BACHELORARBEIT

Anbindung von Blackmagic Design Kamerasystemen an generische Bildmischer

Prototyp-Entwicklung zur Verbesserung und Erweiterung von Kamerakontrollfunktionen und Tally Lights

Version vom 25. März 2024,
Lars Pieperjohanns

Erstprüfer: Prof. Dr. Marco Grimm
Zweitprüferin: M.A. Nathalie Mai

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Department Medientechnik
Finkenau 35
22081 Hamburg

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der Integration eines Drittanbieter-Bildmischers in ein Videoproduktionssystem von BlackmagicDesign (BMD). Das Ziel besteht darin, den vollen Funktionsumfang des BMD-Systems beizubehalten, einschließlich Kontrollfunktionen wie Kamerasteuerung und Farbkorrektur, ohne die Notwendigkeit eines zusätzlichen BMD-Bildmischers. Ein weiterer Mehrwert wird durch eine vereinfachte Bedienung ohne spezialisierte Hardware geschaffen. Kontrollfunktionen, die im Protokoll enthalten sind, können auch dann genutzt werden, wenn sie nicht in der BMD-Software vorgesehen sind.

Dieses wird auf der Grundlage eines bereits vorhandenen Videoproduktionssystems mit einem durch die NewTek TriCaster TC1-Bildmischer umgesetzt und ist auf andere Systeme übertragbar. Anstelle des BMD-Bildmischers wird ein Arduino-Mikrocontroller mit einem BMD 3G-SDI-Shield verwendet. Eine Netzwerkschnittstelle ermöglicht die Steuerung des Arduinos, wobei das OpenSoundControl (OSC) Protokoll für die Kommunikation verwendet wird. Als Beispiel für einen Controller wird die Software TouchOSC verwendet.

Abstract

The present work deals with the integration of a third-party video mixer into a BlackmagicDesign (BMD) video production system. The aim is to retain the full functionality of the BMD system, including control functions such as camera control and color correction, without the need for an additional BMD video mixer. Another added value is created through simplified operation without specialized hardware. Control functions included in the protocol can be utilized even if they are not provided in the BMD software.

This is achieved by implementing a NewTek TriCaster TC1 video mixer based on an existing video production system and is transferable to other systems. Instead of the BMD video mixer, an Arduino microcontroller with a BMD 3G-SDI Shield is used. A network interface allows for the control of the Arduino, with communication facilitated using the OpenSoundControl (OSC) protocol. The TouchOSC software is used as an example controller.

Inhaltsverzeichnis

Abbildungsverzeichnis

Codeblockverzeichnis

1 Motivation

Die Firma Blackmagic Design (BMD) hat in den letzten Jahren ein vollumfängliches Liveproduktions-Ökosystem geschaffen. Dabei wird die komplette Sendekette, von der Kamera bis zum Streaming-Encoder, abgedeckt. Alle Komponenten sind aufeinander abgestimmt, lassen sich meist zentral bedienen und bieten Funktionen, die sonst nur in professionellem Broadcast Equipment zu finden sind. Dabei fallen zwei Punkte besonders auf: die Integration von Tallys und das umfangreiche Kamerakontrollsystem zur Bedienung und Farbanpassung der Kameras. So lassen sich fast alle Funktionen der Kameras von einem Arbeitsplatz aus steuern. Die zentrale Einheit bildet dabei ein BMD Atem Videomischer. Hier laufen die Steuersignale zusammen und werden, gemeinsam mit den Bildsignalen, verarbeitet und an die jeweiligen Geräte verteilt. Soll jedoch ein anderer Bildmischer verwendet werden, geht die zentrale Einheit und damit auch die meisten Vorteile und Funktionen dieses Ökosystems verloren.

In meiner beruflichen Laufbahn als Veranstaltungs- und Medientechniker bin ich bereits mehrfach mit der folgenden Problemstellung konfrontiert worden: Gewünscht sind die Vorteile eines Geschlossen Systems, wie BMD es bietet. Allerdings sind häufig bereits einzelne Komponenten anderer Hersteller vorhanden, die aus unterschiedlichen Gründen weitergenutzt werden sollen. Wenn diese Entscheidung nicht Finanziell begründet wird, dann meist da bestimmte Funktionen, die das BMD Ökosystem nicht abbilden kann.

Insbesondere im Bereich der Bildmischertechnik verfolgt BMD einen eher konservativen Ansatz und verwendet beispielsweise keinerlei Video-over-IP Protokolle. Hybride Bildmischer anderer Hersteller, welche diese Funktionen bieten, halten immer mehr Einzug in die Veranstaltungsbranche. Da sich diese Hersteller aber meist auf eine Disziplin fokussieren und nicht, wie es bei BMD der Fall ist, einen Systemansatz verfolgen, ist es nun erforderlich, Lösungen zu finden, diese beiden Ansätze zu kombinieren.

Diese Situation bietet mir als ambitioniertem Medientechniker die Chance, mich mehrere Monate mit der Aufgabenstellung zu befassen, einen guten Kompromiss durch Kombination unterschiedlichster Systeme zu finden. Hierbei geht es nicht nur darum, das inhaltlich beste Endergebnis zu erarbeiten. Stattdessen stehen vor allem Wirtschaftlichkeit, Bedienbarkeit, Zuverlässigkeit und die Schaffung eines Mehrwertes für das Endprodukt, im Vordergrund.

Versucht man diese Aufgabe nun zu lösen, zeigt sich schnell, dass Kompromisse zu machen sind. Es ist eine große Herausforderung, einen Mehrwert zu schaffen, dieses mit geringem Budget umzusetzen und zuverlässig und gut bedienbar zu gestalten.

Durch Neukombination bereits etablierter Standardkomponenten ist schon ein gutes Ergebnis zu erreichen, allerdings haben Aufbau, Bedienbarkeit und Zuverlässigkeit noch reichlich Potential für Verbesserungen. Es müssen viele Geräte verwendet werden, die auf komplexem Wege miteinander verschaltet werden und teilweise nicht vollständig miteinader kompatibel sind.

Um das bestmögliche Ergebnis zu erreichen, soll eine prototypische Lösung erarbeitet werden, die so weit wie möglich an den gewünschten Funktionsumfang heranreicht.

Ziel dieser Bachelorarbeit ist es, ein praxistaugliches System zu entwickeln, dass die Vorteile des geschlossenen BMD Ökosystems mit den Funktionen eines hybriden Bildmischers vereint.

2 Grundlagen

Um ein gutes Verständnis der folgenden Arbeit zu gewährleisten, wird der aktuelle Stand der verwendeten Videotechnik beschrieben, der Begriff des *hybriden Bildmischers* definiert und am Beispiel des *NewTek TriCaster* erklärt. Es wird die technische Ausgangssituation zur Entwicklung des Systems dargestellt und Probleme der aktuellen Umsetzung aufgezeigt.

2.1 Technischer Hintergrund

Die Videotechnik hat sich in den letzten Jahrzehnten schnell weiterentwickelt. Getrieben durch erhöhte Nachfrage nach Livestreams, teils bedingt durch die Coronapandemie, gab es in den letzten Jahren insbesondere im unteren Preissegment der Videotechnik große Entwicklungssprünge. Auch günstige Kamera- und Bildmischertechnik bietet nun Funktionen, die zuvor nur aus dem High-End-Broadcast bekannt waren. Die Firma Blackmagic Design (BMD) bietet inzwischen Produkte in nahezu jedem Bereich der Videotechnik an. Hierbei wird der Systemansatz verfolgt: Alle Geräte harmonieren miteinander und lassen sich zentral steuern. Es gibt eine zentrale Software, mit der der Bildmischer bedient wird. Zusätzlich lassen sich dort Kameras fernsteuern, Audiomischungen erstellen, Videomaterial aufzeichnen, Playbacks einspielen und Livestreams encodieren. Mit der Hardware von BMD lässt sich so ein komplettes Broadcast System, zu Preisen, bei denen man bei Wettbewerbern, wie Beispielsweise GrassValley, lediglich einzelne Komponenten erhält, aufbauen. Will man nun jedoch zusätzliche Quellen und Geräte in dieses System integrieren, stößt man leicht an Grenzen, vor allem im Bereich der Audio- und Videoschnittstellen. Hier setzt BMD auf bewährte und branchenübliche Anschlüsse, wie z.B. BNC zur SDI-Videoübertragung und XLR zur analogen Audioanbindung an die Videomischer. Die kleineren Bildmischer von BMD verwenden HDMI-Anschlüsse zur Videoübertragung. Zusätzliche Audioschnittstellen sind dort sogar nur per Miniklinke vorgesehen.

Durch den immer häufigeren Einsatz von Netzwerktechnik im Veranstaltungsbereich bietet es sich an, diese vermehrt im Audio- und Videobereich einzusetzen, da es viele Abläufe erleichtert und beispielsweise im Audiosektor seit Jahren erprobt und gut verfügbar ist. Kaum eine Veranstaltung kommt ohne Netzwerkinfrastruktur aus, es wäre somit ohne große Schwierigkeiten möglich, diese mitzunutzen.

AV-Systeme werden üblicherweise auf Basis von TCP/IP-Netzwerken betrieben. Audioprotokolle wie z.B. *Dante* der Firma *Audinate* werden seit Mitte der Nullerjahre erfolgreich eingesetzt. In der Videotechnik hält dieser Wandel nun auch Einzug. Im Bereich der lokalen Videoübertragung scheint sich NDI, einem kostenlosen Video-Over-IP Standard des Herstellers *NewTek*, durchzusetzen. Im Jahre 2019 wurde Version 4 des Protokolls veröffentlicht und erfreut sich immer größerer Beliebtheit. [25] Die 2021 veröffentliche Version 5 von NDI bringt dabei neue Funktionen, wie beispielsweise *NDI Remote*, einer einfachen Lösung um Videoübertagung über das Internet bereitzustellen, mit. Mit dieser Lösung lassen sich Videosignale, visuell nahezu verlustfrei, in üblichen TCP/IP-Netzwerken übertragen. Unkomprimierte Audiosignale, ein Alphakanal und Metadaten sind ebenfalls enthalten. Computer können mit wenigen Klicks als Videoquellen- oder Senken ins Netzwerk eingespeist werden und sind dann frei abonnierbar. Die Signalverteilung wird auf diese Weise stark erleichtert, da bereits vorhandene Netzwerkinfrastruktur genutzt werden kann und es nicht notwendig ist, für jedes Videosignal eine eigene Leitung zu verlegen, bzw. einen dedizierten Eingang am Videomischer zu verwenden.

Diese modernen Funktionen bildet aktuell kein Bildmischer im BMD Ökosystem ab. Zusätzlich sind Funktionen häufig über mehrere Geräte verteilt, die sich zwar verbinden und gemeinsam steuern lassen, allerdings meist zusätzliche Hardware benötigen. Beispielsweise unterstützen ATEM Bildmischer kein Videoplayback. Um dies umzusetzen, wird dann ein *BMD Hyperdeck* benötigt, welches aber mit gleichen Steuersoftware angesprochen werden kann. Möchte man nun Funktionen dieser Art oder eine Kombination dieser Funktionen in einem Gerät verwenden, so bleibt einem nur der Wechsel auf einen Bildmischer eines anderen Herstellers.

2.2 Hybride Bildmischer

Der Begriff *hybrider Bildmischer* ist keine klar definierte Produktkategorie. Daher wird im Folgenden eine Definition zur weiteren Verwendung innerhalb dieser Thesis erarbeitet.

Hybride Bildmischer zeichnen sich durch vielseitige Konnektivität aus, welche sowohl aus konventionellen als auch aus netzwerkbasierten Schnittstellen besteht. Zusätzlich bietet diese Klasse von Geräten einen großen Funktionsumfang, der diverse Anwendungen und Arbeitsplätze der klassischen Broadcast-Technik vereinen kann. Im Gegensatz zu hardwarebasierten Bildmischern sind diese softwaredefinierten Systeme auf Basis gewöhnlicher Computerhardware aufgebaut und werden durch zusätzliche spezialisierte Grafikkarten und Schnittstellen erweitert. Dabei lassen sich zwei Kategorien unterscheiden: Systeme, die vollständig softwarebasiert sind und Systeme, die an eine spezifische Hardware gebunden sind. Als Beispiele für softwarebasierte Systeme sind *Open Broadcaster Software (OBS)*[26], *VMix*[20] oder *Wirecast*[34] zu nennen, mit der Besonderheit, dass es sich bei *OBS* um eine Software unter Open

Source Lizenz handelt, die frei nutzbar ist und von einer großen Community verwendet und weiterentwickelt wird. Bei den hardwarebasierten Systemen sind vor allem die *TriCaster*-Produkte von *NewTek* zu nennen.[24] Diese Systeme werden als fertige Videomischer ausgeliefert, bestehen allerdings aus modifizierten Windows-Computern, welche um die um proprietäre Ein- und Ausgangskarten erweitert wurden. Die Softwarelizenz ist an die Hardware gebunden und lässt sich nicht auf andere Geräte übertragen. Dies bringt sowohl Vor- als auch Nachteile mit sich: Es ist zwar von Vorteil, dass die Hardware genau auf die benötigte Leistung des Systems abgestimmt ist, allerdings sind diese Geräte dadurch nicht erweiterbar und lassen sich nicht an individuelle Bedürfnisse anpassen. Bei softwarebasierten Bildmischern ist das nicht der Fall: Die Lizenz ist unabhängig von der Hardware und die Systeme lassen sich somit auf jeden Anwendungsfall passend zuschneiden.

All diese Systeme zeichnet aus, dass sie viele Funktionen auf einem Gerät vereinen. So lassen sich mit hybriden Bildmischern nicht nur die Eingangsignale verarbeiten und unter Verwendung von Schnitten, Übergängen und Keyern zu einer Sendung zusammenschneiden, sondern sie ermöglichen es auch Videozuspielungen, Live-Grafiken und Countdowns direkt aus dem Bildmischer wiederzugeben. Dazu kommen Audiomischungen, Replays, Aufzeichnugsmöglichkeiten, Einbindung von Webinhalten und Netzwerkschnittstellen, die zusätzliche Kommunikation zu weiteren Geräten ermöglichen. Die genannten Systeme bieten diese Funktionen und verfügen ebenfalls über Streaming-Encoder, sodass ein Livestream direkt aus dem Videomischer abgesetzt werden kann. Während ihrer Lebensdauer lassen sich diese Systeme immer wieder updaten und softwareseitig mit neuen Funktionen versehen, sodass stetig Neuerungen hinzukommen und die Bildmischer immer besser an die Anforderungen angepasst werden können.

Abschließend lassen sich hybride Videomischer wie folgt definieren:

> *Ein softwarebasietes Videoproduktionssystem mit klassischen Bildmischerfunktionen, erweitert um netzwerkbasierte Audio- und Videoschnittstellen, Graphics, Recording, Playback, Audiomischung und Encoding.*

2.3 Besonderheiten des TriCaster

Bei der Betrachtung der hybriden Bildmischer sticht der *TriCaster* besonders hervor. So beschreibt ihn Hersteller *NewTek* mit den Worten:

> The most complete production system available today, TriCaster TC1 represents the continued innovation of the iconic product that defined an industry. Designed with the way you work in mind and equipped with hundreds of advanced

> production capabilities, it has everything you need to do video your way today — and tomorrow.[24]

Der *TriCaster* zeichnet sich dadurch aus, dass er die aufgestellten Anforderungen an hybride Bildmischer erfüllt und als hardwarebasierter Bildmischer auftritt. Somit ist gewährleistet, dass alle Funktionen, die beworben werden, verwendet werden können und es keine Limitierungen durch die Auswahl der Hardware gibt. Dadurch gehen jedoch Aufrüst- und Anpassbarkeit der Hardware verloren. Sollten weitere Funktionen benötigt werden, müssen zusätzliche Geräte angeschafft oder der Bildmischer ausgetauscht werden.

Betrachtet man nun den Funktionsumfang des TriCaster genauer, so fallen dort viele Besonderheiten auf, die andere Bildmischer nur sehr umständlich oder gar nicht bieten können.

Abbildung 2.1: Schaubild aller Quellen des TriCaster TC1 von NewTek.

Der TriCaster stellt insgesamt 16 NDI-Inputs zur Verfügung, von denen 4 Eingänge wahlweise per NDI oder 3G-SDI genutzt werden können. Es gibt zwei integrierte Videoplayer und diverse Standbild- Animation- und Clipebenen. Dazu kommen zwei Streaming-Encoder, die komplett unabhängig verwendet werden können. Es können diverse Signale zeitgleich aufgezeichnet werden. Die maximale Aufzeichnugsdauer ist dabei vom verwenden Format und dem lokalen Speicherplatz abhängig. Aufzeichnungen auf Netzlaufwerke sind ebenfalls möglich. Ausgangsseitig bietet das Gerät vier Mixe, die per SDI und NDI abrufbar sind. Die Signale aller SDI-Eingänge und der bis zu drei Multiviewer-Ansichten sind ebenfalls als NDI-Quellen verfügbar. Zusätzlich zur Hauptebene des Videomischers gibt es vier weitere Mix/Effekt Busse (M/E). Jede dieser M/Es kann auf alle Inputs zugreifen und bietet vier sogenannte Downstream Keyer (DSK). Diese Keyer ermöglichen es, verschiede Videoquellen, Bilder oder Logos freizustellen und anzuordnen. Die M/Es und der Hauptmischer lassen sich alle untereinander verschalten und auch kaskadieren.

Audiotechnisch bietet der TriCaster einen großen internen Funktionsumfang sowie viele externe Schnittstellen. Es stehen 16 Eingänge, mit jeweils vier Kanälen zur Verfügung, welche aus allen beliebigen Schnittstellen gespeist werden können. Diese lassen sich entweder mit den eingebundenen Audiosignalen aus den Videoquellen oder mit zusätzlichen Audiosignalen, entweder analog, über die zwei XLR- und zwei Klinkeneingänge, per zusätzlichem USB-Audiointerface oder per *DANTE* oder anderen AES67 kompatiblen Protokollen, die sich über die *Windows WDM-Treiber* anbinden lasen, bespielen.

Besonders hervorzuheben sind dabei noch zwei Funktionen, die den TriCaster deutlich von anderen Bildmischern abheben: die *Makrofunktion* und die *Compositions*. [24]

2.3.1 Makros

Die Makrofunktion ermöglicht es, alle Funktionen des TriCaster, also jede Handlung, die sich mit Tastatur, Maus und Control Panel ausführen lässt, in abrufbare Sequenzen zu speichern. Diese lassen sich anschließend auf unterschiedlichen Wegen abrufen. Dazu gehören z.B. MIDI-Signale oder Tastaturbefehle. Es lassen sich für jedes Makro bis zu vier Trigger frei setzen. Zur Programmierung der Makros gibt es unterschiedliche Wege. Der einfachste Weg ist dabei das Aufzeichnen. Hierzu legt man eine neues Makro an, gibt ihm einen Namen und startet dann die Aufzeichnung. Nun werden alle Befehle der Benutzeroberfläche aufgezeichnet und in das Makro gespeichert. Standardmäßig wird hierbei auch die Zeit zwischen den jeweiligen Interaktionen aufgezeichnet.

Sollte das einmal nicht gewünscht sein, so lassen sich die einzelnen Schritte der Makros anschließend einfach editieren. Hierzu öffnet man den Edit-Dialog und findet dort alle Schritte des Makros vor. Es werden die einzelnen Funktionen und Ebenen, mit den zugehörigen Quellen und Senken und der benötigten Zeit angezeigt. Diese Punkte lassen sich alle individuell bearbeiten und den jeweiligen Anforderungen anpassen. Die Befehlsstruktur ist dabei einfach gehalten und nahezu in Klartext geschrieben, sodass der Überblick leicht fällt. Das Kopieren von Makros ist ebenfalls vorgesehen, sodass die Erstellung vieler ähnlicher Makros erleichtert wird.

Eine gute Übersicht über die verfügbaren Funktionen im TriCaster gibt NewTek auf dem eigenen Youtube Kanal. Um den Makro Workflow anschaulicher darzustellen, wird das folgende Video empfohlen.[22]

2.3.2 Compositions

Die Composition-Funktion(Comp) ist ein Presetspeicher, welcher zur schnellen Umkonfiguration der M/Es und des Hauptmischers verwendet werden kann. Dabei lassen sich

unterschiedliche Layouts der einzelnen DSKs innerhalb einer M/E abspeichern und wiederherstellen. Diese Funktion ist zwar nicht sonderlich außergewöhnlich, jedoch sind diese Comps beim TriCaster vollständig animiert und lassen sich somit deutlich kreativer nutzen und z.B. auch im aktiven Programmbild fließend bewegen. Dabei können Position, Rotation, Zoom, Crop, Transparenz und Aktivierung aller DSKs des Layers abgespeichert werden. Diese lassen sich dann mit einer voreingestellten Überblendungszeit abrufen. Mittels der Makrofunktion lassen sich also kreative Looks und weiche Übergänge zwischen verschiedenen Bildkompositionen umsetzen. Sie bietet sich somit an, wenn häufig mehrere Bildquellen kombiniert werden sollen. Eine anschauliche Erklärung dieser Funktion gibt die Firma *BC Live Productions* im folgenden Video.[30]

Die Comps lassen sich beispielsweise auch für Vortragssituationen verwenden. Analog zum Beispiel im Video wird dort eine Kameraquelle durch das Präsentationsbild der PowerPoint ersetzt und die Größenverhältnisse angepasst. So lässt sich die Form und der Übergang zu einem sonst eher schichten Bildaufbau hochwertiger gestalten.

2.4 Ausgangssituation

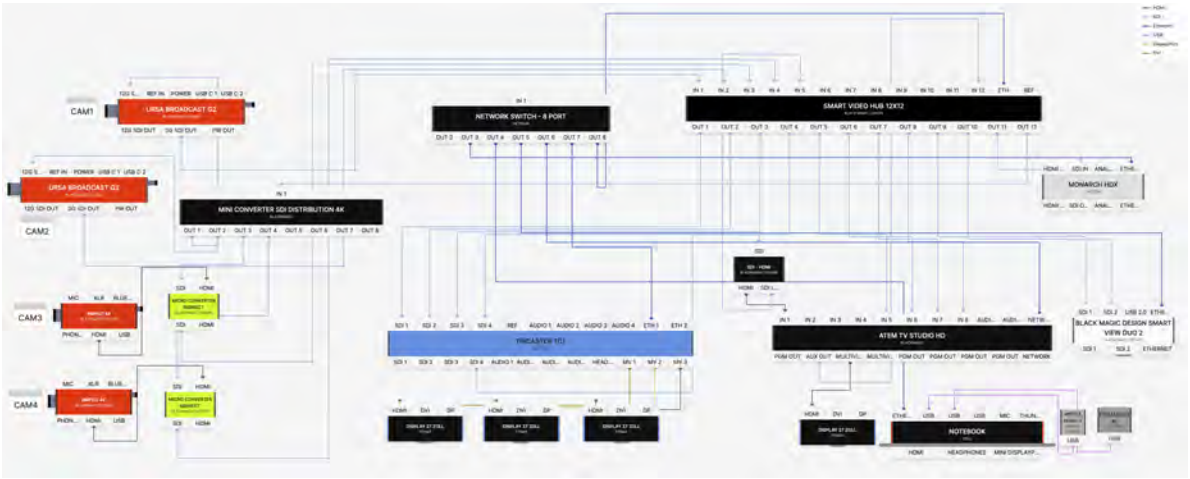


Abbildung 2.2: Signalplan des vorhandenen Kamerakontrollsystems

Im folgenden Kapitel wird eine kurze Übersicht über das bisher verwendete Videosystem geben. Dabei ist die Auswahl des verwendeten Equipments vor allem durch betriebliche und wirtschaftliche Faktoren bedingt. Diverse Geräte sind bereits vorhanden und sollen weiterhin genutzt werden, sodass grundlegende Änderungen an der Systemkonzeptionen im Rahmen dieser Arbeit nicht umsetzbar sind. Der Funktionsumfang und die Verschaltung der vorhanden Technik wird umfassend beschrieben und erklärt.

2.4.1 Hardware

Das aktuell verwendete Bildmischersystem besteht aus der Kombination eines *NewTek TriCaster TC1*[24] und einem *BMD* ATEM *Television Studio HD*[5]. Um die Kamerasignale passend an die Geräte zu verteilen, kommt eine *BMD Smart Videohub 12x12* Kreuzschiene zum Einsatz. Die verwendeten Kameras sind zwei *BMD URSA Broadcast G2*[10] und zwei *BMD Pocket Cinema Camera 4k*[9]. Die Kameras werden direkt an die Kreuzschiene angeschlossen und die Signale von dort an beide Bildmischer verteilt. Um die beiden *BMD Pocket Cinema Camera 4k* anzuschließen, werden HDMI-SDI Signalwandler benötigt, da die Kameras lediglich mit einem HDMI-Ausgang ausgestattet sind. Hierzu werden *BMD Micro Converter BiDirectional SDI/HDMI 3G (BIDI)*verwendet. Diese Geräte ermöglichen nicht nur die Wandlung des Videosignals auf SDI, sondern bieten auch die Möglichkeit, das eingehende SDI-Rücksignal auf HDMI zu wandeln und die Steuersignale zu erhalten.

Der Rückkanal zu den Kameras beginnt im *TriCaster*. Ein SDI-Output sendet das Programmsignal in die Kreuzschiene, welche es in den *ATEM* weiterleitet. Hier liegt das Signal als zusätzliche Videoquelle neben den Kameras an, wird auf dem Aux-Ausgang des *ATEM* wieder ausgespielt und nun auf die Kameras verteilt. Das Durchschleifen des Programmbildes durch den *ATEM* ist notwendig, um die Kamerakontrollsignale in das SDI-Signal einzubetten. Das Aux-Signal lässt sich jetzt an die Kameras zurückführen. Bei den *URSA Broadcast G2* Kameras gibt es einen SDI-Eingang für das Rückbild. Für die beiden *Pocket4k* Kameras kommt die Besonderheit des *BIDI* zum Tragen. Neben der Konvertierung der Videosignale, ermöglicht der *BIDI* auch die Umwandlung der eingehenden SDI-Steuersignale auf HDMI-CEC-Befehle, die dann von der Kamera verarbeitet werden können.

Alle Steuersignale werden zu jeder Kamera übertragen, da diese das gleiche SDI-Rücksignal erhalten. Die Zuordnung der Kameras funktioniert mittels einer ID, die bei den *URSA Broadcast G2* im Menü der Kamera eingestellt werden kann. Bei den *Pocket4K* muss die ID am jeweiligen *BIDI*(mit der BMD Converters Setup Software) gesetzt werden, da die Übertragung der HDMI-CEC-Steuersignale nur für eine Kamera vorgesehen ist.(S. Abb. 2.3)

Ein Vektorskope wird mit der Kreuzschiene verbunden, um damit die Kamerabilder messtechnisch bewerten zu können. Die Umschaltung der Quellen erfolgt über die Kreuzschiene. Alle verwendeten Geräte befinden sich in einem gemeinsamen TCP/IP-Netzwerk, sodass sie von dort mittels Notebooks gesteuert und konfiguriert werden können. An das Notebook ist ein *Elgato Streamdeck XL*[15] angeschlossen, um die Bedienung zu erleichtern und programmierbare und digital beschriftbare Tasten zur Verfügung zu stellen.

2.4.2 Software

Die verwendete Software teilt sich in zwei Kategorien auf. Einerseits die Programme von BMD, zur Bedienung der verbauten BMD-Geräte und andererseits *Bitfocus Companion*[2] als etablierte Open Source Lösung zur einheitlichen Konfiguration und Steuerung der verwendeten Komponenten anderer Hersteller.

Um den *ATEM* zu steuern, wird die Software *ATEM Software Control*[4] benötigt. Diese funktioniert für alle Videomischer aus der *ATEM* Serie und ermöglicht die Steuerung aller Funktionen dieser Geräte. Dazu zählen z.B. Bildmischung, Audiomischung und Kamerasteuerung. In diesem Setup wird die Software nahezu ausschließlich dazu verwendet die Farbkorrektur der Kameras durchzuführen.

Die Kreuzschiene lässt sich zwar auch direkt am Gerät steuern, allerdings ist es deutlich komfortabler die Software *BMD Videohub Control Utility* zu verwenden. Hier lassen sich die Quellen und Senken einfach zuordnen und beschriften. Außerdem lässt sich der Videohub so überall im Netzwerk kontrollieren, ohne sich direkt am Gerät befinden zu müssen.

Um die passenden IDs der Kameras in den *BIDI*-Konvertern einstellen zu können, wird die Software *BMD Converters Setup* verwendet. Ein einfaches Nutzerinterface ermöglicht den, per USB angeschlossenen, Konverter zu konfigurieren und diese Einstellungen zu speichern.

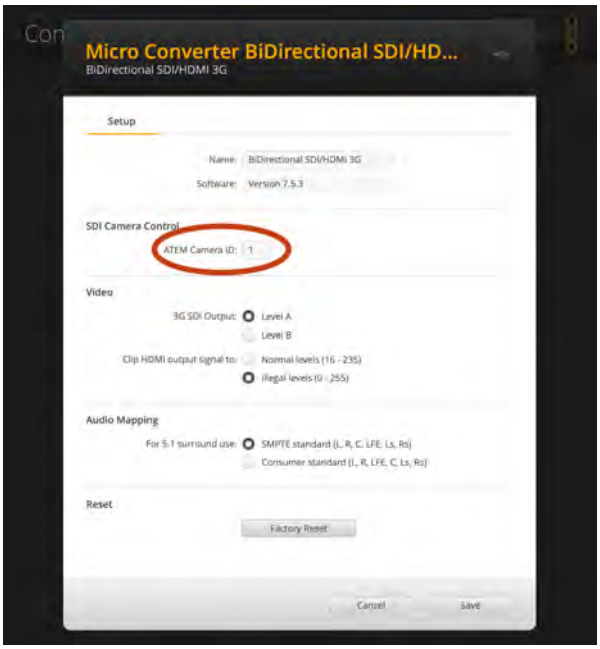


Abbildung 2.3: Camera-ID Einstellung in BMD Converters Setup

Der *Smart Scope* benötigt ebenfalls eine eigene Software, um die unterschiedlichen Funktionen, die dieses Gerät bietet, wie Anzeige des Videobildes, eines Histogramms oder des Vectorscopes, umzuschalten. Dazu wird die *BMD SmartView Software* verwendet.

Um nun ein einfacheres Nutzerinterface und die Verknüpfung mehrerer Geräte umsetzen zu können, wird *Companion*[2] von *Bitfocus AS* verwendet. Die Software ist quelloffen, wird fortlaufend weiterentwickelt und ermöglicht es alle genannten Geräte zentral zu steuern und die Bedienung dieser zu vereinfachen.

Ein zentraler Punkt dieser Anwendung ist es die Verknüpfung zwischen *TriCaster* und *ATEM* zu ermöglichen. Diese Verbindung ist erforderlich, da ansonsten die Tally-Funktion der Kameras nicht genutzt werden kann. Um diese Verbindung zu realisieren gibt es bei Companion, zusätzlich zu den Buttons, eine Möglichkeit *Trigger* zu setzen.

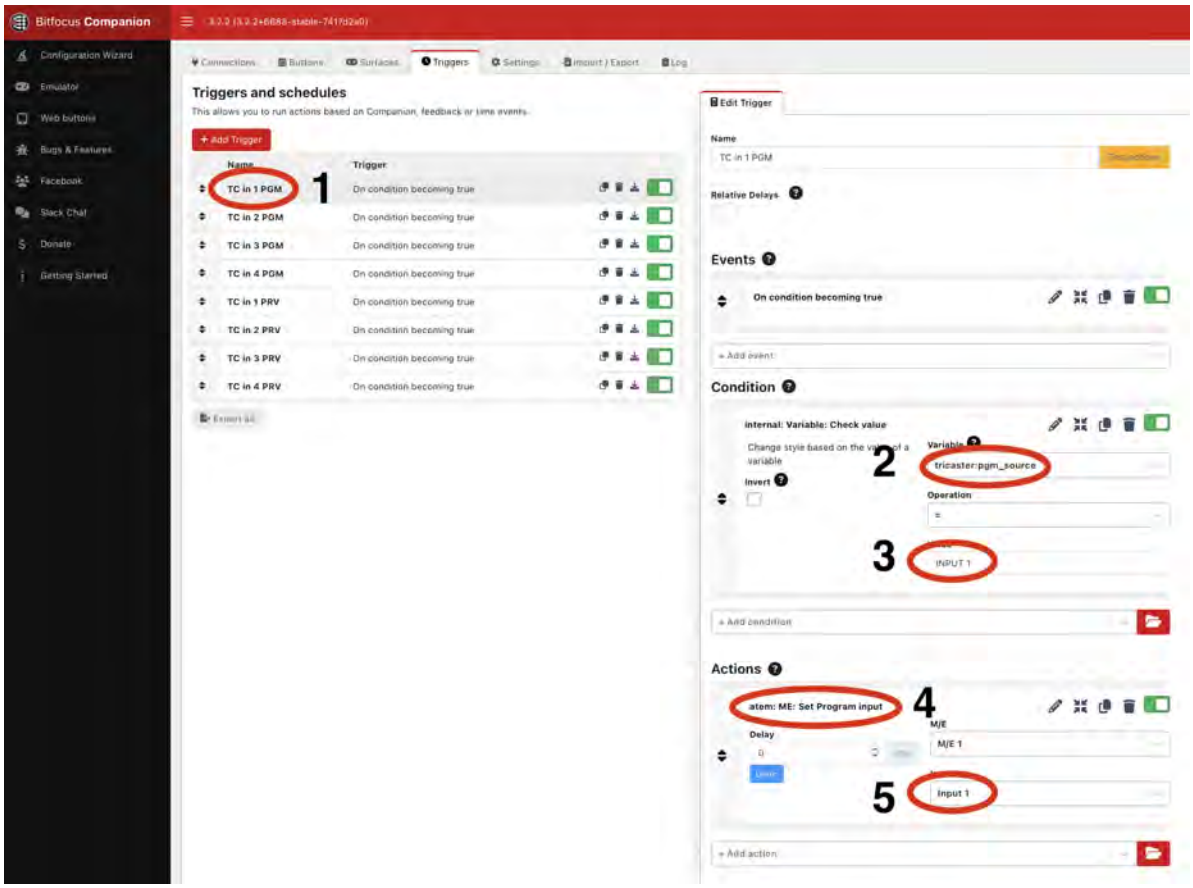


Abbildung 2.4: Annotierte Benutzeroberfläche von Bitfocus Companion zur Konfiguration der Trigger-Funktion

Nach Anwahl des Speicherslots (1) wird die zu überwachende Variable ausgewählt (2) und der erwartete Inhalt festgelegt (3). Anschließend wird die auszuführende Aktion (4) festgelegt und schlussendlich die zu sendende Variable eingetragen (5).

Hierzu ist die Verknüpfung so gesetzt, dass der Trigger auf die Information des TriCaster, ob eine bestimmte Kamera grade im Program ist, wartet und dann anschließend die gleiche Kamera im *ATEM* auswählt und ins Program schaltet. Diese Funktion wiederholt sich für alle vier Kameras, jeweils für Preview und Program. So wird ermöglicht, dass sowohl das Tally als auch die Farbkontrolle zeitgleich funktionieren.

2.4.3 Probleme der aktuellen Umsetzung

Die beschriebene Konfiguration funktioniert grundsätzlich und bietet die erforderlichen Funktionen, jedoch sind einige Verbesserungsmöglichkeiten erkennbar. Hardwareseitig fällt insbesondere auf, dass zwei Bildmischer erforderlich sind, um sämtliche Funktionen bereitzustellen. Dabei erfüllt lediglich ein Bildmischer seine primäre Aufgabe der Verarbeitung sendungsrelevanter Inhalte, während der zweite Bildmischer ausschließlich für die Kamerasteuerung und Tallyy verwendet wird. Um die Kamerasignale auf die Bildmischer zu verteilen und die Verteilung der Rücksignale zu gewährleisten, ist eine zusätzliche Kreuzschiene erforderlich. Diese Technik nimmt viel Platz ein und bringt ein hohes Gewicht mit. Das Case des *ATEM*, inklusive Kreuzschiene und Anschlussfeldern, wiegt etwa 21,5kg und hat die Maße(BxTxH) 54,5x52x28cm.

Die Verbindung der beiden Bildmischer erfordert einen zusätzlichen Computer, welcher ein wesentlicher Bestandteil des Setups ist. Ein Ausfall dieses Computers hat keine direkten Auswirkungen auf das Sendebild, jedoch können keine Farbanpassungen mehr vorgenommen werden und die Tallys funktionieren nicht, da sie von *Companion* abhängig sind.

Im Betrieb ist zu bemerken, dass die Ansteuerung der Tallys zwar im Allgemeinen funktioniert, jedoch mit variabler Latenz. Die normale Reaktionsgeschwindigkeit der Tallys, beispielsweise bei direktem Bildwechsel im *ATEM*, ist nahezu verzögerungsfrei. Wenn jedoch die Quelle am TriCaster gewechselt wird, erhöht sich die Latenz aufgrund der vielen Zwischenschritte erheblich. Bei längeren Betriebszeiten ist festzustellen, dass diese Latenzzeit schwankt. Die Ursachen dafür sind nicht offensichtlich und beeinträchtigen die Kameraarbeit, da es teilweise mehr als eine Sekunde dauern kann, bis der Umschnitt am Tally der Kamera angezeigt wird.

Diese Aspekte ermöglichen zwar den Betrieb des Systems, jedoch ist weder der Komfort noch die Reaktionsfähigkeit oder Zuverlässigkeit gewährleistet. Außerdem wirkt sich die hohe Komplexität des Systems negativ auf die Wirtschaftlichkeit und Wartbarkeit aus.

3 Planung des neuen Systems

Bei der Betrachtung des aktuellen Systems haben sich mehrere Punkte herausgestellt, die die Arbeit mit dem Videosystem erschweren und Ansatzpunkte bieten dieses System zu überarbeiten und damit eine Verbesserung der Leistung und der Bedienung zu erzielen.

3.1 Ziele für die neue Lösung

Bei der Überarbeitung des aktuellen Systems soll es vorrangig darum gehen, den technischen Aufwand zu reduzieren, ohne dabei den Funktionsumfang der jetzigen Lösung zu verlieren. So soll für den Einsatz der Bildtechnik deutlich weniger Hardware benötigt und die Doppelung der Bildmischer abgeschafft werden. Die Funktionen der Kamerakontrolle dürfen dabei nicht verloren gehen. Im Gegenteil bietet es sich sogar an, den aktuellen Funktionsumfang zu erweitern, sollte die Technik dies zulassen. Eine praktische Erweiterung wäre die Möglichkeit eingestellte Werte zwischen den Kameras zu kopieren oder Einstellungen als Startwerte festzulegen. Da für das System ein eigenes Nutzerinterface erstellt werden muss, soll es ein simples Bedienkonzept dafür geben. Zusätzlich soll sich die Bedienoberfläche leicht und frei gestalten lassen, sodass sie mit geringem Aufwand an unterschiedliche Szenarien angepasst werden kann.

Nachfolgend werden konkrete Anforderungen festgelegt, die das neue System erfüllen soll.

3.1.1 Funktionen

Folgende, bereits im Ausgangssystem enthaltene Funktionen sollen erhalten werden.

Steuerung von:

- Blende
- Gain
- Verschlusswinkel

- Weissabgleich und Tint
- Kontrast und Pivot
- Farbton und Sättigung
- Farbe(R,G,B,Y Steuerung für Lift, Gamma und Gain)
- Fokus
- Tally mit dem TriCaster

3.1.2 Hardware

Bei der Hardwareauswahl sollen die Funktionsanforderungen beachtet werden. Zusätzlich ist eine Größenreduktion des Systems gewünscht. Damit verbunden soll auch der Aufwand des Aufbaus und der Inbetriebnahme reduziert werden.

Das neue System sollte über etablierte Standardverbinder, wie SDI und RJ-45 Anschlüsse verfügen. Optional wäre eine permanente Montage innerhalb des TriCaster-Cases.

Zusätzlich zu den genannten Punkten soll die Marktverfügbarkeit und eine breite Nutzerbasis beachtet werden. Dabei bietet es sich an, branchenübliche Systeme zu verwenden, die sich einfach erlernen und austauschen lassen, sollten Verbesserungen oder Reparaturen vorgenommen werden müssen oder der Stand der Technik deutlich fortgeschritten sein. Die Wirtschaftlichkeit sollte ebenfalls Beachtung finden, da eine kosteneffiziente Lösung immer von Vorteil ist. Bereits vorhandene Hardware soll möglichst Verwendung finden.

3.1.3 Software

Bei der Auswahl der Software muss sowohl auf die Funktionsanforderungen als auch auf die Kompatibilität der Hardware geachtet werden. Da die Software den größten Teil des Nutzererlebnisses bestimmen wird, muss ein besonderer Fokus auf die Möglichkeit zur Erstellung von leicht verständlichen und gut bedienbaren Nutzerinterfaces liegen. Wünschenswert ist dabei, dass die Platform eine zukünftige Änderung der Oberflächen ermöglicht.

Standardisierte und branchenübliche Protokolle zur Kommunikation sollen verwendet werden, damit die Möglichkeit gegeben ist, zukünftig weitere und andere Kontrolloberflächen an das System anzubinden. Sollte es bereits bekannte und nutzbare Software geben, ist diese bevorzugt zu verwenden um den Entwicklungsaufwand in einem möglichst geringen Rahmen zu halten.

3.2 Auswahl der Hardware

Die Auswahl der Hardware lässt sich nur in Kombination mit der Bewertung des Funktionsumfangs der zugehörigen Software treffen. Somit spielt Software auch hier ein Rolle, wenn auch im Folgenden nicht explizit erwähnt.

Um nun die passenden Produkte zu den gesetzten Kriterien zu finden, bietet es sich an, nach bereits existierenden Lösungen zu suchen und zu bewerten, ob diese für den gesetzten Anwendungsfall nutzbar sind. Hierbei fällt der Hersteller *Skaarhoj* auf, der unterschiedlichste Kontrollsysteme für den Videobroadcast-Markt anbietet. Dabei hat *Skaarhoj* eine eigene Softwareplattform geschaffen, auf der Ihre Systeme betrieben werden. Diese lassen sich mit einer Vielzahl branchenüblicher Technologien nutzen, sind allerdings größtenteils an die eigene Hardware gebunden, die hochpreisig ist. Kleine Controller liegen oft schon jenseits der 1000€.[32]

Sowohl der Preis als auch die dedizierte Hardware und die Closed-Source Software, die erforderlich ist, um dieses System zu nutzen, disqualifizieren es für den Einsatz im Rahmen dieses Projektes.

Die Suche nach einem preisgünstigen und flexiblen Weg, die in SDI-Signalen eingebetteten Steuersignale zu manipulieren, ergab ein Produkt von BMD, das *3G-SDI Shield*[3]. Dabei handelt es sich um ein Zusatzmodul für Arduino Mikrocontroller, mit dem sich Steuerdaten der Kameras aus einem 3G-SDI Signal extrahieren und einbetten lassen.

Dieses Gerät erfüllt die Anforderungen der Kommunikation mit der Kamera und bietet mit dem angeschlossenen Arduino die Plattform für weitere Schnittstellen. Dafür bietet es sich an, einen klassischen *Arduino UNO*[1] zu verwenden, da dieser software- und hardwareseitig nativ mit dem *3G-SDI Shield* kompatibel und somit sofort einsetzbar ist. Die Kommunikation der beiden Boards findet dabei über I^2C statt, einem seriellen Datenbus, der häufig Anwendung in Bereich der Mikrocontroller findet.[11]

Beim Arduino UNO handelt es sich um ein einfaches Mikrocontroller-Board, das auf einem ATmega328P basiert. Der Arduino bietet diverse analoge und digitale Schnittstellen, jedoch keine native Ethernet-Schnittstelle. Diese lässt sich mit einem weiteren Shield realisieren, welches dann ebenfalls auf den Arduino aufgesetzt werden kann und per SPI, einem weiteren seriellen Bussystem, mit dem Arduino kommuniziert. [13][33] Diese Geräte lassen sich mit einem gemeinsamen 12V Netzteil betreiben, da die Spannungsversorgung der drei Boards per GIPO-Pins verbunden ist.

Da es sich bei diesen Geräten um Mikrocontroller handelt, deren Komponenten und Anschlüsse offen auf ihren entsprechenden Platinen montiert sind, bietet es sich an, ein Gehäuse zu bauen, sodass mechanische Beschädigungen vermieden werden können.

Mit dieser Hardware sind nun die nötigen Anforderungen erfüllt, um netzwerkbasierte Steuersignale in SDI-Signale einzubetten und so eine Kamerasteuerung zu ermöglichen. Sollen nun BMD Kameras verwenden werden, die statt SDI einen HDMI-Anschluss besitzt, muss ein Konverter verwendet werden. Dabei ist es wichtig zu beachten, dass einfache SDI-HDMI Konverter keine Wandlung der Steuersignale, zwischen SDI und HDMI ermöglichen. BMD bietet dazu den *Micro Converter BiDirectional SDI/HDMI 3G*[8] an. Dieser ermöglicht es HDMI-basierte Kameras in die SDI Infrastruktur zu integrieren. Schließt man eine Kamera per HDMI an dieses Gerät an, so wandelt der Konverter das Bildsignal und gibt es via SDI aus. Legt man nun das Rücksignal mit den Steuerdaten an den SDI-Eingang an, werden die SDI-Steuerdaten in HDMI-kompatible Steuerdaten umgewandelt und an den HDMI-Eingang der Kamera weitergeleitet, da das HDMI-Protokoll bidirektional aufgebaut ist.

3.3 Auswahl der Software

Im Bereich der Software bietet es sich ebenfalls an, sich an bereits etablierten Anwendungen zu orientieren. Da die bereits ausgewählte Hardware nun der begrenzende Faktor ist, reduzieren sich die Softwareanwendungen auf einen kleinen Bereich. BMD stellt für das *3G-SDI Shield* einige Beispielprojekte für die Kamerasteuerung zur Verfügung. Diese eignen sich gut um einen ersten Funktionstest durchzuführen und die Syntax der zum *3G-SDI Shield* zugehörigen Arduino-Bibliothek kennenzulernen. Der nächste Anlaufpunkt um bereits vorhandene Projekte zu finden ist GitHub, eine Website zur kollaborativen Entwicklung und Veröffentlichung von quelloffener Software.

Kombiniert man nun die Suchbegriffe *Arduino, Shield, SDI, Blackmagic, BMD, Tally* und *Camera* so stößt man auf wenige dutzend Projekte, die mehr oder weniger mit dem geplanten Anwendungsfall übereinstimmen oder zumindest einzele Parallellen aufweisen.

Insbesondere das Projekt *bmControl*[19] des Nutzers *jg33*[17] erweist sich hier als hilfreich, da es bereits mehrere der benötigten Funktionen implementiert. So verwendet dieses Projekt eine Netzwerkschnittstelle und eine klare Syntax zur Eingabe der Steuerdaten per OSC, einem nachrichtenbasierten Netzwerkprotokoll, dass in der Medientechnik häufige Verwendung findet. Die Kommunikation der OSC-Befehle an das BMD *3G-SDI Shield* ist ebenfalls integriert.

Da dieses Projekt unter der Lizenz *GNU GPLv3*[16] veröffentlicht wurde und somit frei nutzbar und modifizierbar ist, wird es als Grundlage der nachfolgenden Programmierung verwendet und an die gesetzten Anforderungen angepasst.

Der nächste Schritt ist die Auswahl einer Software zur Erstellung der Kontrolloberflächen. Hierbei soll das OSC-Protokoll verwendet werden und das Programm idealerweise plattformunabhängig sein, um einen möglichst flexiblen Anwendungsspielraum zu gewährleiten.

Zusätzlich soll die Möglichkeit geboten sein, eine flexible Nutzeroberfläche zu gestalten. Bei der Suche nach einer passenden Anwendung bot sich das Programm *TouchOSC* des Herstellers *HEXLER* auf. Diese Software wurde im Rahmen des Studiums bereits verwendet und erfüllt alle gestellten Anforderungen. Es kann OSC-Nachrichten in nahezu beliebiger Form ausgeben und bietet die Möglichkeit, sehr flexible Nutzerinterfaces zu erstellen. Die Plattformunabhängigkeit ist ebenfalls gegeben, da TouchOSC auf Windows, macOS und Linux, sowie iOS und Android verwendbar ist und sogar einen Editor auf Mobilgeräten zur Verfügung stellt. [18]

3.4 Schnittstellen und Protokolle

Dieser Abschnitt gibt einen Überblick über verwendete Schnittstellen und Protokolle. Dabei werden die Hauptprotokolle wie SDI oder HDMI als grundlegend bekannt vorausgesetzt, spezielle Teilfunktionen wie die Implementierung von Steuersignalen werden jedoch - sofern für diese Arbeit relevant - im folgenden genauer erläutert. Es wird grob auf die gesamten Protokolle eingegengen und anschließend die Besonderheiten weiter ausgeführt.

3.4.1 SDI - Austastlücke

Zur Bildübertragung wird in diesem Projektaufbau das 3G-SDI Protokoll verwendet. Dieses baut auf den vorhergegangenen Versionen von SDI auf und ist im *SMPTE 424M* Standard aus dem Jahr 2006 definiert. Dieses serielle Protokoll bietet die Möglichkeit unkomprimierte 10Bit, differenzcodierte Videosignale mit einer maximalen Auflösung von 1080p, bei maximal 60fps, zu übertragen. Der Name *3G* resultiert dabei aus der maximalen Bandbreite von 2.970 GBit/s.

Neben der Videoübertragung beinhaltet das 3G-SDI Signal eine Austastlücke, die aus der analogen Videoübertragung übernommen wurde. Dadurch, dass in diesem Signalabschnitt keine Videodaten übertragen werden ergibt sich die Möglichkeit, an dieser Position Audiodaten oder sonstige Datenworte zu übertragen. Vgl. [31, S. 155 f.]

Die frei nutzbare Bandbreite in der Austastlücke des SDI-Signals verwendet BMD um dort ihr *SDI Camera Control Protocol* zu übertragen. Dabei richtet sich diese Implementierung nach *SMPTE 291M*. [6, S. 107]

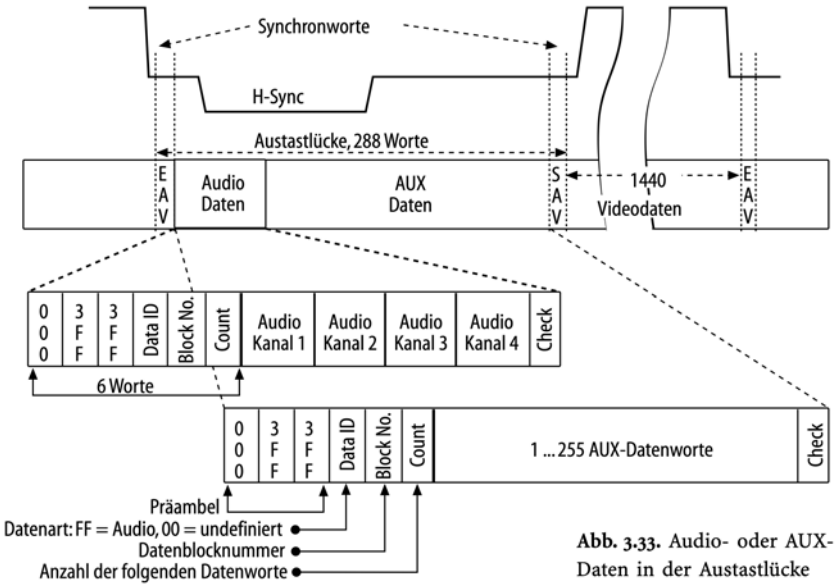


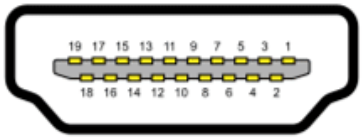
Abb. 3.33. Audio- oder AUX-Daten in der Austastlücke

Abbildung 3.1: Austastlücke im Videosignal [31, S. 156]

3.4.2 HDMI - CEC

Da der HDMI Standard auch im professionellen Sektor immer häufiger vertreten ist, liegt es nahe, dass auch BMD ihre kompakten Kameras ausschließlich mit einem HDMI-Anschluss ausstattet. Um diese Kameras nun auch ansteuern zu können, wie es beim *SDI Camera Control Protocol* der Fall ist, braucht es auch im HDMI-Protokoll einen entsprechend nutzbaren Rückweg. Hierfür verwendet BMD das *Consumer Electronics Control* (CEC) Protokoll, das bereits seit HDMI Version 1.0 spezifiziert und mit HDMI Version 1.1 implementiert wurde. Neben den Highspeed-Datenverbindungen zur Bildübertragung finden sich im HDMI-Anschluss mehrere Datenleitungen, die für zusätzliche Kommunikation, wie z.B. HDCP, Fast-Ethernet oder 5V Spannungsversorgung ausgelegt sind. [14]

Für das CEC Protokoll wird dabei eine Ader, welche im HDMI A Stecker auf Pin 13 aufgelegt ist, verwendet, um eine bidirektionale Kommunikation aufzubauen. Der Inhalt des Protokolls wird von BMD leider nicht offengelegt, dementsprechend lässt sich nicht ohne größeren Aufwand bestimmen, wie der genaue Aufbau der Kamerabefehle ist. Die Vermutung liegt nahe, dass sich der Aufbau an der Form des *SDI Camera Control Protocol* orientiert, da die gleichen Funktionen implementiert werden müssen. Ein gutes Indiz dafür ist das Reverse-Engineering-Projekt von Martijn Braam, der in zwei Blogposts auf seiner Website, ausführlich darüber berichtet, wie er versucht das CEC-Signal zwischen einem Bildmischer und einer BMD Kamera zu analysieren. Dabei kommt er letztendlich zum Fazit, dass sich die Strukturen der Implementierung stark gleichen. [12]



Pin	Signal
Pin 01	TMDS Data 2+
Pin 03	TMDS Data 2 -
Pin 05	TMDS Data 1 Schirmung
Pin 07	TMDS Data 0+
Pin 09	TMDS Data 0-
Pin 11	TMDS Takt Schirmung
Pin 13	CEC
Pin 15	SCL
Pin 17	DDC/CEC/HEC Erdung
Pin 19	Hut-Plug-Erkennung HEC Data + (HDMI 1.4)

Pin	Signal
Pin 02	TMDS Data 2 Schirmung
Pin 04	TMDS Data 1+
Pin 06	TMDS Data 1-
Pin 08	TMDS Data 0 Schirmung
Pin 10	TMDS Takt+
Pin 12	TMDS Takt-
Pin 14	HEC Data - (ab HDMI 1.4)
Pin 16	SDA
Pin 18	+5V DC Spannung

Abbildung 3.2: Belegung eines HDMI A Steckverbinders [14]

Um die Wandlung zwischen SDI CameraControlProtocol und HDMI CameraControlProtocol zu ermöglochen bietet BMD die *MicroConverter Bidirectional* an. Diese ermöglichen die Konvertierung von SDI zu HDMI und umgekehrt, bei vollem Erhalt der Kamerakontrollfunktionen.[8]

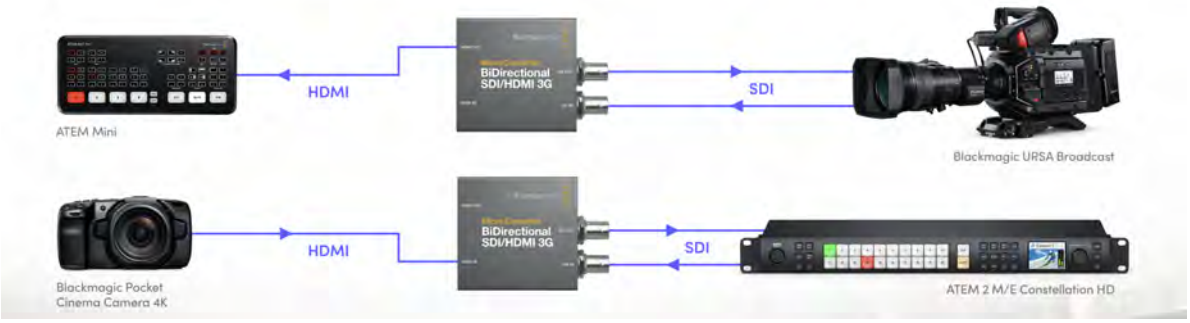


Abbildung 3.3: Signalfluss der BIDI Converter [8]

3.4.3 OSC - Open Sound Control

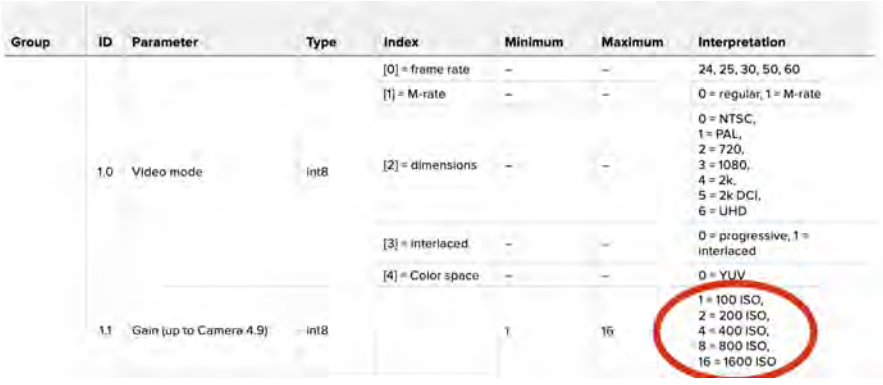
Open Sound Control (OSC) ist ein Kommunikationsprotokoll, das hauptsächlich in der Audio- und Multimedia-Industrie verwendet wird, um Daten zwischen verschiedenen Geräten und Softwareanwendungen, in Echtzeit, auszutauschen. Dabei handelt es sich um einen effizienten Standard der unabhängig vom Transportprotokoll ist, jedoch meist per UDP übertragen wird.

OSC unterstützt etablierte Datentypen wie Fließkommazahlen (float) und Integer (int) mit einer Wortbreite von 32Bit. Zusätzlich sind Zeitstempel im NTP-Format und *Strings*, also Textinhalte, verfügbar.

Die Übertragungsstruktur ist nach folgendem Muster aufgebaut: Adresse, Typ, Nachricht. OSC Nachrichten können als einzelne *Packets* oder als *Bundle*, der Zusammenfassung mehrerer Nachrichten, versendet werden. Die Sender von OSC-Nachrichten werden dabei als *OSC-Client* und die Empfänger als *OSC-Server* bezeichnet. OSC ist weit verbreitet und in viele Anwendungen integriert. Dazu zählen beispielsweise AbletonLive, QLab, Resolume, TouchDesigner, Max/MSP, PureData, Unity und Companion. [21] [29]

4 Praktische Umsetzung

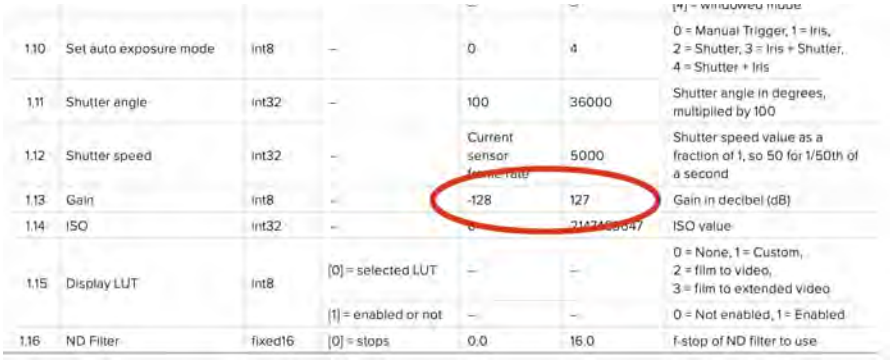
In diesem Kapitel wird der praktische Aufbau des Kontrollsystems dargestellt. Da das verwendete BMD *3G-SDI-Shield* bereits 2016 veröffentlicht wurde und weitere Inhalte und Funktionen teilweise knapp oder unvollständig dokumentiert sind, wird die Softwareentwicklung permanent an einem Testsystem überprüft. Beispielsweise ist die BMD Dokumentation zum *3G-SDI-Shield*, in der aktuellsten Version 1.1, vom 06.07.2018, und somit schon etwa sechs Jahre alt ist.[3] Die derzeit aktuelle Version der Dokumentation des CameraControlProtokolls ist aus dem Jahr 2020 und weist offensichtliche Ungenauigkeiten auf. Beispielsweise ist in Abb.4.1 ein Ausschnitt aus dem CameraControlProtocol der Wertebereich zur Gainsteuerung so definiert, dass es lediglich fünf fest definierte Werte gibt. Moderne BMD Kameras, die weiterhin mit diesem Protokoll angesteuert werden können, reagieren zwar auf diese Werte, bieten allerdings größere Gainbereiche mit feineren Unterteilungen. So ist beispielsweise in der aktuellen Anleitung der BMD Pocket Cinema Cameras[7, S.162] die gleiche Kontrollfunktion mit einem Wertebereich von *-127* bis *128* angegeben. Passend dazu ist auch die ID zur Steuerung verändert worden. Diese beiden Funktionen lassen sich so parallel verwenden und neuere Kameras reagieren auf beide Eingaben. Leider werden nicht alle Anleitungen und Dokumentationen von BMD aktualisiert, wenn eine eue Kamera auf den Markt gebracht oder das Kontrollprotokoll ergänzt wird.



Group	ID	Parameter	Type	Index	Minimum	Maximum	Interpretation
				[0] = frame rate	–	–	24, 25, 30, 50, 60
				[1] = M-rate	–	–	0 = regular, 1 = M-rate
	1.0	Video mode	int8	[2] = dimensions	–	–	0 = NTSC, 1 = PAL, 2 = 720, 3 = 1080, 4 = 2k, 5 = 2k DCI, 6 = UHD
				[3] = interlaced	–	–	0 = progressive, 1 = interlaced
				[4] = Color space	–	–	0 = YUV
	1.1	Gain (up to Camera 4.9)	int8		1	16	1 = 100 ISO, 2 = 200 ISO, 4 = 400 ISO, 8 = 800 ISO, 16 = 1600 ISO

Abbildung 4.1: Ausschnitt aus der SDI-Shield Anleitung mit veralteter Definition der unterstützten Gain-Intervalle.[6][S.162]

Der Arduino Quellcode *bmControl* von Autor *jg33* aus den Jahr 2016[19] ist nur knapp dokumentiert, sodass dort eher auf experimentelles Entwickeln in kleinen Schritten zurückge-



						[4] = windowed mode
1.10	Set auto exposure mode	int8	–	0	4	0 = Manual Trigger, 1 = Iris, 2 = Shutter, 3 = Iris + Shutter, 4 = Shutter + Iris
1.11	Shutter angle	int32	–	100	36000	Shutter angle in degrees, multiplied by 100
1.12	Shutter speed	int32	–	Current sensor frame rate	5000	Shutter speed value as a fraction of 1, so 50 for 1/50th of a second
1.13	Gain	int8	–	-128	127	Gain in decibel (dB)
1.14	ISO	int32	–	0	2147483647	ISO value
1.15	Display LUT	int8	[0] = selected LUT	–	–	0 = None, 1 = Custom, 2 = film to video, 3 = film to extended video
			[1] = enabled or not	–	–	0 = Not enabled, 1 = Enabled
1.16	ND Filter	fixed16	[0] = stops	0.0	16.0	f-stop of ND filter to use

Abbildung 4.2: Ausschnitt aus der PocketCinemaCamera Anleitung mit aktueller Definition der Gaineinstellungen[7]

griffen werden muss, um so sicherstellen zu können, dass die implementierten Funktionen den gewünschten Effekt erzielen. Ebenfalls sind viele Teile des Codes für weitere Kommunikation, beispielsweise Rückkommunikation vom Arduino via OSC vorgesehen. Der Code bietet fast alle benötigten Funktionen dieses Projektes, sodass kaum eigener Code geschrieben wird. Die vorhandene Codebasis wird umfassend erklärt und angepasst, sodass die Möglichkeit der Erweiterung gegeben wird.

4.1 Aufbau des Systems

Um nun mit der Entwicklung beginnen zu können, wird zunächst die Hardware in Betrieb genommen. Der Übersichtlichkeit halber werden die Geräte dem Signalfluss nach geordnet.

Die Eingabe der Signale beginnt mit einem iPad, welches per WLAN an einen Router angeschlossen ist und einen DHCP-Server zur Verfügung stellt. Neben dem Arduino Uno, mit seinen zwei Shields, dem Network Shield und dem *3G-SDI-Shield*, wird hierfür noch ein 12V Netzteil benötigt. Schließt man dieses an das SDI-Shield an, so werden die weiteren Komponenten ebenfalls mit Energie versorgt. Das SDI-Shield des Arduino stellt einen Eingang und einen Ausgang, jeweils ausgeführt als BNC-Anschluss, zur Verfügung. An den Eingang wird der Bildmischer angeschlossen, der das Programmsignal liefert. Für den Testaufbau ist dieser allerdings nicht notwendig, da das SDI Shield bei fehlendem Eingangssignal ein Schwarzbild erzeugt und somit trotzdem Steuersignale übertragen werden können. Der SDI-Ausgang wird mit einem BMD *BIDI* verbunden, der das SDI-Eingangssignal in ein HDMI-Signal wandelt. Dieses HDMI-Signal lässt sich dann mit der BMD *Pocket4K* verbinden.

4.2 Erste Versuche mit dem *CameraControlProtocol*

Zu Beginn des praktischen Aufbaus bietet es sich an, zuerst einen Funktionstest der Handware zu machen. Dabei lassen sich erste Einblicke in die Struktur und Logik der Software und Bauteile gewinnen. BMD liefert zu den *3G-SDI-Shield* nicht nur eine Arduino Library, mit der das Board angesprochen werden kann, sondern auch exemplarische Codebeispiele. Unter anderem gibt es dort den Beispielsketch *TallyBlink*. In diesem Sketch wird die Kommunikation zwischen Arduino und *3G-SDI-Shield*, unter Zuhilfenahme der *BMDSDIControl.h* Library, aufgebaut. Im Setup wird die Verwendung als *TallyOverride*, also die Verwendung des *3G-SDI-Shield* als Erzeuger der Tally-Nachrichten, deklariert und Pin 13, also die rote LED auf dem Arduino Board, als Output definiert.

Im Loop geht es nun darum, das Program Tally(Rot) der Kamera 1 blinken zu lassen. Dieses wird im Sekundentakt ein- und ausgeschaltet. Mit dem Befehl *setCameraTally* lassen sich die Tallys aller Kameras einzelnen ansprechen. Hier wird nun das Tally der Kamera 1 eingeschaltet und nach einer Verzögerung von einer Sekunde wieder ausgeschaltet. Anschließend beginnt der Loop von vorn. Zur Kontrolle wird die Rote LED auf dem Arduino zeitgleich angesprochen.

```
#include <BMDSDIControl.h>                    // need to include the library
BMD_SDITallyControl_I2C sdiTallyControl(0x6E);  // define the Tally object using I2C
                                              // using the default shield address
void setup()
{
  sdiTallyControl.begin();                    // initialize tally control
  sdiTallyControl.setOverride(true);          // enable tally override
  pinMode(13, OUTPUT);                        // initialize pin 13 as an output
}

void loop()
{
  digitalWrite(13, HIGH);                     // turn the LED ON
  sdiTallyControl.setCameraTally(             // turn tally ON
    1,                                        //Camera Number
    true,                                     // Program Tally
    false                                     // Preview Tally
  );

  delay(1000);                                // leave it ON for 1 second
  digitalWrite(13, LOW);                      //turn the LED OFF
  sdiTallyControl.setCameraTally(             // turn tally OFF
    1,                                        // Camera Number
    false,                                    // Program Tally
    false                                     // Preview Tally
  );
  delay(1000);                                // leave it OFF for 1 second
}
```

Codeblock 4.1: 3G-SDI Shield Beispielcode *TallyBlink*

Wird nun der Ausgang des *3G-SDI-Shield* mit dem SDI-Eingang einer Kamera mit ID 1 verbunden, so blinkt dort das rote Tally. Hiermit ist die erfolgreiche Kommunikation von Arduino zu Kamera bestätigt.

Abbildung 4.3: Erster Testlauf mit leuchtendem Tally

4.3 Test des Arduino Codes *bmControl*

Nachdem die Schnittstelle zwischen Arduino und Kamera erfolgreich getestet wurde, geht es nun an den eigentlichen Code. Dazu wird als Basis, wie bereits in Kap. 3.3 erwähnt, der Code *bmControl* von Autor *jg33* verwendet. Auch hier wird im ersten Schritt ein Test der Kommunikation der verschiedenen Bauteile durchgeführt. Dazu wird das Setup aus dem ersten Test verwendet und erweitert. Der Arduino wird, zusätzlich zum 3G-SDI-Shield, noch mit einem Ethernet-Shield ausgestattet, sodass eine Netzwerkschnittstelle verwendet werden kann. Die Verwendung dieses Shields ist bereits im Code vorgesehen. Um die Eingabe der OSC Befehle zu realisieren werden nun noch zwei weitere Geräte benötigt. Ein Router, der mit dem Shield verbunden wird und ein Gerät, mit dem OSC-Befehle gesendet werden können. Dazu wird die Software TouchOSC verwendet. Diese wird in Kap. 4.4 genauer beschrieben. Vorerst wird das Programm nur zum Funktionstest verwendet. Um die Ausgabe der Kamerakontrollsignale zu prüfen wird eine *BMD Pocket4k* Kamera und ein *BIDI* verwendet.

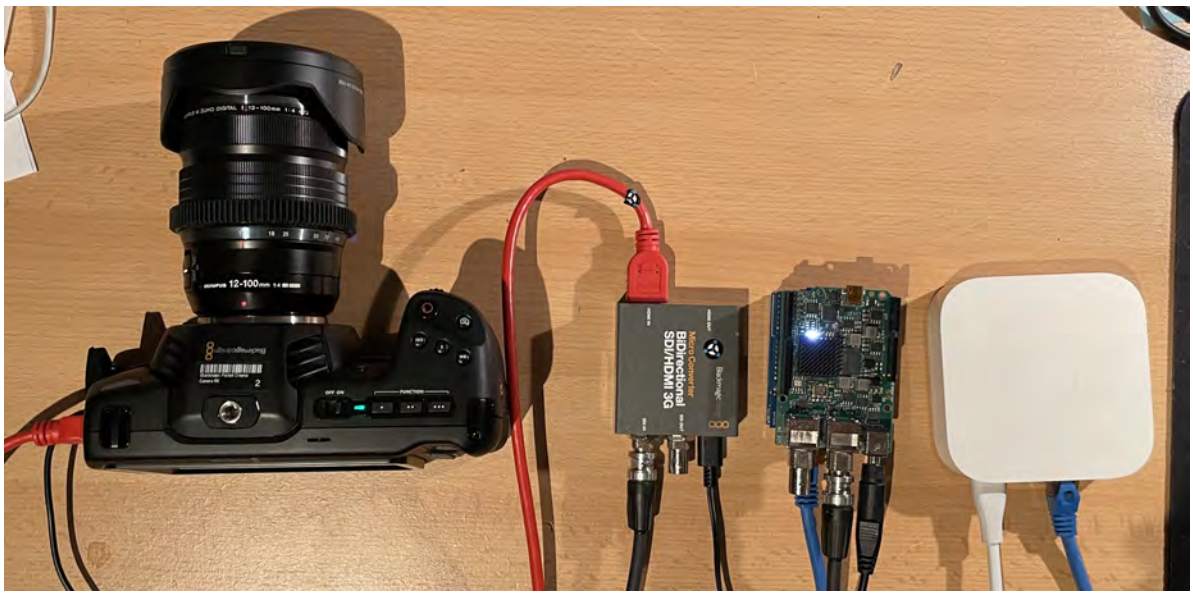


Abbildung 4.4: Testaufbau für die Softwareentwicklung

4.3.1 Grundeinstellungen und Kommunikation zum Arduino

Um die Kommunikation zwischen TouchOSC und dem Arduino herzustellen benötigt man die IP-Adresse und den zuvor festgelegten Port des Ethernet-Shields. Diese sind statisch und werden im Code festgelegt. Ausgangsseitig werden ebenfalls IP und Port festgelegt, die den rückwärtigen Kommunikationskanal definieren.

```
  // Constants //
#define DEFAULT_OSC_RECEIVE_PORT 8888
#define DEFAULT_IP_ADDRESS 10,0,0,4
#define STATUS_LED_PIN 9
#define NUM_CAMERAS 2 //number of cameras to be controlled

// Ethernet Stuff //
EthernetUDP Udp;
IPAddress targetIp(10,0,0,255);
byte mac[] = \{0x90, 0xA2, 0xDA, 0x10, 0x63, 0x69\};
byte ip[] = \{DEFAULT_IP_ADDRESS\};
char packetBuffer[256];  //buffer to hold incoming packet,
char  sendBuffer[] = "acknowledged...";        // a string to send back

// OSC Stuff //
int oscReceivePort = DEFAULT_OSC_RECEIVE_PORT;
float replyPort = 8000; //where to respond to pings and gets
byte replyIp[]  = \{ 10,0,0,255 \};
```

Codeblock 4.2: Netzwerkeinstellungen in *bmControl*

Die IP-Adressen und Ports müssen in TouchOSC übernommen werden, allerdings in umge-

kehrter Definition, da der sendende Port in TouchOSC identisch zum empfangenden Port im Arduino sein muss.

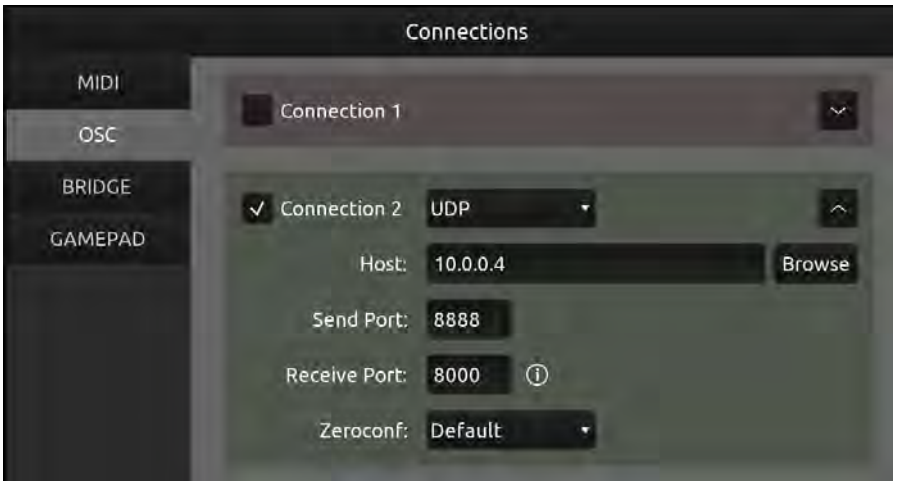


Abbildung 4.5: Netzwerkeinstellungen in TouchOSC

Hiermit ist die Grundlage der Kommunikation gegeben. Um zu überprüfen, ob die Verbindung funktioniert, muss nun die inhaltliche Kommunikation getestet werden. Dazu enthält der Code die Funktion *pingPong*, eine einfache Logik, die auf die eingehende OSC-Nachricht */ping* mit der Ausgabe der OSC-Nachricht */pong* antwortet und zusätzlich eine serielle Textausgabe des Arduino erzeugt. Somit lässt sich zuverlässig erkennen, ob eine erfolgreiche Kommunikation zwischen den zwei Komponenten besteht. Hierzu werden nun in TouchOSC zwei Buttons angelegt. Der erste Button enthält die Nachricht */ping* und der zweite BUtton die Nachricht */pong*. (Vgl. Abb. 4.6) Wenn die Kommunikation erfolgreich aufgebaut ist, sollte beim Auslösen des Buttons */ping* der Button */pong* aufleuchten und im Serial-Monitor die Nachricht *pinged* ausgegeben werden.

4.4 TouchOSC

Nach dem erfolgreichen Test der Kommunikation soll nun eine Benutzeroberfläche in TouchOSC erstellt werden. Die Entwicklung dieser Oberfläche ist dabei durch zwei Faktoren beschränkt: Die zu steuernden Funktionen der Kamera und die verschiedenen Eingabearten von TouchOSC und ihre Einstellungsmöglichkeiten. Dabei sollen alle geplanten Kameraparameter abgebildet werden und sich gut bedienen lassen.

Hierfür wurden ebenfalls einzelne Tests vorgenommen, bevor das Layout festgelegt wurde. Zuerst wurde exemplarisch die Kamerafunktion *Blende*(engl. *Aperture*) implementiert. Dazu wurde in TouchOSC ein neuer Fader mit dem OSC-Befehl: */bmc/1/aperture* angelegt. Fader

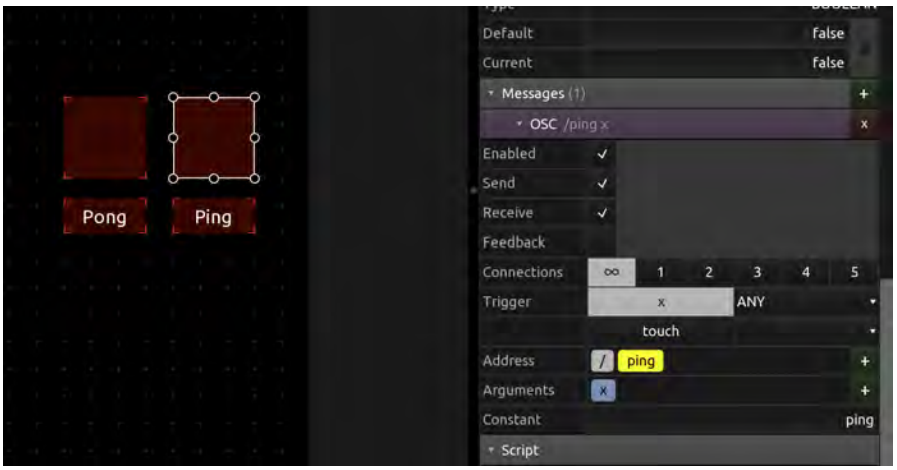


Abbildung 4.6: pingPong Test in TouchOSC

arbeiten innerhalb von TouchOSC mit Fließkommazahlen im Bereich von 0 bis 1. Da dieser Wertebereich nicht zur Anforderung des SDI-Shields passt und im Arduino-Code nicht umgewandelt wird, sollte der Wertebereich des Faders innerhalb von TouchOSC angepasst werden.

4.4.1 Anpassung der Fader

Neben der Optik und Funktionsweise eines Faders, lassen sich in TouchOSC auch die ausgegeben Wertebereiche und Datentypen des Faders beeinflussen.

Der erste Anlaufpunkt sind die Werte, die der Fader innerhalb von TouchOSC erzeugt. Hier gibt es zwei Parameter: Der erste Parameter wird als *Touch* bezeichnet und stellt einen binären Indikator bereit, der die Berührung des Faders abbildet. Dieser findet aktuell keine Verwendung. Der zweite Parameter ist *X*. Bei diesem handelt es sich um eine Fließkommazahl (*float*) im Bereich von 0 bis 1, welche durch die Positionierung des Faders verändert wird. Innerhalb von TouchOSC werden diese Variablen immer als Fließkommazahl verarbeitet. Soll dieser Wert nun verändert werden, sodass er zum erwarteten Wertebereich des SDI-Shields passt, findet sich diese Möglichkeit in der OSC Ausgabe der Variable. (Vgl. Abb. 4.7)

Die OSC-Ausgabe bietet die Möglichkeit, die Form der OSC-Nachricht anzupassen und in diese Nachricht die vom Fader erzeugte Variable einzuarbeiten. Die Variable wird dabei unter dem Begriff *Argument* in die Nachricht integriert. Im Untermenü des *Arguments* finden sich nun die gesuchten Möglichkeiten zur Skalierung des Wertes, also der Definition eines neuen Wertebereichs, in den die Fließkommazahl übertragen wird. Dabei gibt es dort zusätzlich noch die Möglichkeit, den Datentyp zu verändern. Hierbei stehen die Typen *Boolean, Integer, Float*

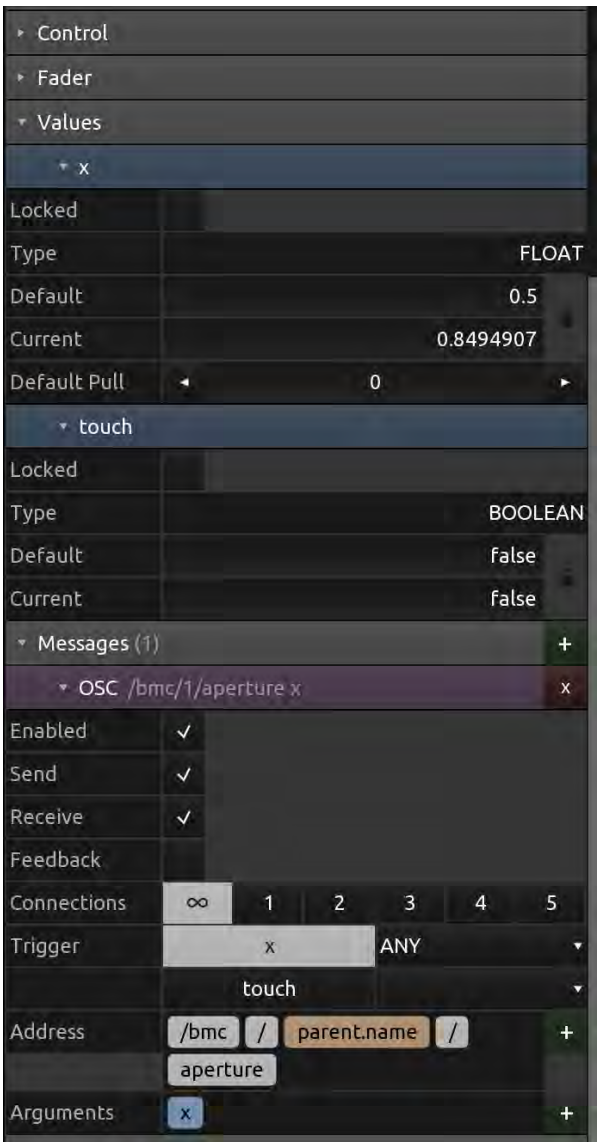


Abbildung 4.7: Fader Einstellungen in TouchOSC

und *String* zur Auswahl. In diesem Fall bleibt der Dateityp *Float* erhalten, da der Arduino-Code diesen Typen erwartet.

Neben der Variation der Werte lässt sich auch die Eingabe des Nutzers beeinflussen. Zusätzlich zur *Absolute* Reaktion des Faders, also einer direkten Beeinflussung des Fader-Wertes durch den Nutzer, lässt sich auch der *Relative* Modus verwenden. Hiermit wird eine präzisere Steuerung der Fader ermöglicht, um diese beispielweise platzsparender anordnen zu können. Die Empfindlichkeit lässt sich dabei über einen Faktor einstellen.

Abbildung 4.8: OSC Arguments in TouchOSC

4.4.2 Weitere Eingabemöglichkeiten

Neben den Fadern bietet TouchOSC noch weitere Eingabemöglichkeiten. Hier sind vor allem die *Buttons* und *Radios* zu erwähnen.

Buttons sind simple Eingabemöglichkeiten, die unterschiedliche Modi bieten: Wenn der Button-Typ als *Momentary* definiert ist, gibt der Button einen Wert von 1 aus, während er berührt wird. Die übrige Zeit hat der Button den Wert 0. Die weiteren Modi *Toggle Release* und *Toggle Press* bieten beide die Funktion, den Button durch eine Berührung einzuschalten. Dabei unterscheiden Sie sich jedoch darin, dass *Toggle Press* eingeschaltet ist, sobald er berührt wird und *Toggle Release* erst beim loslassen aktiviert wird. Die Buttons werden erst wieder deaktiviert, wenn sie erneut gedrückt werden.

Radios sind eine zusätzliche Möglichkeit, die sich zwischen Button und Fader befindet. Sie ermöglichen die gezielte Auswahl von diskreten Werten. Die Optik erinnert an einen Fader mit fest eingestellten Schritten. So lassen sich, ähnlich wie beim Fader, unterschiedliche Werte ausgeben, jedoch sind diese in ganzzahlige Stufen unterteilt. Somit bietet sich diese Komponente an, um sie für Variablen zu verwenden, bei denen die Eingabe von Fließkommazahlen nicht gewünscht ist.

Um ein funktionales und übersichtliches Nutzerinterface herzustellen, sind Beschriftungen unerlässlich. Dazu bietet TouchOSC die Möglichkeit *Label* zu verwenden. Neben der Beschriftung von anderen Eingabemöglichkeiten bieten Label zusätzlich die Funktion, Werte anzuzeigen. Diese Werte können sowohl TouchOSC intern als auch extern erzeugt werden. Um nun eine Beschriftung für z.B. einen Fader zu erstellen, wird zunächst ein neues Label erstellt und positioniert. Wenn nun ein veränderbarer Wert, beispielsweise der Ausgabewert des Faders, angezeigt werden soll, ist die Herangehensweise die Folgende: In der *Messages* Sektion des zu beschriftenden Faders lässt sich eine interne Nachricht erstellen, bei der das Ziel der Nachricht das Textfeld des Labels ist. Dazu wird die Variable X des Faders ausgewählt und

an das Textfeld des Labels gesendet. Diese Werte lassen sich, analog zur OSC-Nachricht, in einen neuen Wertebereich übertragen, sodass die Anzeige unabhängig von den OSC-Werten skaliert werden kann. Als Beispiel lässt sich die Einstellung des *Shutter Angle* nennen. Das SDI-Shield erwartet, laut Anleitung[6, S. 20] einen Wert im Bereich von 100 bis 36000. Dieser Wert beschreibt den Verschlusswinkel der Kamera, von 0,01 bis 360 Grad, multipliziert mit dem Faktor 100, sodass eine feinere Unterteilung möglich ist. Um die Übersichtlichkeit des Nutzerinterfaces zu wahren, bietet es sich an auf die Nachkommastellen zu verzichten, sodass für das Label der Wertebereich von 0 bis 360 gesetzt werden kann.

4.5 Layouts in TouchOSC

Um in TouchOSC gut nutzbare Kontrolloberflächen erstellen zu können, sollte zunächst definiert werden, welche Inhalte und Funktionen verwendet werden sollen. Für eine effiziente Positionierung der Funktionen bietet es sich zusätzlich an, die Häufigkeit der Nutzung zu bewerten. Gute Quellen der Inspiration sind dabei artverwandte Systeme anderer Hersteller, da diese bereits etabliert sind und sich daraus möglicherweise Layouts und Bedienungsabläufe ableiten lassen.

4.5.1 Layout der ersten Kontrolloberfläche

Das Ziel der ersten Kontrolloberfläche soll es sein, eine übersichtliche Möglichkeit zu bieten, bis zu vier Kameras einzurichten und Farbkorrekturen vorzunehmen. Hierbei sollen die Funktionen, die in Kap. 3.1.1 als Anforderung an das System gesetzt wurden, umgesetzt werden.

- Blendensteuerung
- Gainsteuerung
- Verschlusswinkelsteuerung
- Weissabgleich und Tint
- Kontrast und Pivot
- Farbton und Sättigung
- Farbkorrektur(R,G,B,Y Steuerung für Lift, Gamma und Gain)
- Fokussteuerung der Kamera

Der Schwerpunkt liegt dabei auf der Farbkorrektur. Dazu sollen für Lift, Gamma und Gain jeweils die R, G, B, Y Werte gut zugänglich sein.

Da sich diese Funktionen größtenteils mit der Farbkorrektur-Oberfläche der BMD *ATEM Software Control* Software decken, bietet es sich an, an dieser Stelle einen Blick auf das Layout dieser App zu werfen.

Abbildung 4.9: ATEM Software Control(Version: 9.0.1): Farbkorrektur-Oberfläche

Am oberen Rand findet sich die Auswahl aller verfügbaren Kameras. Hier lässt sich die zu bearbeitende Kamera auswählen. An der linken Seite finden sich die Grundeinstellungen der Kamera, gebündelt an einem Ort. Diese Einstellungen werden, nach der initialen Einrichtung, weniger oft verwendet. Hingegen groß und mittig angeordnet finden sich die Einstellungen zur Farbkorrektur. Diese sind gut zu erreichen und stehen im Fokus dieser Ansicht.

Für das Layout in TouchOSC wird ein ähnlicher Ansatz verwendet. Am oberen Rand wird die Auswahl der vier Kameras mittels der Funktion *Pager* ermöglicht. Dieser *Pager* ermöglicht es mehrere Unterseiten in einer TouchOSC Oberfläche zu verwenden. Der *Pager* bietet zudem den Vorteil, dass die Variable der jeweils ausgewählten *Pager*-Seite in die OSC-Nachricht integriert werden kann und dadurch die Auswahl der Kamera abgebildet werden kann.

Auf der einzelnen Kameraseite steht die Farbkorrektur im Vordergrund und nimmt die gesamte Breite der Seite ein. Zur Steuerung werden hier Fader verwendet, die blockweise angeordnet sind. Dabei sind diese Blöcke aus *R G B Y* Werten zu *Lift*, *Gamma* und *Gain*

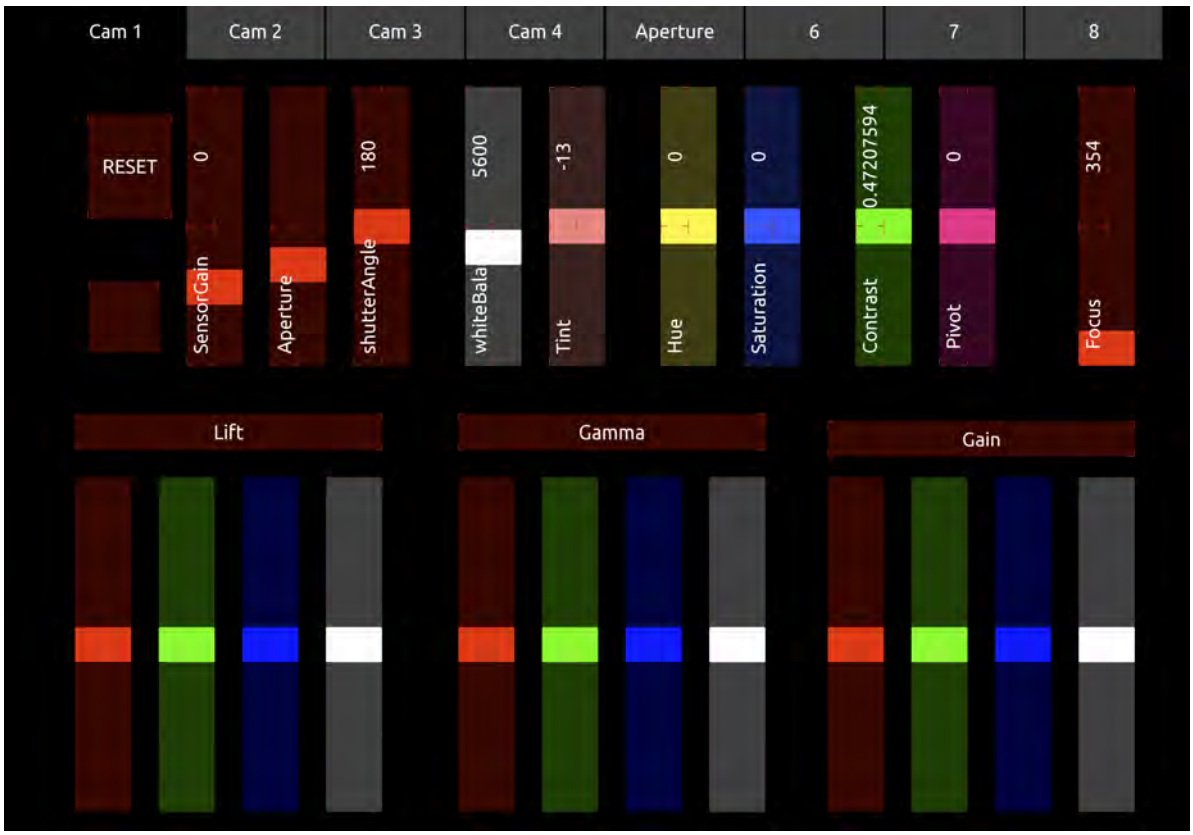


Abbildung 4.10: Layout der ersten Kontrolloberfläche zu Kamerasteuerung in TouchOSC

zugeordnet. Zur besseren Unterscheidbarkeit sind die Fader, passend zur Funktion, eingefärbt und blockweise beschriftet. Um an dieser Stelle eine feine Einstellung zu ermöglichen, sind die Fader im *Relative Mode* mit dem Faktor 5 angelegt und nehmen etwa die halbe Höhe des Bildschirms ein.

Die Grundfunktionen Gain, Blende, Verschlusswinkel, Weißabgleich und *Tint* sind in der oberen linken Ecke angeordnet und ebenfalls als Fader, im *Relative Mode* mit dem Faktor 10 ausgeführt. Danach folgen die Einstellungen für *Hue* und *Saturation*, sowie *Contrast* und *Pivot*. Am rechten Rand ist der Fader für den *Focus* angeordnet.

4.5.2 Layout der zweiten Kontrolloberfläche

Die Idee der zweiten Kontrolloberfläche verfolgt einen anderen Ansatz als die erste Oberfläche. Hier sollen vor allem die Einstellungen Platz finden, die während einer Produktion am meisten benötigt werden. Für gewöhnlich findet die Einrichtung der Kameras im Vorfeld statt. So sind Farben und Belichtungszeiten vor Beginn einer Sendung eingerichtet und bedürfen häufig keiner weiteren Aufmerksamkeit. Durch variierende Bildausschnitte und unterschiedliche

Lichtverhältnisse am Set, ergibt sich allerdings trotzdem die Notwendigkeit, die Helligkeiten der Kameras anzupassen. Dazu wird meist die Blendeneinstellung verwendet. Sollte diese allerdings an ihre Grenzen stoßen, da die Blende beispielsweise bereits vollständig geöffnet ist, wird zusätzlich der *Gain* verwendet.

Abbildung 4.11: Layout der zweiten zur Steuerung der Kameraparameter entwickelten Kontrolloberfläche in TouchOSC

Im Gegensatz zur ersten Kontrolloberfläche, bei der eine Vielzahl von Funktionen einer ausgewählten Kamera verfügbar sind, sollen hier nun Gain und Blende aller Kameras zugleich bedienbar sein, um ein schnelles und unkompliziertes Eingreifen in die Belichtung zu ermöglichen. So ergibt sich beispielsweise die Chance, dass der Operator des Bildmischers die Helligkeiten der Kameras anpassen kann, ohne auf einen Bildtechniker angewiesen so sein.

Dieses ist insbesondere für kleine Produktionen sehr praktisch, da häufig nicht genug Personal zur Verfügung steht um alle klassischen Broadcast-Positionen zu besetzen. Nach eigener Erfahrung fällt hier meist zuerst der Arbeitsplatz des Bildtechnikers weg, da diese Aufgaben größtenteils von der Sendung erledigt werden können.

Die Funktionen zu Erstellung dieses Nutzerinterfaces sind bereits auf der ersten Seite vorhanden und bedürfen nur kleineren Anpassungen. Der erste Schritt besteht darin in *Pager* eine leere Seite anzuwählen. Der bereits mit den passenden Einstellungen konfigurierte Fader lässt sich nun von der ersten Kameraseite kopieren und für den neuen Einsatzzweck modifizieren.

Um die Kontinuität der Werte über alle Seiten zu gewährleisten, ist es erforderlich, die korrespondierenden Fader miteinander zu verlinken. Dazu bietet TouchSOC die Funktion,

lokale Befehle zu senden. Lokale Befehle lassen sich im selben Menüpunkt wie OSC-Befehle anlegen. Um diese Verlinkung nutzen zu können, ist es erforderlich, die Einstellungen bei allen zusammengehörigen Fadern durchzuführen. Es wird eine eine neue, lokale Nachricht erstellt und dieser wird der andere Fader als Ziel zugewiesen.

Sind alle Fader entsprechend konfiguriert, beeinflussen sie sich gegenseitig. Um nun das doppelte senden der gleichen OSC-Nachrichten zu verhindern, werden alle OSC-Befehle auf der neu erstellten Kontrolloberfläche entfernt.

Um diese Ansicht optisch intuitiver zu gestalten werden die Fader nun noch eingefärbt und mit den passenden Kameranummern beschriftet.

4.5.3 Reset der Kameraparameter

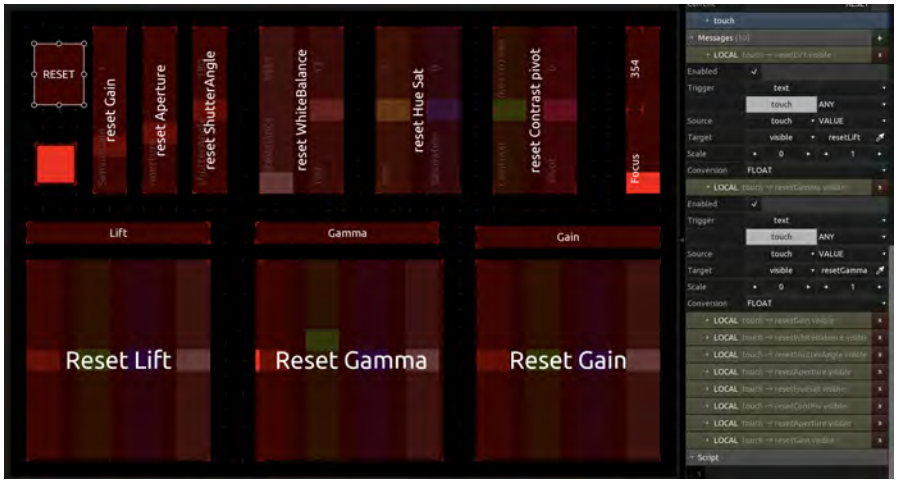


Abbildung 4.12: Ansicht der Resetfunktion in TouchOSC

Um die Kameraparameter wieder auf ihre Ausgangswerte zurücksetzen zu können, wurde die Funktion *Reset* implementiert. Dieses geschieht nach Funktionsgruppen, da in der Praxis vermutlich eher einzelne Parameter zurückgesetzt werden und nicht die gesamte Kamera. Dabei war es wichtig zu verhindern, dass diese Resets versehentlich betätigt werden, da dieses zu sichtbaren Sprüngen im Kamerabild führen kann.

Um diese Funktion umzusetzen, wurden zunächst Reset-Buttons für die jeweiligen Gruppen von Funktionen angelegt und oberhalb der Fader platziert. Anschließend wurde die Sichtbarkeit der Buttons deaktiviert, um diese anschließend mit einem Aktivierungsknopf temporär wieder anzeigen zu können. Dazu wurde ein lokaler TouchOSC-Befehl für jeden Reset-Button angelegt, der auf die Berührung des Labels reagiert und damit die Sichtbarkeit ermöglicht. Der Umweg über einen separaten Auslöser verhindert das versehentliche Zurücksetzen der Funktionen. (S. Abb. 4.12)

4.6 Beispielhafter Signalverlauf eines Steuerbefehls

Um ein besseres Verständnis des Gesamtsystems zu erreichen, wird der Signalfluss eines exemplarischen Steuerbefehls, über alle Komponenten, an Beispiel der Gainsteuerung von Kamera 1, dargestellt. (S. Abb. 4.13)

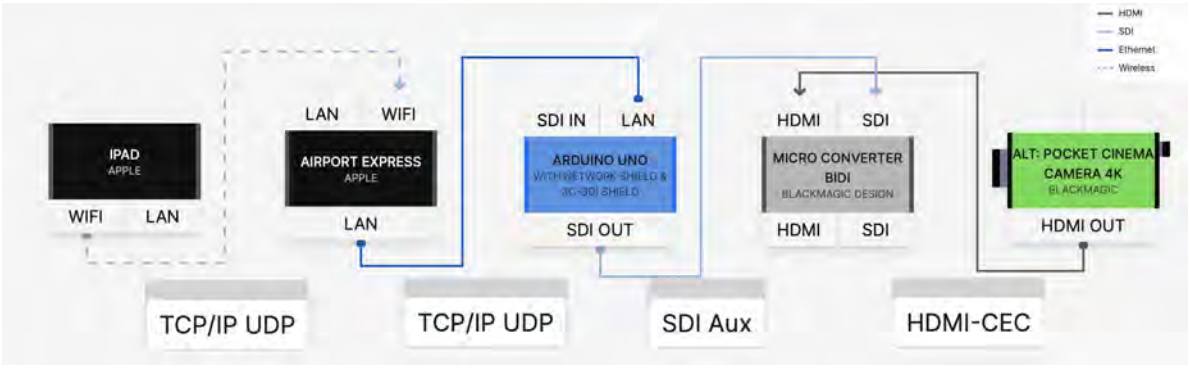


Abbildung 4.13: Signalfluss des Testsetups

Der erste Schritt ist dabei die Eingabe eines Wertes in der TouchOSC Oberfläche auf dem iPad. Diese Variable wird nun auf zwei Wegen weiterverarbeitet. Der erste Weg ist die interne Verarbeitung, in Form einer lokalen Nachricht an den zugehörigen Fader auf der zweiten Kontrolloberfläche.

Der zweite Weg ist die Umwandlung dieser Variable in eine OSC-Nachricht. Hier wird die Variable mit der angelegten Nachricht an die IP-Adresse des Arduinos gesendet. Dies geschieht per UDP und wird dann vom Router, der die Nachricht des iPads zuerst empfängt, an den Arduino weitergeleitet. Im Arduino wird das Paket empfangen und auf Fehler überprüft. Wenn keine Fehler vorliegen, wird das Paket an die zugehörige Funktion weitergeleitet.

```
    // --- receive a bundle  --- //
2   if ((size = Udp.parsePacket()) > 0) {
      Serial.println("Packet␣Received...");
4     while (size--) {
        bndl.fill(Udp.read());
6       Serial.print(".");
      }
8     if (!bndl.hasError()) {
        Serial.println("no␣err");
10      if (bndl.size() > 0) {
          static int32_t sequencenumber = 0;
12        bndl.route("/ping", pingPong);
          bndl.route("/bmc", parseBmcMsg);
14        (...)
```

Codeblock 4.3: Fehlerkontrolle und Routing in *bmControl*

Beginnt der OSC-String nun mit */bmc* so wird er an die Funktion *parseBmcMsg* weitergeleitet.

```
// --- OSC Messages --- //
void parseBmcMsg(OSCMessage &_msg, int offset) {
  char address[256];
  _msg.getAddress(address,0);
  int cam = getAddressSegment(address,1).toInt(); //get address segment 1
  String cmd = getAddressSegment(address,2); //get address segment 2
  int rgby = getAddressSegment(address,3).toInt(); //get address segment 3

  if (cmd == "aperture") {
    setAperture(cam, _msg.getFloat(0));
  } else if (cmd == "focus") {
    setFocus(cam, _msg.getFloat(0));
  } else if (cmd == "exposure") {
    setExposure(cam, _msg.getInt(0));
  } else if (cmd == "sensorGain") {
    setSensorGain(cam, _msg.getInt(0));
  } else if (cmd == "whiteBalance") {
    setWhiteBalance(cam, _msg.getInt(0));
    (...)
  }else {
    char buff[16];
    _msg.getAddress(buff,0);
  }
}
```

Codeblock 4.4: Funktion *parseBmcMsg* aus *bmControl*

Innerhalb dieser Funktion wird die OSC-Nachricht zunächst in ihre einzelnen Segmente aufgeteilt und in neue Variablen geschrieben. Das erste Segment */bmc* wird dabei als Segment 0 bezeichnet. Das nächste Segment beinhaltet die Nummer der anzusprechenden Kamera und wird in der Variable *cam* gespeichert. Das dritte Segment beinhaltet die anzusprechende Funktion und wird in der Variable *cmd* abgelegt. Um die Fehlersuche zu erleichtern, lassen sich die Werte noch mittels der Funktion *Serial.print()* auf der seriellen Konsole ausgeben.

Im nächsten Schritt wird der Inhalt der Vairable *cmd* ausgewertet und anschließend der komplette OSC-String an die darauffolgende Funktion übertragen. In diesem Beispiel ist der Inhalt *sensorGain*.

Beim Aufruf der Funktion *setSensorGain* wird die Kameranummer sowie der OSC-String an die Funktion übergeben.

```
void setSensorGain(int _camera, int _value) {
  // FORMAT: Int8  -128 to 127    Gain in decibel (dB)
  cameras[_camera].sensorGain = _value;
  sdiCam.writeCommandInt8(_camera, 1, 13, 0, cameras[_camera].sensorGain);  //
}
```

Codeblock 4.5: Funktion *setSensorGain* aus *bmControl*

Diese Funktion setzt nun die empfangene Kameranummer und den OSC-String in den *sdiCam.wirteCommandInt8* Befehl ein und überträgt diese damit an das *3G-SDI-Shield*.

An dieser Stelle verlässt der Befehl den Arduino und wird per SDI an den BMD BIDI übertragen. Hier findet nun die Umwandlung vom SDI-Signal in ein HDMI-CEC Signal statt. Dieses CEC-Signal wird an die Kamera weitergeleitet. Hier löst der Befehl nun die Steuerung des Gains aus.

4.7 Implementierung zusätzlicher Funktionen am Beispiel *Contrast*

In dem bisher erstellen Kontrollsystem sind vorrangig Funktionen implementiert worden, die bereits in *bmControl* vorhanden waren. Sollen nun weitere Funktionen integriert werden, so ist das recht einfach umzusetzen, da die Basis dazu bereits geschaffen wurde.

Die Dokumentation des BMD CameraControlProtocols [6, S. 19 ff.] enthält eine lange Liste an Funktionen, die in das Protkoll integriert sind, allerdings nicht im *ATEM Software Control* verfügbar sind. Sollen diese Funktionen ebenfalls umgesetzt werden, so lässt sich der vorhandene Code leicht dazu anpassen. Das wird beispielhaft an der Funktion *Contrast* gezeigt. Diese Funktion ist zwar auch im *ATEM Software Control* verfügbar, alleridngs nicht im Code *bmControl* vorgesehen.

Da vom Arduino-Code, laut Dokumentation[6, S. 19 ff.], Fließkommazahlen erwarte werden, bietet es sich an, einen Fader zu verwenden. Der Name der OSC-Nachricht wird festgelegt als *contrast*.

Um diese Nachricht jetzt im Arduino verarbeiten zu können, müssen folgende Dinge umgesetzt werden: Zuerst wird eine neue Variable, mit dem Namen *contrast* angelegt. Der Typ der Variable ist *float*. Dise Variable wird in einen *Struct*, einem Array, das unterschiedliche Datentypen kombiniert, abgelegt.

```
1  // Camera State //
   struct Camera{
3    int id;
     float aperture;
5    float focus;
     int exposure;
7    int sensorGain;
     int whiteBalance;
9    int tint;
     float lift[4];
11   float gamma[4];
     float gain[4];
13   float hue;
     float sat;
```

```
15      float contrast;
        float pivot;
17      int shutterAngle;
    };
```

Codeblock 4.6: Struct der Kameravariablen aus *bmControl*

In nächsten Schritt wird eine neue Funktion angelegt, die es ermöglicht, die eingehenden Werte zusammen mit der passenden Kameranummer, in die Ausgangsfunktion für das *3G-SDI-Shield* zu übertragen. Hier lässt sich die Funktion *setHue* kopieren und anpassen, da die Anforderungen aus dem *3G-SDI-Shield* Datenblatt übereinstimmen.

```
  void setHue(int _camera, float _value){
2   cameras[_camera].hue = _value;
    float colorCombined[2] = {cameras[_camera].hue, cameras[_camera].sat};
4   sdiCam.writeCommandFixed16(_camera,8,6,0,colorCombined);
}
6 void setSat(int _camera, float _value){
    cameras[_camera].sat = _value;
8   float colorCombined[2] = {cameras[_camera].hue, cameras[_camera].sat};
    sdiCam.writeCommandFixed16(_camera,8,6,0,colorCombined);
10 }
  void setContrast(int _camera, float _value){
12  cameras[_camera].contrast = _value;
    float contrastCombined[2] = {cameras[_camera].pivot, cameras[_camera].contrast};
14  sdiCam.writeCommandFixed16(_camera,8,4,0,contrastCombined);
}
16 void setPivot(int _camera, float _value){
    cameras[_camera].pivot = _value;
18  float contrastCombined[2] = {cameras[_camera].pivot, cameras[_camera].contrast};
    sdiCam.writeCommandFixed16(_camera,8,4,0,contrastCombined);
20 }
```

Codeblock 4.7: *setContrast* Funktion in *bmControl*

8.4	Contrast Adjust	fixed16	[0] pivot	0	1	default 0.5
			[1] adj	0	2	default 1.0
8.5	Luma mix	fixed16	–	0	1	default 1.0
8.6	Color Adjust	fixed16	[0] hue	-1	1	default 0.0
			[1] sat	0	2	default 1.0

Abbildung 4.14: Ausschnitt aus der Funktionsübersicht des CameraControlProtokolls [6, S. 23]

Eine Kopie der Funktion wird nun angepasst und *hue* wird dabei durch *contrast* ersetzt. Da diese Variablen paarweise übertragen werden, gibt es noch weiterre Variablen, die verändert werden sollten. Die Funktion *float contrastCombined[2]* fasst die zwei Variablen für den *Contrast* und den *Pivot Point* zusammen, sodass diese gemeinsam in die Übertragungsfunktion des *3G-SDI-Shield* geschrieben werden können. Zuletzt wird noch die Ausgangsnachricht

angepasst, da die *Contrast* Funktion an eine andere ID des *3G-SDI-Shield* gesendet werden muss.

Das Routing in der Funktion */parseBmcMsg* muss ebenfalls ergänzt werden.

```
// --- OSC Messages --- //
void parseBmcMsg(OSCMessage &_msg, int offset) {
(...)
  } else if(cmd== "hue"){
    setHue(cam, _msg.getFloat(0));
  } else if (cmd=="sat"){
    setSat(cam, _msg.getFloat(0));
  } else if (cmd=="contrast"){
    setContrast(cam, _msg.getFloat(0));
      } else if (cmd=="pivot"){
    setPivot(cam, _msg.getFloat(0));
(...)
      }
```

Codeblock 4.8: Angepasste Funktion *parseBmcMsg* aus *bmControl*

Nach diesen Anpassungen im Code lässt sich nun der Kontrast der Kamera auf der TouchOSC Oberfläche verändern.

4.8 Aufbau des fertigen Systems

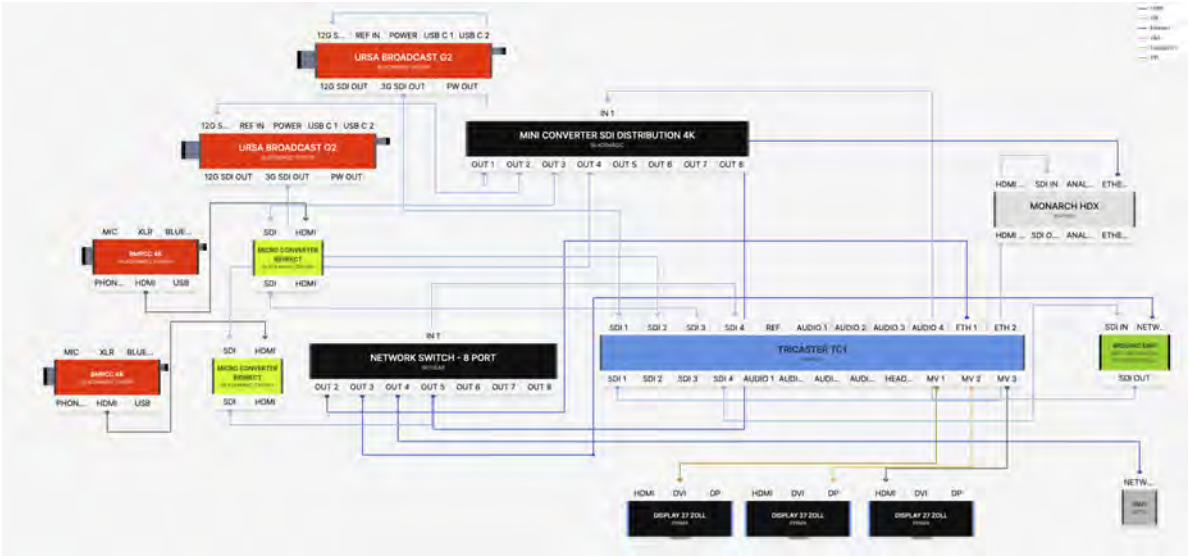


Abbildung 4.15: Signalplan des neuen Kontrollsystems

Nachdem sich das Testsetup als funktional herausgestellt hat, soll nun noch das engültige Setup zusammengestellt werden. Dazu wird das vorher genutzte System mit den zwei Bildmischern modifiziert und um das Testsetup ergänzt.

Der bisher zur Bildsteuerung nötige ATEM-Bildmischer und die Kreuzschiene aus dem alten Setup werden entfernt und durch das neue System mit dem Arduino, inklusive der Shields, ersetzt. Vergleicht man nun den Signalplan des unsprünglichen Setups(Abb. 2.2) mit dem aktuellen Signalplan(Abb. 4.15), so ist es gelungen, die Komplexität der Verschaltung deutlich zu reduzieren.

Das Programmbild des TriCaster wird zwar noch durch den Kameracontroller geschliffen, allerdings werden in diesem Setup die Befehle für die Tallys des TriCasters per SDI weitergeleitet und müssen nicht erst aufwändig auf einen zweiten Bildmischer übersetzt werden. Der Einsatz von *Companion* ist somit nicht länger erforderlich und der eingesetzte Laptop, der zur Steuerung der weggefallenden Komponenten diente, ist ebenfalls obsolet, da zur Steuerung nun ein iPad verwendet wird.

Somit ergibt sich eine deutliche Reduktion des Materials, sowie der Komplexität des Aufbaus. Damit lässt sich Zeit und Material sparen und das Potential für Fehler, sowohl technischer- als auch menschlicher Art wird reduziert.

4.9 Konstuktion und Bau eines Gehäuses

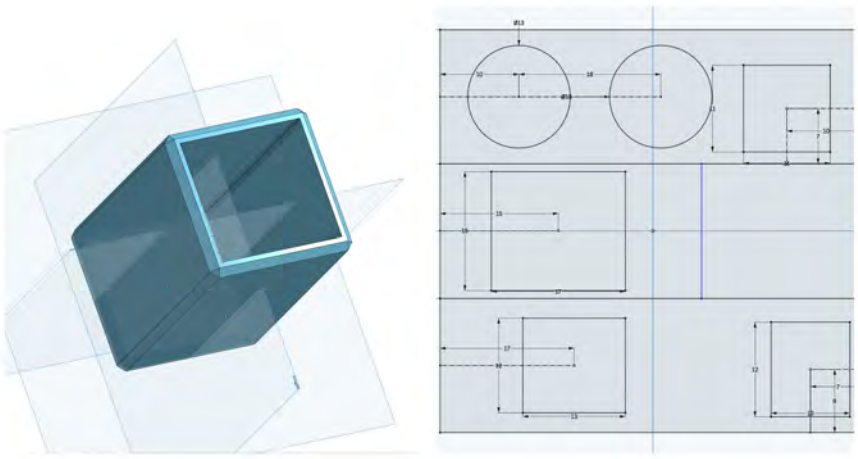


Abbildung 4.16: CAD Zeichnug des Gehäuses links 3D- Ansicht des gesamten Gehäuses. Rechts Aufsicht der Frontplatte.

Um nun die Hardware vor Beschädigungen zu schützen wird ein einfaches Gehäuse konstruiert, das anschließend mit einem 3D-Drucker produziert werden soll.

Zur Konstruktion wird die Software *Onshape*[28] verwendet, da diese für Privatanweder kostenfrei zur Verfügung steht und im Browser verwendet werden kann.

Der erste Schritt der Konstuktion ist die Vermessung des Kameracontrollers. Dazu werden die groben Außenmaße des Arduinos, inklusive der Shields, ermittelt und ein leerer Quader erstellt. Dabei wird die Seite mit den Anschlüsse offengelassen. Auf der Rückseite des Bauteils wird ein Lochraster zur Belüftung der Komponenten eingelassen.

Abbildung 4.17: Passungen der Fontplatte: Erste Probepassung(l) und finale Frontplatte(r)

Da die Anschlüsse gut erreichbar sein müssen, wird der Controller auf der Anschlussseite vermessen, sodass die genauen Positionen der Anschlüsse ermittelt und in die CAD-Zeichnung übernommen werden können. Diese erste Iteration des Bauteils kann nun extrudiert und gedruckt werden, um die Passform zu überprüfen. Nach etwa 15 Minuten Druckzeit lässt sich nun die Passform überprüfen und die Zeichnung anpassen, falls nicht alle Anschlüsse in die vorgesehen Ausschnitte passen. Sind diese Korrekturen vorgenommen, so lässt sich das finale Gehäuseteil drucken.

Onhape biete die Möglichkeit, das Projekt online einzusehen und zu kopieren, sodass individuelle Anpassungen vorgenommen werden können. [27]

Im letzten Schritt wird nun der Arduino in das Gehäuse eingebaut und mit der Anschlusseite verschlossen. Dabei sind die Toleranzen so gewählt, dass die Bauteile durch eine Presspassung in Position gehalten werden.

Abbildung 4.18: Kameracontroller im fertigen Gehäuse

5 Auswertung

In der Auswertung wird zusammengetragen, ob die gestellten Anforderungen erreicht werden konnten. Zusätzlich wird ein Vergleich gezogen, ob sich eine Verbesserung zum vorherigen System ergeben hat.

5.1 Anforderungen erfüllt?

Die Anforderungen an die Kamerasteuerung aus Kap. 3.1 konnten vollständig umgesetzt werden. So ermöglicht das System die Steuerung aller gesetzten Kameraparameter. Das Nutzerinterface ist klar strukturiert, besteht aus etablierten Werkzeugen wie Fadern und Buttons und legt den Fokus auf die Farbkorrektur der einzelnen Kameras. Ergänzt wird dies durch eine weitere Ansicht, die es ermöglicht, den *Gain* und die *Blende* aller Kameras auf einer Oberfläche zu kontrollieren. Alle Funktionen sind beschriftet und lassen sich bei Bedarf zurücksetzen.

Durch die Verwendung eines Arduino Uno, dem BMD 3G-SDI-Shield und einem Ethernet-Shield ergibt sich ein kompaktes System, das alle benötigten Funktionen zur Verfügung stellt. Die Technik ist gut verfügbar, verwendet Standard-Anschlüsse zur Kommunikation und Stromversorgung und bewegt sich in einem Preisrahmen deutlich unter 200 Euro.

Die Software TouchOSC bietet die Grundlage für weitere Bedienoberflächen und ist leicht zu erlernen. Die Plattformunabhängigkeit ist dabei ein großer Vorteil, da die eingesetzte Hardware dadurch leicht austauschbar wird und man nicht an ein spezifisches Gerät gebunden ist.

Alle Komponenten bieten die Möglichkeit der zukünftigen Erweiterung des Systems und der Code *bmControl* sieht bereits einige Erweiterungen vor.

5.2 Vergleich zum vorherigen System

Vergleicht man den finalen Signalplan des entwickelten Systems(Abb. 4.15) mit dem Signalplan der vorherigen Lösung(Abb. 2.2), ist die reduzierte Anzahl an Geräten und die verringerte Komplexität der Verkabelung klar zu erkennen. Hierdurch reduziert sich die Aufbauzeit und Fehleranfälligkeit deutlich.

5.2.1 Formfaktor

Das vorherige System wiegt etwa 21,5kg und hat die Außenmaße (BxTxH): 54,5x52x28cm. Vergleicht man diese Maße mit dem neue Controller, so ist ein deutlicher Unterschied festzustellen. Der Controller misst etwa 10x6,5x6,5cm zuzüglich Steckernetzteil. Das Gesamtgewicht liegt bei etwa 400 Gramm. Für den Einsatz in der realen Produktionsumgebung müssen beide Geräte selbstverständlich in entsprechenden Gehäusen verbaut werden. Um einen fairen Vergleich zu ziehen sollten diese Maße und Gewichte, inklusive der Gehäuse, verwendet werden. Es lässt sich joedoch festhalten, dass das Gerät deutlich kleiner und leichter als sein Vorgänger ist.

Abbildung 5.1: Das bisherige Kontrollsystem im Case

5.2.2 Reaktionszeit der Tallys

Die Reaktionszeit des neuen Systems wird auf theoretische Weise bestimmt. Die hohen Latenzen des vorherigen System entstehen vermutlich insbesondere durch die Verknüpfung der beiden Bildmischer, die durch *Companion* verbunden werden.

Beim neuen System gibt der TriCaster den Befehl per SDI an das 3G-SDI Shield weiter. Hier wird das Signal nicht zusätzlich verarbeitet, sondern direkt an die Kameras weitergeleitet.

Laut Handbuch des 3G-SDI Shields beträgt die interne Verarbeitungszeit der vom Arduino gesendeten Werte weniger als ein Frame.[6, S. 17] Die Verarbeitungszeit des per SDI eingehenden Signals wird nicht spezifiziert.

Nimmt man nun an, dass das SDI-Signal ähnlich schnell verarbeitet wird wie das Signal des Arduinos, so ergibt sich daraus eine Laufzeit von einem Frame. Wenn noch ein Unsicherheitsfaktor von 100% angenommen wird, ergibt sich somit eine Latenz von zwei Frames, die bei 50fps eine Laufzeit von 40ms ergeben. Vergleich man diese Latenz mit der Durchsatzlatenz des TriCaster, die laut technischen Daten bei etwa 4 Frames liegt, so sollte diese Verzögerung im vertretbaren Rahmen liegen, da sich eine theoretische Systemlatenz von etwa 6 Frames, also 120ms, ergibt.[23]

Die theoretisch ermittelten Werte sehen vielversprechend aus. Allerdings ist ein Praxistest vor dem ersten Einsatz zwingend erforderlich, um sicherzustellen, dass das gesamte System wie erwartet funktioniert und insbesondere die Laufzeiten eingehalten werden. Dieser Test konnte an dieser Stelle leider nicht mehr stattfinden, da kein Testaufbau zur Verfügung stand.

5.3 Ausblick

Da das Projekt eine flexible Basis zu Weiterentwicklung bieten soll, werden hier einige Ideen für die Zukunft festgehalten. Dabei sind sowohl grundlegend neue Funktionen, als auch Funktionen, deren zukünftige Verwendung bereits vorbereitet ist, enthalten.

5.3.1 Mehrbenutzer Funktionalität

Um eine sinnvolle Mehrbenutzerfunktionalität herzustellen, ist es erforderlich, dass der Arduino die Parameterverwaltung übernimmt und eingestellte Werte an die TouchOSC-Instanzen senden kann. Aktuell verwaltet die verwendete TouchOSC Instanz die Kameraparameter. Sollte diese nun einmal ausfallen, so beginnt die Benutzeroberfläche mit den voreingestellten Parametern. Die Kamerabilder werden zunächst nicht verändert, da eine Übertragung der gesendeten Parameter erst durch Nutzerinteraktion geschieht. Sobald jedoch ein neuer Parameter gesendet wird, der einen Sprung zu vorher eingestellten Parametern aufweist, so ist dieser Sprung im Kamerabild deutlich zu erkennen. Eine weiche Interpolation von größeren Parametersprüngen findet nicht statt. Das gleiche Verhalten ist auch zu erwarten, wenn mehrere Geräte ohne eine zentrale Steuerung und Verteilung der aktuellen Parameter Werte senden. Im Arduino-Code *bmControl* ist dieser Rückkanal bereits vorgesehen, wurde jedoch vorerst aus Zeitgründen nicht implementiert oder getestet.

5.3.2 Speichern von Kameraeinstellungen

Neben der zentralen Verwaltung der Kameraparameter ist es denkbar, auch eine Langzeitspeicherung der Werte im Arduino umzusetzen. Dazu ließen sich die relevanten Werte im EEPROM, einem auf dem Arduino verbauten nichtflüchtigen Speicher, ablegen. Dieser kann mittels spezieller Befehle beschrieben und auch nach längerer Spannungslosigkeit weiterhin ausgelesen werden. Allerdings stellt der verwendete Arduino UNO dazu nur 1024Byte zur Verfügung. Alternativ ließe sich ein externer Speicher verwenden. Das Ethernet-Shield bietet dazu die Möglichkeit MicroSD-Karten zu verwenden. Bei der Umsetzung der Speicherlösung böte es sich an, mehrere Speicherplätze vorzusehen, sodass beispielweise Presets für einzelne Kameras oder Locations abgelegt werden könnten.

5.3.3 Übertragen von Kameraeinstellungen

Eine zusätzlich nützliche Funktion wäre die Übertragung von Kameraeinstellungen auf weitere Kameras. Ist eine gute Grundeinstellung für einen Kameratypen gefunden, ließe sich diese dann auf die weiteren Kameras übertragen. Diese Funktion würde eine deutliche Zeitersparnis bei der Einrichtung vieler Kameras ähnlichen Typs bringen.

5.3.4 Tallyanzeige in TouchOSC

Um ein versehentliches Eingreifen in eine Kamera, die aktuell im Programmbild zu sehen ist, zu verhindern, könnte das eingehende Tally-Signal vom Arduino ausgelesen und in OSC-Nachrichten übersetzt werden. Damit ließe sich dann beispielsweise eine Kameraseite sperren, sollte diese Kamera aktuell im Programmbild gezeigt werden.

6 Fazit

Abschließend lässt sich festhalten, dass die Anbindung von Blackmagic Design Kamerasystemen an generische Bildmischer erfolgreich umgesetzt werden kann, ohne dabei wichtige Kamerakontrollfunktionen und Tallys zu verlieren.

Der entwickelte Prototyp zur Kamerasteuerung ermöglicht die volle Kontrolle über alle relevanten Kamerafunktionen, sowie die Weiterleitung der Tally-Befehle des TriCaster. Durch die Verwendung von TouchOSC als plattformunabhängiges und benutzerfreundliches Interface bietet das System eine hohe Anpassbarkeit und Erweiterbarkeit. Die Verwendung von Standardkomponenten wie dem Arduino Uno, dem BMD 3G-SDI-Shield und einem Ethernet-Shield ermöglicht es, ein kompaktes System herzustellen, das zusätzlich einfach zu beschaffen und kostengünstig nachzubauen ist. Die offenen Plattformen der beteiligten Systeme eröffnen zudem vielfältige Möglichkeiten für zukünftige Weiterentwicklungen und Anpassungen an spezifische Anforderungen. Das OSC-Protokoll, das für die Kommunikation zwischen den Komponenten verwendet wird, ermöglicht darüber hinaus eine flexible Steuerung des Controllers durch andere Geräte und eröffnet somit Möglichkeiten für die Einbindung in bestehende AV-Systeme.

Da es sich bei diesem Kontrollsystem um einen mit limitierten Mitteln entwickelten Prototypen handelt und dieser nicht umfassend getestet werden konnte, sollte die professionelle Einsatzbereitschaft des Projektes kritisch hinterfragt werden. Der Controller bietet allerdings eine gute Basis zur weiteren Entwicklung und für umfangreiche Praxistests. Würden weitere Funktionen integriert, so ließe sich ein deutlicher Mehrwert zur etablierter Technik schaffen.

Abschließend lässt sich sagen, dass die Entwicklung dieses Prototypen eine spannende und bereichernde Erfahrung war. Es ist jedoch bedauerlich, dass es die Notwendigkeit eines solchen Gerätes gibt. Wünschenswert wäre die Standardisierung von Kamerakontrollprotokollen, sodass alle Bildmischer diese implementieren können und damit flexiblere Kombinationen von Kameratechnik und Bildmischern ermöglicht würden.

Literatur

[1] Arduino. *Store - Arduino Uno Rev3*. 2024. URL: https://store.arduino.cc/collections/boards-modules/products/arduino-uno-rev3.

[2] Bitfocus AS. *Companion Website*. 2024. URL: https://bitfocus.io/companion (besucht am 18. 03. 2024).

[3] BlackmagicDesign. *3G-SDI Shield Entwickerwebsite*. 2024. URL: https://www.blackmagicdesign.com/de/developer/product/arduino (besucht am 16. 03. 2024).

[4] BlackmagicDesign. *ATEM TV Studio Software*. 2024. URL: https://www.blackmagicdesign.com/de/products/atemtelevisionstudio/software (besucht am 18. 03. 2024).

[5] BlackmagicDesign. *Blackmagic Design stellt den neuen ATEM Television Studio HD vor*. 2017. URL: https://www.blackmagicdesign.com/de/media/release/20170206-01 (besucht am 18. 03. 2024).

[6] BlackmagicDesign. *Installation and Operation Manual Blackmagic 3G-SDI Shield for Arduino*. 2020. URL: https://documents.blackmagicdesign.com/UserManuals/ShieldForArduinoManual.pdf?_v=1581580811000 (besucht am 16. 03. 2024).

[7] BlackmagicDesign. *Installation and Operation Manual Blackmagic Pocket Cinema Camera*. 2023. URL: https://documents.blackmagicdesign.com/UserManuals/BlackmagicPocketCinemaCamera4KManual.pdf?_v=1681714811000 (besucht am 16. 03. 2024).

[8] BlackmagicDesign. *Micro Converter - Die weltweit kleinstenUSB-Broadcast-Konverter*. 2024. URL: https://www.blackmagicdesign.com/de/products/microconverters (besucht am 17. 03. 2024).

[9] BlackmagicDesign. *Pocket Cinema Camera*. 2024. URL: https://www.blackmagicdesign.com/products/blackmagicpocketcinemacamera (besucht am 26. 02. 2024).

[10] BlackmagicDesign. *URSA Broadcast G2*. 2024. URL: https://www.blackmagicdesign.com/products/blackmagicursabroadcast (besucht am 26. 02. 2024).

[11] Botand. *I2C-Bus – was ist das und wofür wird er verwendet*. 2024. URL: https://botland.de/blog/i2c-bus-was-ist-das-und-wofuer-wird-er-verwendet/ (besucht am 18. 03. 2024).

[12] Martijn Braam. *Reverse engineering the BMD camera HDMI control, part 2*. 2023. URL: https://blog.brixit.nl/reverse-engineering-the-bmd-camera-hdmi-control-part-2/ (besucht am 17. 03. 2024).

[13] AZ Delivery. *Ethernet Shield W5100 mit MicroSD-Karten Slot*. 2024. URL: https://www.az-delivery.de/products/ethernet-shield-w5100 (besucht am 18. 03. 2024).

[14] Welt der Elektronik. *HDMI CEC Funktion*. 2024. URL: https://video-kabel.de/blog/hdmi-cec/#:~:text=HDMI%20CEC%20ist%20eine%20zus%C3%A4tzliche,der%20HDMI%20Version%201.3%20eingeflossen. (besucht am 17. 03. 2024).

[15] Elgato. *Streamdeck XL Produktseite*. 2024. URL: https://www.elgato.com/de/de/p/stream-deck-xl (besucht am 18. 03. 2024).

[16] Free Software Foundation. *GNU General Public License 3*. 2024. URL: https://www.gnu.org/licenses/gpl-3.0.de.html (besucht am 18. 03. 2024).

[17] Jesse Garrison. *GitHubHomepage*. 2024. URL: https://github.com/jg33/bmControl (besucht am 16. 03. 2024).

[18] Hexler. *TouchOSC Website*. 2024. URL: https://hexler.net/touchosc (besucht am 18. 03. 2024).

[19] jg33. *bmControl*. 2016. URL: https://github.com/jg33/bmControl (besucht am 16. 03. 2024).

[20] StudioCoast Pty Ltd. *vMix Website*. 2024. URL: https://www.vmix.com (besucht am 18. 03. 2024).

[21] Midisoft - Rolf Meurer. *OSC kurz und knapp erklärt*. 2011. URL: https://midisoft.de/Hints_and_Tools/OSC_Open_Sound_Control.html (besucht am 18. 03. 2024).

[22] NewTek. *TriCaster TC1 Get Started Training Chapter 10 - Workflow Automation*. 2019. URL: https://www.youtube.com/watch?v=FY4iWLgfq2Y (besucht am 18. 03. 2024).

[23] NewTek. *Tricaster TC1 Website*. 2023. URL: https://www.newtek.com/tricaster/tc1/tech-specs/ (besucht am 16. 08. 2023).

[24] NewTek. *Tricaster TC1 Website*. 2024. URL: https://www.newtek.com/tricaster/tc1/ (besucht am 26. 02. 2024).

[25] NewTek/Vizrt. *NewTek and Vizrt announce NDI® 4 and revolutionize production from live to post*. 2019. URL: https://www.vizrt.com/news-articles/broadcasting/newtek-ndi-4/ (besucht am 18. 03. 2024).

[26] OBS. *OBS Website*. 2024. URL: https://obsproject.com/de (besucht am 18. 03. 2024).

[27] Onshape. *Projekt: ArduinoSDIShield*. 2024. URL: https://cad.onshape.com/documents/fc377674bdd63b061ec071c5/w/1a79d1164f93a9b54f595abc/e/e234dc51da67315a30326487 (besucht am 21. 03. 2024).

[28] Onshape. *Website*. 2024. URL: https://www.onshape.com/de/ (besucht am 21. 03. 2024).

[29] OpenSoundControl. *OpenSoundControl.org*. 2021. URL: https://opensoundcontrol.stanford.edu (besucht am 18. 03. 2024).

[30] BC Live Productions. *Building Comps in TriCaster TC2 Elite*. 2023. URL: https://www.youtube.com/watch?v=cyP5gF4UJlc (besucht am 18. 03. 2024).

[31] Ulrich Schmidt. *Professionelle Videotechnik :* 7., aktualisierte und erweiterte Auflage. Springer eBook Collection. Berlin ; Springer Vieweg, 2021. URL: https://doi.org/10.1007/978-3-662-63944-3.

[32] Skaarhoj. *Homepage*. 2024. URL: https://www.skaarhoj.com/ (besucht am 18. 03. 2024).

[33] Edis Techlab. *Wie funktioniert SPI – einfach erklärt!* 2024. URL: https://edistechlab.com/wie-funktioniert-spi/?v=3a52f3c22ed6 (besucht am 18. 03. 2024).

[34] Telestream. *WireCast Website*. 2024. URL: http://www.wirecast.io/en/ (besucht am 18. 03. 2024).

Anhang

Der Anhang dieser Arbeit befindet sich auf dem der Printausgabe beigelegten Datenträger. Er umfasst:

- Die TouchOSC Projektdatei
- Den bearbeiteten Arduino-Code *BmControl*
- Das erstelle 3D-Model des Gehäuses im Step-Format

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel

Anbindung von Blackmagic Design Kamerasystemen an generische Bildmischer

Prototyp-Entwicklung zur Verbesserung und Erweiterung von Kamerakontrollfunktionen und Tally Lights

selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Hamburg, 25. März 2024