
Ein robuster Algorithmus, um Signalrauschen herauszufiltern, ist der Median. Es werden alle Werte einer Sammlung sortiert und der Wert, der sich in der Mitte befindet entspricht dem Median. (Huang, Yang und Tang 1979)
Somit haben im Gegensatz zum *Mean*, dem Durchschnitt, vereinzelte stark abweichende Werte weniger Einfluss auf das Gesamtergebnis. Diese Eigenschaft qualifiziert den Median für diese Anwendung.
In *Algorithmus 5* wird aus der Eingabe eines drei- bzw. vierdimensionalen *ndarray* (numpy 2021) der Median über zwei oder drei Achsen berechnet. Bei sehr stark rauschenden Kameras kann zusätzlich über mehrere Einzelbilder eines Videoausschnitts gemittelt werden. Der Rückgabewert ist ein eindimensionales *ndarray* (numpy 2021) mit drei Werten, das I_{cam} darstellt.
Dieser Workflow von Input eines Videosignals hin zu I_{cam} ist in Abbildung 3.8 nochmals visualisiert.



(a) Aufgenommenes Standbild der Kamera, Gamma-kodiert mit *S-log 2*

(b) Ausschnitt 200px (links) und visualisierter Medianwert I_{cam} (rechts)[a]

[a]Wurde zur Visualisierung des Rauschens mit Gamma 1.5 exportiert



(c) Verteilung der Werte des G-Kanals pro Bildspalte, errechneter Median des Gesamtausschnitts

Abbildung 3.8: Beispielhafte Visualisierung: Vom aufgenommenem Bild bis zum Farbwert I_{cam}

3.3.3 Linearisierung von Ausgabe- und Eingabe-Bilddaten

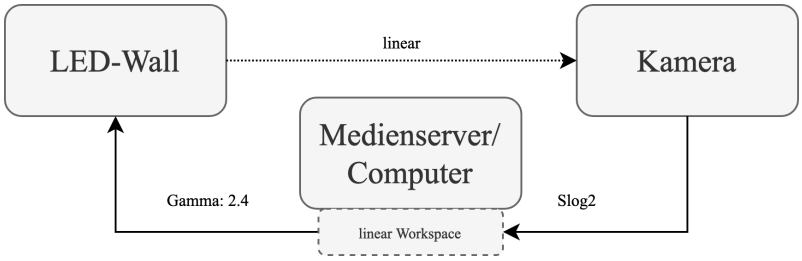


Abbildung 3.9: Schematische Darstellung: Linearer Workflow des Gesamtsystems

In Abschnitt 3.2.1 *Bestandteile: LED-Wall und Kamera* wurde dargestellt, wie der lineare Zusammenhang zwischen dem dargestellten Bild der LED-Wall und dem aufgenommenen Bild der Kamera besteht. In Abbildung 3.9 ist dieser lineare Zusammenhang zwischen den Ausgabebilddaten und Eingabebilddaten grob skizziert.
Um die später in Abschnitt 3.3.4 *Korrekturberechnung* aufgezeigte Korrektur mithilfe von Matrizen zu ermöglichen, muss ein linearer Zusammenhang zwischen Input und Output gegeben sein. (Westland, Ripamonti und Cheung 2012b)
Das bedeutet, dass die beiden Werte I_{out} und I_{cam} jeweils zur Verarbeitung im Bereich des Medienservers/ Computers im linearen Arbeitsraum vorliegen müssen. Somit ergibt sich, dass eine Transformation von I_{out} zu $I_{out,gamma}$ und von $I_{cam,gamma}$ zu I_{cam} definiert werden muss.

Linear-Workflow: Input

Um $g : I_{cam,gamma} \rightarrow I_{cam}$ aufzustellen, müssen die EOCF (electronic-optical-conversion-function), bzw. deren Inverse, die OECF (optical-electronic-conversion-function) der Kamera bekannt sein. Beachtet werden muss das bei der Einrichtung der Kamera. Beides wird in dieser Arbeit auch Gamma-Kodierung bzw. -Dekodierung genannt. Im Falle von *S-Log 2* als EOCF ist g die dekodierende Funktion von *S-log2*.
In der Bibliothek (colour 2021) gibt es für die Kodierung und Dekodierung verschiedener Gammafunktionen entsprechende Module. In diesem Fall wird $I_{cam,gamma}$ als S-log2-kodiert angenommen (siehe *Algorithmus 6*).

Linear-Workflow: Output

Um $h : I_{out} \rightarrow I_{out,gamma}$ zu definieren, muss bekannt sein, mit welcher OECF der LED-Controller das eingehende Videosignal verarbeitet. Deshalb muss entsprechend der Ein-

stellungen des LED-Controllers die EOCF implementiert werden. Im Fall einer Gamma-Einstellung von 2.4 im LED-Controller gilt für jeden Einzelwert $p\ h : p \rightarrow p^{2.4}$. Umgekehrt gilt mit $i : I_{out,gamma} \rightarrow I_{out}$, $i : p \rightarrow p^{\frac{1}{2.4}}$
Dafür kann ebenfalls die Bibliothek colour 2021 genutzt werden (siehe *Algorithmus 6*).

3.3.4 Kompensation des Ausgabebildes

Für die vereinfachte Darstellung der folgenden Konzepte, sind alle Daten ($I_{original}$, I_{cam}, I_{mod}) als im linearen Arbeitsraum vorliegend anzunehmen. Ferner wird davon ausgegangen, dass $I_{original}$ linear auf I_{cam} abbildet.
Es gilt:

$$I_{cam} = M_{cam} \cdot I_{original} \tag{3.1}$$

mit M_{cam} als 3×3 - Matrix, die die lineare Abbildung $I_{original} \rightarrow I_{cam}$ beschreibt. (Hasche und Ingwer 2016, S. 60f) und (Westland, Ripamonti und Cheung 2012d)

Es wird analog zu (James u. a. 2021) und (Buchholz und Raisch 2021) von Folgendem ausgegangen:

$$I_{cam,mod} = I_{original} = M_{cam} \cdot I_{mod} \tag{3.2}$$

Diese Formel erläutert die von James u. a. 2021 und Buchholz und Raisch 2021 genutzte Methode mathematisch. Hierbei wird das anzuzeigende Bild $I_{original}$, bevor es an die LED-Wall übergeben wird, modifiziert zu I_{mod}. Sodass es nach der Verarbeitung, Transformierung und Anzeige auf der LED-Wall und der Aufnahme, Transformation sowie Verarbeitung der Kamera $I_{cam,mod}$ dem originalen Bild $I_{original}$ entspricht. Damit kompensiert die Korrektur von $I_{original}$ hin zu I_{mod} die Transformationen zwischen Ausgabe- und Eingabe-Bild.

Herleitung: Verwendung einer Korrektur-Matrix für das anzuzeigende Bild

Wie von Westland, Ripamonti und Cheung 2012d beschrieben, lässt sich aus Formel 3.1 mithilfe der inversen Matrix M_{cam}^{-1}, bzw. der Pseudoinversen M_{cam}^{+} bei nicht deterministi-

schen Gleichungssystemen folgende Gleichung aufstellen:

$$I_{original} = M_{cam}^{-1} \cdot I_{cam} \tag{3.3}$$

Mit I_{mod} definiert als:

$$I_{mod} = M_{mod} \cdot I_{original} \tag{3.4}$$

Formel 3.4 in 3.2 ergibt:

$$I_{original} = M_{cam} \cdot (M_{mod} \cdot I_{original}) \tag{3.5}$$

Multipliziert man Formel 3.5 mit der inversen Matrix M_{cam}^{-1} bzw. der Pseudoinversen M_{cam}^{+} von links folgt:

$$M_{cam}^{-1} \cdot I_{original} = M_{mod} \cdot I_{original}$$

$$M_{cam}^{-1} = M_{mod} \tag{3.6}$$

Unter der Bedingung einer linearen Abbildung von $I_{original}$ auf I_{cam} lässt sich schließen, dass die Korrekturmatrix M_{cam}^{-1} bzw. M_{cam}^{+}, im Folgenden M_{CC}, der Matrix zur Modifizierung des Outputs M_{mod} entspricht. Es gilt folglich:

$$M_{CC} = M_{mod}$$

Um M_{CC} zu bestimmen wird eine Referenz-Matrix M_R *(Colors original)* von Werten für $I_{original}$ und eine Test-Matrix M_T *(Colors measured)* mit den entsprechenden Werten für I_{cam} benötigt. Für ein deterministisch lösbares Gleichungssystem mit einer 3×3-Matrix als M_{CC}, sind M_R und M_T ebenso 3×3-Matrizen. Wenn die Anzahl der Vektoren-Paare mehr als drei ist, spricht man von einem über-determinierten Gleichungssystem. Im Falle von Bilddaten ist das oft sinnvoll, da sie meist Rauschen beinhalten. In diesem Fall wird die Matrix gebildet, die den kleinsten quadrierten Fehler *(least square error)* ergibt. (Westland, Ripamonti und Cheung 2012d)

Messung von Referenz- und Testwerten

Am Anfang einer Korrektur steht die Sammlung M_R *(Colors original)* von Werten für $I_{original}$. Diese Werte werden jeweils über die LED-Wall ausgegeben und direkt über die Kamera aufgenommen. Alle Daten werden so verarbeitet, dass I_{cam} bzw. M_T *(Colors measured)* ebenso wie $I_{original}$ bzw. M_R im linearen Arbeitsraum vorliegt. Somit sollte zwischen jedem Wertepaar $I_{original}$ und I_{cam} ein linearer Zusammenhang bestehen.

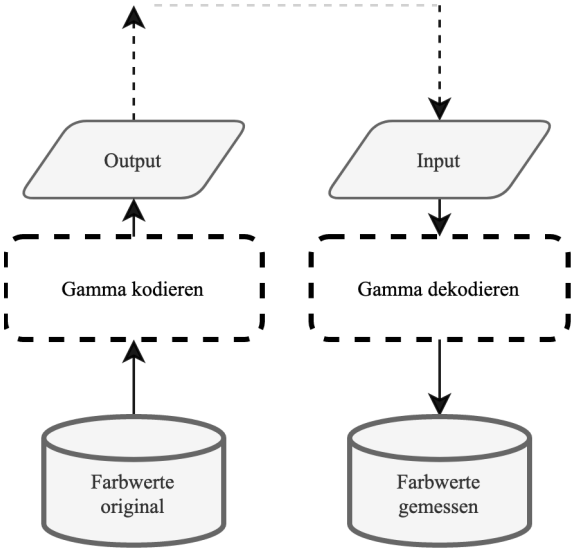


Abbildung 3.10: Schematische Darstellung: Durchführung der Messungen pro $I_{original}$ in M_R *(Colors original)*

Korrektur-Matrix Berechnung

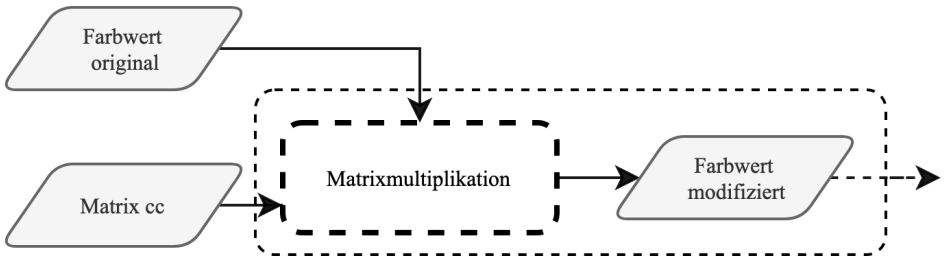


Abbildung 3.11: Schematische Darstellung: Durchführung der Korrektur-Matrix Berechnung

Die Gleichung $I_{original} = M_{CC} \cdot I_{cam}$ gilt und kann mit M_R und M_T wie folgt gelöst werden:

$$M_R = M_{CC} \cdot M_T$$

Die berechnete Matrix M_{CC} wird für die weitere Verwendung zwischengespeichert.

Transformation mit Korrektur-Matrix

Mit M_{CC} kann I_{mod} für jeden beliebigen Wert $I_{original}$ bestimmt werden. Somit können auch nicht eingemessene Farbwerte als $I_{original}$ korrekt angezeigt werden:

$$I_{mod} = M_{CC} \cdot I_{original}$$

Der Wert I_{mod} wird ausgegeben und wieder aufgenommen. Folglich entspricht I_{cam} $I_{original}$, unter der Grundvoraussetzung von Formel 3.1.

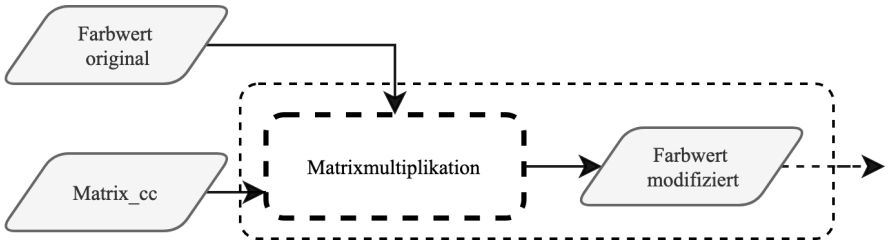


Abbildung 3.12: Schematische Darstellung: Durchführung der Korrektur für I_{mod} *(Color modified)*

Die Korrektur gilt jeweils für den Blickwinkel α, aus dem sie gemessen wurde. Diese Erkenntnis leitet sich daraus ab, dass sich die Strahldichte-Verteilung L_e pro Wellenlänge λ der LED-Wall je nach Blickwinkel bei gleichem Originalbild $I_{original}$ verändert. Genaue Ausführungen zur Veränderung dieser Parameter finden sich in Kapitel 4.

Implementierung in Python

In Python kann die Erstellung der Korrekturmatrix wie folgt implementiert werden:
Da $M_{mod} = M_{CC}$ gilt, muss $M_R = M_{CC} \cdot M_T$ gelöst werden. Das ist möglich mit der Funktion *matrix_colour_correction_Cheung2004(M_T, M_R)* der Bibliothek colour 2021. Aus dem Rückgabewert dieser Funktion M_{CC} kann mithilfe der Bibliothek numpy 2021 mit *linalg.pinv(M)* die inverse Matrix M_{CC}^{-1} bzw. Pseudoinverse gebildet M_{CC}^{+} werden. Rückgabewerte der Funktion *getCorrectionMatrix* sind M_{CC} und $M_{CC}^{-1} = M_{cam}$ (siehe *Algorithmus 7*).
Mit M_{cam} ist es möglich für Debugging-Zwecke eine Voraussage für den aufgenommenen

Wert I_{cam} zu treffen und Werte zu überprüfen.

Die Berechnung des modifizierten Bilds I_{mod} ergibt sich aus der Multiplikation von $I_{original}$ mit der Korrekturmatrix M_{CC} (siehe *Algorithmus 8*).

3.3.5 Blickwinkelabhängige Kompensation des Ausgabe-Bilds

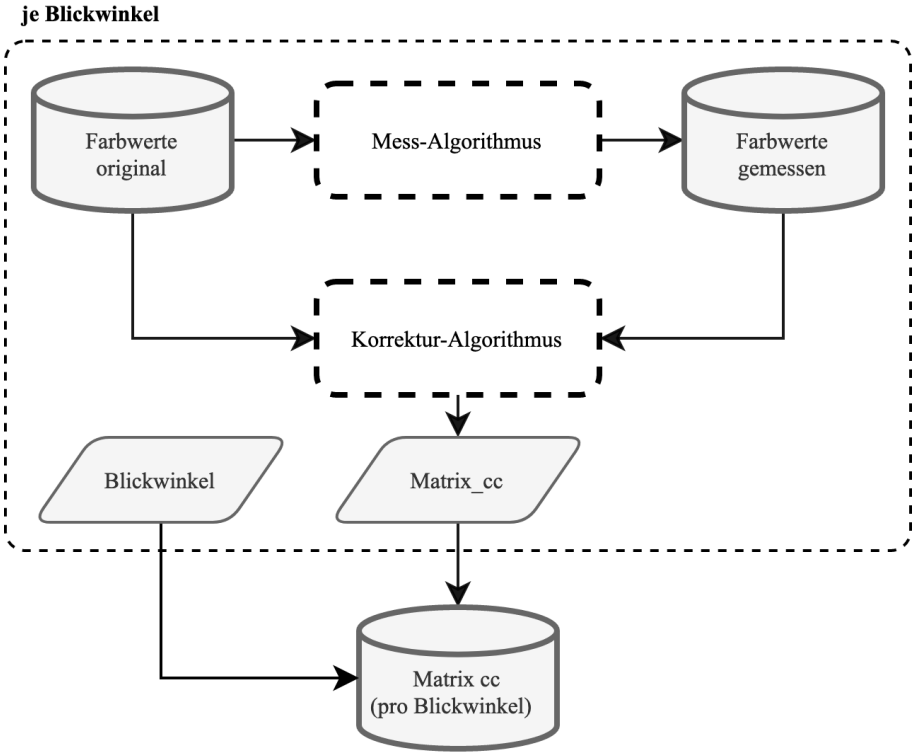


Abbildung 3.13: Schematische Darstellung: Blickwinkelabhängige Messungen und Korrektur-Matrix Berechnungen

Die grundlegende Idee der blickwinkelabhängigen Korrektur ist die Messungen und Korrekturen je einzumessendem Blickwinkel α anzuwenden. Rückgabewerte, bzw. die benötigten Daten um die Korrektur für beliebige Blickwinkel zu interpolieren, sind die Korrektur-Daten $M_{CC,angle}$ und die korrespondierenden Blickwinkel α *(Matrix cc, pro Blickwinkel).*

In einer Schleife wird für jeden Blickwinkel α ein Eintrag angelegt, in den die Korrekturmatrix M_{CC} zwischengespeichert wird. *Algorithmus 9*
Zu Debugging-Zwecken werden zusätzlich die originalen Farbwerte M_R und gemessenen Farbwerte M_T, sowie die jeweilige inverse Matrix bzw. Pseudoverse M_{cam} gespeichert.

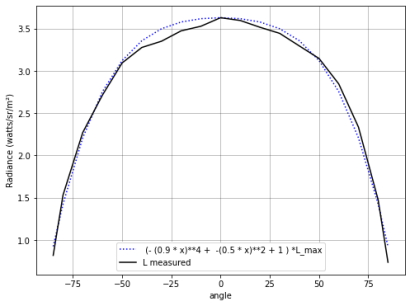
3.3.6 Interpolations Algorithmus

Das Ziel einer blickwinkelabhängigen Korrektur ist, dass mehrere, aber nicht alle möglichen Winkel, vermessen werden müssen. Aus diesem Grund muss zwischen den Korrekturen der eingemessenen Blickwinkel α interpoliert werden. Die Frage die sich hieraus ergibt ist: Wie soll interpoliert werden?

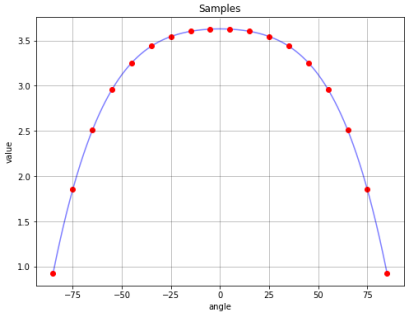
Hier soll beschreiben werden, wie aus der Menge n Korrekturdaten $f(\alpha_1), f(\alpha_2), ..., f(\alpha_n)$ für verschiedene Blickwinkel die Daten herangezogen werden, aus denen interpoliert werden soll. Ebenso auf welche Weise zwischen diesen Daten interpoliert werden soll, mit dem Ergebnis des unbekannten Werts $f(\alpha)$ für α.

k-nächste Nachbarn

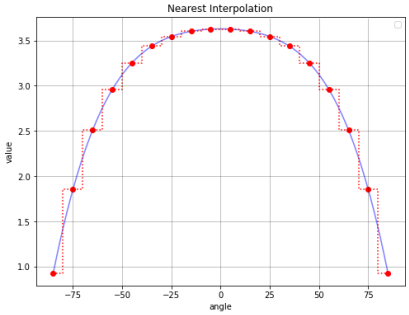
Unabhängig der Methode, die zur Interpolation genutzt wird, werden k Werte $\{f(\alpha_1), f(\alpha_2), ..., f(\alpha_k)\}$ benötigt zwischen denen Interpoliert wird. Es wird der Blickwinkel α genutzt um k-nächste Werte dieses Blickwinkels α in der Sammlung der eingemessenen Blickwinkel *(Matrix cc pro Blickwinkel)* zu finden. Für eine lineare Interpolation reicht $k = 2$, also beide Nachbarwerte des Blickwinkels α (siehe *Algorithmus 10*).



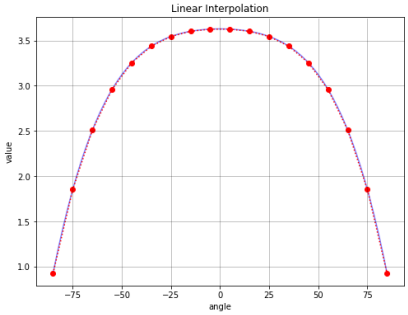
(a) Modellierte Funktion auf gemessene Werte der Strahldichte, siehe Abschnitt 4.5



(b) Simulation gemessener Samples pro 10°



(c) *Nearest* Interpolation



(d) Lineare Inteprolation

Abbildung 3.14: Visualisierung von Interpolations-Methoden mit Strahldichte-Veränderung nach Kapitel 4

Nearest Interpolation

Um ein Resampling, aus der Menge an Datenpunkten der Korrekturen für unbekannte Punkte durchzuführen, wäre eine einfach zu implementierende Methode:
Der Wert des nächsten Nachbarn *nearest neigbor mit* $k = 1$ wird angenommen. Die Interpolations-Methode *Nearest* stimmt allerdings mit der Veränderung der Luminanz über verschiedene Blickwinkel hinweg nicht ausreichend überein (siehe Abbildung 3.14(c)).

Lineare Interpolation

In dieser Methode ist die lineare Interpolation implementiert. Zwischen den zwei nächsten Werten $f(\alpha_0), f(\alpha_1)$ von $f(\alpha)$,aus dem *k-nearest* Algorithmus mit $k = 2$ besteht eine Strecke. Es gilt:

$$f(\alpha) = f(\alpha_0) + \frac{f(\alpha_1) - f(\alpha_0)}{\alpha_1 - \alpha_0}(\alpha - \alpha_0)$$

In *Algorithmus 11* wird die Funktion *interp1d* aus scipy 2021 genutzt. Die eindimensionale Interpolations-Funktion reicht aus, da in dieser Arbeit nur die Veränderung über den Blickwinkel der Y-Achse betrachtet wird.

3.3.7 Kompensation des Ausgabebildes für einen nicht vorher eingemessenen Blickwinkel α

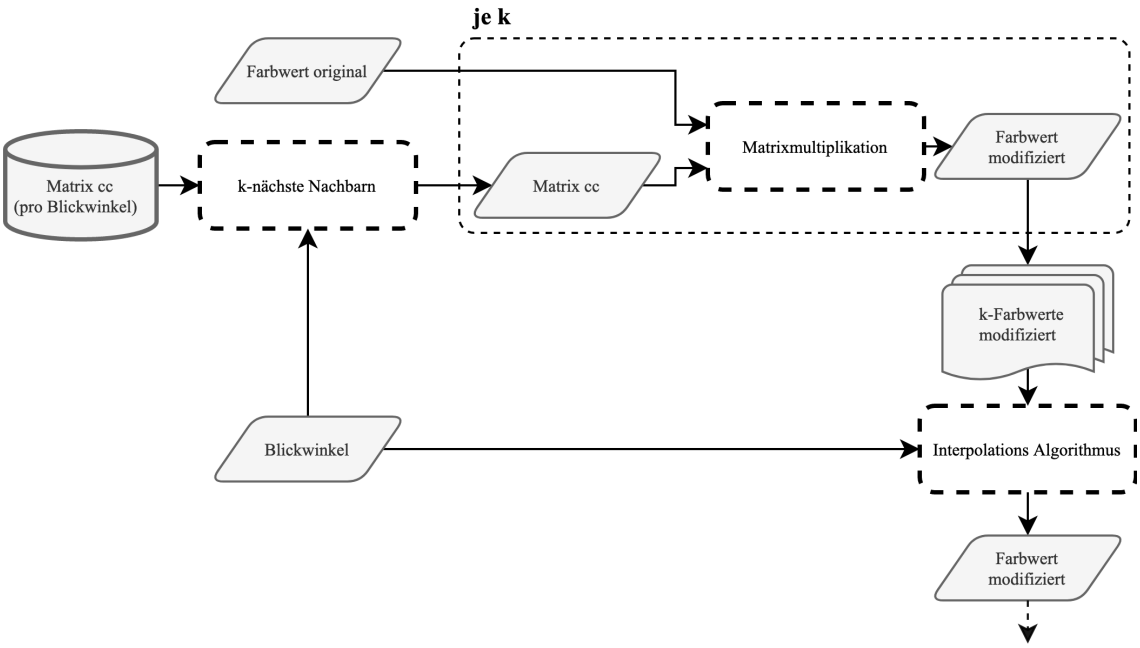


Abbildung 3.15: Gesamtübersicht: Algorithmus zur Korrektur eines Blickwinkel α

Der Ansatz dieser Arbeit, ist die Werte $\{f(\alpha_1), f(\alpha_2), ..., f(\alpha_k)\}$, also k Nachbarn des zu korrigierenden Winkels α zu interpolieren. Das heißt, die bereits korrigierten Werte I_{mod,α_k} des jeweiligen Winkel $\alpha_0, \alpha_1, ..., \alpha_k$ werden dem Interpolations-Algorithmus übergeben. Für jeden Winkel der Rückgabe des *k-nearest Algorithmus 10* wird aus der Sammlung der Messungen die jeweilige Korrekturmatrix für eine Korrektur von $I_{original}$ genutzt. Die resultierenden modifizierten Farbwerte $I_{mod,\alpha_0}, I_{mod,\alpha_1}, ..., I_{mod,\alpha_k}$ werden als Sammlung mit dazugehörigen Winkeln $\alpha_0, \alpha_1, ..., \alpha_k$ zwischengespeichert *Colors modified*.
Sodass der Interpolations-Algorithmus, mit Hilfe des Blickwinkel α, aus dieser Sammlung den Wert für $I_{mod,\alpha}$ interpoliert. *Algorithmus 12*.
Für die lineare Interpolation gilt $k = 2$. Das System ist modular, weil z.B. nur der Interpolations-Algorithmus oder Korrektur-Algorithmus ausgetauscht werden könnte.

3.4 Evaluierung, Verifizierung der Korrektur

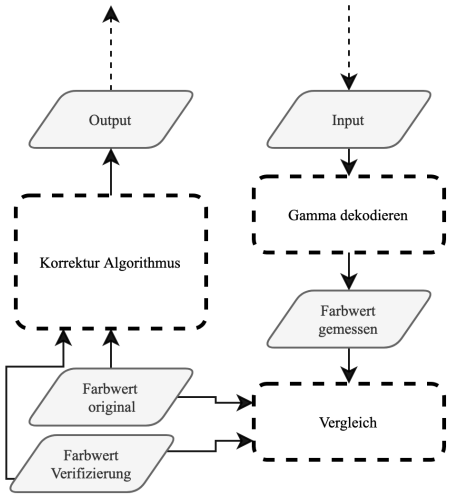


Abbildung 3.16: Schematische Darstellung: Evaluierung einer erfolgten Korrektur

Um nach einer Korrektur durch die Kompensation des Ausgabebildes diese auch bewerten zu können, muss von einem Farbwert $I_{original}$ der modifizierte I_{mod} für diesen Blickwinkel α angezeigt und aufgenommen I_{cam} werden. Danach kann I_{cam} mit $I_{original}$ abgeglichen werden. Mit einer weiteren Sammlung an Verifizierungs-Farbwerten M_{verify}, kann der Algorithmus zusätzlich auf noch unbekannte Ausgabefarbwerte für $I_{original}$ getestet werden. Hiermit ist es möglich eine Aussage über die Allgemeingültigkeit des Algorithmus' zu machen.

Je nach Fehlerfunktion kann aus dem Unterschied von I_{cam} und $I_{original}$ eine Bewertung des Systems erfolgen. Die in dieser Arbeit verwendete Methode ist die absolute Differenz zwischen $I_{original}$ und I_{cam} im linearen Arbeitsraum und Farbraum *ITU BT.709*. Also gilt:

$$Error = |I_{original} - I_{cam}|$$

Daraus kann die Summe über alle absoluten Fehler *SAE - sum of absolute errors* der Messungen n vor und nach der Korrektur gebildet werden.

$$\sum_{i=1}^{n} |I_{original} - I_{cam}|$$

Wenn sich diese beiden SAE-Werte signifikant unterscheiden, lässt sich ableiten, ob die Korrektur eine Verbesserung darstellt.

4 Messungen der blickwinkelabhängigen Veränderung der emittierten Strahlung einer LED-Wall

Im Rahmen dieser Arbeit wurde das verwendete LED-Panel aus verschiedenen Blickwinkeln mit einem Spektrometer vermessen.

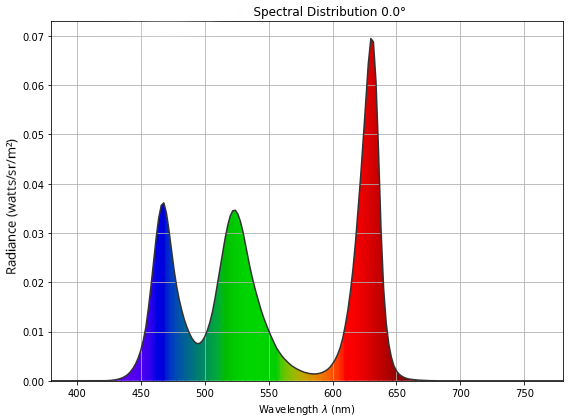


Abbildung 4.1: Strahldichte L_e pro Wellenlänge λ für Blickwinkel $\alpha = 0°$, angezeigt *(1,1,1)*

4.1 Ziel der Messungen

Ziel der Messung ist es, eine Aussage über die Veränderung der Parameter Strahldichte *radiance* L_e in $Wsr^{-1}m^{-2}$ pro Wellenlänge λ in nm über verschiedene Blickwinkels zu treffen. Daraus abgeleitet werden kann die Veränderung der Leuchtdichte *luminance* L_V in $\frac{cd}{m^2}$. Aus der Verteilung der Strahldichte L_e und deren Veränderung lassen sich Rückschlüsse über die Veränderung der angezeigten Farbtöne ziehen.

Basierend auf der Veränderung dieser Parameter, über verschiedene Blickwinkel α, soll ein besseres Verständnis darüber erlangt werden, welche Auswirkungen eine Veränderung des Blickwinkels α auf das aufgenommene Bild einer Kamera haben kann. Daraus lässt sich schließen, welche Parameter korrigiert werden müssen bzw. in welcher Form eine Korrektur möglich ist. Mit dem primären Ziel ein homogenes Bild durch die Kamera über

verschiedene Blickwinkel α hinweg zu erhalten.

Diese Daten sollen eine fundierte Aussage darüber erlauben, ob und warum eine blickwinkelabhängige Korrektur sinnvoll ist und inwiefern diese möglich ist.

4.2 Aufbau - spektrale Messungen über verschiedene Blickwinkel

Für den Versuch wurden folgende Materialien und Software verwendet:

- LED Panel: inspireLED s3.6b Indoor *ICT AG*
- LED Prozessor: inspireLED MCTRL660 *ICT AG* und *Xi'an NovaStar Tech Co., Ltd.*
- LED Controller-Software: (NovaLCT 5.4.3 2021)
- Computer: MacBook Pro 14“ 2021, MacOS 12.1 (21C52) *Apple Inc.*
- Zuspielung: (Qlab 4.6.11 2021)
- Spektrometer: SpectraScan® PR-670 *Photo Research - JADAK a Novanta Inc. company*
- Objektiv-Spektrometer: MS-7.5 *Photo Research - JADAK a Novanta Inc. company*
- Datenerfassung per USB: (SpectraWin™2 2019)
- Datenauswertung: (Python 3.9.7 2021) mit (pandas 2021), (colour 2021) und (matplotlib 2021)

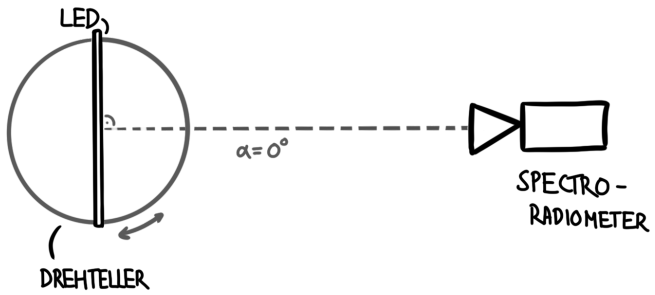


Abbildung 4.2: Aufbau: blickwinkelabhängige Messungen mit Spektrometer

Spektrometer

Wie in Abbildung 4.2 *(Aufbau)* dargestellt, befindet sich das Spektrometer in einem Abstand von 2,15m (gemessen vom Objektiv) zum LED-Panel und ist senkrecht auf dieses gerichtet. Ziel ist es den Abstand und die Ausrichtung ähnlich zur später verwendeten Kamera zu wählen. Außerdem wird das Spektrometer auf die Höhe des Mittelpunkts des LED-Panels gebracht, sodass die Blickachse senkrecht zum Boden und dem LED-Panel ist. Kontrolliert wird die richtige Justage mithilfe eines Testpatterns auf dem LED-Panel, das nur die mittleren zwei Pixelreihen bzw. Pixelspalten markiert. Durch den Sucher des Spektrometers sollte der Mittelpunkt zwischen diesen markierten Pixeln liegen. Außerdem wird mit Hilfe einer Schnur und des Winkelmessers der Blickwinkel $\alpha = 0°$ verifiziert. In Abschnitt 5.1.3 wird erläutert, wie das LED-Panel samt Drehteller und Winkelmesser eingerichtet werden.

Das Spektrometer *SpectraScan® PR-670* wird per USB-Tethering durch (SpectraWin™2 2019) gesteuert. Das Objektiv *MS-7.5* ist so defokussiert, dass das LED-Panel wie eine homogene Fläche wirkt und keine einzelnen SMD-3in1-LEDs erkennbar ist. Die Einstellungen des Spektrometers sind wie folgt:

Aperture - *1 deg*, Exposure Time - *adaptive*, Measurement Speed - *Normal*, Sensivity - *Standard*, Sync - *Automatic*, Average - *1 cycles*

Diese Einstellungen liefern für diesen Aufbau die am besten reproduzierbaren Ergebnisse.

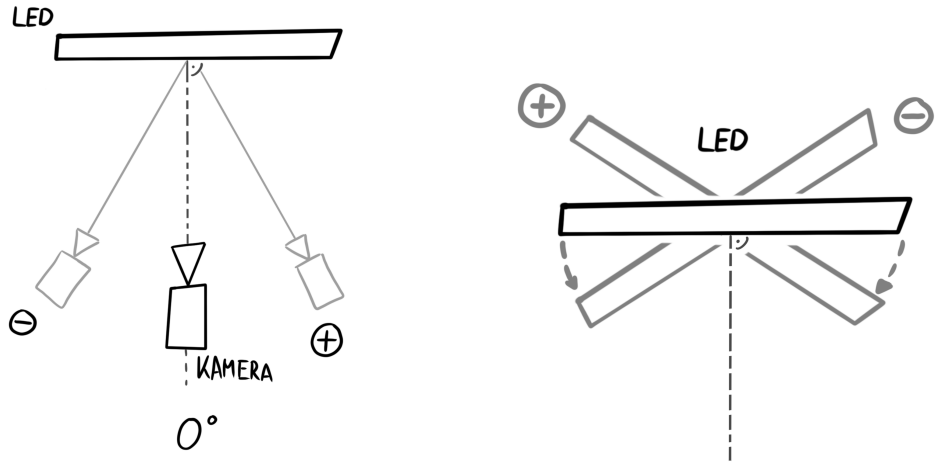


Abbildung 4.3: Festlegung, wie der Blickwinkel α zu interpretieren ist

LED-Panel

Das LED-Panel befindet sich auf einem Drehteller. Dieser ermöglicht es, anstelle der Kamera um einen bestimmten Pivotpunkt, das LED-Panel selbst um die eigene Achse zu rotieren. Die detaillierte Ausführung zum Aufbau von Drehteller und LED-Panel finden sich in Abschnitt 5.1.3. Der Blickwinkel α orthogonal zum LED-Panel wird im Folgenden auf 0° gesetzt. Negativ ist ein Blickwinkel α, wenn die Kamera sich links der orthogonalen Achse befindet bzw. positiv, wenn sich diese rechts davon befindet, siehe Abbildung 4.3 *(Blickwinkel α - Festlegung)*. In dieser Arbeit wird nur die Veränderung der Winkelkoordinate in horizontaler Richtung betrachtet.

Zuspielung LED-Panel

Der LED-Controller wurde in NovaLCT 5.4.3 2021 frisch konfiguriert und der Wert Gamma wurde auf 2.2 eingestellt. Die zu zuspielenden Grafiken waren 16bit tiefe und 128x128px große TIFF-Dateien im RGB Format. Zugespielt wurden folgende Farbwerte vollflächig, in gegebener Reihenfolge, pro Blickwinkel:

1. White (R = 1.0, G = 1.0, B = 1.0)
2. Red (R = 1.0, G = 0.0, B = 0.0)
3. Green (R = 0.0, G = 1.0, B = 0.0)
4. Blue (R = 0.0, G = 0.0, B = 1.0)

Versuchsumgebung

Die Messungen wurden in einem abgedunkelten Zimmer durchgeführt, sodass kein Tageslicht vorhanden war. Allerdings konnten die Wände nicht mit schwarzem, lichtabsorbierendem Material abgehängt werden. Deshalb kann minimales reflektiertes Streulicht nicht ausgeschlossen werden. Die Messreihe über verschiedenen Blickwinkel α fanden jedoch unter denselben Bedingungen statt. Deshalb ist davon auszugehen, dass die Aussagekraft der Veränderung über verschiedene Blickwinkel α nicht eingeschränkt ist.

4.3 Versuchsablauf - spektrale Messungen über verschiedene Blickwinkel

Es wurden für folgende Blickwinkel α Daten mit dem Spektrometer erhoben.

$$\alpha \in \{-85°, -80°, -70°, -60°, -50°, -40°, -30°, -20°, -10°, \\ 0°, 10°, 20°, 30°, 40°, 50°, 60°, 70°, 80°, 85°\}$$

Für jeden Blickwinkel α wurden die in 4.2 erwähnten Farbwerte vollflächig über das LED-Panel ausgespielt und jeweils eine Messung mit dem Spektrometer erstellt.
Für jede Messreihe pro Blickwinkel wurden folgende Parameter gemessen bzw. errechnet von (SpectraWin™2 2019):

- Strahldichte *radiance* L_e in $W*sr^{-1}*m^{-2}$ pro Wellenlänge λ in nm in $2nm$ Schritten, von 380 bis inkl. $780nm$
- Strahldichte *radiance* L_e in $W * sr^{-1} * m^{-2}$ (Integral unter Strahldichte pro Wellenlänge)
- Leuchtdichte *luminance* L_V in $\frac{cd}{m^2}$ (Integral unter V-Lambda-Kurve und Strahldichte pro Wellenlänge)
- XYZ-Koordinaten sowie xy- und u'v'-Koordinaten der Strahldichte-Verteilung

Diese Daten wurden per Exportfunktion aus SpectraWin™2 2019 in eine Textdatei im CSV-Format geschrieben und via pandas 2021 mit Python 3.9.7 2021 weiterverarbeitet.

4.4 Datenaufbereitung - Spektrometer

Um die Daten der CSV-Datei aus SpectraWin™2 2019 in Python 3.9.7 2021 zu verarbeiten, wird die Bibliothek pandas 2021 verwendet. Je Blickwinkel α liegt eine Messreihe vor. Alle verfügbaren Messreihen werden in ein Dataframe-Objekt überführt, die gesammelt in einem Dictionary *data_frames* verfügbar sind. Jede Spalte in der CSV-Datei repräsentiert eine Messung. Alle Daten in den Zellen werden als Zeichenkette *(str)* interpretiert und müssen, um verarbeitet zu werden, in Fließkommazahlen *(float)* gewandelt werden. Die numerischen Werte liegen in Kommaschreibweise vor. Damit Python diese als Fließkommawert *(float)* interpretieren kann, müssen alle Kommata in Punkte gewandelt

werden. Hierfür werden die Hilfsfunktionen *convertCommaDecimaToFloat(x)* und *change-ToFloat(x)* genutzt. *Algorithmus* 1

Mit den Bibliotheken colour 2021 und matplotlib 2021 lassen sich aus den Daten des Spektrometers Abbildungen zur Datenauswertung erstellen.

4.5 Auswertung der Daten des Spektrometers

Aus den gesammelten Daten des Spektrometers pro Blickwinkel α sollen hauptsächlich zwei Fragen beantwortet werden:
Wie verändert sich die Strahldichte L_e pro Wellenlänge λ bzw. die Leuchtdichte L_V über verschiedene Blickwinkel α?
Wie verändert sich der Farbton des Angezeigten über verschiedene Blickwinkel α?

4.5.1 Strahldichte-Verteilung über verschiedene Blickwinkel α

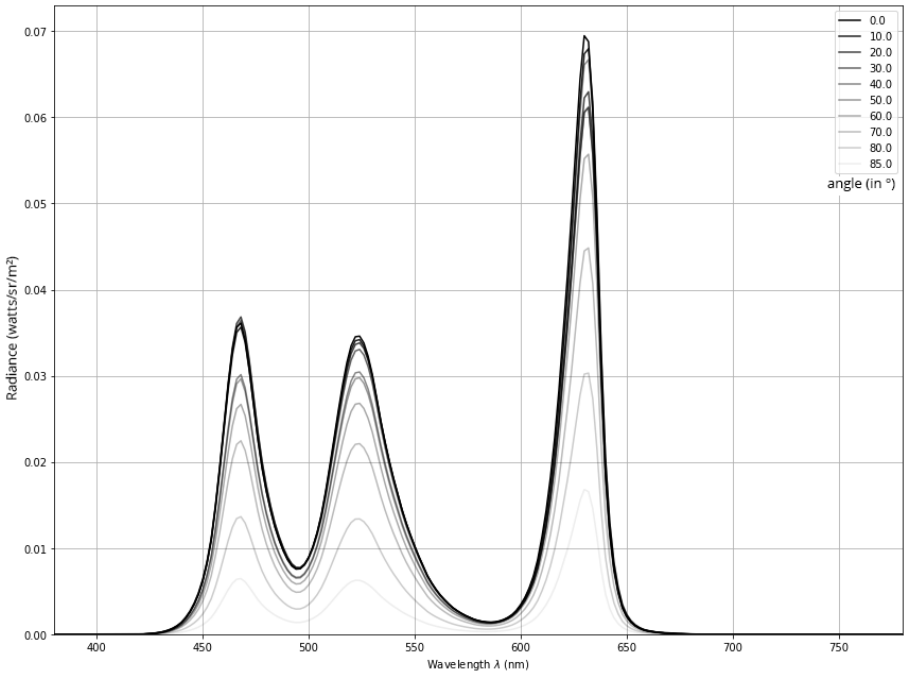


Abbildung 4.4: Strahldichteverteilung L_e pro λ rechts der Blickachse 0°, Angezeigter Farbwert: (1,1,1)

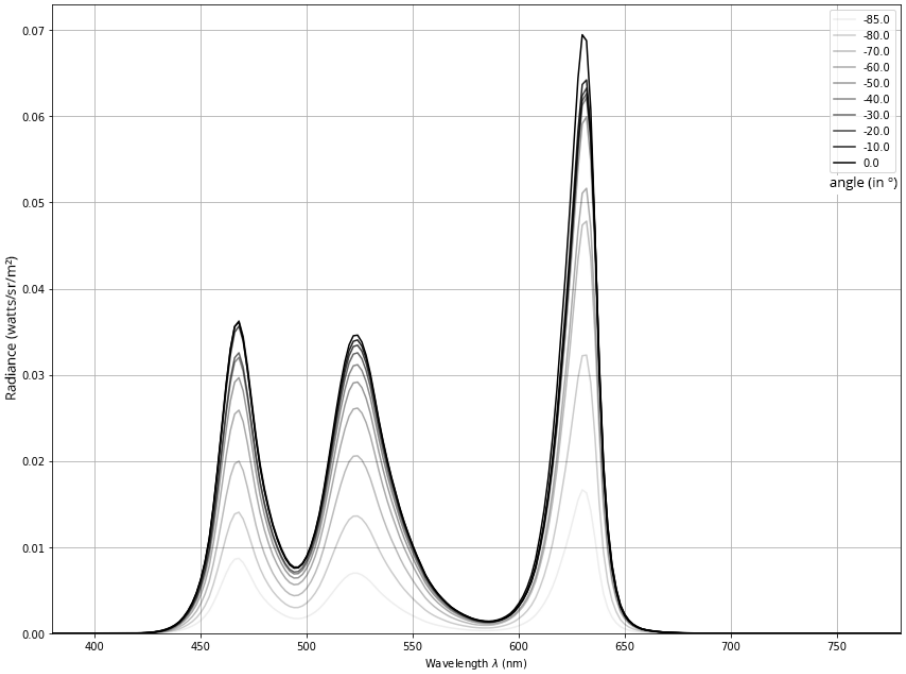


Abbildung 4.5: Strahldichteverteilung L_e pro λ links der Blickachse 0°,
Angezeigter Farbwert: (1,1,1)

In den Abbildungen 4.4 und 4.5 ist die Veränderung der Verteilung der Strahldichte L_e pro Wellenlänge λ für die Blickwinkel $\alpha = 0°$ bis zu $\alpha = 85°$ bzw. $\alpha = -85°$ zu ablesbar.

Aus den Daten ist klar erkennbar, dass die Strahldichte L_e abseits der senkrechten Blickachse abnimmt. Je größer der absolute Blickwinkel α ist, desto kleiner wird die Strahldichte für jede Wellenläge λ. Die Daten deuten für die 10°-Schritte bis zu $\alpha = \pm 40°$ eine kleinere Veränderung an, als darüber hinaus. Diese Beobachtung soll überprüft werden.

In der folgenden Abbildung 4.6 ist die akkumulierte Strahldichte L_e für den angezeigten Farbwert Weiß *(1, 1, 1)* und deren Veränderung über verschiedene Blickwinkel dargestellt. Hier ist der Abfall deutlich zu erkennen. Je weiter sich die Beobachtung von der Mittelachse $\alpha = 0°$ entfernt, desto größer ist die Veränderung pro $\Delta\alpha = \pm 10°$. Außerdem ist eine Symmetrie an der Mittelachse bei 0° erkennbar.

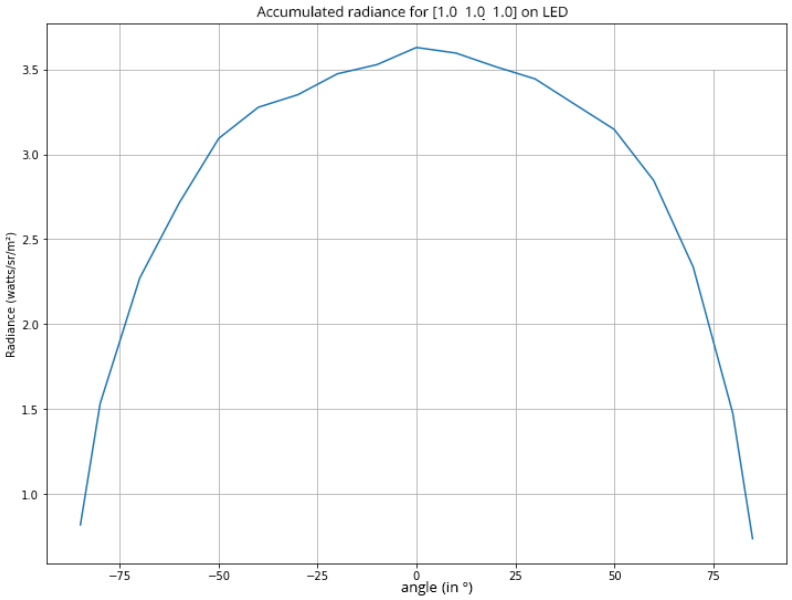
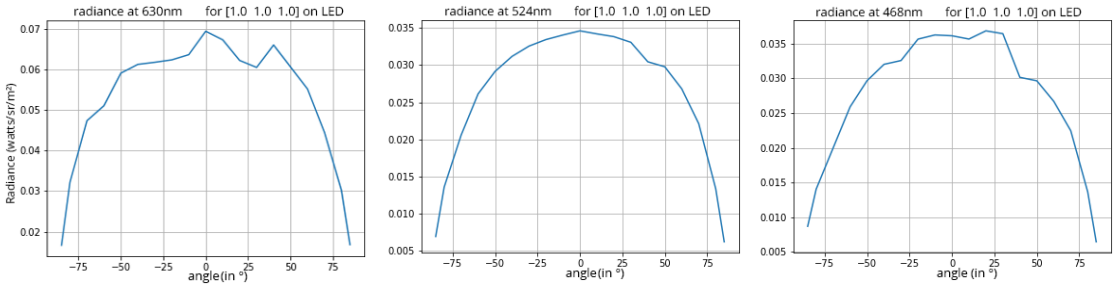


Abbildung 4.6: Akkumulierte Strahldichte L_e pro Blickwinkel α, Angezeigter Farbwert: (1,1,1)

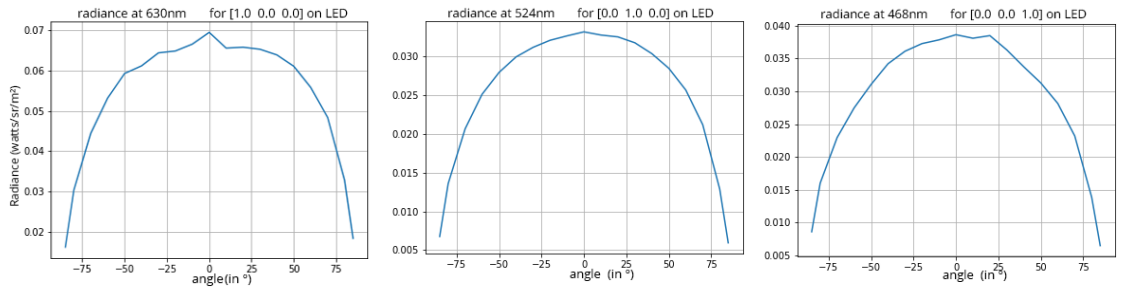
Zusätzlich wird für die Ausgabe von Weiß die Veränderung der Strahldichte für die Peak-Wellenlängen *(630nm, 524nm, 468nm)* berechnet und in Abbildung 4.7 dargestellt. In Abbildung 4.8 findet sich die gleiche Darstellung, jedoch für jeweils die angezeigten Farbwerte *Rot*, *Grün* und *Blau*.

Die Daten in Abbildung 4.6, 4.7 und 4.8 deuten alle auf eine gute Modellierbarkeit hin. Besonders zeigen sie die Problematik auf, die es beim Abfilmen von LED-Walls gibt: Je größer der absolute Blickwinkel, desto geringer ist die Strahldichte aller LEDs.



(a) Rot-Peak: 630nm (b) Grün-Peak: 524nm (c) Blau-Peak: 468nm

Abbildung 4.7: Strahldichte L_e je LED für jeweilige Peak-Wellenlänge λ_{peak}, pro Blickwinkel α, angezeigter Farbwert: (1,1,1)



(a) Peak - rote LED: 630nm (b) Peak - grüne LED: 524nm (c) Peak - blaue LED: 468nm

Abbildung 4.8: Strahldichte L_e je LED für jeweilige Peak-Wellenlänge λ_{peak}, pro Blickwinkel α, Angezeigte Farbwerte: *(a)* (1,0,0), *(b)* (0,1,0), *(c)* (0,0,1)

4.5.2 Leuchtdichte über verschiedene Blickwinkel α

Für die Leuchtdichte L_V ist, wie es mathematisch zu erwarten ist, das gleiche Verhalten beobachtbar. Interessant ist insbesondere das Verhältnis, zwischen $L_{V,max}$ und $L_{V,\alpha}$, da die Leuchtdichte L_V als photometrische Einheit der menschlichen Helligkeitswahrnehmung eher entspricht als die Strahldichte L_e.

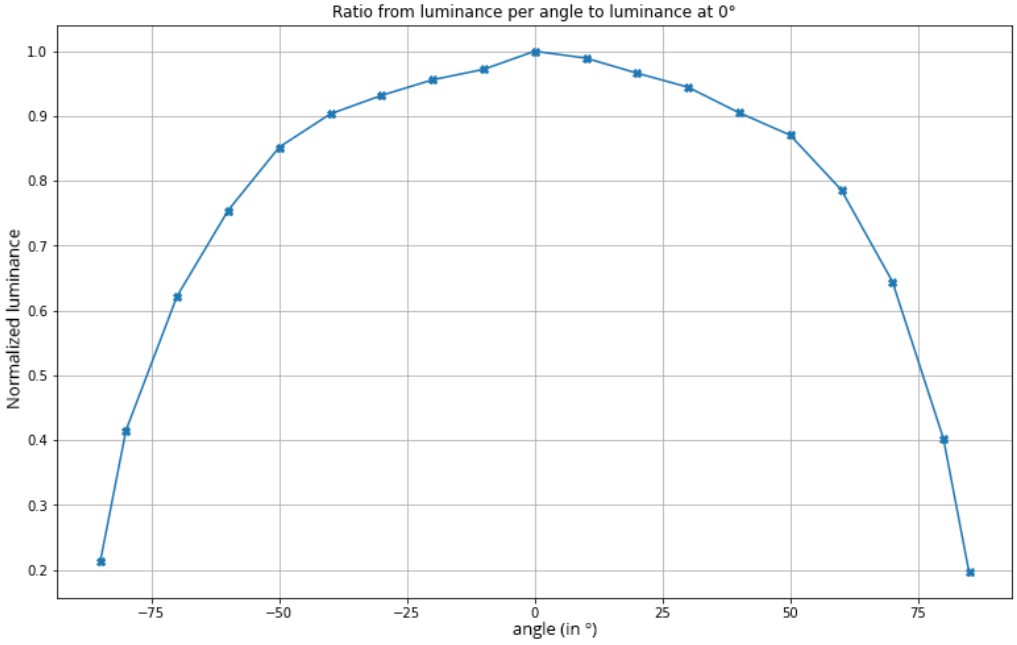


Abbildung 4.9: Verhältnis der Leuchtdichte $L_{V,\alpha}$ pro Blickwinkel α zu $L_{V,0°}$, Angezeigter Farbwert: (1,1,1)

In Abbildung 4.9 ist das Verhältnis der Leuchtdichte L_V pro Winkel α zur Leuchtdichte L_V für $\alpha = 0°$ dargestellt. Hier ist deutlich erkennbar, dass die Leuchtdichte stark abnimmt,

wenn der absolute Winkel zur senkrechten Blickachse mehr als 40° bzw. 50° beträgt. In dem Datenblatt von ICT ist zu diesem Panel ein Einblickwinkel von 140°, also von −70° bis 70°, angegeben (ICT AG o. D.[b]). Nach Prak 2021 beschreibt der Einblickwinkel den Winkel, in dem eine LED-Wall noch 50% ihrer Leuchtdichte erreicht.

Wie hier gemessen wurde, beträgt für diese Blickwinkel α die Leuchtdichte L_V nur noch 64,5% bzw. 62,2% der Leuchtdichte L_V für $\alpha = 0°$. Die Notwendigkeit für eine blickwinkelabhängige Korrektur begründet sich hierdurch. Besonders für Aufbauten, bei denen LED-Walls orthogonal zueinander stehen, wie von disguise 2021 empfohlen. Hier ist der Standard-Blickwinkel einer Kamera bei $\alpha = \pm 45°$ für beide LED-Walls.

4.5.3 Farbverschiebung über verschiedene Blickwinkel α

Nachdem der Einfluss des Blickwinkels α auf die gesamte Strahldichte L_e bzw. Leuchtdichte L_V erläutert wurde, soll in diesem Abschnitt der Einfluss auf den Farbton unabhängig von der Helligkeit untersucht werden. Aus den Daten des Spektrometers lassen sich nicht nur die oben genutzten Werte auslesen, sondern ebenso die xy- und u'v'-Koordinaten der Messung. Die Werte für xy nach *CIE 1931* und u'v' nach *CIE 1971* sind ein geräteunabhängiger Farbwert aus zwei Koordinaten, die den Farbton bestimmen. Das Koordinatensystem für u'v' ist so verzerrt, dass ein Unterschied in u'v' egal in welchem Bereich proportional dem wahrgenommenen Unterschied gleicht. (Hasche und Ingwer 2016)

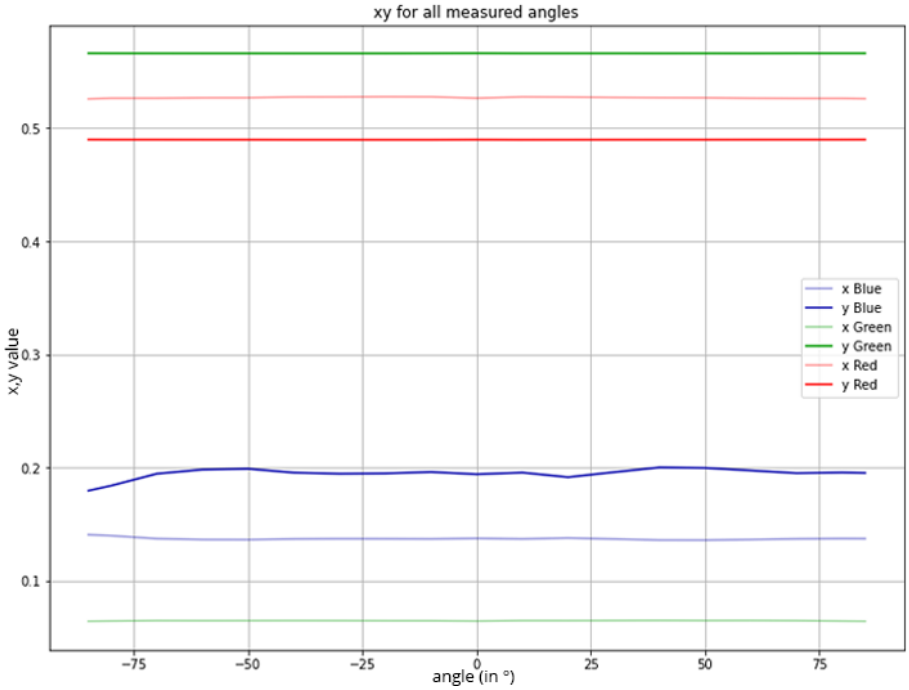


Abbildung 4.10: Veränderung über Blickwinkel α von x,y nach CIE 1931, Angezeigte Farbwerte: (1,0,0), (0,1,0), (0,0,1)

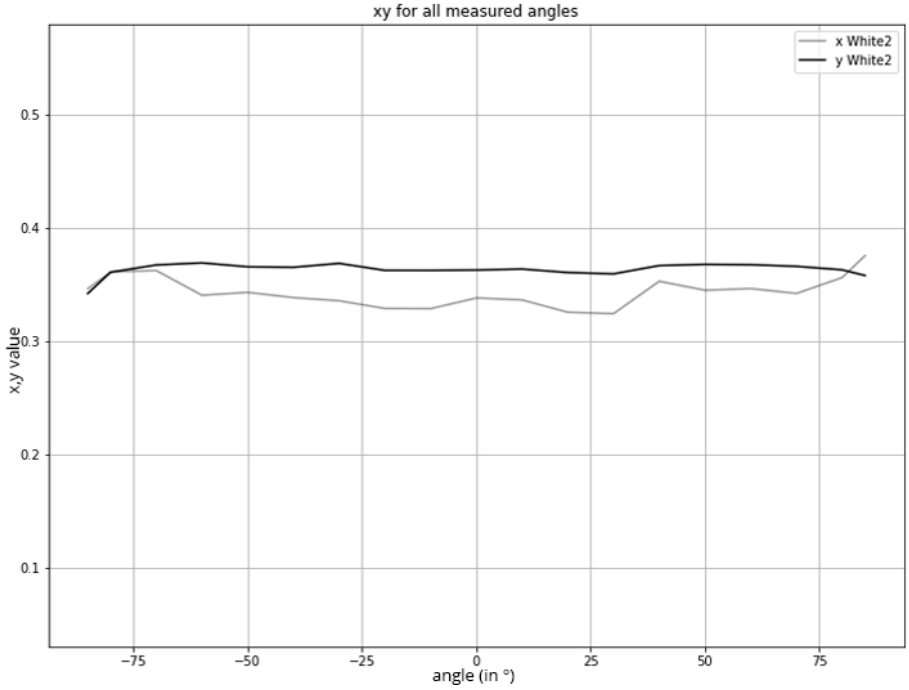


Abbildung 4.11: Veränderung über Blickwinkel α von x,y nach CIE 1931, Angezeigter Farbwert: (1,1,1)

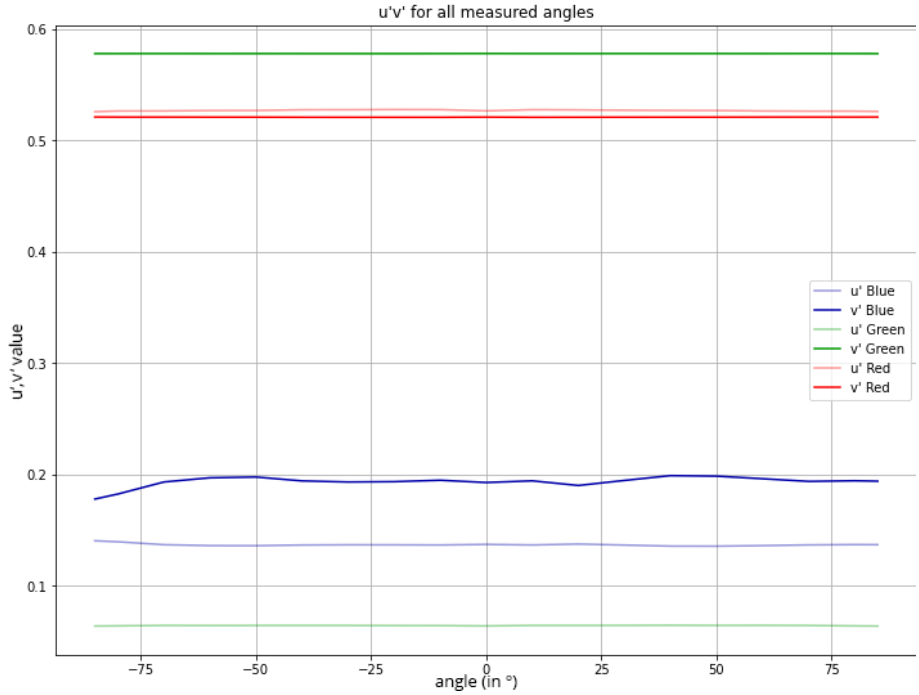


Abbildung 4.12: Veränderung über Blickwinkel α von u',v' nach CIE 1976, Angezeigte Farbwerte: (1,0,0), (0,1,0), (0,0,1)

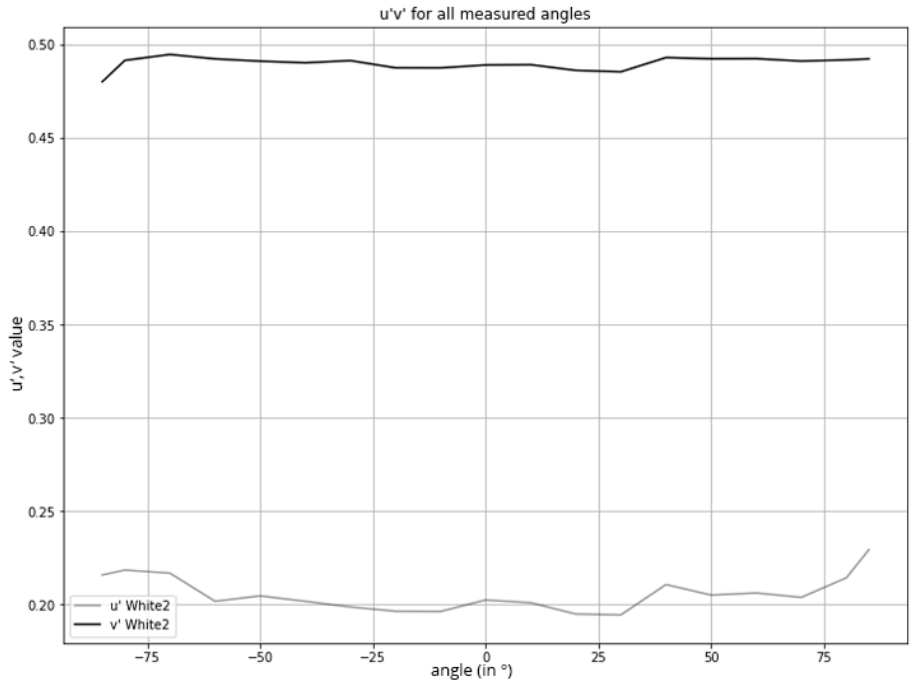


Abbildung 4.13: Veränderung über Blickwinkel α von u',v' nach CIE 1976, Angezeigter Farbwert: (1,1,1)

Veränderung von xy und u'v'

Für die angezeigten Farbwerte Rot, Grün und Blau (siehe Abbildung 4.10 und 4.12) ist keine merkliche Änderung über den Blickwinkel α erkennbar. Die Werte für Blau im Bereich von $\alpha < 70°$ können als Messfehler gewertet werden. Da die Messungen aus Abbildung 4.8 *(c)* auf ein symmetrisches Verhalten schließen lassen und diese Veränderung nur im Bereich links der senkrechten Blickachse auftreten.
Betrachtet man dagegen die Veränderung der Werte für Weiß in Abbildung 4.11 und 4.13, über die Blickwinkel α, erkennt man für die Koordinaten x bzw. u' eine annähernd symmetrische Veränderung in den Blickwinkeln $|\alpha| > 50°$. Das Verhalten deutet auf eine Farbverschiebung in diesem Bereich hin, sodass eine Korrektur für den Blickwinkel $\alpha = 0°$ nicht anwendbar ist. Stattdessen sollten gesonderte Korrekturen für mehr als nur einen Blickwinkel α errechnet werden.

Strahldichte der Peaks im Verhältnis

Um geäußerte Vermutung zur Farbtonverschiebung zu überprüfen, soll die Veränderung pro LED im Verhältnis betrachtet werden. Das kann aus der Verteilung der Strahldichte L_e pro Wellenlänge λ gelesen werden. Es wird die Strahldichte an der Stelle bzw. der Wellenlänge λ genutzt, die für die jeweilige LED als Peak angekommen wird. Das ist für Rot *630nm*, für Grün *524nm* und für Blau *468nm*.
Wenn für die Messungen der Farbwerte von Rot, Grün und Blau die Strahldichte $L_{e,\lambda,\alpha}$ für die Wellenlänge λ pro Blickwinkel α im Verhältnis zu seiner jeweiligen maximalen Strahldichte $L_{e,\lambda,0°}$ für Blickwinkel $\alpha = 0°$ und Wellenlänge λ dargestellt wird, erhält man Abbildung 4.14.

Sofern es keine Farbverschiebung über die Messungen hinweg gibt, sollten alle Linien der Abbildung übereinander liegen. Da hier der Verlust, im Verhältnis zum jeweiligen Maximalwert, gleich bleiben müsste, um den gleichen Farbton zu erhalten.

Im Bereich von ca. $\pm 60°$, ist gut zu erkennen, dass das nicht der Fall ist. Dort scheint die Strahldichte der Blauen-LED schneller abzunehmen als die der anderen LEDs. Dieser leichte Trend ist auch für den Farbwert *(1, 1, 1)* in Abbildung 4.15 zu erkennen.

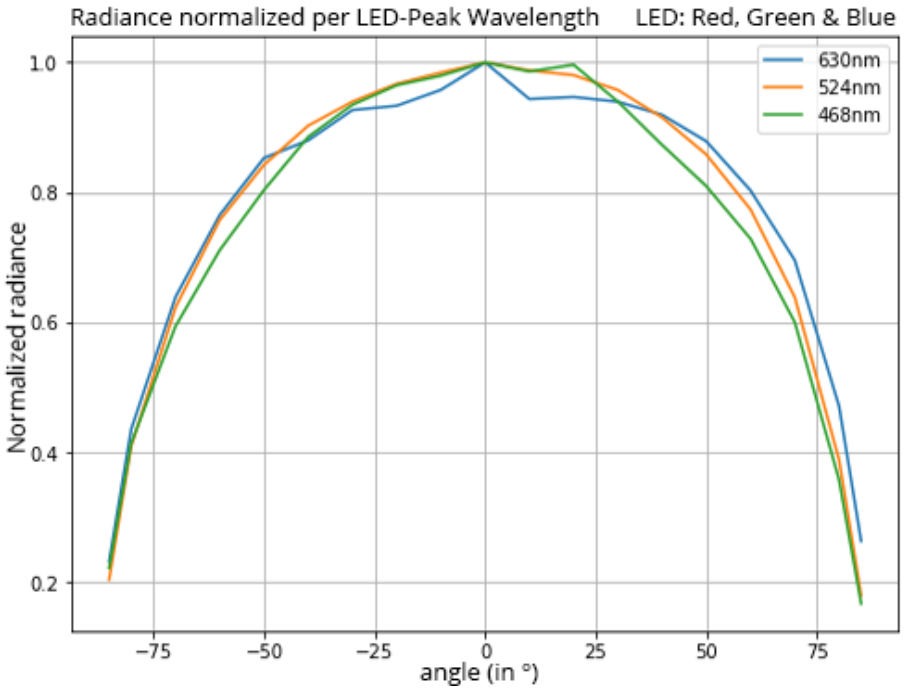


Abbildung 4.14: Angezeigte Farbwerte (1,0,0), (0,1,0), (0,0,1): Verhältnis von $I_{e,\alpha}$ zu $I_{e,0^\circ}$

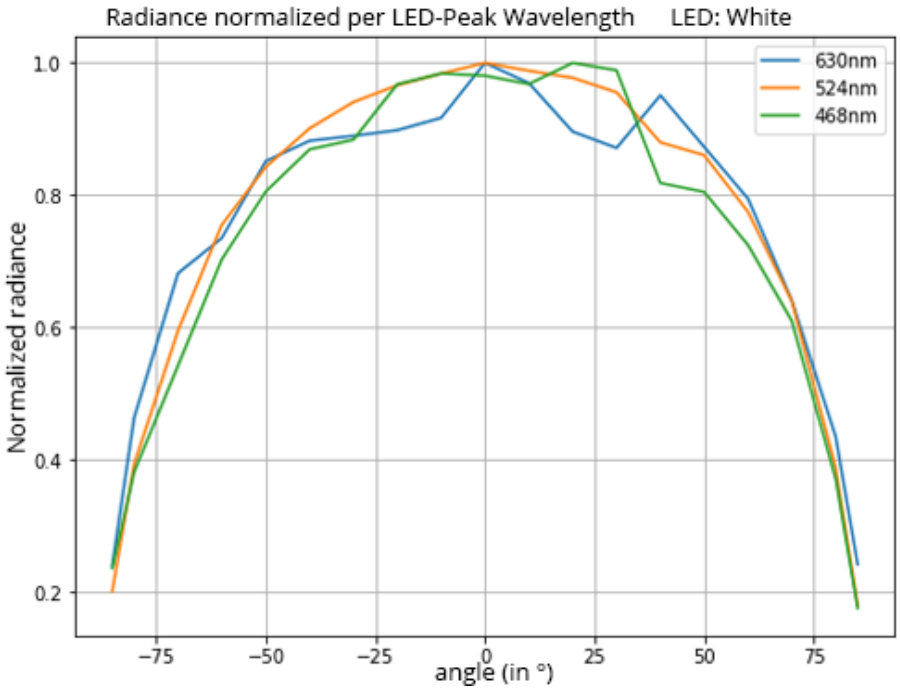


Abbildung 4.15: Angezeigter Farbwert (1,1,1): Verhältnis von $I_{e,\alpha}$ zu $I_{e,0^\circ}$

4.6 Bedeutung der spektralen Messungen für eine blickwinkelabhängige Korrektur

Die Ergebnisse aus den Versuchen mit dem Spektrometer zeigen, dass die Veränderung des Blickwinkels α auf das aufgenommene Bild einer Kamera einen signifikanten Einfluss hat. Zum Einen hat das Abfallen der Strahldichte einen großen Einfluss auf das Bild der Kamera, je größer der absolute Blickwinkel α ist. Zum Anderen scheint es eine leichte Farbverschiebung für Bereiche der möglichen Blickwinkel α zu geben.

Beide Sachverhalte zeigen, welche Parameter eine blickwinkelabhängige Korrektur korrigieren muss. Um die Strahldichte optimal zu korrigieren, muss außerdem beachtet werden, dass sich diese Korrektur immer an der geringsten Strahldichte orientieren muss. Das hier verwendete LED-Panel hat eine maximale Strahldichte L_e mit dem Farbwert Weiß von 3.629 $Wsr^{-1}m^{-2}$ bei 0°, jedoch fällt dieser bei einem Blickwinkel 80° auf 1.533 $Wsr^{-1}m^{-2}$. Ein Korrekturalgorithmus sollte immer die Möglichkeit haben, den Bildwert zu verringern. Das ist besonders relevant für die Einstellung des LED-Controllers.
Da nicht nur die Strahldichte angepasst werden muss, sondern auch der Farbton, ist der in Kapitel 3 vorgeschlagene Workflow eine 3×3-Matrix zu nutzen als folgerichtig zu bewerten.

Zwei Dinge finden in der *Methode* dieser Arbeit keine Verwendung, sind dennoch von Relevanz für zukünftige Implementierungen. Erstens zeigen die Daten dieser Analyse, dass die Veränderungen sich symmetrisch zur senkrechten Blickachse verhalten. Eine Einmessung nur auf einer Seite der Blickachse ist deshalb denkbar. Zweitens deuten die Abbildungen der hier erhobenen Daten daraufhin, dass sich besonders der Abfall der Strahldichte L_e modellieren lässt. Wenn dieses Modell der Veränderung der Strahldichte L_e für eine LED-Wall bekannt ist, können zwei Messungen, für $\alpha_0 = 0°$ und α_{max}, zum Einmessen reichen. So könnten neue Kameras in einem bekannten LED-Wall-Setup deutlich schneller eingemessen werden.

5 Implementierung und Evaluierung der Methode zur blickwinkelabhängigen Korrektur

Inwiefern die erläuterte Methode zur blickwinkelabhängigen Korrektur einer LED-Wall für die Verwendung mit einer Kamera funktionsfähig ist, soll in diesem Kapitel untersucht werden. Die einzelnen Bestandteile und die Gesamtmethode werden auf ihre Umsetzbarkeit und Funktionalität überprüft und evaluiert. Dafür werden der im Folgenden genutzte Testaufbau nachvollzogen, die Implementierungshilfen erläutert und Ergebnisse aus den Messreihen dargestellt und interpretiert.

5.1 Aufbau und Material

5.1.1 Verwendete Hardware-Komponenten

- LED-Panel: inspireLED s3.6b Indoor *ICT AG*
- LED-Prozessor: inspireLED MCTRL660 *ICT AG* und *Xi'an NovaStar Tech Co., Ltd.*
- Drehteller, Durchmesser 45cm
- Computer: MacBook Pro 14“ 2021, MacOS 12.1 (21C52) *Apple Inc.*
- Inputkarte: Blackmagic UltraStudio Recorder 3G *Blackmagic Design Pty. Ltd.*
- Kamera: FS5 II *Sony - PXW-FS5M2*
- Objektiv: 18 bis 105mm F4 *Sony - SELP18105G*

5.1.2 Verwendete Software und Bibliotheken

- Zuspiel-Software Computer: (Qlab 4.6.11 2021)
- Inputkarte Auslesen: (FFMPEG 2021) mit (Decklink Software Development Kit 10.11.2 2021)

- (Python 3.9.7 2021), mit folgenden Bibliotheken:
  - (colour 2021)
  - (numpy 2021)
  - (ffmpeg-python 2021)
  - (matplotlib 2021)
  - (scipy 2021)
- LED Controller-Software: (NovaLCT 5.4.3 2021)
- Inputkarte Auslesen Möglichkeit Zwei: (DaVinci Resolve 17.4.2 2021)

5.1.3 Übersicht des Systems im konkreten Aufbau

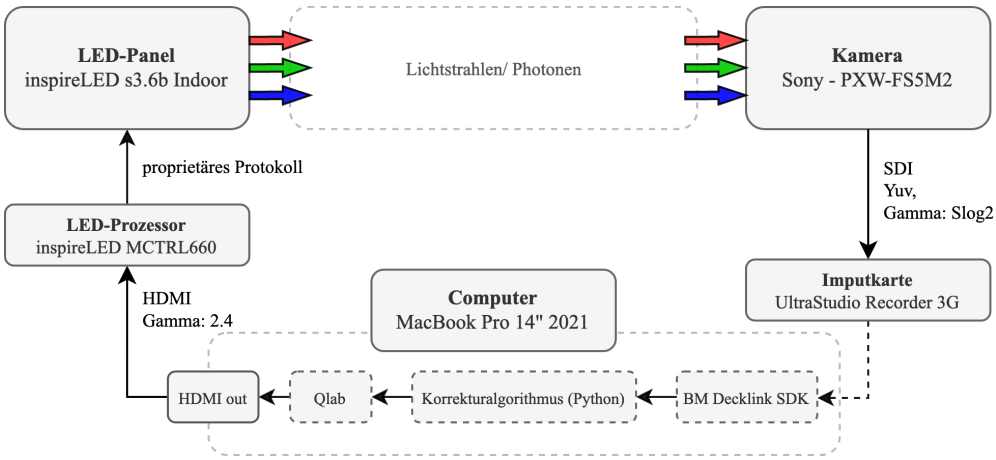


Abbildung 5.1: Schematische Darstellung des konkreten Versuchsaufbaus

Für die Evaluierung, der in Kapitel 3 erläuterten Methode, werden die oben genannten Komponenten im Gesamtsystem implementiert (siehe Abbildung 5.1). Der Computer ist das zentrale Element. Dieser sendet ein Videosignal aus Qlab 4.6.11 2021 über den Standard HDMI-Ausgang an den LED-Prozessor *MCTRL660*. Das Videosignal ist Gamma-kodiert, mit $\gamma = 2.4$. Dieser übergibt das verarbeitete, nun proprietäre Signal weiter an das LED-Panel. In dieser Arbeit wird ein LED-Panel der Serie *inspireLED s3.6b* genutzt. Die verbauten RGB-SMD-LEDs zeigen das Videosignal pulsweitenmoduliert an. Die Kamera *FS5M2* wandelt die Strahldichte der Szene in ein elektrisches Signal und verarbeitet dieses zu einem digitalen Videosignal. Das Signal wird über SDI in YCbCr, Gamma-kodiert mit *S-Log2* an die Inputkarte *UltraStudio Recorder 3G* übergeben. Das entsprechende SDK

(Decklink Software Development Kit 10.11.2 2021) und die in Abschnitt 5.2.2 besprochene selbstgeschriebene Hilfsfunktionen wandeln diesen Input in ein von Python 3.9.7 2021 mit numpy 2021 weiter verarbeitbares *ndarray*.

5.1.4 Physischer Versuchsaufbau

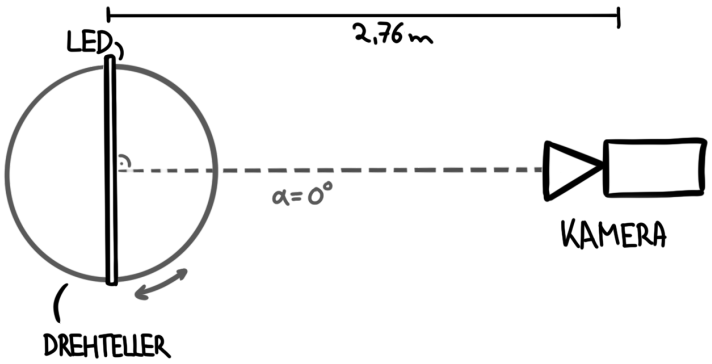


Abbildung 5.2: Aufbau LED-Panel und Kamera

Für dieses Testszenario wird kein LED-Wall-Studio genutzt. Der Aufbau ist so weit vereinfacht, dass er dennoch die essentiellen Bestandteile, die für die Evaluierung der Methode relevant sind, abbilden kann.
Daher wird auf ein Tracking-System verzichtet. Stattdessen steht das einzelne LED-Panel auf einem Drehteller. Der Blickwinkel α wird von einem Winkelmesser abgelesen und händisch in das System eingetragen (siehe Abbildung 4.3). Die senkrechte Blickachse der Kamera auf das LED-Panel sei $\alpha = 0°$.

Aufbau: LED-Panel

Das LED-Panel steht auf seinen *Füßen*. Diese sind ca. 10cm lange und 4cm breite Metallelemente und befinden sich auf der Unterseite des *inspireLED*-Cabinets. So aufgestellt steht das Panel 87,3° zur Bodenebene, also aus Sicht der Kamera leicht nach hinten gekippt. Solch eine Neigung ist für eine LED-Wall generell nicht unüblich und stört nicht, da sie gleichbleibend ist.
So steht das LED-Panel auf einem Drehteller. Damit davon ausgegangen werden kann, dass sich das LED-Panel exakt um seine Mitte, also die Mitte der Bildebene dreht, muss

der Mittelpunkt M des Drehtellers bestimmt werden.

Das LED-Panel steht parallel zu einer Geraden durch den bestimmten Mittelpunkt M so, dass die Drehachse in der Bildebene des LED-Panels liegt. Unter dem Drehteller befindet sich ein auf DIN-A3 ausgedruckter Winkelmesser. In der Grundausrichtung ($\alpha = 0°$) sind die Gerade durch den Mittelpunkt M orthogonal zum LED-Panel und die Markierung für $\alpha = 0°$ aneinander ausgerichtet. Erstellt wurde der Winkelmesser mit Adobe Illustrator 2022 2022. Der Mittelpunkt dieses Winkelmessers entspricht im Aufbau dem Mittelpunkt M des Drehtellers. Um diffuse Reflexionen vom Drehteller zu vermeiden, wird der Bereich vor dem LED-Panel mit schwarzem Stoff ausgelegt.

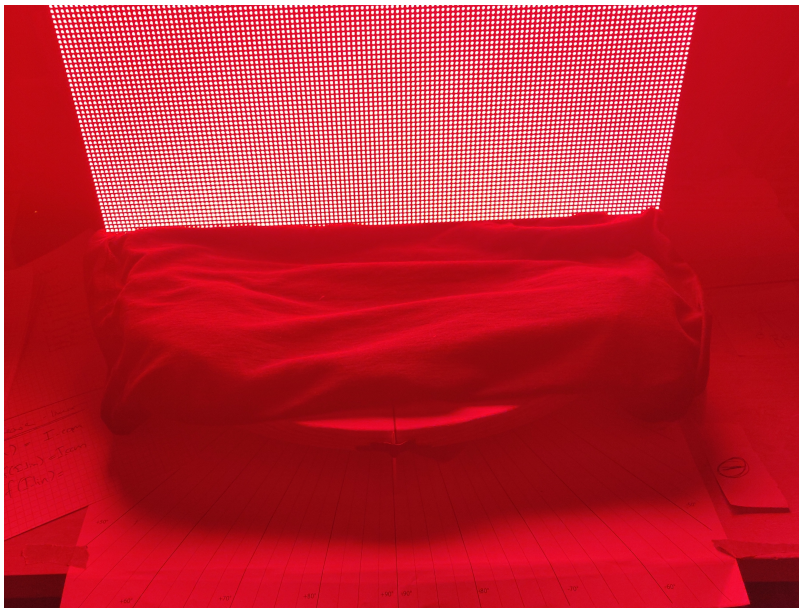


Abbildung 5.3: LED-Panel auf dem Drehteller, darunter der Winkelmesser

Aufbau: Kamera

Die Kamera befindet sich auf einem Stativ, genau 2,76m entfernt zur Bildebene des LED-Panels, gemessen vom Sensor. Gewählt wird die maximal mögliche Entfernung im Raum des Versuchsaufbaus, um den Moiré-Effekt zu vermeiden. Zuerst wird die Kamera so justiert, dass sie sich parallel zur Bodenebene befindet. Dann wird die Höhe des Stativs so eingestellt, dass eine horizontale Linie auf den zwei mittleren Pixellinien des LED-Panels in der Bildmitte der Kamera liegt. Die Blickachse soll senkrecht zum LED-Panel sein. Dafür wird eine dünne Schnur zwischen dem Mittelpunkt des Winkelmessers und dem Mittelpunkt des Stativs der Kamera gespannt. Das Stativ wird nun so positioniert, dass

die 0° Markierung unter der Schnur liegt. Um nun die Drehachse des Panels im Verhältnis zur Kamera zu verifizieren, wird eine vertikale Linie auf den zwei mittleren Pixelspalten des LED-Panels dargestellt. Im Kamerabild sollte sie gerade durch die Bildmitte verlaufen. Falls das nicht der Fall ist, muss die Kamera geschwenkt werden. Danach wird das LED-Panel auf dem Drehteller vorsichtig an die maximal möglichen Blickwinkel α positioniert. Für diese Blickwinkel muss die markierte Pixelspalte im Kamerabild gleich verlaufen und positioniert sein wie für $\alpha = 0°$. Durch die Verifizierung kann davon ausgegangen werden, dass die Blickachse ausreichend senkrecht für die Versuche ist.

Einrichtung: Ausgabe-Bild

Computer Wie das auszugebende Bild entsteht wird in Abschnitt 3.3.1 genauer beschrieben. Der Rückgabewert des in Python 3.9.7 2021 programmierten Algorithmus' ist eine TIFF-Datei. Diese Datei mit dem Namen *img_out.tiff* wird von Qlab 4.6.11 2021 ausgespielt. Der Grund, für diesen Testaufbau Qlab 4.6.11 2021 zu nutzen, liegt hauptsächlich in der Möglichkeit, für Autoupdates von Dateien im Livebetrieb. Wenn sich die anzuzeigende Datei ändert, in diesem Fall *image_out.tiff*, wird mit einer kurzen Verzögerung von ca. einer Sekunde die neue Datei ausgegeben. Außerdem ist es möglich den Videoausgang pixelgenau zu bespielen. Das ist für die Bespielung des LED-Panels, besonders für Testbilder, essentiell.

Der Videoausgang ist der Standard-HDMI Ausgang des *MacBook Pro 14"2021*. Die Ausgabeeinstellungen sind folgende: Gamma - 2.4, Farbraum - Rec.709, siehe Abbildung 5.4 *(Ausgabe-Einstellungen)*. Wichtig für die Reproduzierbarkeit ist in den Einstellungen des Hauptbildschirms *True Tone* zu deaktivieren. Diese Funktion hätte Auswirkungen auf alle angeschlossenen Displays.

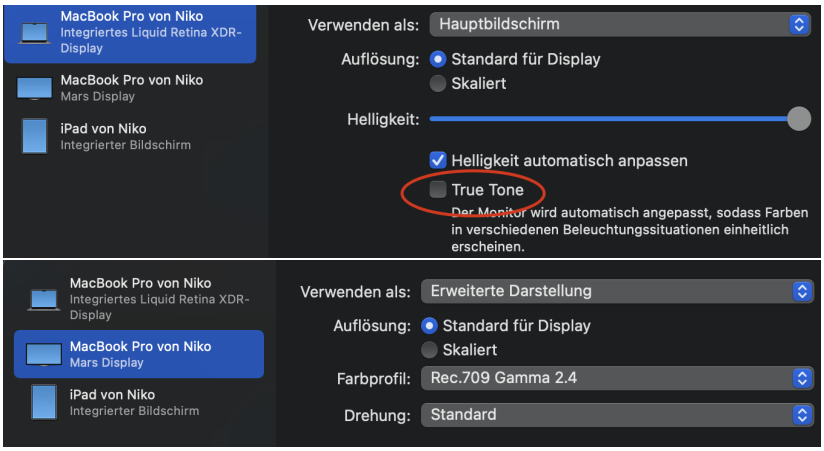
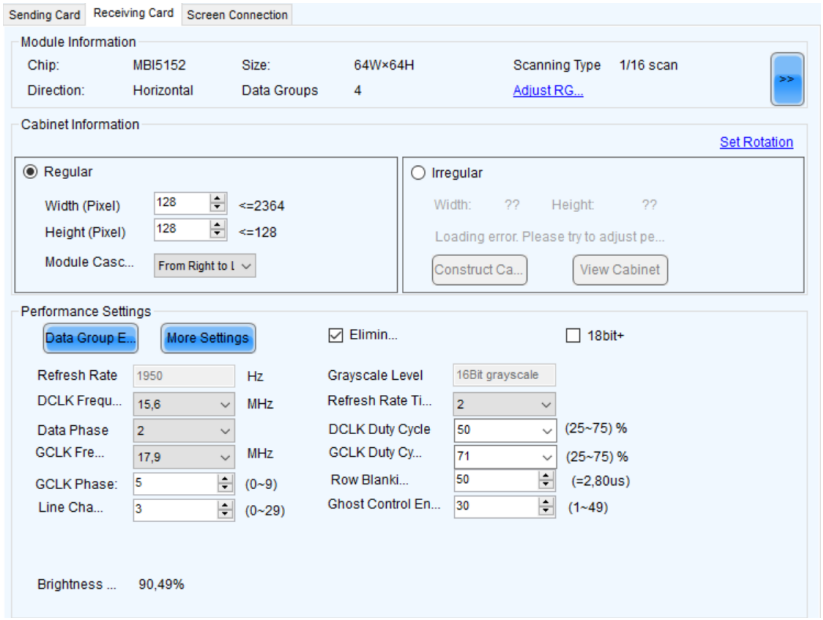


Abbildung 5.4: Ausgabeeinstellungen MacBook Pro 14"2021

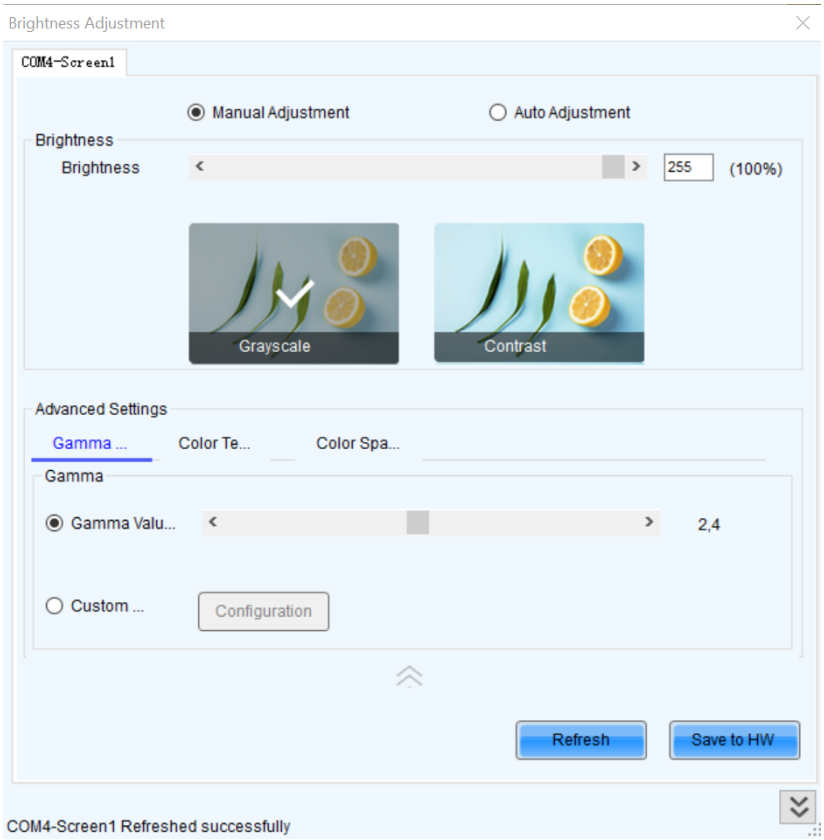
Einrichtung: LED Controller

Der hier genutzte LED-Controller *inspireLED MCTRL660* bekommt sein Signal durch den HDMI-Eingang oder DVI-Eingang. (ICT AG o. D.[a]) In diesem Fall wird er via HDMI bespielt. Das LED-Panel ist durch einem CAT5-Kabel an einem der *RJ45 Gigabit Ethernet* Ports mit dem LED-Controller verbunden. Hierdurch kann die Receivingcard des LED-Panels angesteuert und es können Bilddaten an das Panel übergeben werden.

Mit einem Windows-Laptop und NovaLCT 5.4.3 2021 kann der Controller konfiguriert werden. Für den Versuchsaufbau wird die mitgelieferte *.rcfg*-Datei in Controller und LED-Panel neu eingelesen. Außerdem wird das eingehende Videosignal mit einem Gamma $\gamma = 2.4$ interpretiert, siehe Abbildung 5.5 *(NovastarLCT-Einstellungen)*.



(a) Einstellungen nach Einlesen der *.rcfg*-Datei



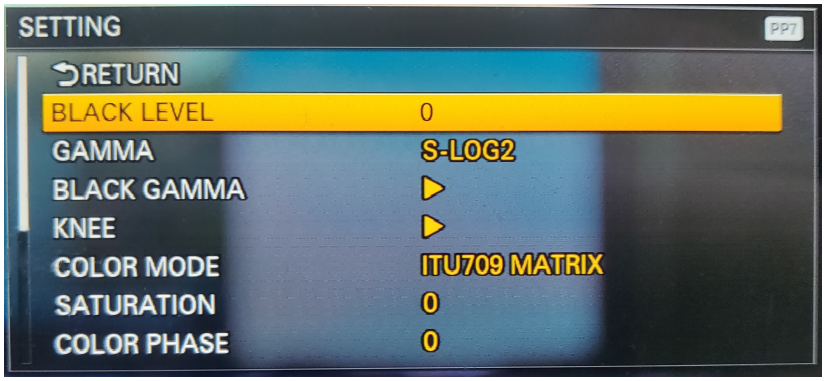
(b) Gamma-Einstellungen

Abbildung 5.5: Einstellungen in NovaLCT 5.4.3 2021 für *inspireLED MCTRL660*

Einrichtung: Kameraeinstellungen

Die Kamera ist senkrecht zum LED-Panel auf der Position $\alpha = 0°$ ausgerichtet und filmt dieses. Die Versuche finden hauptsächlich nachts sowie ohne Umgebungslicht statt. Die Kamera nimmt in *1080/25p* auf, verarbeitet wird das Signal wie in *Picutreprofile7* angegeben. Der *color mode* wird auf *ITU709* gestellt und das *Knee* wird auf *Manual* geändert, da sonst ungewollte nicht reproduzierbare adaptive Veränderungen auftreten könnten. Der Gamma wird auf *S-Log2* belassen. Das aufgenommene Videosignal wird via SDI weitergereicht, siehe Abbildung 5.6.

(a) Videooutput Einstellungen



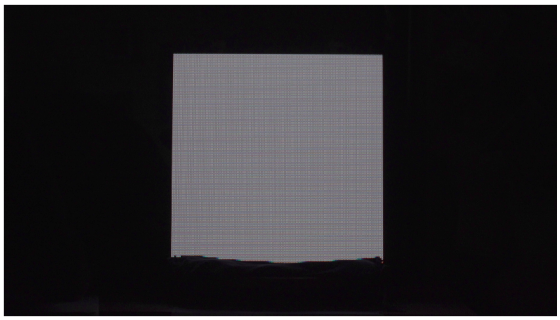
(b) Pictureprofile Einstellungen

Abbildung 5.6: Einstellungen Kamera, *Sony - PXW-FS5M2*

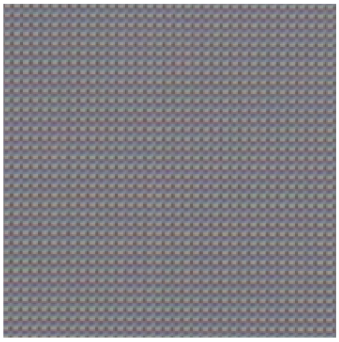
Die Brennweite der Kamera wird so eingestellt, dass genügend Bildinhalt vom LED-Panel gefüllt wird, sodass der in Abschnitt 3.3.2 erläuterte Denoise-Algorithmus genügend Bildinformation erhält. Dennoch müssen genügend RGB-SMD-LEDs auf einen Bildpixel fallen, um Moirepattern zu vermeiden. In diesem Aufbau wird das bei einem Wert *Zoom = 57* erreicht.

Eine weitere Besonderheit in den Einstellungen der Kamera besteht darin, den Shutter-Angle der Kamera auf 300° zu setzen. Da bei diesem Aufbau kein Taktgeber implementiert ist, kommt es mit einem Shutter-Angle von 180° vereinzelt zu Scanline-Artefakten. Diese treten auf, wenn es zu einer temporalen Unterabtastung der LED-Panels kommt. (Kintrup 2022)

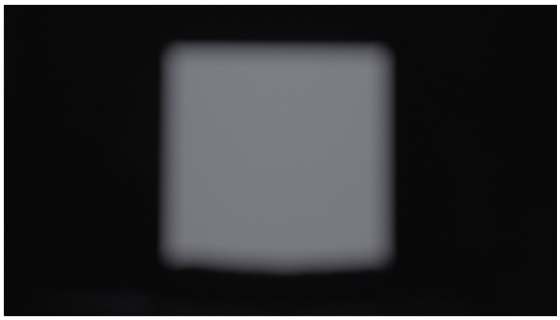
Abbildung 5.7: Scanline-Artefakt bei einem Shutter-Angle 180°



(a) Fokus auf 1.8m



(b) Ausschnitt für Median-Berechnung



(c) Fokus auf 0.4m

(d) Ausschnitt für Median-Berechnung

Abbildung 5.8: Fokus-Einstellungen des Objektivs zur Verhinderung von Moiré-Effekten

Um Moiré-Effekte zu verhindern, muss die Blende des Objektivs auf die geöffnetste Einstellung gesetzt werden. Für das 18 bis 105 mm F4 *Sony - SELP18105G* entspricht das einem Wert von F4.0. Idealerweise sollte der Fokuspunkt für eine Farbkorrektur ungefähr dem entsprechen, der später für die Produktion genutzt wird. In diesem Fall war das nicht möglich, siehe Abbildung 5.8 *(Verschiedene Fokus-Einstellungen)* . Deshalb wird für diesen Versuch durchgängig der Fokuspunkt auf 0.4m gesetzt, die minimalste Einstellung des Objektives.

In Folge dessen müssen die verbleibenden Parametern für die Belichtung, Gain und ND-Filter, gewählt werden. Für diesen Aufbau, ist es nicht erforderlich die Kamera so zu belichten, dass ein Subjekt im Vordergrund richtig belichtet dargestellt wird, da allein die Funktionsfähigkeit der Methode verifiziert werden soll. Für weitere, praxisnähere Versuche, müsste sich daran orientiert werden. Aus diesen Gründen wird der Gain auf 0db belassen, um möglichst rauschfreie Bilder zu produzieren.
Der variable ND-Filter wird auf *1/27* eingestellt. Dieser Wert wird so ermittelt, dass ein auf der dem LED-Panel angezeigtes Weiß gerade noch nicht im Bereich des Clippings ist. Dafür wird in DaVinci Resolve 17.4.2 2021 das Livebild der Kamera über die Inputkarte eingelesen. Dann wird es in einen Arbeitsfarbraum von Rec.709 gewandelt. Über den *Waveform-Monitor* können die Bildwerte betrachtet werden. Der variable ND-Filter wird entsprechend eingestellt. Das LED-Panel steht generell auf einer *Brightness* von 100%. Die Ausgabehelligkeit des Panels wird so nicht beschränkt und garantiert die beste Ausgabequalität.

Einrichtung: Inputkarte-Computer

Um das Videosignal der Kamera zu verarbeiten, muss es in den Computer eingelesen werden. Hierfür wird ein *UltraStudio Recorder 3G* verwendet. Die Kamera überträgt das Videosignal via 3G-SDI, 1080p in YCrCb 4:2:2. Die Daten der Inputkarte können über das Decklink Software Development Kit 10.11.2 2021 ausgelesen werden. Dessen Bibliothek ist allerdings in *C++* geschrieben und für Python 3.9.7 2021 ist zum Zeitpunkt der Versuche keine native Lösung verfügbar.

Aus diesem Grund wird das Videosignal (n Frames) mit Hilfe von FFMPEG 2021 und des

python-bindings ffmpeg-python 2021 als *ndarray* numpy 2021 eingelesen und zwischengespeichert.

Einlesen von Videosignalen via Python mit Decklink-Inputkarten

In *Algorithmus 13* wird die Funktion *getDecklinkFrames(frames=2, height=1080, width=1920, rawformat="yuv422p10")* genutzt um n Frames der Inputkarte auszulesen und als vierdimensionales *ndarray* numpy 2021 im RGB-Format auszugeben. Um die volle Bit-Tiefe von 10bit über die Verarbeitung hinweg zu erhalten, sind folgende Flags im *output* von ffmpeg-python 2021 zu verwenden: *format="rawvideo"* und *pix_fmt="rgb48le"*. Hierdurch wird das eintreffende Kamerasignal eingelesen, in ein RGB-Farbmodell gewandelt mit 16bit pro Kanal *(rgb48le)* und schließlich von FFMPEG 2021 in die Standardausgabe als *binary*-Buffer geschrieben. Die Ausgabe wird abgegriffen und in die Variable *out* zwischengespeichert. Mithilfe von *numpy.frombuffer(out, dtype=np.uint16)* numpy 2021 kann dieser Buffer in ein eindimensionales *ndarray* numpy 2021 eingelesen werden. Essentiell ist, dass *pix_fmt* und der angegebene *dtype* übereinstimmen. Schlussendlich wird das eindimensionale Array in die richtige Form, also *shape = (frames, height, width, 3)* gebracht und als Rückgabewert der Funktion ausgegeben.

5.2 Durchführung der Implementierung und Evaluierung der Methode

Im Folgenden soll der Algorithmus, der in Kapitel 3 beschriebenen Methode, überprüft werden. Dazu werden die Einzelkomponenten in verschiedenen Stufen im vorher beschriebenen Aufbau evaluiert.

5.2.1 Komponente: Matrixkorrektur und Matrixberechnung

Wie in Abschnitt 3.3.4 beschrieben, ist die Korrektur durch eine Matrixmultiplikation im Gesamtsystem essentiell. Hier soll die Funktion des erstellten Algorithmus' 9 verifiziert werden.

2D Testszenario

Die Implementierung der Korrektur soll überprüft werden, vorerst mit Testdaten. Folgende Annahme wird hierfür getroffen:

$$I_{cam}, I_{original}, I_{verify} \in \mathbb{R}^2$$

$$\text{Mit diesen Parametern: } M_T = \begin{pmatrix} 0.8 & 0.1 \\ 0.1 & 0.7 \end{pmatrix},\ M_R = \begin{pmatrix} 1 & 0.2 \\ 0.3 & 1 \end{pmatrix},\ I_{verify} = (0.8, 0.6)$$

In *Algorithmus 14* wird aus M_T und M_R mit der Funktion *numpy.linalg.slove(x,y)* von numpy 2021 M_{CC}^{-1} berechnet. Damit kann die Transformation von $I_{original}$ zu I_{cam} simuliert werden. In dem Beispiel werden die Transformationen von r und g aus M_T und M_R dargestellt. Außerdem wird p als I_{verify} zu y_p $I_{cam,verify}$ transformiert (siehe Abbildung 5.9). Für I_{mod} wird die inverse Matrix, also M_{CC}, berechnet, mit *numpy.linalg.inv(M)* aus numpy 2021. Die Matrix M_{CC} multipliziert mit I_{verify} ergibt I_{mod} x_p_mod. Wenn nun die selbe Transformation wie oben angewendet wird, gleichen sich $I_{cam,mod}$ y_p_mod und $I_{original}$ x_p. Visuell bestätigt wird der Sachverhalt in Abbildung 5.10 aus *Algorithmus 14*. Daraus lässt sich schließen, dass die beschriebene Methode in Python 3.9.7 2021 für zwei Dimensionen richtig implementiert ist.

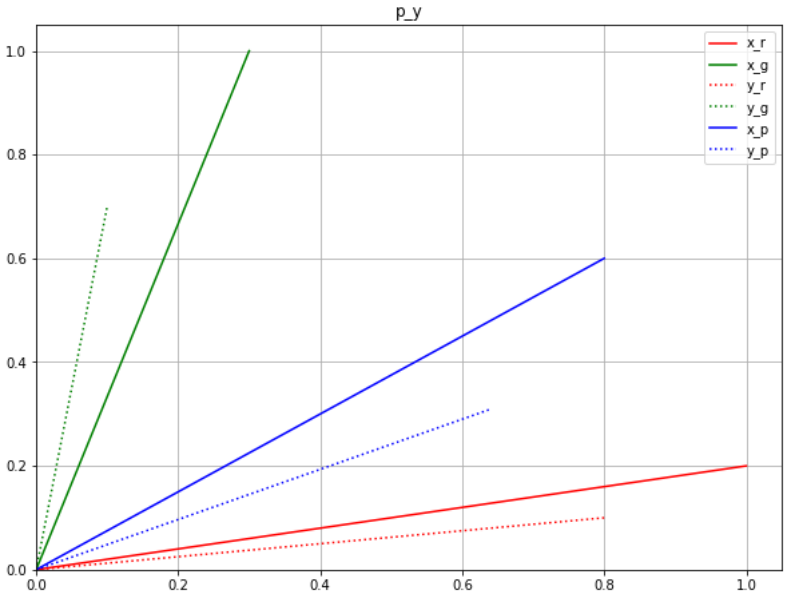


Abbildung 5.9: 2D Verifizierung der Methode, Darstellung der Referenz- und Testdaten Präfix x für alle $I_{original}$ und y für alle I_{cam}

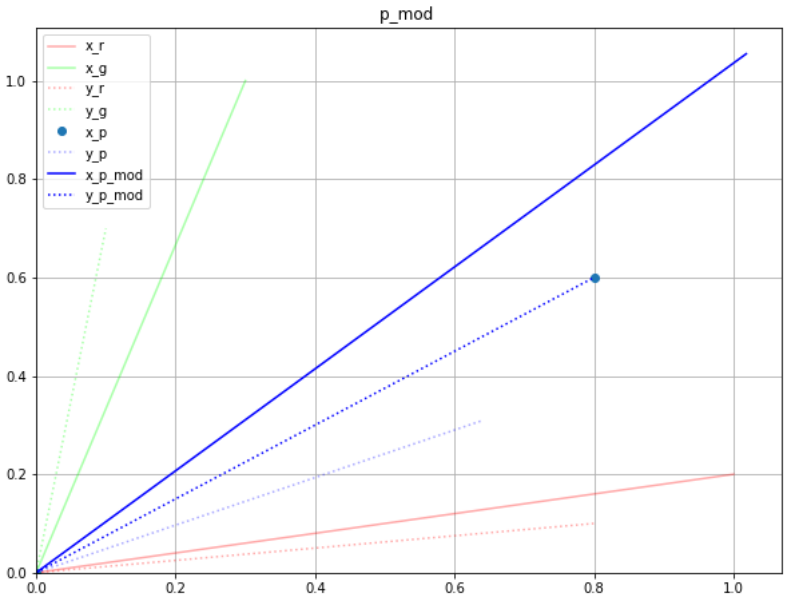


Abbildung 5.10: 2D Verifizierung der Methode, durchgeführte Korrektur in 2D-Raum I_{mod} für $I_{original} = x\_p$

3D Testszenario

Das gleiche Szenario wird in abgewandelter Form auf dreidimensionale Daten angewendet. Dieses Testszenario entspricht den Daten, die später real in den Algorithmus übergeben

werden, hier gilt:

$$I_{cam}, I_{original}, I_{verify} \in \mathbb{R}^3$$

$$\text{Mit diesen Werten: } M_T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, M_R = \begin{pmatrix} 0.8 & 0.1 & 0.05 \\ 0.1 & 0.7 & 0.21 \\ 0.1 & 0.1 & 0.95 \end{pmatrix}, I_{verify} = (0.8, 0.6, 0.4)$$

Hier ist dieselbe Funktion (siehe *Algorithmus 15*) *matrix_colour_correction_Cheung2004* aus colour 2021 so implementiert, wie sie im späteren Verlauf für die Anwendung im Real-Szenario genutzt wird.

Dadurch erkennbar ist in Abbildung 5.12, dass $I_{original}$ *x_p* dem Wert von $I_{cam,mod}$ *y_p_mod* gleicht. Daraus lässt sich schließen, dass diese Komponente (Korrekturalgorithmus) funktionsfähig ist. Solange die Abbildung $I_{original} \rightarrow I_{cam}$ durch eine Matrixmultiplikation beschrieben werden kann, gilt diese Berechnung.

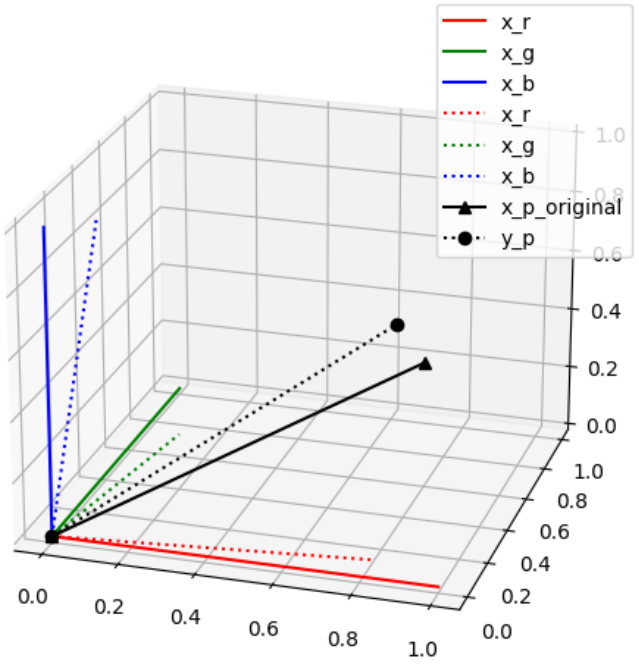


Abbildung 5.11: 3D Verifizierung der Methode, Darstellung der Referenz- und Testdaten, Präfix x für alle $I_{original}$ und y für alle I_{cam}

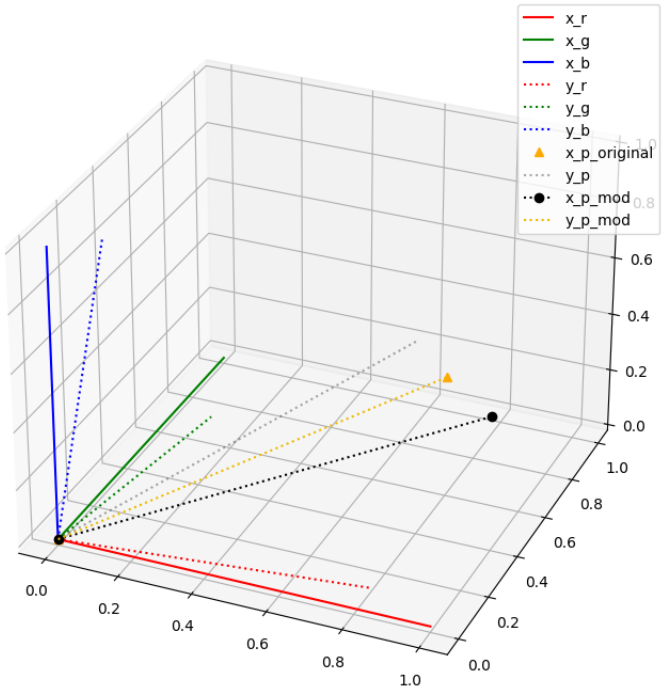


Abbildung 5.12: 3D Verifizierung der Methode, Darstellugn der Korrektur, $I_{mod} = x\_p\_mod$, $I_{original} = x\_p\_original$ und $I_{cam,neu} = y\_p\_mod$

5.2.2 Komponente: Decklink-Inputkarte in Python

Die Verarbeitung der Bilddaten, wie in Abschnitt 5.1.4 beschrieben, muss ebenso verifiziert werden. Die Art und Weise, wie Video-Frames über FFMPEG 2021 in Python 3.9.7 2021 eingelesen werden, wird überprüft.

Dafür wird mit der in Abschnitt 5.1.4 erwähnten Methode ein Video-Frame aufgenommen und gespeichert. Das geschieht ebenfalls mit Hilfe eines zweiten Programms, hier *DaVinci Resolve 17.4.2 2021*. Das wird für insgesamt drei verschiedene Grautöne auf dem LED-Panel wiederholt. Beide Bilddaten liegen nun pro Pixel $p_{i,j}$ an der Stelle i, j in der Form $p \in \mathbb{R}^3$, also *RGB* vor. Hieraus soll jeweils der Median berechnet werden. Der Bildauschnitt, in Abschnitt 3.3.2 erläutert, wird mit *80x80 px* gewählt. Das Videosignal ist Gamma-kodiert, in *S-log2*. Die Rückgabewerte der Median-Berechnungen finden sich in Tabelle 5.1. Sowohl DaVinci Resolve 17.4.2 2021, als auch FFMPEG 2021 exportieren für diesen Versuch die Daten kodiert in 8bit.

Tabelle 5.1: Frame aufgenommen aus *DaVinci* und *FFMPEG*

Programm	**Messung 1**	**Messung 2**	**Messung 3**
DaVinci	126 126 127	100 97 99	66 64 67
FFMPEG	127 126 127	100 98 100	67 64 68

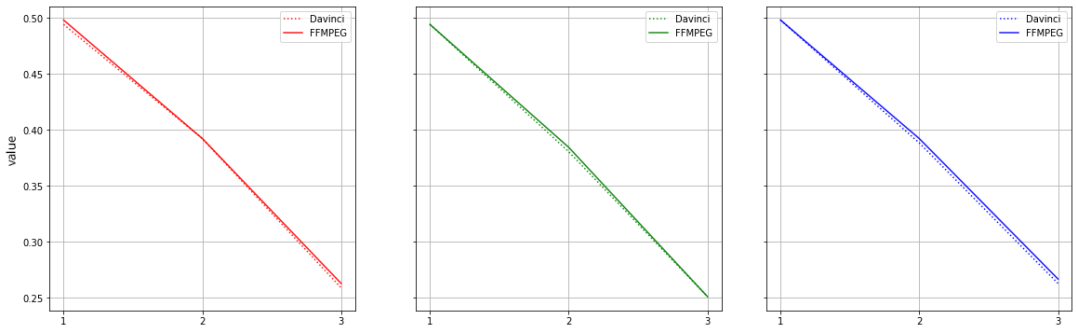


Abbildung 5.13: Daten aus Tabelle 5.1, Bilddaten aus *DaVinci* zu *FFMPEG*

Die Daten in Tabelle 5.1, visualisiert in Abbildung 5.13, geben Aufschluss darüber, dass die Nutzung und Implementierung von FFMPEG 2021 funktionsfähig ist. Die leichten Schwankungen in einzelnen Kanälen können mit dem relativ klein gewählten Ausschnitt von *80x80px* und dem damit einhergehenden Rauschen begründet werden. Die Bilddaten können nicht zeitgleich von FFMPEG 2021 und DaVinci Resolve 17.4.2 2021 verarbeitet werden.

10bit aus Bilddaten

Die Methode, um Videodaten in 8bit von der Inputkarte abzugreifen, scheint funktionsfähig. Im nächsten Schritt muss verifiziert werden, dass die Methode auch die volle Bit-Tiefe von 10bit über den Verarbeitungsprozess hinweg erhält. Für die Bestätigung wird die Annahme getroffen, dass die geringste Wertedifferenz innerhalb eines Bilds häufig zwischen Pixeln und ihren Nachbarpixeln ist.
Wenn mit einem in 16bit kodierten Pixelwert eines Kanals $p_{i,j} \in \{0, 1, 2..., 65535\}$ in der Zeile i an der Stelle j Folgendes gilt:

$$63 < |p_{i,j} - p_{neighbor}| < 256,$$

dann ist der Werteabstand kleiner als das, was ein ursprünglich 8bit tief kodiertes Signal, das auf 16bit hoch skaliert wird, auflösen kann. Somit kann durch diese Methode bestätigt werden, dass ein Bild ursprünglich in 10bit kodiert war und erhalten wird. In Python 3.9.7 2021 wird die Differenz zwischen Nachbarpixeln mit der Methode *numpy.diff()*, aus numpy 2021, implementiert, siehe *Algorithmus 16*. Hier werden nur die unmittelbaren Nachbarpixel geprüft, nicht die diagonal liegenden. Es wird separiert zeilenweise und spaltenweise überprüft, wie groß die Differenz ist. Falls diese größer als 63 und kleiner als 256 ist, wird in einem Zähler um Eins hochgezählt. Standardmäßig wird nur der Grün-Kanal überprüft.

Ein aufgenommener Videoframe, mit der in 5.1.4 erläuterten Methode über FFMPEG 2021, ergibt *330 242* Unterschiede im Grün-Kanal, *63 322* im Rot-Kanal und *175 969* im Blau-Kanal, die zu klein für ein 8bit kodiertes Bild wären.

Im Umkehrtest wird die in 5.1.4 erwähnte Flag für *pix_fmt* auf *"rgb24le"* gesetzt und numpy 2021 interpretiert den Buffer als *uint8*. Dieses *ndarray* wird dann auf *dtype=uint16* hoch skaliert. Die Methode aus *Algorithmus 16* ergibt nun *0* Unterschiede in jedem Kanal. Das bestätigt, dass die Kameradaten 10bit kodiert erhalten in Python 3.9.7 2021 als *ndarray* vorliegen.

5.2.3 Komponente: Median - Rauschminimierung

Die in Abschnitt 3.3.2 erwähnte Funktion, um aus einem bzw. mehreren verrauschten Video-Frames den Wert $I_{cam} \in \mathbb{R}^3$ zu bestimmen, soll hier verifiziert werden. Die Ausschnitts-Größe von *80x80px* wird nach mehreren Versuchen auf *200x200px* angepasst.
Hier bleibt der Wert stabil über drei Messreihen zu je 50 Messungen mit *frames=1*, siehe Abbildung 5.14. Daraus lässt sich schließen, dass ein Frame und die angepasste Größe des Bildausschnitts *(200x200px)* beides ausreichend ist.

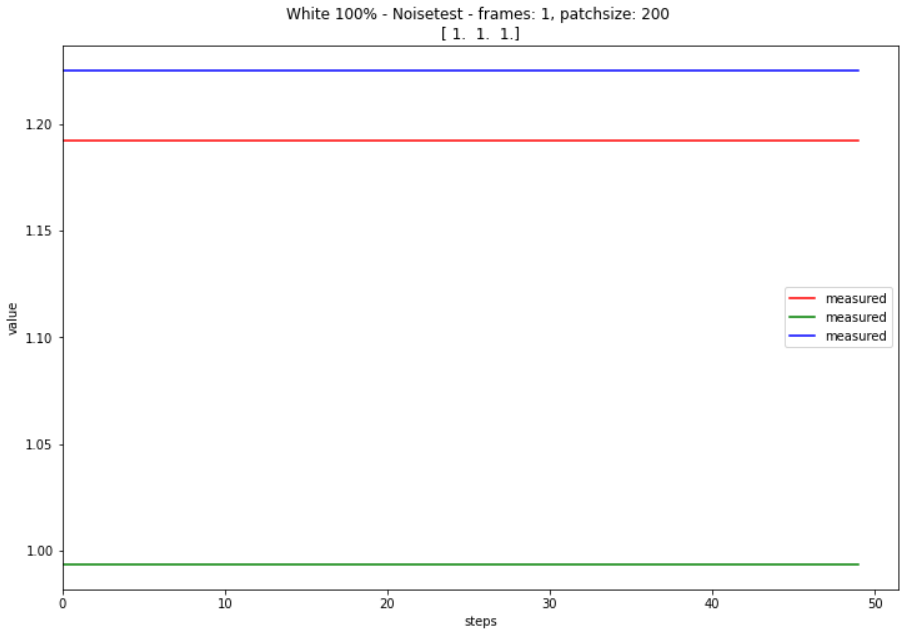


Abbildung 5.14: Veränderung des Medians (linearer Arbeitsraum) aus einem Bildausschnitt von *200x200px* für 50 aufgenommene Frames

5.2.4 Komponente: Linearer Arbeitsraum - Linearität zwischen $I_{original}$ und I_{cam}

Wie in Abschnitt 3.3.3 erklärt, müssen, damit $I_{original}$ und I_{cam} im linearen Arbeitsraum vorliegen, der Input des Computers und der Output des Computers Gamma-dekodiert bzw. -kodiert werden. Die richtige Anwendung dieser Funktionen soll hier verifiziert werden: Für die Abbildung $f : I_{original} \rightarrow I_{cam}$ muss mit $c \in \mathbb{R}$ als Konstante Folgendes gelten:

$$f(c \cdot I_{original}) = c \cdot I_{cam}$$

Das heißt, wenn $I_{original}$ mit dem Faktor $c = 0.5$ multipliziert und ausgegeben wird, dann kann, falls I_{cam} und $I_{original}$ bekannt sind, $I_{cam,neu}$ vorausgesagt werden. Folglich sollten sich $I_{cam,neu}$ und I_{cam}, multipliziert mit demselben Faktor c, gleichen.

$$I_{cam,neu} = c \cdot I_{cam}$$

Dieses Verhalten soll mit verschiedenen Farbwerten für $I_{original} \in \mathbb{R}^3$ überprüft werden. Dafür wird zuerst der Wert $I_{original}$ ausgeben und I_{cam} zwischengespeichert. Für n Schritte der Konstante c zwischen 0 und 1 $c \in [0, 1]$, wird $I_{original}$ mit c multipliziert und ausgege-

ben. Für jede der n Konstanten c_i wird der Input vorausgesagt $I_{predicted,c_i} = c_i \cdot I_{cam}$ und gemessen $I_{measured,c_i}$. Wenn die Werte von $I_{predicted,c_i}$ und $I_{measured,c_i}$ immer übereinstimmen, kann angenommen werden, dass beide Daten im linearen Arbeitsraum vorliegen und $f : I_{original} \to I_{cam}$ linear ist.

Linearitäts-Versuch 1

Für Weiß $I_{original} = (1, 1, 1)$ wird dieser Versuch mit $n = 4$ durchgeführt, siehe Abbildung 5.16. Hierfür wird der LED-Controller und das LED-Panel frisch konfiguriert. Für die Messreihe aus Abbildung 5.16 wird die Standard-Einstellungen so belassen, wie sie sind.

Für die Messreihe in Abbildung 5.17, wird die *Brightness*-Einstellungen in (NovaLCT 5.4.3 2021) so eingestellt, dass I_{cam} nahezu $I_{original}$ gleicht, siehe Abbildung 5.15.
Wie in den Abbildungen zu sehen, stimmt die Vorhersage in beiden Fällen nahezu mit dem Gemessenen überein. Deswegen ist davon auszugehen, dass die Methode zur Linearisierung aus Abschnitt 3.3.3 für $I_{original} = (1, 1, 1)$ wirksam ist.

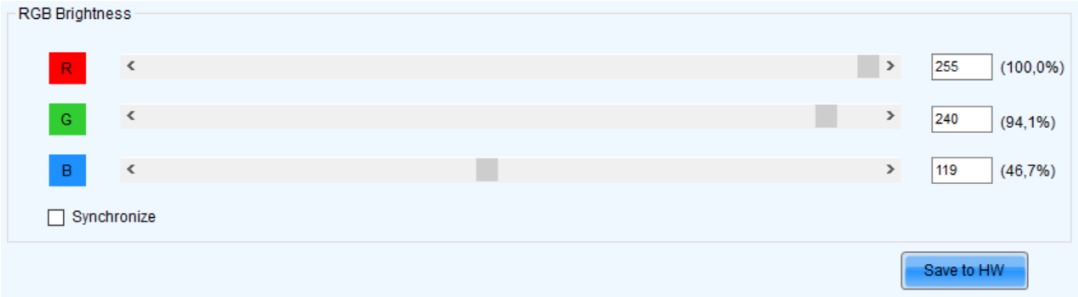


Abbildung 5.15: RGB Brightness - Einstellungen in (NovaLCT 5.4.3 2021)

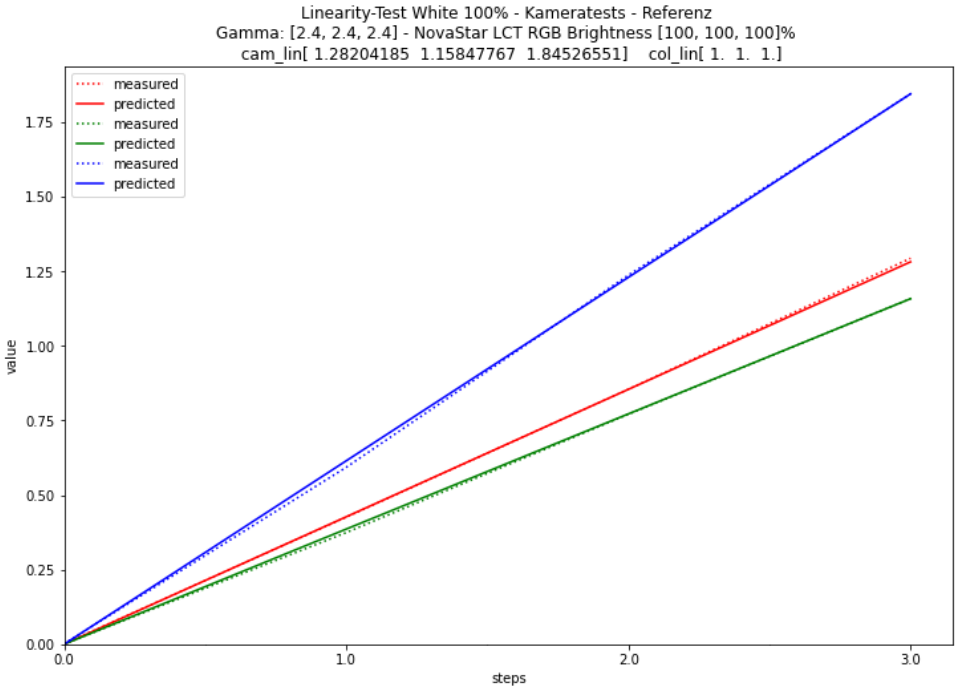


Abbildung 5.16: Linearitäts-Nachweis: Brightness auf 100% für R,G,B in NovaLCT 5.4.3, *gemessen* zu *vorausgesagt*

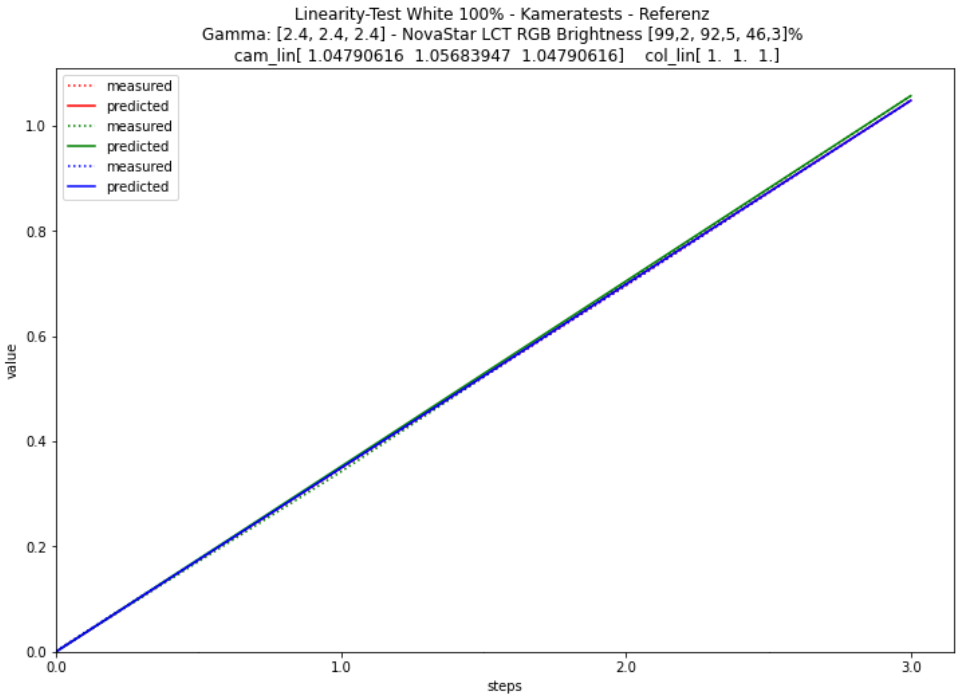


Abbildung 5.17: Linearitäts-Nachweis: Brightness auf 99.2%, 92.5%, 46,3% für R,G,B in NovaLCT 5.4.3, *gemessen* zu *vorausgesagt*

Linearitäts-Versuch 2

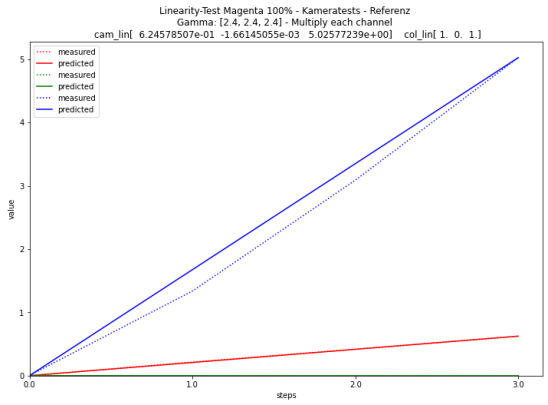
In einem weiteren Versuch werden mit den Einstellungen, wie in Abbildung 5.15 *NovastarLCT Einstellungen zu Brightness*, nicht nur Graustufen ausgegeben. Der Versuch wird durchgeführt für:

$$I_{original} = \{(1,0,1),(1,1,0),(0,1,1),(1,0,0),(0,1,0),(0,0,1)\},$$

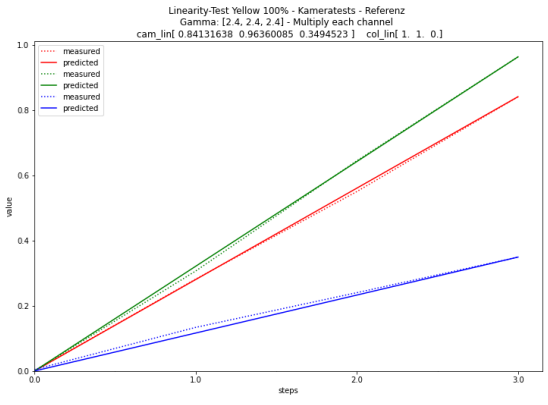
mit $n = 3$ für alle Messungen. Die Messung für Grün wird mit $n = 5$ wiederholt um das, in den Daten der Messreihe besonders erkennbare, Artefakt genauer aufzulösen und um einen Messfehler auszuschließen (siehe Abbildung 5.18).
Zu erwarten war, dass wie in Abbildung 5.18(b) für Gelb erkennbar, die spektralen Bänder der einzelnen SMD-RGB-LEDs nicht mit denen der Farbfilter des Sensors übereinstimmen. Daraus folgt, wie in diesem Beispiel zu sehen ist, dass der Blau-Kanal der Kamera mit angesprochen wird, auch wenn auf dem LED-Panel keine blaue LED aktiv ist. Wie in den Daten zu erkennen ist, verhalten sich allerdings unterschiedliche ausgegebene Farben verschieden über deren Helligkeitsabstufungen. So ist für den Blau-Kanal ein konträres Verhalten über die Helligkeitsabstufungen bei Magenta und Cyan zu erkennen. Besonders auffällig ist allerdings der Grünkanal, der für Weiß in den vorherige Versuchen und für Gelb in diesem Versuch, sehr gut vorausgesagt werden konnte. Jedoch deutet der Grünkanal für Cyan und Grün eine Art Gammakurve an.

Dieses nicht lineare Verhalten ist nach angezeigtem Farbwert unterschiedlich nichtlinear. Das deutet darauf hin, dass in diesem Aufbau keine zufriedenstellende allgemeingültige Korrektur möglich sein wird.
Die Ursache für dieses Verhalten kann mit den Mitteln des Versuchsaufbaus nicht eingegrenzt werden. Da dieser Versuch für Weiß funktioniert, kann daraus geschlossen werden, dass die unterschiedliche Verarbeitung zwischen Output und Input des Computers passiert.
Im nächsten Schritt müsste verifiziert werden, dass die ausgegebenen Farbwerte und Helligkeitsabstufungen am Ausgang so Gamma-kodiert werden, dass sie so im linearen Arbeitsraum vorliegen, wie der LED-Controller die SMD-RGB-LEDs linear ansteuert. Dies ließe sich mit einem Spektrometer oder einem anderen Messinstrument untersuchen, das

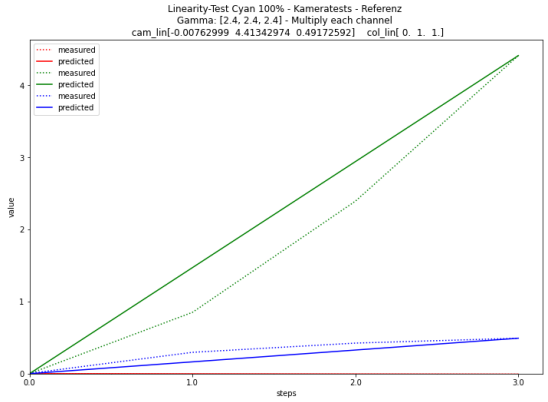
die Strahldichte pro Wellenlänge L_e, λ messen kann. Für die in Kapitel 5 beschriebene Versuchsreihe zur Implementierung und Evaluierung der Methode stand dies nicht zur Verfügung.



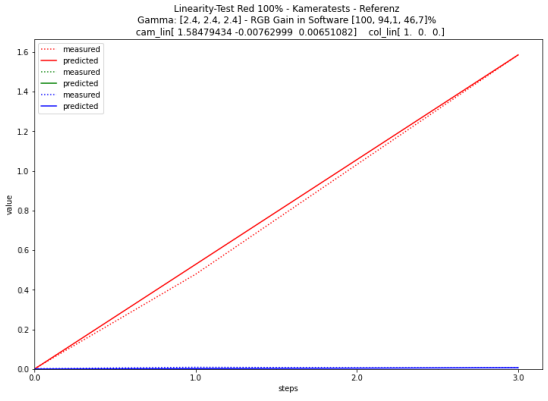
(a) $I_{original} = (1, 0, 1)$, $n = 3$



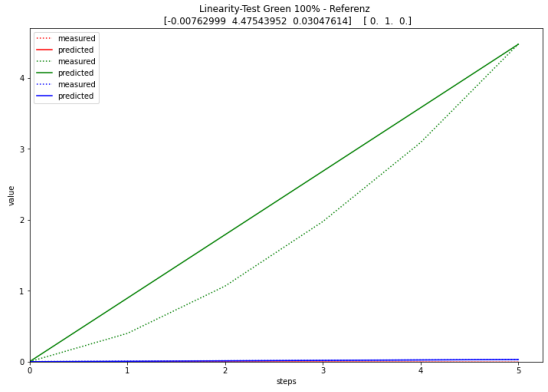
(b) $I_{original} = (1, 1, 0)$, $n = 3$



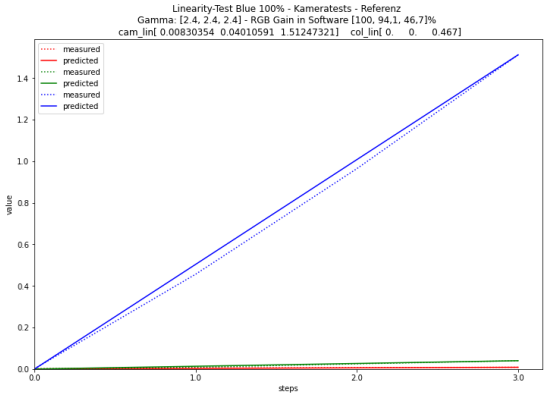
(c) $I_{original} = (0, 1, 1)$, $n = 3$



(d) $I_{original} = (1, 0, 0)$, $n = 5$



(e) $I_{original} = (0, 1, 0)$, $n = 5$



(f) $I_{original} = (0, 0, 1)$, $n = 5$

Abbildung 5.18: Linearitäts-Nachweis für verschiedene Werte $I_{original}$, *gemessen* zu *vorausgesagt*

Linearitäts-Versuch 3

Um eine Korrekturmatrix M_{CC} deterministisch zu berechnen, sind nur drei Testsamples notwendig. (Westland, Ripamonti und Cheung 2012d) Aus den erhobenen Daten des vorherigen Versuchs ist zu erkennen, dass sich die Ausgabefarben *Rot* und *Blau* annähernd linear verhalten (siehe Abbildung 5.18). Daraus lässt sich die Vermutung ableiten, dass es möglich sein könnte, die Gammakodierung für den Grün-Kanal zu verändern, um für *Grün* ein lineares Verhalten zu erwirken. Deshalb wird die Funktion um $I_{original}$ zu Gamma-kodieren erweitert. Die Kanäle Rot und Blau werden weiterhin mit $\gamma = 2.4$ kodiert. Der Grün-Kanal wird jedoch in diesem Versuch mit $\gamma_1 = 2.4, \gamma_2 = 3.9$ kodiert.

Die Ergebnisse in Abbildung 5.20 mit einem Gamma $\gamma = 3.9$ für Grün deuten auf ein lineares Verhalten hin. Diese Einstellung erwirkt allerdings das gewünschte Verhalten nur für die Ausgabe von Helligkeitsabstufungen Grün *(0,1,0)*, nicht für Weiß *(1,1,1)* oder andere Farbwerte. Durch die angepasste Gammakodierung der Ausgabe, lässt sich der Grün-Kanal für die Helligkeitsabstufungen unter anderem von Weiß nicht mehr linear vorhersagen.

Daraus folgt, dass nur für Rot, Grün und Blau als Ausgabe-Farbwerte unter der Verwendung der erweiterten Gammakodierung ein lineares Verhalten nachgewiesen werden kann. Deswegen werden die Versuche nur mit diesen Ausgabe-Farbwerten durchgeführt mit der erweiterten Gammakodierung. Das hat zur Folge, dass die Aussagekraft über die allgemeine Funktionsfähigkeit des Algorithmus' zur Korrektur eingeschränkt ist. Da unter anderem zur Verifizierung keine weiteren Farbwerte als I_{verify} ausgegeben werden, wie es in Abschnitt 3.4 angedacht ist.

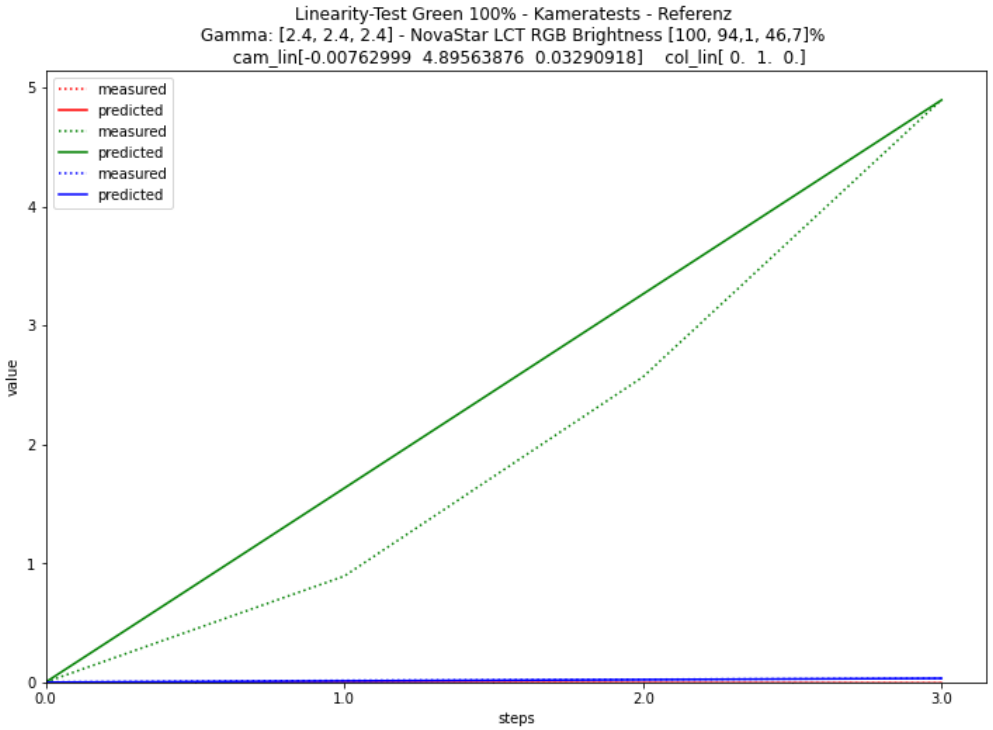


Abbildung 5.19: Linearitäts-Nachweis, Gamma des Ausgabebilds im G-Kanal: 2.4, Helligkeitsabstufungen von (0,1,0)

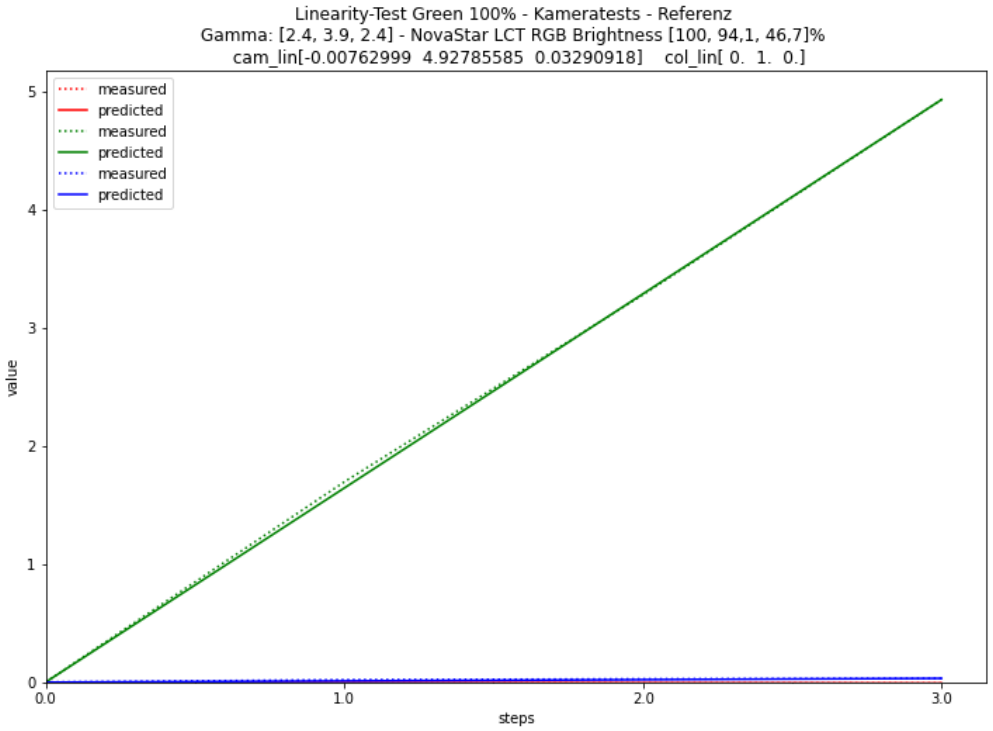


Abbildung 5.20: Linearitäts-Nachweis, Gamma des Ausgabebilds im G-Kanal: 3.9, Helligkeitsabstufungen von (0,1,0)

5.2.5 Komponente: Farbkorrektur, Berechnung von I_{mod}

Um die Möglichkeiten der Korrektur über verschiedene Blickwinkel zu evaluieren, muss die Funktion für die Korrektur selbst verifiziert werden. Die in Abschnitt 3.3.4 beschriebene Methode zur Korrektur wird in Abschnitt 5.2.1 theoretisch verifiziert. Folgend soll diese Methode für einen Blickwinkel $\alpha = 0°$ angewendet werden. Aus den vorangegangenen Evaluierungen der Einzelkomponenten des Workflows ergeben sich für den Versuch die Testsamples:

$$I_{original} = \{(1,0,0),(0,1,0),(0,0,1)\}$$

Wie in Abschnitt 3.3.4 *(Korrekturberechnung)* erklärt, wird für jeden Wert von $I_{original}$ eine Messung I_{cam} gemacht. Aus diesen gesammelten Test- und Referenzsamples M_T, M_R kann die Korrekturmatrix M_{CC} errechnet werden. Diese wird angewendet, siehe *Algorithmus 8*, um I_{mod} auf dem LED-Panel darzustellen. Nun wird die Differenz pro Kanal zwischen $I_{original}$ und $I_{cam,neu}$ berechnet. Das ist der absolute Fehler dieser Messung pro Farbkanal, siehe Abbildung 5.21 *(b, d, f)*. Die Summe des absolute Fehlers *SAE - sum of absolute error* pro Farbkanal, ist die Summe dieser Differenzen. Das Gleiche wird mit $I_{original}$ und I_{cam} durchgeführt, der hieraus errechnete absolute Fehler pro Messung bzw. die *SAE* muss durch die Korrektur signifikant verbessert werden.

Farbkorrektur 1 - $I_{original}$ aus Rot, Grün, Blau

Aus Abbildung 5.21 ist klar zu erkennen, dass die SAE durch die in Abschnitt 3.3.4 *(Korrekturberechnung)* beschriebene Methode verringert werden konnte, unter der oben genannten Einschränkung für M_R.

Die *SAE* pro Farbkanal, ist in jedem Kanal signifikant gesunken zwischen der ersten Messung *ohne Korrektur* und der zweiten Messung *mit Korrektur*. Der *SAE* ist im Rotkanal von 1.072 zu $1.199 \cdot 10^{-1}$, im Grünkanal von 3.681 zu $7.987 \cdot 10^{-2}$ und im Blaukanal von $5.686 \cdot 10^{-1}$ zu $6.670 \cdot 10^{-2}$ gesunken.

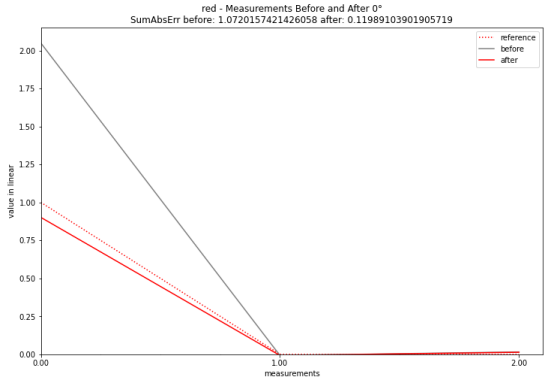
Mit den Ergebnissen aus diesem Versuch kann, unter denselben Bedingungen, die Möglichkeit einer blickwinkelabhängigen, interpolierten Korrektur, nach der in Abschnitt 3.3.7 *(blickwinkelabhängige Korrektur)* beschriebenen Methode, getestet und deren Funktionalität evaluiert werden.

Farbkorrektur 2 - $I_{original}$ aus sechs zufälligen Farbwerten

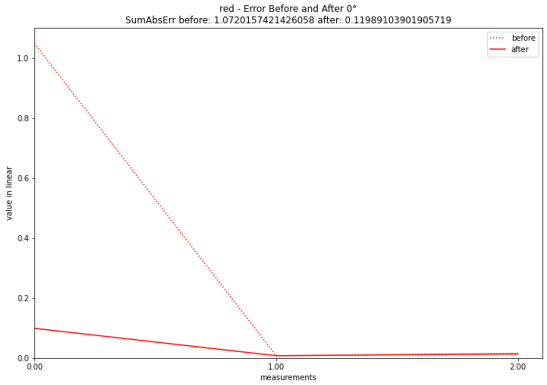
Nach der signifikanten Verbesserung der *SAE* für $(1,0,0),(0,1,0),(0,0,1)$ als $I_{original}$, steht die Frage im Raum: Kann es trotz der Ergebnisse aus Abschnitt 5.2.4 *Evaluierung der Linearisierung* möglich sein, für andere Farbwerte als $(1,0,0),(0,1,0),(0,0,1)$ eine Korrektur durchzuführen? Dafür werden zufällig sechs Farbwerte generiert.

$$I_{original} = \{(0.553, 0.950, 0.758), (0.835, 0.324, 0.225), (0.005, 0.301, 0.792),$$
$$(0.519, 0.011, 0.141), (0.262, 0.360, 0.437), (0.886, 0.232, 0.874)\}$$

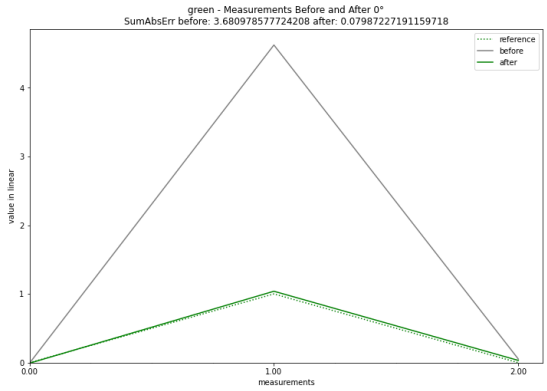
Die *SAE* verändert sich hier nicht signifikant. Im Rotkanal verschlechtert er sich sogar von 0.9580 zu 1.086. In jedem Fall ist für diesen Versuch bei mehr als der Hälfte der Messungen keine Verbesserung durch die Korrektur erkennbar. Daraus muss für die folgenden Versuche gefolgert werden, entsprechend der Empfehlung aus Abschnitt 5.2.4 *(Evaluierung der Linearisierung)*, ausschließlich mit $(1,0,0),(0,1,0),(0,0,1)$ für $I_{original}$ zu arbeiten.



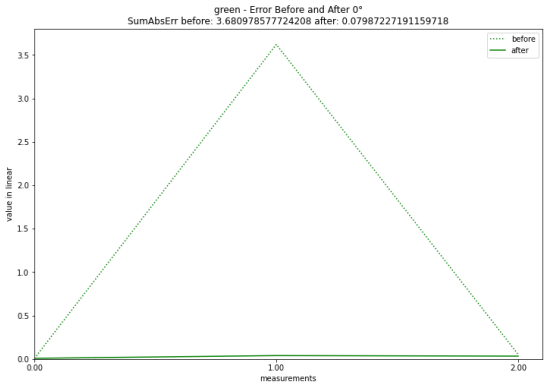
(a) Rotkanal: $I_{original}$ (gepunktet) zu I_{cam} und $I_{cam,neu}$



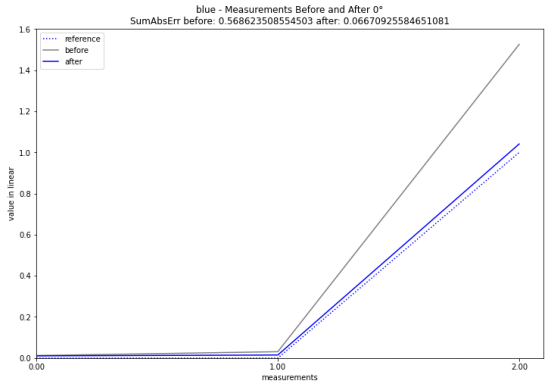
(b) Rotkanal: Absoluter Fehler I_{cam} vs. $I_{cam,neu}$



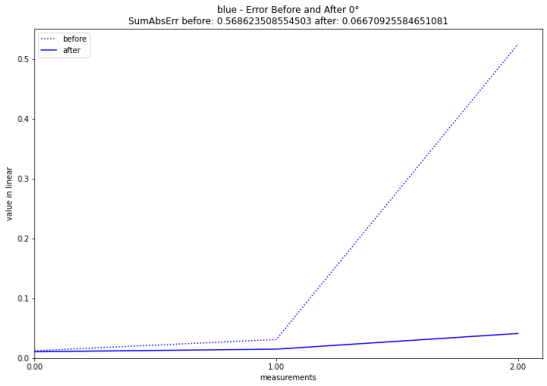
(c) Grünkanal: $I_{original}$ (gepunktet) zu I_{cam} und $I_{cam,neu}$



(d) Grünkanal: Absoluter Fehler I_{cam} vs. $I_{cam,neu}$

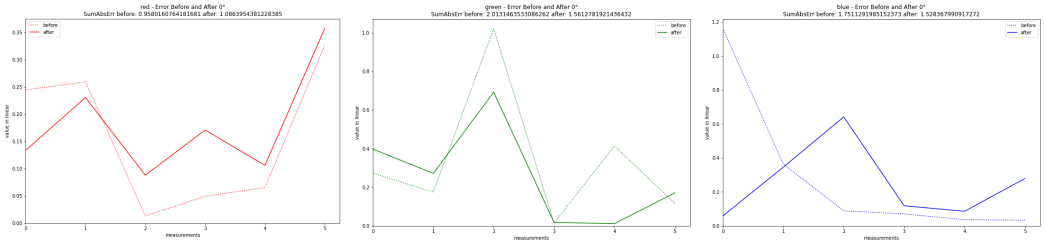


(e) Blaukanal: $I_{original}$ (gepunktet) zu I_{cam} und $I_{cam,neu}$



(f) Blaukanal: Absoluter Fehler I_{cam} vs. $I_{cam,neu}$

Abbildung 5.21: Farbkorrektur 1: $\alpha = 0°$ mit Rot, Grün, Blau
Werte vor und nach der Korrektur und absoluter Fehler



(a) Rotkanal (b) Grünkanal (c) Blaukanal

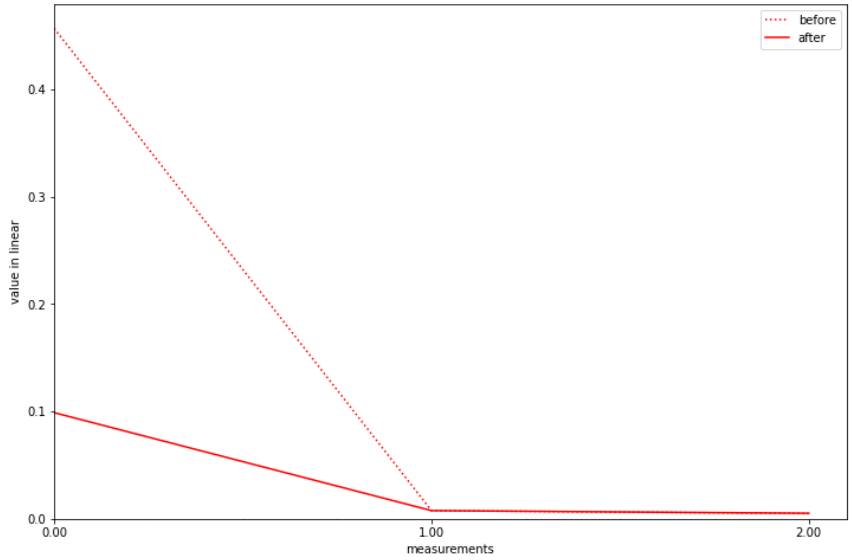
Abbildung 5.22: Farbkorrektur 2: Absoluter Fehler vor (gepunktet) und nach der Korrektur mit sechs verschiedenen Farbwerten für $I_{original}$

5.2.6 Evaluierung Gesamtsystem: blickwinkelabhängige Korrektur

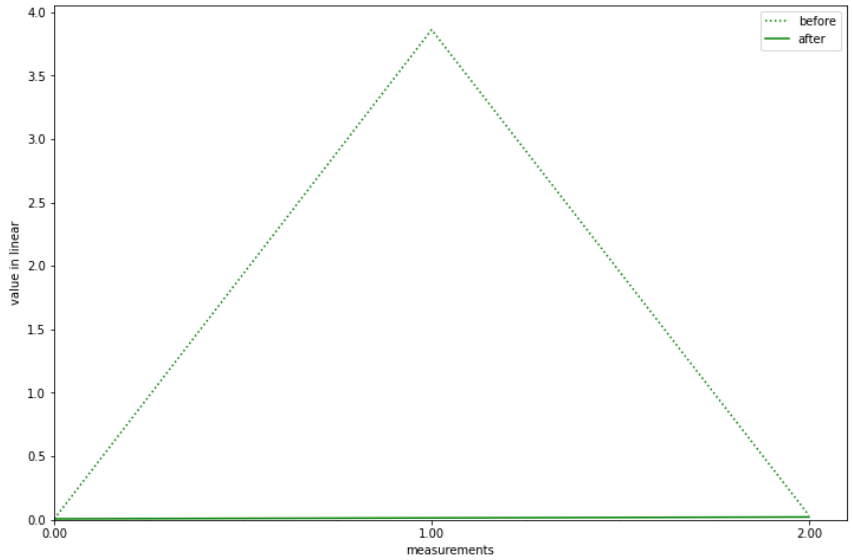
Wie in Abschnitt 3.3.7 *blickwinkelabhängige Korrektur* beschrieben, soll eine Korrektur für einen Blickwinkel α berechnet werden, der nicht vorher eingemessen wird. Die Korrektur bzw. $I_{mod,\alpha}$ soll aus der Korrektur der zwei nächsten Blickwinkeln α_0, α_1 linear interpoliert werden. Die Referenz-Farbwerte seien: $I_{original} = \{(1,0,0),(0,1,0),(0,0,1)\}$
Die Angaben für den Blickwinkel α werden entsprechend Abbildung 4.3 in den Algorithmus eingetragen. Eingemessen werden soll das System für $\alpha_{messung} = \{0°, -20°, -40°\}$. Der nicht eingemessene Blickwinkel α für die Interpolation sei $\alpha = -30°$. Die Werte können vom Winkelmesser unterhalb des Drehtellers abgelesen werden.
Aus den Messungen wird in der Interpolation die Korrektur für die beiden nächsten Nachbarn berechnet und, entsprechend des Blickwinkels, linear interpoliert. Im Gegensatz zu Abschnitt 5.2.5 *(Evaluierung der Korrektur)* wird der dargestellte Farbwert I_{mod} nicht für diesen Blickwinkel α speziell gemessen.

blickwinkelabhängige Korrektur inkl. Interpolation: Versuch 1 $\Delta\alpha = 20°$

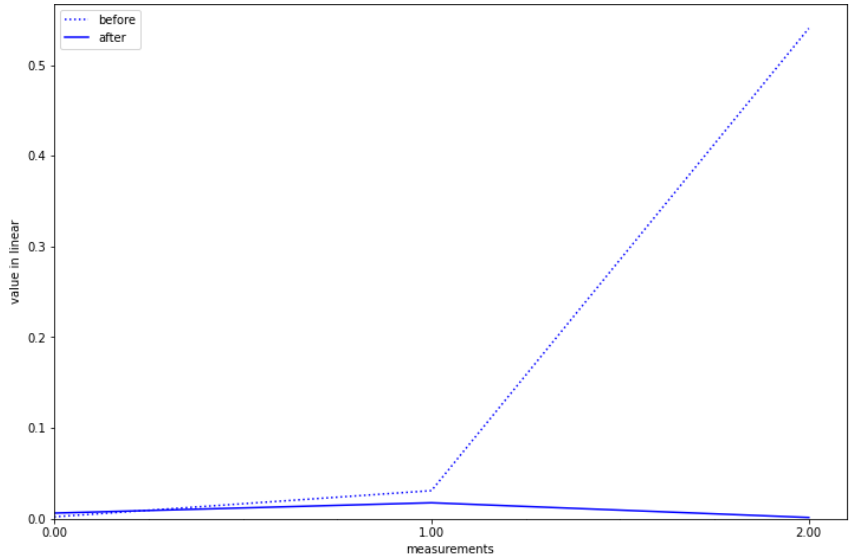
Die beiden nächsten Blickwinkel werden errechnet und ergeben korrekterweise
$\alpha_0 = -20°, \alpha_1 = -40°$. Um die Funktionsweise des Algorithmus zu verifizieren, wird in Abbildung 5.23 der Absolute Fehler pro Kanal für I_{cam} und $I_{cam,neu}$, also vor und nach der Korrektur, dargestellt. Der *SAE* hat sich pro Farbkanal vor und nach der Korrektur signifikant verändert:
Rot: $4.693 \cdot 10^{-1}$ zu $1.116 \cdot 10^{-1}$
Grün: 3.903 zu $4.450 \cdot 10^{-2}$
Blau: $5.735 \cdot 10^{-1}$ zu $2.529 \cdot 10^{-2}$



(a) Rotkanal



(b) Grünkanal



(c) Blaukanal

Abbildung 5.23: Interpolation 1: Absoluter Fehler vor und nach der Interpolation $\alpha = -30°$ interpoliert aus $-20°$ und $-40°$

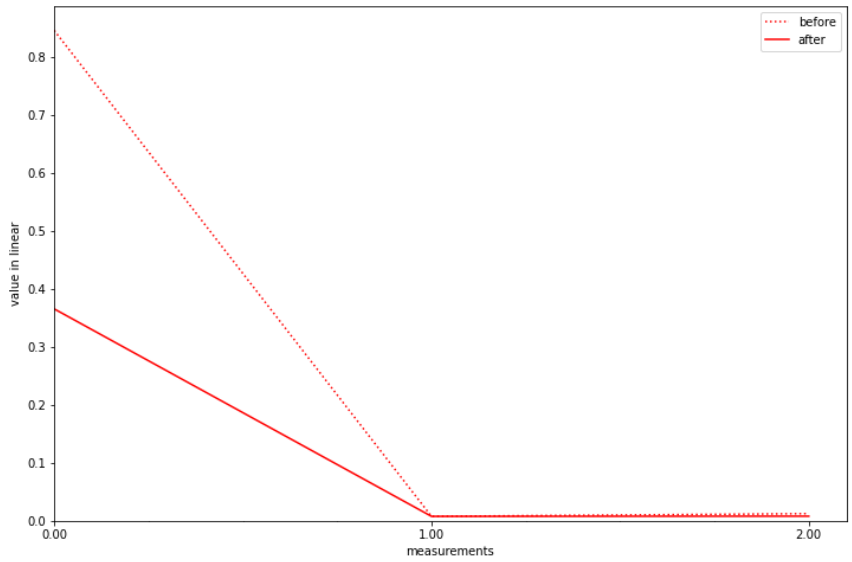
Vergleich mit und ohne Interpolation

Im Kontext dieser signifikanten Veränderung ist es interessant die Verbesserung dieses Versuches mit derjenigen zu vergleichen, die ohne Interpolation erreichbar ist. Für dieses Beispiel, wenn direkt für den Blickwinkel $\alpha = -30°$ eingemessen wird, siehe Abbildung 8.1 im Anhang. In einem vorherigen Versuch wird dieser Blickwinkel selbst eingemessen.
Die Messungen hier sind allerdings unter dem Gesichtspunkt zu betrachten, dass das LED-Panel für diesen früheren Versuch schon deutlich *(8 Stunden)* länger im aktiven Betrieb war. Aus diesem Grund sind Unterschiede der Strahldichteverteilung auf Grund der Temperatur erwartbar, siehe Schanda, Muray und Kranicz 2015. Der Effekt der temperaturabhängigen Veränderung wird kurz in Abschnitt 2.3.2 umrissen.
Aus diesem Grund lassen sich die Werte nicht direkt vergleichen. Das Verhältnis zwischen der jeweiligen *SAE* davor und danach sollte jedoch eine Aussage darüber erlauben, ob die Korrektur durch die Interpolation signifikant schlechter ist, siehe Tabelle 5.2.

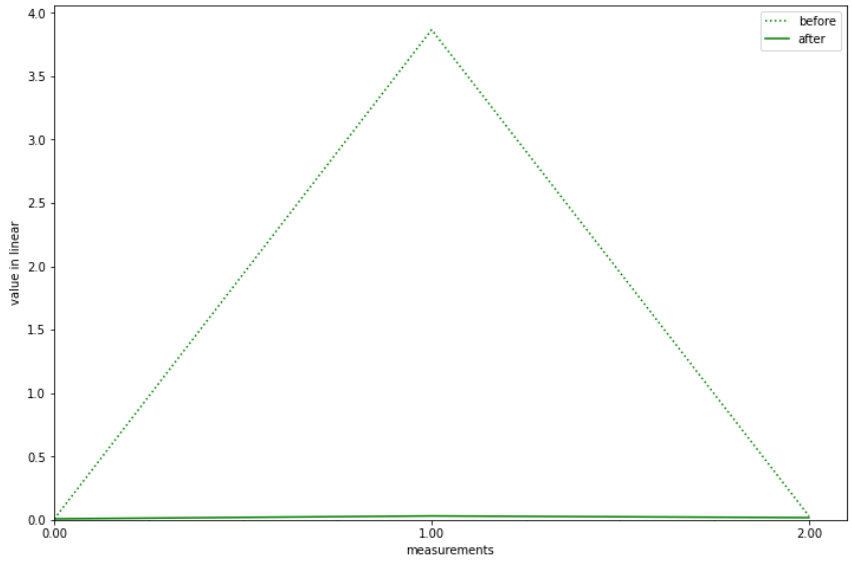
Aus diesen Daten wird ersichtlich, dass die hier verwendete Methode zur Interpolation vergleichbar mit der Messung für den Winkel selbst ist. Daraus kann abgeleitet werden, dass die Interpolation in diesem Fall funktioniert.

blickwinkelabhängige Korrektur inkl. Interpolation: Versuch 2 $\Delta\alpha = 40°$

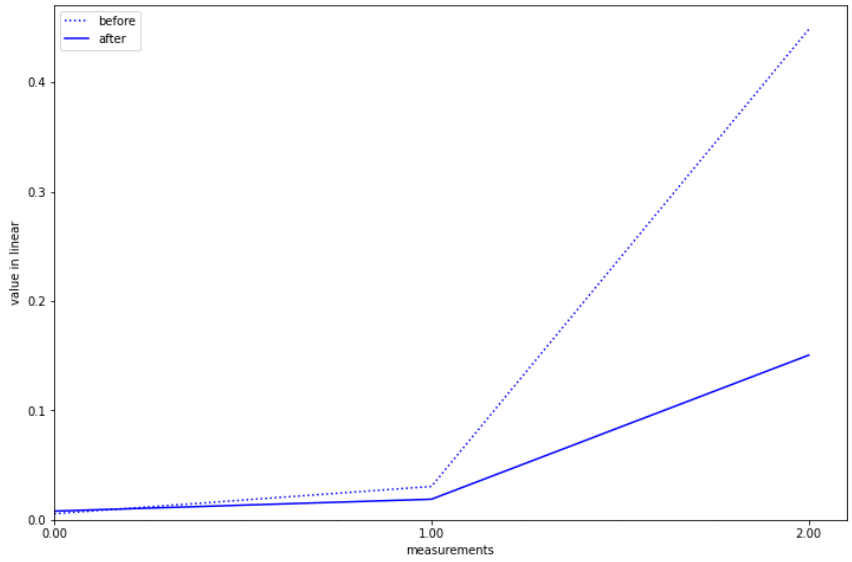
Um die Grenzen dieses Systems zu testen soll nun aus den gemessenen Blickwinkel $\alpha_0 = 0°, \alpha_1 = -40°$ der Wert für I_{mod} für $\alpha = -30°$ interpoliert werden. Die Ergebnisse finden sich in Abbildung 5.24
Hier wird ebenso die Verbesserung der *SAE* berechnet und verglichen, siehe Tabelle 5.3. Zu erkennen ist, dass es einen signifikanten Unterschied zwischen *Interpolation 1* und *Interpolation 2* gibt, außer für die Verbesserung im Grünkanal. Daraus lässt sich trotzdem schließen, dass die Methode zur Interpolation funktionsfähig ist.
Die Güte der Interpolation hängt allerdings davon ab, wie gut die initiale Korrektur pro Blickwinkel ist und in welchem Abstand $\Delta\alpha$ Messungen pro Winkel durchgeführt werden. Bei der *Interpolation 1* zeigt die Datenlage. dass eine Auflösung mit $\Delta\alpha = \pm 20°$ jedoch als ausreichend ist.



(a) Rotkanal



(b) Grünkanal



(c) Blaukanal

Abbildung 5.24: Interpolation 2: Absoluter Fehler vor und nach der Interpolation $\alpha = -30°$ interpoliert aus $0°$ und $-40°$

Tabelle 5.2: Verbesserung zwischen *SAE* davor und danach in Prozent für $\alpha = 30°$ für die Methoden *interpoliert* und *direkt Eingemessen*

Methode	**Rot**	**Grün**	**Blau**
Interpoliert	76,22 %	98,86 %	95,59 %
Eingemessen	88,04 %	97,39 %	88,82 %

Tabelle 5.3: Verbesserung zwischen *SAE* davor und danach in Prozent für $\alpha = 30°$ für die Methoden *interpoliert mit* $\Delta\alpha = \pm 20°$, *interpoliert mit* $\Delta\alpha = \pm 40°$ und *direkt Eingemessen*

Methode	**Rot**	**Grün**	**Blau**
Interpolation 01	76,22 %	98,86 %	95,59 %
Interpolation 02	55,99 %	98,59 %	63,41 %
Eingemessen	88,04 %	97,39 %	88,82 %

6 Diskussion

In dieser Arbeit sollte die Möglichkeit erörtert werden eine LED-Wall so zu korrigieren dass sie abgefilmt, durch eine Kamera über mehrere Blickwinkel, ein gleichbleibendes Bild mit gleichbleibenden Farbwerten ergibt.
Durch die Messungen mit einem Spektroradiometer konnte die Relevanz unterstrichen werden. Je größer die Abweichung zur senkrechten Blickachse, desto größer wird der Abfall der Strahldichte. Außerdem wurde gezeigt, dass durch den leicht unterschiedlichen Strahldichten-Abfall je Blickwinkel pro LED eine dreidimensionale Korrektur notwendig ist. Das aus den Daten geschlossene Wissen zur Symmetrie und die Indizien zur Modellierbarkeit bieten eine Grundlage für aufbauende Forschung.
Die hier aufgeführte Methode zur Implementierung einer blickwinkelabhängigen Korrektur ist auf Grund der Versuchsergebnisse, als grundsätzlich funktionsfähig anzusehen. So wurde die von James u. a. 2021 beschriebene Methode nicht nur erweitert, sondern auch nachvollziehbar dokumentiert, sodass zukünftige Forschung die verwendeten Softwarekomponenten weiterverwenden, abwandeln und erweitern kann.

Die durchgeführten Versuche und deren Aussagekraft sind auf die Verwendung des genutzten LED-Panels beschränkt. Deshalb kann es sein, dass es bei der Reproduktion zu abweichenden Ergebnissen kommen kann.
Für die Verifizierung des Korrektur-Algorithmus' generell bzw. der blickwinkelabhängen Korrektur mussten Einschränkungen in Kauf genommen werden. Auf Grund der nicht linear abbildenden Mischfarben, wurde die Beschränkung auf *Rot*, *Grün* und *Blau* als Ausgabe-Farbwerte festgelegt, mit der speziell angewendeten Gammakodierung im Grün-Kanal. Das schränkt die Aussage über die Allgemeingültigkeit des Korrektur-Algorithmus' ein, nicht jedoch über die der Methode zur Interpolation. Hier ist davon auszugehen, dass diese mit einer anderen allgemeingültigen Korrektur ebenso nutzbar ist.
Grundlegend ist es möglich, mit der hier aufgeführten Methode, blickwinkelabhängig LED-Walls zu korrigieren.

7 Fazit

Diese Arbeit zeigt, dass die hier nachvollzogene und erweiterte Methode grundsätzlich wirksam ist.
Des Weiteren zeigen die Daten, die mit dem Spektrometer erhoben wurden einerseits deutlich die Relevanz einer blickwinkelabhängigen Korrektur und bieten gleichzeitig die Grundlage um den genauen Einfluss des Blickwinkels auf das Gesamtbild zu bewerten.

Es wurde gezeigt, wie eine Korrektur konkret durchgeführt werden kann und welche einzelnen Komponenten dafür notwendig sind. Für die Korrektur wurde eine Transformation, die aus Referenz- und Testsamples für das Kamerabild berechnet wird, auf den Video-Ausgang zur LED-Wall angewendet. Außerdem wird erläutert, wie eine Interpolation zwischen verschiedenen Korrekturen für verschiedene Blickwinkel einer Kamera durchgeführt werden kann, um die LED-Wall blickwinkelabhängig zu korrigieren. Diese ausgearbeitete Methode wurde im Detail erläutert und der entwickelte Code im Anhang quelloffen verfügbar gemacht. Der modulartige Aufbau erlaubt es einzelne Komponenten gezielt zu erweitern oder zu verbessern.
Aus den Messungen zur Veränderung der Strahlung des verwendeten LED-Panels über verschiedene Blickwinkel geht eindeutig hervor, dass die Strahldichte über die Blickwinkel hinweg immer stärker abfällt und ein leichter Color-Shift erkennbar ist.
Die entwickelte Methode wurde mit einer Einschränkung an benutzbaren Testfarben evaluiert. Unter dieser Bedingung zeigt sich die Methode als wirksam.

8 Ausblick

LED-Walls in Kombination mit Kameras zu nutzen ist ein Workflow, der sich im Moment etabliert und sehr wahrscheinlich ein relevantes Thema bleiben wird. Wie in dieser Arbeit gezeigt, ist für diese Anwendung ein grundlegendes Farbmanagement essentiell. Wenn eine bewegte Kamera genutzt wird, sollte blickwinkelabhängig korrigiert werden. Die Frage, die es zu erörtern gilt ist, an welcher Stelle diese Korrektur besser verortet sein sollte, entweder bei der Ausgabe des Bilds am Computer bzw. Medienserver selbst oder im LED-Controller, wie von *Megapixel* angedacht.

Für weitere Forschung im Bereich der Farbkorrektur solcher Workflows, sollte die Linearität zwischen den Bilddaten im Computer und der wirklich emittierten Strahlung der LED-Wall untersucht werden. Die durchgeführten Versuche zeigen, dass weitere Studien mit verschiedenen LED-Panels und Testsamples gleich verteilt im linearen Arbeitsraum aus dem ganzen RGB-Würfel einen erheblichen Beitrag leisten können.

Anstelle der hier verwendeten linearen Interpolations-Methode ließen sich auch andere Methoden testen, wie eine Spline-Interpolation oder eine aus den Daten eines Spektrometers modellierte Funktion. Wenn die Veränderung nach einem Modell vorhersagbar ist, müssten weniger Blickwinkel eingemessen werden. Des Weiteren ließen sich LED-Panels, mit einem besseren Einblickwinkel in der vertikalen Achse, erfolgversprechend diese Blickwinkelveränderung einmessen, sodass die Korrektur bilinear interpoliert werden kann.

Eine weitere Frage, die sich für die Anwendung in LED-Wall-Studios ergibt, ist: Wie verhält sich die Methode für große LED-Walls und wie muss sie angepasst werden? Da hier der Blickwinkel zum nächsten zu sehenden Punkt der LED-Wall ein anderer ist, als der zum weit entferntesten Punkt derselben LED-Wall. Ebenso relevant für die Implementierung in die Praxis ist die Echtzeit-Korrektur in einer entsprechenden Umgebung. Hierfür könnte die Methode z.B. als Shader eingebettet in *Unreal Engine* in Kombination mit *nDisplay* implementiert werden.

Wie unter anderem von McElvain und Gish 2013 ausgearbeitet, ist davon auszugehen, dass es in Zukunft neue und komplexere Methoden zur Farbraum-Transformationen geben wird. Dies wird zur Folge haben, dass die Komponenten zur Berechnung der Kompensation bzw. der Korrektur verändert werden müssen.

Abkürzungsverzeichnis

AD	Analog-Digital
CIE	Commission Internationale de l'Éclairage *(International Commission on Illumination)*
CMF	color matching function
CSV	Comma-separated values
DVI	Digital Visual Interface
EOCF	electronic-optical-conversion-function
HDMI	High Definition Multimedia Interface
LED	light emitting diode, Leuchtdiode
LUT	Look Up Table
OECF	optical-electronic-conversion-function
PCB	printed circuit board
PLCC	Plastic leaded chip carier
PWM	pulse width modulation
SAE	sum of absolute error
TIFF	Tagged Image File Format

Glossar

Pixelpitch Abstand der Mittelpunkte zweiter Pixel, meist in mm

Pulsweiten-Modulation Durch zeitlich unterschiedlichen Abstand beim Wechseln der Spannung zwischen 0 und 1 können unterschiedliche Helligkeitswerte einer LED gesteuert werden, ohne deren Kennlinie kennen zu müssen. Hierdurch kann ein Bildsignal linear auf einem LED-Display angezeigt werden.

Moiré-Effekt Ein optischer Effekt, bei dem durch die Überlagerung von z.B. zwei regelmäßiger Raster ein periodisches Raster mit speziellen Strukturen entsteht, das in keinem der Ursprungsmuster vorhanden ist und sich mit der Überlagerung verändert. Besonders sichtbar bei Abfilmen von Displays.

Scanline-Artefakte Temporale Artefakte, die durch die Ansteuerung der LEDs durch die Treiberchips entstehen. Visuell sichtbar sind sie als dunkle Linien in einem eigentlich für das Auge homogenen Bild. (Kintrup 2022)

Shader Meist schwarze Plastikmaske, die die Zwischenräume zwischen Einzelpixeln eines LED-Moduls ausfüllt, um den Schwarzwert zu verbessern. Hierdurch wird Auflicht so gut wie möglich absorbiert.

Shutter-Angle Länge der Verschlusszeit bei Videos bzw. Film im Verhältnis zur Frame-Rate. Ein Shutter-Angle von 360° entspricht einem Verhältnis von 1.

Literaturverzeichnis

Adobe Illustrator 2022 (2022). *Adobe Inc.* URL: https://www.adobe.com/de/products/illustrat.html.

Buchholz, Leon und Lauritz Raisch (2021). "Können On Location Drehs durch Virtual Productionersetzt werden? Erstellung einer Colormanagement Pipeline für einen Locationscanning Virtual Production Workflow und die beispielhafte Anwendung und Validierung anhand eines filmischen Vergleiches." Hochschule der Medien Stuttgart.

Cheung, Vien u. a. (Jan. 2004). "A comparative study of the characterisation of colour cameras by means of neural networks and polynomial transforms". In: *Coloration Technology* 120.1, S. 19–25. DOI: 10.1111/j.1478-4408.2004.tb00201.x.

colour (2021). *Open Source.* Licensed under [New BSD License]. URL: https://github.com/colour-science/colour.

DaVinci Resolve 17.4.2 (2021). *Blackmagic Design Pty. Ltd.* URL: https://www.blackmagicdesign.com/de/products/davinciresolve/.

Decklink Software Development Kit 10.11.2 (2021). *Blackmagic Design Pty. Ltd.* Blackmagic Design Decklink Software Development Kit libraries 10.11.2. URL: https://www.blackmagicdesign.com/support/family/capture-and-playback.

disguise (Nov. 2021). *disguise UserGuideVersion-r19.* p. 1875f, aufgerufen am 07.01.22. Disguise Technologies Limited. 88-89 Blackfriars Road, London SE18HA. URL: https://disguise.app.box.com/v/r19userguide.

FFMPEG (2021). *Open Source.* Version N-105038-g30322ebe3c, Licensed under [GNU Lesser General Public License (LGPL) version 2.1]. URL: https://ffmpeg.org/.

ffmpeg-python (2021). *Open Source.* URL: https://github.com/kkroening/ffmpeg-python.

FKT (Juni 2021). “Studiowechsel auf Knopfdruck: Flexible LED-Hintergrund- beleuchtung für SWR Mainstage”. In: *FKT Fachzeitschrift für Fernsehn, Film und elektronische Medien*. abgerufen am 23.02.2022. URL: https://www.fkt-online.de/archiv/artikel/2021/fkt-6-2021/29478-studiowechsel-auf-knopfdruck-flexible-led-hintergrund-beleuchtung-fuer-swr-mainstage/.

Graßmann, Hermann (1853). “Zur Theorie der Farbenmischung”. In: *Annalen der Physik und Chemie* 165.5, S. 69–84. DOI: 10.1002/andp.18531650505.

Green, Phil und Lindsay MacDonald (2011). *Colour engineering: achieving device independent colour*. John Wiley & Sons.

Hansen, Ryan (Apr. 2010). “Contrast Ratios in Outdoor LED Displays”. In: *Information Display* 26.4, S. 8–12. DOI: 10.1002/j.2637-496x.2010.tb00246.x.

Hasche, Eberhard und Patrick Ingwer (2016). *Game of Colors: Moderne Bewegtbildproduktion*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-43889-3.

Hochman, Jeremy und Keith Harrison (2021). *Megapixel Technology Overview Calibration Metadata*. Techn. Ber. abgerufen am: 25.01.2022. Megapixel Visual Reality. URL: https://megapixelvr.com/calibration-metadata/.

Huang, T., G. Yang und G. Tang (Feb. 1979). “A fast two-dimensional median filtering algorithm”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 27.1, S. 13–18. DOI: 10.1109/tassp.1979.1163188.

Hyun, Joonho, Suk-Ju Kang und Young Hwan Kim (Dez. 2016). “Configurable Controller for High-Resolution LED Display Systems”. In: *Journal of Display Technology* 12.12, S. 1594–1601. DOI: 10.1109/jdt.2016.2621009.

ICT AG (o. D.[a]). *inspireLED MCTRL660*. ICT AG. Erscheckweg 1, 72664 Kohlberg.

– (o. D.[b]). *inspireLED s3.6b Indoor*. ICT AG. Erscheckweg 1, 72664 Kohlberg.

James, Oliver u. a. (Juli 2021). “Colour-Managed LED Walls for Virtual Production”. In: *ACM SIGGRAPH 2021 Talks*. ACM. DOI: 10.1145/3450623.3464682.

Janglin Chen Wayne Cranton, Mark Fihn (2016). *Handbook of and Visual Display and Technology*. Hrsg. von Janglin Chen, Wayne Cranton und Mark Fihn. Springer International Publishing. DOI: 10.1007/978-3-319-14346-0.

Kintrup, Jana (2022). “Untersuchung von zeitlich bedingten Artefakten im Wechselspiel von Kameras mit Global beziehungsweise Rolling Shutter mit LED-Wänden”. Magisterarb. TECHNISCHEN HOCHSCHULE KÖLN.

Lee, Dong-Hyeok u. a. (13. März 2017). “A camera-based color calibration of tiled display systems under various illumination environments”. In: *Journal of Information Display* 18.2, S. 73–85. DOI: 10.1080/15980316.2017.1291454.

Li, Jinmin und G. Q. Zhang (2019). *Light-Emitting Diodes*. Hrsg. von Jinmin Li und G. Q. Zhang. Springer International Publishing. DOI: 10.1007/978-3-319-99211-2.

Löffler-Mang, Martin (Dez. 2011). “Spektrometer”. In: *Optische Sensorik*. Vieweg+Teubner Verlag, S. 192–202. DOI: 10.1007/978-3-8348-8308-7_23.

matplotlib (2021). *Open Source*. Licensed under [BSD-style license]. URL: https://matplotlib.org/.

Maxwell, James Clerk (1857). “XVIII.—Experiments on Colour, as perceived by the Eye, with Remarks on Colour-Blindness”. In: *Transactions of the Royal Society of Edinburgh* 21.2, S. 275–298. DOI: 10.1017/s0080456800032117.

McElvain, Jon und Walter Gish (Jan. 2013). “Camera Color Correction Using Two-Dimensional Transforms”. In: *Color and Imaging Conference*.

Netflix Inc. (2021). *On-Set Infrastructure*. abgerufen am 06.01.2022. URL: https://partnerhelp.netflixstudios.com/hc/en-us/articles/360061949514-On-Set-Infrastructure.

NovaLCT 5.4.3 (2021). *Xi’an NovaStar Tech Co., Ltd.* URL: https://novastar.shop/download-software/.

NovaStar (2020). *NovaLCT LED Configuration Tool for Synchronous Control System User Manual*. abgerufen am 12.01.2022. Xi’an NovaStar Tech Co., Ltd. URL: https://novastar.shop/wp-content/uploads/2019/10/NovaLCT-LED_Synchronous_Control_System_User_Manual_V5.3.01.pdf.

numpy (2021). *Open Source*. Licensed under [NumPy license]. URL: https://numpy.org.

pandas (2021). *Open Source*. Licensed under [BSD 3-Clause "Newör "Revised"License]. URL: https://pandas.pydata.org/.

Photo Research (2019). *PR-655/670 SpectraScan® User Manual*. abgerufen am 14.01.2022. Photo Research. 7279 William Barry Blvd. North Syracuse, NY 13212. URL: https://www.jadaktech.com/products/photo-research/spectrascan-pr-655/.

Prak, Marina (2021). *ROE Visual Thesaurus LED technology for Virtual Production*. Techn. Ber. abgerufen am 08.12.2021. ROE Visual. URL: https://www.roevisual.com/en/knowledge-and-support/thesaurus-led-technology-for-virtual-production-xr-film.

Python 3.9.7 (2021). URL: https://www.python.org.

Qlab 4.6.11 (2021). *Figure 53, LLC.* URL: https://qlab.app.

Schanda, J., K. Muray und Balazs Kranicz (Juni 2015). "LED colorimetry". In: *9th Congress of the International Colour Association*. Hrsg. von Robert Chung und Allan Rodrigues. SPIE. DOI: 10.1117/12.464616.

Schmidt, Ulrich (2013). *Professionelle Videotechnik*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-38992-4.

Schwirten, Tobias und Nina Chen (Nov. 2015). "Grundprinzipien von SMD-LEDs im Einsatz von professionellen Videomodulen". In: *FKT Fachzeitschrift für Fernsehn, Film und elektronische Medien*. abgerufen am 26.12.2021, S. 503–506. URL: https://www.fkt-online.de/fileadmin/user_upload/Bilder/FKT/Themenseiten/Prolight___Sound/2015_11_503.pdf.

scipy (2021). *Open Source*. Licensed under [BSD 3-Clause "Newör "Revised"License]. URL: https://scipy.org/.

Selan, Jeremy (17. Okt. 2012). *Cinematic Color VES*. Techn. Ber. abgerufen am: 07.12.2021. VES Technology Committee. URL: https://raw.githubusercontent.com/jeremyselan/cinematiccolor/master/ves/Cinematic_Color_VES.pdf.

Seymour, Mike (März 2020). *Art of LED Wall Virtual Production, Part Two: 'How you make one'*. abgerufen am 05.01.2022. URL: https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-sets-part-two-how-you-make-one/.

Sharma, Abhay (Juli 2004). “Introduction”. In: *Understanding Color Management*. John Wiley & Sons, S. 1–35. DOI: 10.1002/9781119223702.ch1.

Sharma, Gaurav und H. Joel Trussell (Juli 1997). “Digital color imaging”. In: *IEEE Transactions on Image Processing* 6.7, S. 901–932. DOI: 10.1109/83.597268.

Slawsky, Richard (2020). *LED Displays in Movie & TV Production*. Techn. Ber. abgerufen am 05.01.2022. Digital Signage Today. URL: https://www.digitalsignagetoday.com/resources/led-displays-in-movie-tv-production/.

SpectraWin™2 (2019). *Jadak - A Novanta Company*. URL: https://www.jadaktech.com/products/photo-research/spectrascan-pr-670/.

Stiny, Leonhard (2015). *Aktive elektronische Bauelemente*. p. 130–143. Springer Fachmedien Wiesbaden. DOI: 10.1007/978-3-658-09153-8.

Thielemans, Robbie (2016). “LED Display Applications and Design Considerations”. In: *Handbook of Visual Display Technology*. Springer International Publishing, S. 1735–1748. DOI: 10.1007/978-3-319-14346-0_76.

Westland, Stephen, Caterina Ripamonti und Vien Cheung (Mai 2012a). “Characterisation of Cameras”. In: *Computational Colour Science using MATLAB®*. John Wiley & Sons, Ltd, S. 143–157. DOI: 10.1002/9780470710890.ch10.

– (Mai 2012b). “Colour Management”. In: *Computational Colour Science using MATLAB®*. John Wiley & Sons, Ltd, S. 119–129. DOI: 10.1002/9780470710890.ch8.

– (Mai 2012c). “Display Characterisation”. In: *Computational Colour Science using MATLAB®*. John Wiley & Sons, Ltd, S. 131–141. DOI: 10.1002/9780470710890.ch9.

– (Mai 2012d). “Linear Algebra for Beginners”. In: *Computational Colour Science using MATLAB®*. John Wiley & Sons, Ltd, S. 13–18. DOI: 10.1002/9780470710890.ch2.

Wolf, Stephen (2003). *Color Correction Matrix for Digital Still and Video Imaging Systems*. Techn. Ber. In NTIA Technical Memorandum TM-04-406, abgerufen am 18.12.2021. U.S. DEPARTMENT OF COMMERCE. URL: https://www.its.bldrdoc.gov/publications/details.aspx?pub=04-406.

Anhang

Datenaufbereitung Specotroradiometer

Code in Python

Algorithm 1: Convert Strings with comma to float

```
import pandas as pd
def changeToFloat(x):
    #changeToFloat, if not sure element is convertable to float
    if x == "_00": #special for this usecase
        x = 0
    try:
        return float(x)
    except:
        return x
def convertCommaDecimaToFloat(x):
    #replaces every , with .
    x = x.str.replace(',','.')
    x = x.apply(changeToFloat)
    return x
names = [path[-10:-7] for path in all_filepaths]
order = ["-85","-80","-70","-60","-50","-40","-30","-20","-10","_00","+10","
    +20","+30","+40","+50","+60","+70","+80","+85"]
#names should equal order
# dict with name: filepath
all_files = {names[i]: all_filepaths[i] for i in range(len(all_filepaths))}
# Big dataset for start (in order)
data_frames = {name : None for name in order}
for name in order:
    path = all_files[name]
    df = pd.read_csv(path, encoding="ISO-8859-1" ,sep=";", index_col=0) #
        reads csv, important with right encoding
    df = df.apply(convertCommaDecimaToFloat) #replaces every , with .
    realNames2={df.columns[7]:'White',
                df.columns[6]:'Gray75',
```

```
                df.columns[5]: 'Gray50',
                df.columns[4]: 'Gray25',
                df.columns[3]: 'Red',
                df.columns[2]: 'Green',
                df.columns[1]: 'Blue',
                df.columns[0]: 'White2'}
    df = df.rename(columns=realNames2)# Change M1 to White
    data_frames[name] = df
```

Algorithm 2: Aus data_frames einzelne Veränderung herausgreifen

```
import colour
definition = "White"
definition_spec = []
definition_dict = []
max_value = []
for name in order:
    df = data_frames[name]
    df_snippet = df.loc["380":"780",definition]
    df_snippet = df_snippet.rename(changeToFloat(name))
    max_value.append(df_snippet.max())
    definition_spec.append(df_snippet)
max_value = max(max_value)*1.05
```

Daten - Evaluierung

Absoluter Fehler für $\alpha = 30°$ gemessen

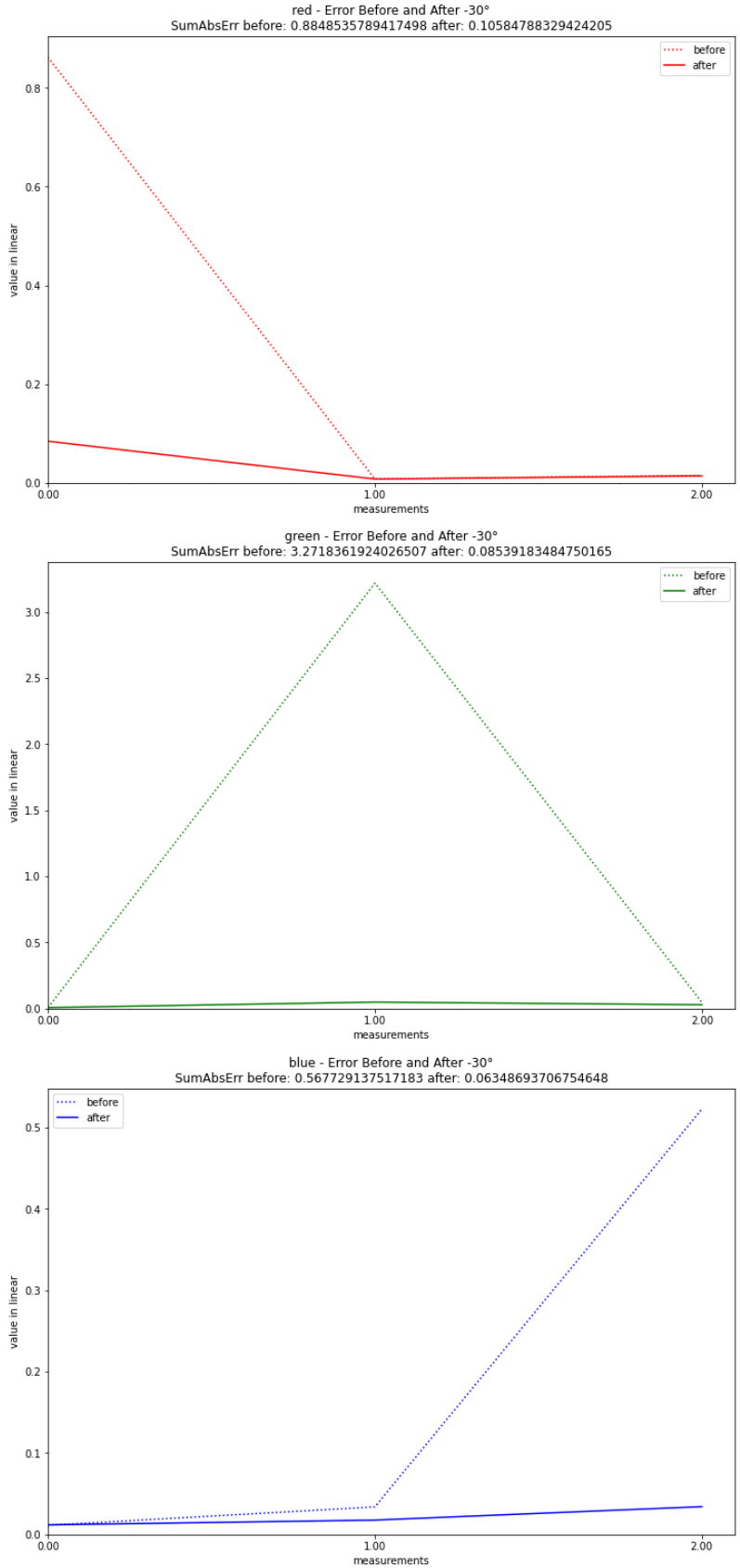


Abbildung 8.1: Anhang: Absoluter Fehler für $\alpha = -30°$ gemessen und korrigiert

blickwinkelabhängige Farbkorrektur

Code in Python

Algorithm 3: Aus Color -Output ein Ausgabe-Bild berechnen

```
import numpy as np
import time
import os
from colour import write_image
# generate OUTPUT IMG
def outputColor(c_out, width=128, height=128):
    """
    Saves a 16bit tiff from color to assets als img_out.tiff for qlab to
        show
    Parameters
    ----------
    c_out : ndarray (3,)
        Color, to display
    width : int, optional
        The default is 128.
    height : int, optional
       The default is 128.
    Returns
    -------
    None
    """
    path = os.path.join("../assets", "img_out.tiff")
    c_out = np.array(c_out)
    # if a value is smaller than 0 gets clamped to 0 or bigger than 1 gets
        clipped
    # Cause -0.1 would become 0.9 through colour.write_image

    c_out = np.clip(c_out, 0, 1)
    img_out = np.tile(c_out, (height, width, 1))
    colour.write_image(img_out, path, bit_depth="uint16", method="Imageio")
    time.sleep(2) # hold, for Qlab to recognize changed output img
```

Algorithm 4: Aus einem Video im ndarray-Format einen Bildausschnitt ausschneiden

```
import numpy as np
#get a cutout from the middle, default 80px by 80px
def getCutOut(video, width=80, height=None):
    """
    Cuts out a part of a video file from the middle
    Parameters
    ----------
    video : 4dim ndarray
    width : int, optional
        The default is 80.
    height : int, optional
        if not filled = width. The default is 80.
    Returns
    -------
    TYPE
        video / ndarray.shape(frames, width, height, 3)
    """
    video = frameToVideo(video)
    if height == None:
        height = width
    video_height = video.shape[1]
    video_width = video.shape[2]
    video_middle = [int(video_height / 2), int(video_width / 2)]
    cutout_coordinates = [video_middle[0] - int(height/2),video_middle[1] -
        int(width/2)] # topleft corner of coutout beginning
    cutoutVideo = []
    for i, frame in enumerate(video):
        cutoutVideo.append(frame[cutout_coordinates[0]:cutout_coordinates
            [0]+height, cutout_coordinates[1]:cutout_coordinates[1]+width,:])

    return np.array(cutoutVideo)

#create a 4 dimensional ndarray from one frame
def frameToVideo(frame):
    if frame.ndim < 4: # if only a frame is inputted
        video = np.array([frame])
    else:
        video = frame
    return video
```

Algorithm 5: Berechnet aus einem Video den Median nur spatial oder auch temporal

```
import numpy as np
# denoises ndarray cutout with the use of the median either only spatial or
    also temporal
def denoiseMedian(cutout):
    if cutout.ndim == 4:
            cutout_median = np.median(cutout, axis=(0,1,2))
    elif cutout.ndim == 3:
            cutout_median = np.median(cutout, axis=(0,1))
    else:
            print("Problem with video capture...")
            return
    return cutout_median
```

Algorithm 6: Gamma-kodierung bzw. -dekodierung von Rec709 mit Gamma

```
import numpy as np
import colour
#SLOG2 Values mapped to linear
def slog2_to_linear(video, bit_depth=10, in_normalised_code_value=False):
    # in_normalised[...] deactivates a legal to full conversion
    return log_decoding_SLog2(video, bit_depth, in_normalised_code_value) #
        from colour.models

gamma_BT70924 = 2.4
def BT70924_to_linear(img_gamma, gamma = gamma_BT70924):
    #Preserve: The definition will preserve any negative number, i.e. The
        Foundry Nuke behaviour.
    #Clamp: The definition will clamp any negative number to 0.
    gamma = np.array(gamma)
    img_lin = colour.gamma_function(img_gamma, gamma,
        negative_number_handling="Preserve")
    return img_lin
def linear_to_BT70924(img_lin, gamma = gamma_BT70924):
    #Preserve: The definition will preserve any negative number, i.e. The
        Foundry Nuke behaviour.
    #Clamp: The definition will clamp any negative number to 0.
    gamma = np.array(gamma)
    img_gamma = colour.gamma_function(img_lin, (1/gamma),
```

```
        negative_number_handling="Clamp")
    return img_gamma
```

Algorithm 7: Gibt Colorcorrection-Matrix aus für Referenz-Daten und Test-Daten

```
import numpy as np
from colour.characterisation import matrix_colour_correction_Cheung2004

#returns correctionMatrix for I_cam_lin and I_lin, as M_T and M_R
def getCorrectionMatrix(I_cam_lin, I_lin):
    M = matrix_colour_correction_Cheung2004(I_cam_lin, I_lin)
    M_inv = np.linalg.pinv(M)
    #M_np_inv = np.linalg.solve(I_lin, I_cam_lin) #alternative to colour lib
    #M_np = np.linalg.pinv(M_scipy)
    print(f"M * M_inv should be identity: \n{np.around(M_inv.dot(M), 
        decimals=1)}") #to check if pseudoinverse is right
    return M, M_inv
```

Algorithm 8: Gibt I_{mod} von M_{CC} aus

```
import numpy as np
#returns I_mod_lin > modified Output color
def getModifiedOutputColor(I_lin_element, M_cc):
    I_lin_element = np.asarray(I_lin_element)
    I_mod_lin = M_cc.dot(I_lin_element)
    if np.min(I_mod_lin) < 0:
        print(f"{20*'#'}\nCAREFUL negative Values in I_mod_lin, need to be 
            handled in outputColor\n{20*'#'}")
    return I_mod_lin
```

Algorithm 9: Führt Messung und Korrektur pro aufzunehmenden Blickwinkel aus

```
import numpy as np
import time
import videotools #own functions to grab frames from decklink card

def addMeasurement(suffix="0", patchsize=200, frames = 1):
    time.sleep(2)
    video = videotools.getDecklinkFrames(frames)
    video = videotools.getFloatVideo(video)
```

```
    # get only part where led is visible (patch by patch from middle)
    cutout = videotools.getCutOut(video, patchsize)
    # get median out of cutout in slog2
    cutout_median = denoiseMedian(cutout)
    if cutout_median == None:
        print("Problem with video capture...")
        return
    I_mea_slog = cutout_median
    I_mea_lin = slog2_to_linear(cutout_median)
    return I_mea_slog, I_mea_lin

def measureAndCalculateMatrix(angle, measurementsDatabase):
    measurementsDatabase[f"{angle}"] = dict()
    I_cam = np.zeros(shape=I_lin.shape)
    for idx in range(samplesPerAngle):
        outputColor(I_gamma[idx])
        I_mea_slog, I_mea_lin = addMeasurement(f"_{angle}deg_mea01_{idx+1}")
        I_cam[idx] = I_mea_lin
    measurementsDatabase[f"{angle}"]["I_lin"] = I_lin
    measurementsDatabase[f"{angle}"]["measurements"] = I_cam
    # Correction Matrix
    I_lin = measurementsDatabase[f"{angle}"]["I_lin"]
    I_cam = measurementsDatabase[f"{angle}"]["measurements"]
    M_cc, M_cam = getCorrectionMatrix(I_cam, I_lin)
    measurementsDatabase[f"{angle}"]["M_cc"] = M_cc
    measurementsDatabase[f"{angle}"]["M_cam"] = M_cam

#intitialize Parameters > Color original
red = np.array([1, 0, 0], dtype=float)
green = np.array([0, 1, 0], dtype=float)
blue = np.array([0, 0, 1], dtype=float)
mainColors = [red, green, blue]
I_lin = np.array(mainColors)
I_gamma = linear_to_BT70924(I_lin)
#Database for correction-data-storage
measurementsDatabase = dict()  #[angle, num Colorpatches, 3]

# LOOP for each angle - Doing the measurements
while True:
```

```
    angle = input("Messung zu ____deg:")

    measureAndCalculateMatrix(angle, measurementsDatabase)

    q = str(input("If you are finished, >type: 'yes'")).lower()
    if q == "yes" or q == "y":
        break
```

Algorithm 10: Findent k nächste Elemente (Blickwinkel) zu Input-Blickwinkel

```
import numpy as np
# Function to search the specified array 'angles' for key 'angle_input'
# using the binary search algorithm
def binarySearch(angles, angle_input):
    low = 0
    high = len(angles) - 1
    while low <= high:
        mid = low + (high - low) // 2
        if angles[mid] < angle_input:
            low = mid + 1
        elif angles[mid] > angle_input:
            high = mid - 1
        else:
            return mid      # key found
    return low              # key not found


# Function to find the 'k' closest elements to 'angle_input' in a sorted
    integer array 'angles'
def findKClosestElements(angles, angle_input, k):
    """
    Find k clostest Elements in angles to angle_input
    ----------
    angles : int, float
        input array of numbers.
    angle_input : int, float
        angle_input number.
    k : int
        number of elements.
```

```
    -------
    """
    angles = np.array(list(angles), dtype=int)
    # find the insertion point using the binarSearch function
    i = binarySearch(angles, angle_input)

    left = i - 1
    right = i
    # run `k` times
    while k > 0:
        # compare the elements on both sides of the insertion point `i`
        # to get the first `k` closest elements
        if left < 0 or (right < len(angles) and abs(angles[left] -
            angle_input) > abs(angles[right] - angle_input)):
            right = right + 1
        else:
            left = left - 1
        k = k - 1
    # return `k` closest elements
    return angles[left+1: right]

if __name__ == '__main__':
    x = dict()
    angles = [10, 12, 15, 17, 18, 20, 25]
    for n in angles:
        x[f"{n}"] = 10
    angle_input = 16
    k = 2
    x_keys = x.keys()
    print(findKClosestElements(x_keys, angle_input, k))
```

Algorithm 11: Erstellt Interpolationsfunktion und interpoliert linear zwischen k Blickwinkeln für Input-Blickwinkel

```
from scipy import interpolate
def getInterpolationFunction(k_nearest, angles):
    interpFunc = interpolate.interp1d(k_nearest, angles, axis=0, kind="
        linear")
    return interpFunc
```

```
def getInterpolatedValue(angle_input, interpFunc):
    value = interpFunc(angle_input)
    return value
```

Algorithm 12: Gibt den interpolierten Ausgabewert für angle_input aus

```
# Doing the anglular interpolated Correction for a color I_original_lin
def getModifiedOutputColor_interpolated(angle_input, I_original_lin,
    k_nearest):
    # Correcton matrix-correction per angle and doing correction on
        I_original_lin for each angle_k
    for angle in k_nearest:
        M_cc = np.array(edited_measurementsDatabase[f"{angle}"]["M_cc"])
        I_mod_lin_element = getModifiedOutputColor(I_original_lin, M_cc)
        I_mod_lin_list.append(I_mod_lin_element)
    #list of k calculated values for k angles
    I_mod_lin_list = np.array(I_mod_lin_list)
    #doing interpolation
    interpFunc = interpolate.interp1d(k_nearest, I_mod_lin_list, axis=0,
        kind="linear")
    I_mod_lin =  interpFunc(angle_input)
    print(f"Interpolated␣I_mod_lin{I_mod_lin}")
    return I_mod_lin

#read measurementdata from Database
edited_measurementsDatabase = measurementsDatabase
sortedKeys = np.array(list(edited_measurementsDatabase.keys()), dtype=int)
sortedKeys.sort()

k = 2 # for interpolation method: linear!
k_nearest = findKClosestElements(sortedKeys, angle_input, k)

# Colors original
angle_input = int(input("For␣which␣angle␣interpolate"))
I_original_lin = colorOriginal

# doing Intepolation and output modified Color
getModifiedOutputColor_interpolated(angle_input, I_original_lin)
I_mod_gamma = linear_to_BT70924(I_mod_lin)
```

```
outputColor(I_mod_gamma)
```

Algorithm 13: Nimmt n Frames der (Decklink)-Inputkarte auf

```
import ffmpeg
import numpy as np

def getDecklinkFrames(frames=2, height=1080, width=1920, rawformat="
    yuv422p10"):
    """
    Parameters
    ----------
    frames : int
        DHow many frames to capture. The default is 2.
    width : int
        The default is 1080.
    height : int
        The default is 1920.
    rawformat :
        special ffmpeg type for inputtype.
        The default is "yuv422p10".
    Returns
    -------
    video : ndarray - uint16, shape(frames, height, width, 3)
    """
    # _ for error-handling
    out, _ = (
        ffmpeg
        .input(raw_format=rawformat, filename="UltraStudio Recorder 3G",
            format='decklink')
        .output('pipe:', format='rawvideo', pix_fmt="rgb48le" , vframes=
            frames, loglevel='warning') # loglevel shows less
        .run(capture_stdout=True)
    )
    video = (
        np
        .frombuffer(out, np.uint16) #16bit encoded to support 10bit video
            input
        .reshape([frames, height, width, -1])
```

```
    )
    return video

# Utilities
#create a 4 dimensional ndarray from one frame
def frameToVideo(frame):
    if frame.ndim < 4: # if only a frame is inputted
        video = np.array([frame])
    else:
        video = frame
    return video

#get a float value from a default uint16 value
def getFloatVideo(video, bit_depth=16):
    """
    Parameters
    ----------
    video : 4 dim, ndarray, uint16
    bit_depth : int
        The default is 16.
    dtype : np.dtype
        Output-Type. The default is np.float64.

    Returns
    -------
    ndarray - dtype=float64

    """
    video = video/(2**bit_depth-1)
    return video
```

Algorithm 14: 2D-Test für Matrixkorrektur mit I_mod aus M_{CC}

```
import numpy as np
from matplotlib import pyplot as plt
x = np.array([[1, 0.2],
              [0.3, 1]])
y = np.array([[0.8, 0.1],
             [0.1, 0.7]])
```

```
x_p = np.array([0.8, 0.6])

M_cc_inv = np.linalg.solve(x, y) #Is same as M_cam
M_cc = np.linalg.inv(M_cc_inv)  #Is same as M_cc
y_p = M_cc_inv @ x_p
x_p_mod =  M_cc @ x_p
y_p_mod =  M_cc_inv @ p_mod
```

Algorithm 15: 3D-Test für Matrixkorrektur mit I_mod aus M_{CC}

```
from colour.characterisation import matrix_colour_correction_Cheung2004
x = np.array([[1., 0.,0.],
              [0., 1., 0.],
              [0., 0., 1.]])
y = np.array([[0.8, 0.1, 0.05],
              [0.1, 0.7, 0.21],
              [0.1, 0.1, 0.95]])
x_p = np.array([0.8, 0.6, 0.4])

M = matrix_colour_correction_Cheung2004(y, x)
M_inv = np.linalg.pinv(M)

y_p = M_inv.dot(p)
x_p_mod = M.dot(p)
y_p_mod = M_inv.dot(x_p_mod)
```

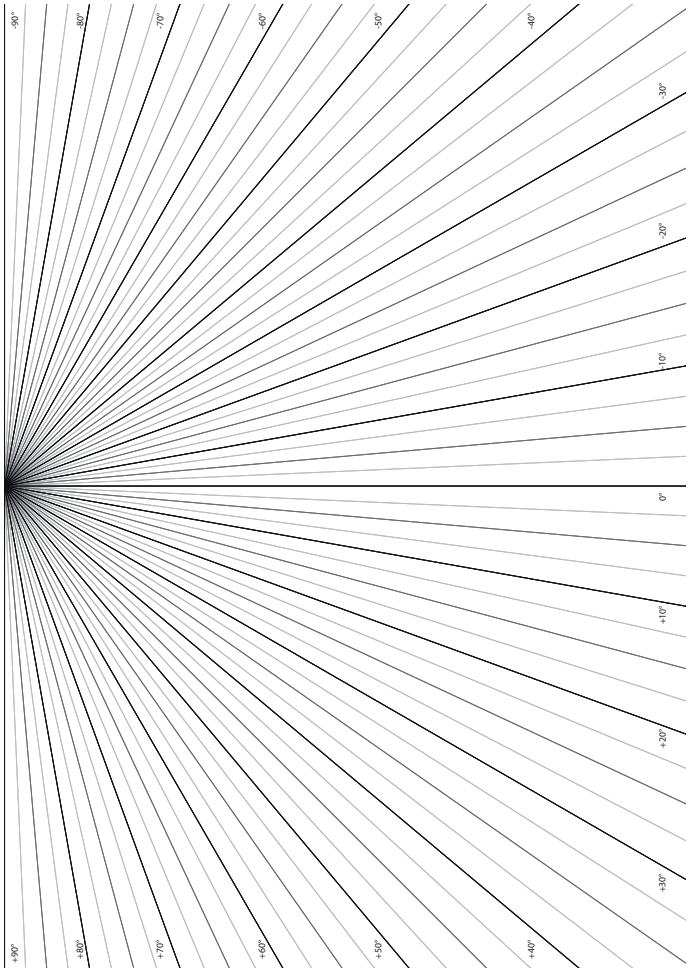
Algorithm 16: 10bit Test

```
import colour
import numpy as np

def check10Bit(img, channel=1):
    """
    Parameters
    ----------
    img : TYPE
        input Image in a numpyformat. Needs to be in dtype=uint16
    channel : int, 0,1,2
        Which channel to check. The default is 1 > green in RGB.
    """
```

```
    c = 0
    img = np.array(img, dtype="uint16")
    for j in range(2):
        img_ = np.diff(img[:,:,channel], axis=j)
        iterable_array = np.nditer(img_[:,:], flags=['multi_index'])
        for i in iterable_array:
            if 63 < i < 256:
                c = c + 1
    print(f"{c} pixel differences are to small for 8bit and right for 10bit"
        )

def check10BitPath(path, channel=1):
    img = colour.read_image(path, bit_depth="uint16")
    check10Bit(img, channel)
    return img
```

Winkelmesser

Original auf DIN-A3, zur Visualisierung skaliert:



Winkelmesser.pdf (nicht maßstabsgetreu)