



GEORG-SIMON-OHM
HOCHSCHULE NÜRNBERG

Georg-Simon-Ohm Hochschule Nürnberg

Fakultät Elektrotechnik Feinwerktechnik Informationstechnik efi

Projekt MatRigX

Gesamtdokumentation von

**Christian Neubauer, Johannes Kiesel, Christian Siemer
und Armin Voit**

Media Engineering (B-ME 6)

**„Gesamtdokumentation für das Projekt MatRigX im
Rahmen des Studienprojektes in Media Engineering“**

Sommersemester 2012

Bestätigung gemäß § 35 (7) RaPO

Christian Neubauer, Johannes Kiesel, Christian Siemer und Armin Voit

Wir bestätigen, dass wir die Gesamtdokumentation mit dem Titel:

„Gesamtdokumentation für das Projekt MatRigX im Rahmen des Studienprojektes in Media Engineering“

selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet haben.

Datum: Dienstag, 31. Juli 2012

Roadmap

After Effects

Tätigkeit	Dokument	Beteiligung
Video Rendering der Aufnahmen	../Bilder/ ../Video/	Kiesel
Tracking und Kalibrierung der Frames	../Video/	Kiesel
Interpolation der Aufnahmen durch Twixtor	../Video	Kiesel

Auslösen der Kameras

Tätigkeit	Dokument	Beteiligung
Recherche CHDK		Siemer/Voit
Installation CHDK	../CHDK/	Siemer/Voit
Erstellen von CHDK Skripten zum Auslösen der Kameras und zur Datenübertragung	../CHDK/	Siemer/Voit
Entwicklung eines Binärzählers	../Arduino/	Siemer
Auslösen mit Arduino	../Arduino/	Siemer

Kalibrierung/Hardware

Tätigkeit	Dokument	Beteiligung
Konstruktion Multikamerarig		Neubauer
Bau Multikamerarig		Neubauer
Darstellung Grid		Siemer/Voit
Verifikation der Ausrichtungsgenaugkeit		Voit

Kalibrierung/Software

Tätigkeit	Dokument	Beteiligung
Recherche openframeworks/ openCV	../OpenCV/	Kiesel/

		Neubauer/ Siemer/ Voit
Festlegen der Libraries und Installation	../Code/	Kiesel/ Neubauer/ Voit
Laden der Bilder in openCV	../Code/	Kiesel/ Voit
Farberkennung/ Markererkennung	../Code/	Kiesel/ Neubauer/ Voit
Morphologische Operationen zur Markerisolation	../Code/	Voit
Auslesen der Markerpositionen	../Code/	Kiesel/ Voit
Berechnung der affinen/perspektivischen Transformationsmatrix	../Code/	Kiesel /Neubauer/ Voit
Affine/perspektivische Transformation der Bilder	../Code/	Kiesel/ Neubauer/ Voit
Einbinden des Codes in openframeworks	../Code/	Neubauer
Interpolation der Einzelbilder zu einer Vorschauanimation	../Code/	Neubauer

Präsentation des Projekts

Tätigkeit	Dokument	Beteiligung
Zwischenpräsentation	../dokumente/	Kiesel/ Neubauer/ Siemer/ Voit
Abschlusspräsentation	../dokumente/	Kiesel/ Neubauer/ Siemer/ Voit

Projektorganisation

Tätigkeit	Dokument	Beteiligung
Ideenfindung	../dokumente/	Kiesel/ Neubauer/ Siemer/ Voit
Organisation	../dokumente/	Voit

Aufgabenverteilung	../dokumente/	Voit
Protokollierung	../dokumente/	Kiesel/ Neubauer/ Siemer/ Voit
Projektplan	../dokumente/	Voit
Pflichten-/Lastenheft	../dokumente/	Kiesel/ Neubauer/ Siemer/ Voit
Interne und externe Projektkommunikation	../dokumente/	Voit
Gesamtdokumentation	../dokumente/	Siemer/ Voit
Wiki-Dokumentation	Projekt-Server	Voit
Dokumentation des Codes	../Code/	Kiesel/ Neubauer/ Voit
Materialbeschaffung		Neubauer/ Siemer

Inhaltsverzeichnis

Bestätigung gemäß § 35 (7) RaPO	2
Roadmap	3
1. Ausgangssituation	8
2. Projektidee	8
3. Projektorganisation	10
3.1 Rollenverteilung.....	10
3.2 Lastenheft.....	11
3.3 Zeitplan.....	12
3.4 Meetings.....	14
3.5 Blog	15
3.6 Dateiverwaltung	15
3.7 Wiki	16
4. Grobkalibrierung.....	17
4.1 Recherche: Rigbau	18
4.2 Rigbau.....	20
4.3 Grid	24
4.4 Verifikation der Ausrichtungsgenauigkeit des Rigs	26
5. Canon Hack Development Kit (CHDK).....	28
5.1 Sprachen	29
5.2 Installation.....	29
5.3 Auslösen.....	32
5.4 Konsolensteuerung der Kamera	35
5.5 Simultane Triggerung	36
5.6 Synchronitätsnachweis (ohne Arduino)	37
5.7 Datenübertragung.....	38
6. Schaltung	39
6.1 Aufbau	40

6.2 Arduino:	43
6.3 Technische Daten:	44
6.4 Programmierschnittstellen.....	46
6.5 Code:	48
6.6 Schaltungsaufbau:	48
6.7 Probleme:	51
6.8 Problemlösung.....	51
6.9 Synchronität.....	55
6.9.1 Verifizierung LCD.....	55
6.9.2 Verifizierung Arduino.....	57
7. Feinkalibrierung.....	62
7.1 Recherche: openCV & openFrameworks	62
7.2 Image-load Funktion	63
7.3 Markererkennung.....	63
7.4 Morphologische Operationen.....	65
7.5 Transformationsmatrix.....	66
7.6 Implementierung des openCV Codes in openFrameworks.....	67
7.7 Lienare Interpolation mit Alpha Blending	67
7.8 Codedokumentation	68
8. Professionelle Software	69
8.1 Adobe After Effects	69
8.2 Twixtor.....	70
9. Soll/Ist-Vergleich.....	73
10. Resümee	74
 Literaturverzeichnis	77
Internetquellen	77
Abbildungsverzeichnis	78

1. Ausgangssituation

Ausgangssituation für das Projekt mit dem Namen "MatRigX" war die, im 6. Semester des Studiengangs Media Engineering an der Georg-Simon-Ohm Hochschule vorgesehene Projektarbeit. Ziel der Projektarbeit war die Konzipierung und Realisierung einer Idee gemeinschaftlich innerhalb einer Gruppe. Hierbei wird im Gegensatz zu anderen Studiengängen ein besonderer Schwerpunkt auf die Gruppenarbeit gelegt. Die Projektarbeit erstreckte sich über das komplette Sommersemester 2012.

Angedacht war, dass die Studenten ihr bisher bestehendes Projekt aus dem Wintersemester 2011/2012 weiterführen und vertiefen, jedoch stand es den Studenten letztendlich dabei frei, ob sie ihr Projekt fortführen oder ein neues Thema wählen. Die, aus den Studenten Johannes Kiesel, Christian Neubauer, Christian Siemer und Armi Voit, gegründete Gruppe MatRigX hat sich wie jede andere Gruppe des Studiengangs Media Engineering dazu entschieden, ein neues Projekt zu beginnen, um somit den eigenen Horizont erweitern und neue Impressionen aus anderen technischen Bereichen einsammeln zu können.

Als Auftraggeber traten Prof. Dr. Stefan Röttger und Prof. Dr. Matthias Hopf in Erscheinung.

2. Projektidee

Seinen Ursprung hatte dieses Projekt im ersten gemeinsamen Treffen der Studenten des Studienfachs Media Engineering aus dem 6. Semester, einer Studentin des 4. Semesters, sowie den betreuenden Professoren, Prof. Dr. Stefan Röttger, Prof. Dr. Matthias Hopf, Prof. Dr. Heinz Brünig, Prof. Dr.(USA) Ralph Lano und Prof. Dr. Hans-Georg Hopf. Nach einer regen Diskussion und zahlreichen Vorschlägen von Seiten der Studenten,

aber auch der Professoren entwickelte sich unser Projekt ausgehend von der Idee eines Studenten. Der Vorschlag bestand in einer Art Kamerafahrt, die einen großen Wert auf eine künstlerische und gestalterische Komponente legte, was jedoch nicht den benötigten technischen Bestandteil aufweisen konnte, den es für den Studiengang bedarf. Diese Idee wurde dann von den Professoren weiterentwickelt und so angepasst, dass sie in einen Projektrahmen passte, der für Media Engineering angemessen war. Die Kamerafahrt blieb bestehen, jedoch wurde das gestalterische Element in den Hintergrund gerückt und der Fokus auf ein Anderes gelegt. Die Idee war nun die Nachbildung einer sogenannten Bullet Time.

Bullet Time(engl. bullet: Projektil und time: Zeit) bezeichnet in der Filmkunst einen Spezialeffekt, bei dem der Eindruck einer Kamerafahrt um ein in der Zeit eingefrorenes Objekt herum entsteht. Dieser Spezialeffekt erlaubt, schnelle Geschehnisse, zum Beispiel fliegende Pistolenkugeln, genau und von verschiedenen Blickwinkeln aus zu sehen. Ebenso sind Verlangsamungen bis hin zum Stillstand und sogar rückwärts laufende Zeit möglich. Neben der Verwendung in Actionfilmen wird Bullet Time auch als Spielelement in Videospielen verwendet.¹

Wohingegen ein Bullet Time Effekt in einem Spiel relativ einfach erzeugt werden kann, da der 3D Raum meist schon besteht und lediglich die virtuelle Kamera, deren Bild letztendlich der Benutzer sieht, bewegt werden muss, ist bei einer cineastischen Variante der Bullet Time schon etwas mehr Aufwand nötig, um eine Fahrt im zeittoten Raum zu simulieren. Hierbei bedient man sich eines sogenannten Rigs mit einem Array von Kameras. Diese werden auf dem Rig, was frei übersetzt soviel heißt wie Aufbau, nebeneinander gestellt und zeitgleich oder leicht sukzessiv ausgelöst je nach Wunsch. Die eigentliche Kamerafahrt besteht nun darin, von einem Bild zum anderen zu springen, sprich die Bilder der Kameras nacheinander ablaufen zu lassen, um so den gewünschten Kamerafahrteffekt zu erzeugen.

¹ http://de.wikipedia.org/wiki/Bullet_Time (Stand: 13.07.2012).

Der Film Matrix hat dieses Genre revolutioniert. Noch heute wird beim Betrachter eine Bullet Time mit den berühmten Szenen der Protagonisten des Films assoziiert. An den überragenden Aufnahmen des Films orientierend, haben wir uns ebenfalls dazu entschlossen solch eine Bullet Time zu simulieren.

3. Projektorganisation

Eine gelungene Projektdurchführung ist gekennzeichnet von einer akribischen Projektorganisation. Dies erforderte in unserem Fall regelmäßig veranstaltete Meetings, die Errichtung eines Blog, sowie das Erstellen einer Dropbox als Lösung einer gemeinsamen Dateiverwaltung und das Einrichten eines Gits zur Darstellung und Kollaboration unseres gemeinsamen Codes.

Diese gemeinsamen Meetings und Dokumentations- bzw. Organisationstools werden auf den folgenden Seiten näher erläutert.²

3.1 Rollenverteilung

Die Rollenvergabe soll erzielen, komplexe Aufgaben, die sich über einen längeren Zeitraum erstrecken, zu strukturieren, um somit die Arbeit in einer Gruppe zu erleichtern und einen harmonischen Arbeitsfluss zu gewährleisten. Ist die Rollenverteilung erfolgt, ist es ebenfalls von Nöten die Kommunikation zu seinen Teammitgliedern aufrecht zu halten und neuste Fortschritte bei eigenen Aufgaben mit der Gruppe zu teilen.

² <http://wiki.ohm-hochschule.de/roettger/index.php/Project-R3DS/Projektorganisation>
(Stand: 02/2012)

Werden diese Arbeitstechniken eingehalten, ist der Grundstein für eine gelungene Projektarbeit gesetzt.

Das Team des Projekts „MatRigX“ hat sich ebenfalls für eine, wenn auch nicht so strikte, Rollenvergabe entschieden. Armin Voit wurde von den Projektmitgliedern einstimmig zum Projektleiter ernannt.³ Aufgrund der Tatsache, dass diese die wichtigste Rolle ist, da der Projektleiter nach Außen sowohl Gruppe als auch das Projekt repräsentiert, wurde dieser Part von uns offiziell im ersten Sitzungsprotokoll festgehalten. Die restlichen Rollen wurden nicht formal niedergeschrieben, sondern in gemeinsamer Einigung mündlich definiert. Die gemeinsam ausgearbeitete Aufgabenverteilung sah folgendes vor.

Die Aufgabe eines Jeden von uns war es, darüber hinaus abwechselnd Sitzungsprotokolle der wöchentlich stattfindenden Teammeetings zu erstellen und diese daraufhin in digitaler Form den anderen Mitgliedern zur Verfügung zu stellen. Ebenfalls in den gemeinsamen Verantwortungsbereich fiel die Erstellung und Instandhaltung eines eigenen Blogs, um somit neuste Prozesse schriftlich festzuhalten und der Öffentlichkeit präsentieren zu können, sowie auf Seiten der nicht öffentlichen Arbeit die Errichtung und ständige Aktualisierung einer Dropbox, die alle Teammitglieder, sowie die Betreuer zeitnah auf den gleichen Stand bringt und die gemeinsame Nutzung einer eigens für das Projekt gegründeten Facebook Gruppe zum schnellen Informationsaustausch.⁴

3.2 Lastenheft

Das Lastenheft (teils auch *Anforderungsspezifikation*, *Anforderungskatalog*, *Produktskizze*, *Kundenspezifikation* oder

³ 1.Protokol_10.04.12.pdf, S.1.

⁴ <http://wiki.ohm-hochschule.de/roettger/index.php/Project-R3DS/Aufgabenverteilung> (Stand 02/2012)

Requirements Specification) beschreibt die Gesamtheit der Forderung des Auftraggebers an die Lieferungen und Leistungen eines Auftragnehmers.⁵ In einem ersten Meeting hat das Projektteam gemeinsam die Grundlage für das Lastenheft gelegt und Themen spezifiziert. Zusammen haben wir diese Spezifizierung dann in Worte gefasst und ein konkretes Lastenheft erstellt, in dem die Forderungen des Auftraggebers, in unserem Fall Prof. Dr. Stefan Röttger und Prof. Dr. Matthias Hopf, niedergeschrieben wurden.

3.3 Zeitplan

Die Erstellung und konsequente Einhaltung eines Zeitplan von Beginn eines Projektes an ist maßgeblich für das Gelingen eben dieses ausschlaggebend. Dabei müssen gleich mehrere Dinge berücksichtigt werden.

Zum einen muss sich der Projektleiter einen Überblick über den Umfang der Projektarbeit verschaffen. Dies kann jedoch nur ein grober Schätzwert werden, da im Laufe eines jeden Projekts immer neue Schwierigkeiten auftreten können, die zu Beginn nicht einsehbar waren und dann viel Zeit in Anspruch nehmen können. Auch Aufgaben, die zu Beginn als schnell vollendbar angesehen werden, können sich im Laufe des Projekts als größeres Hindernis darstellen, was wiederum eine entsprechende Anpassung des Zeitplans zur Folge hat, um gegebenenfalls wichtigere Aufgaben noch dementsprechend zu priorisieren. Natürlich kann auch der Fall eintreten, dass eine Aufgabe schneller als geplant abgearbeitet wird. Die, sich daraus ergebenden, frei werden Ressourcen müssen dementsprechend neu verwaltet werden und mit nächsten, eigentlich erst später vorgesehenen, Aufgaben beauftragt werden. Dies ist auch gleich ein weiterer Punkt, der bei der Zeitplanung berücksichtigt werden muss.

⁵ <http://de.wikipedia.org/wiki/Lastenheft> (Stand: 15.06.2012).

Wie hoch ist die Anzahl der Ressourcen und über welche Kompetenzen verfügen diese? Zu Beginn muss sich der Projektleiter einen Überblick verschaffen, wie viele Personen er für welches Einsatzgebiet einteilen kann und über welche Kompetenzen, diese Personen verfügen. So ist es von größter Wichtigkeit schon vor dem eigentlichen Projekt im gemeinsamen Gespräch zu eruieren, in welchen Themengebieten die Projektmitglieder am versiertesten sind und so wissensspezifisch Aufgaben zu verteilen, um ein möglichst effektives und effizientes Arbeiten garantieren zu können. Ein Punkt, der sich auch oft, wie z.B. auch in unserem Fall, negativ auf den Zeitplan auswirken kann, sind äußere Faktoren. Hierbei kann sich der Projektleiter und das Team nur sehr schlecht bis gar nicht darauf einstellen, da das Projektteam keinen Einfluss auf diese äußeren Faktoren hat. In unserem Fall hieß das konkret, dass wir erst nach drei Monaten alle unsere Kameras zu Verfügung gestellt bekommen hatten. Da wir keinerlei Einfluss darauf hatten, wann die Kameras letztlich geliefert werden, konnten wir den Zeitplan auch nicht dementsprechend anpassen. Die äußeren Faktoren gelten demnach als das schwerste zu kalkulierende Element bei der Erstellung von Zeitplänen.

Ein Zeitplan ist also ein ständig zu überwachendes Dokument, auf das durchgehend reagiert werden muss.

Der Projektmanager wurde mit der Aufgabe vertraut einen Zeitplan für das Projekt MatRigX zu erstellen. Der, gemeinsam mit dem Team ausgearbeitete, Zeitplan sah vier Phasen für unser Projekt vor. Die erste Phase beschäftigte sich mit dem Rigbau. In dieser Phase wurde bereits bestehende Rigs betrachtet und bewertet. Im Anschluss daran wurde unser eigenes Rig konzipiert und konstruiert.

Phase zwei sah vor, eine synchrone Auslösung zu bewerkstelligen, sowie die Realisierung einer Fernbedienung mittels dem Open-Source Mirco Controller Arduino.

Die dritte Phase widmete sich der Kalibrierung unserer Bilder. Dies hieß eine eigene Software zu schreiben, mit Hilfe der freien Bibliotheken openCV und openFrameworks. Darin enthalten war eine Markererkennung und das Erstellen einer Transformationsmatrix.

Als vierte und letzte Phase war die Interpolation und die Ausgabe unserer bearbeiteten Bilder vorgesehen. Dazu wurde eine lineare Interpolation, auch Alpha-Blending genannt, erzeugt.

Die vier Phasen sollten dabei aber keineswegs so verstanden werden, dass sie von chronologischer Natur sind.

3.4 Meetings

Zur optimierten Kommunikation sowie Stärkung und Verbesserung der Teamarbeit legten wir gemeinsam fest, dass wir wöchentlich ein Treffen abhalten werden, in welchem neuste Ergebnisse präsentiert und wiederum neue Aufgaben verteilt werden können. Die Aufgabe eines Jeden von uns war es, darüber hinaus abwechselnd Sitzungsprotokolle der wöchentlich stattfindenden Teammeetings zu erstellen und diese daraufhin in digitaler Form den anderen Mitgliedern und betreuenden Professoren zur Verfügung zu stellen.⁶

Im späteren Verlauf trafen sich gezielt auch nur Teile des Projektteams, um so explizit die gemeinsame projektbezogene Vertiefungsrichtung zu bearbeiten, z.B. Anwendung der Transformationsmatrix auf die von uns geschossenen Bilder.

⁶ <http://wiki.ohm-hochschule.de/roettger/index.php/Project-R3DS/Meetings> (Stand: 02/2012).

3.5 Blog

Unsere Arbeit sollte sowohl für jetzige als auch zukünftige Zeiten Informationen über Fortschritt, Probleme, Schwierigkeiten, Lösungswege etc. öffentlich festhalten. Aus diesem Grund wurde uns und allen anderen Projektteams nahe gelegt unseren Projektfortschritt in einem öffentlichen Blog niederzuschreiben.⁷

Der Blog ist unter folgendem Link zu erreichen:

<http://www.matrigx.blogspot.de>

Unsere Aufgabe bestand darin, einen Blog zu erstellen und ihn mit den neusten Erkenntnissen und Fortschritten unseres Projektes zu füllen. Da der Blog unser Projekt nach außen hin repräsentiert hat, haben wir stets, sobald neue Projektfortschritte erzielt wurden, diese daraufhin als neuen Post im Blog angelegt. Oft wurden die Posts auch mit Bildern vom Projekt unterlegt, um sich somit ein besseres Bild vom aktuellen Projektstand machen zu können.

3.6 Dateiverwaltung

Um ein effektives und effizientes Kollaborieren zu gewährleisten, entschieden wir uns für die Softwarelösung Dropbox. Dropbox ist eine Internetdienstleistung, mit der man Dateien auf verschiedenen Rechnern synchronisieren und bearbeiten kann. Jede Datei, die man in den freigegebenen Ordner auf dem Rechner speichert, wird sofort auf allen anderen Rechnern, die auch auf diesen Ordner zugreifen können,

⁷ <http://schorsch.efi.fh-nuernberg.de/mewiki/index.php/Project-BME-2012-07-MatRigX/Blog> (Stand: 07/2012).

synchronisiert. Somit ist gewährleistet, dass jedem Teammitglied zu jeder Zeit die gleichen Daten zur Bearbeitung des Projekts zur Verfügung stehen.

Dropbox ist mittlerweile für alle gängigen Plattformen verfügbar. Dazu zählen unter anderem Windows, Mac OS, Linux, Android und iOS. Auch ein Zugriff per Webbrowser auf die Daten ist möglich.⁸

Im späteren Verlauf unseres Projekts wurde uns von unseren betreuenden Professoren geraten, unseren bisherigen Code in einem speziellen Verwaltungssystem, einem sogenannten Git abzulegen, um somit leichter kollaborieren zu können. Daraufhin entschieden wir uns für die Seite Github.com. Github ist ein webbasierter Hosting-Dienst für Software-Entwicklungsprojekte. Er verwendet namensgebenderweise das Versionsverwaltungs-System Git.⁹ Unsere Aufgabe war es nun ein Git, eine ordentliche Struktur für das Repository zu erstellen und unseren Code hochzuladen. Des Weiteren fiel es in unseren Aufgabenbereich neuere Versionen unseres Codes hochzuladen, um unser Git auf dem neusten Stand zu halten. Unser kompletter Code ist auf dem Git des Projektserver der Hochschule einzusehen

3.7 Wiki

Um unseren Projektverlauf und die -ergebnisse nachhaltig festzuhalten und sie für eine breite Masse z.B. nachfolgenden Studenten zugänglich zu

⁸ <http://schorsch.efi.fh-nuernberg.de/mewiki/index.php/Project-BME-2012-07-MatRigX/Dateiverwaltung> (Stand: 07/2012).

⁹ <http://de.wikipedia.org/wiki/GitHub> (Stand: 18.07.2012).

machen, haben wir die wichtigsten Punkte in einem Wiki auf dem Projektserver der Ohm Hochschule dokumentiert.¹⁰

Für die Erstellung des Wikis hat sich der Projektleiter bereit erklärt. Wichtig war es hierbei einen Überblick über das Thema zu liefern, ohne sich dabei im Detail zu verlieren. Das Wiki soll als Anlaufstelle für zukünftige Projekte gedacht sein, die sich über unser Projekt informieren und gegebenenfalls darauf aufbauen wollen. Weiterführende und vertiefende Informationen zu unserer Projektarbeit soll die Gesamtdokumentation liefern.

4. Grobkalibrierung

Um ein möglichst gutes Endresultat zu erzielen, haben wir uns darauf geeinigt und auch im Lastenheft festgehalten, dass wir unsere aufgenommenen Bilder, vor der Ausgabe als Video, kalibrieren müssen. Dies wurde auch ein Schwerpunkt unseres Projekts. Zuerst überlegten wir wie wir die Bilder kalibrieren könnten. Relativ schnell wurde uns klar, dass wir hierfür zwei Arten der Kalibrierung benötigen. Zuerst sollte eine Grobkalibrierung durchgeführt werden, welche schon vor dem eigentlichen Aufnehmen der Bilder von statten geht. Diese Kalibrierungsart wurde mittels unserer Hardwarekomponenten erzielt. Als zweiter Schritt zur Optimierung unseres Endprodukt, dem fertigen Videos, sahen wir eine Feinkalibrierung vor. Diese wurde dann auf Softwarebasis realisiert.

¹⁰ <http://schorsch.efi.fh-nuernberg.de/mewiki/index.php/Projects/BME-2012-07-MatRigX> (Stand: 07/2012).

4.1 Recherche: Rigbau

Um unser Ziel, eine Bullet Time zu simulieren, in die Tat umsetzen zu können, benötigten wir eine Vorrichtung, mit der es möglich war, alle acht Kameras gleichzeitig darauf zu platzieren. Eine solche Vorrichtung wird in Fachkreisen Rig genannt. Dieses dient einer äquivalenten Ausrichtung der einzelnen Kameras. Im Bereich der professionellen Filmproduktion bestehen Rigs meist aus Aluminium oder Carbon. Aufgrund der Tatsache, dass diese Materialen sehr leicht sind, jedoch große Robustheit und Präzision aufweisen, erfreuen sich Rigs aus eben diesen Stoffen größter Beliebtheit. Da unser Projekt eine studentisches Projekt war und ohne finanzielle Zuschüsse auskommen musste, somit ein No-Budget-Projekt war, mussten wir einen Kompromiss zwischen Präzision und Preis beim Rigbau beachten. Wir haben uns daraufhin geeinigt, ein Rig aus Holz zu bauen. Nun stand die Frage im Raum, wie das Rig auszusehen hat.

Bevor es jedoch soweit war, haben wir uns zuvor betrachtet, welche prinzipiellen Aufbauten es bei Rigs gibt. Nach einiger Recherche konnten wir zwei prinzipielle Bauweisen nachweisen. Zum einen gibt es den geradlinigen Aufbau. Hierbei werden die Kameras nebeneinander auf dem Rig platziert und alle parallel zueinander, in die gleiche Blickrichtung, ausgerichtet.



Abbildung 1: Geradliniges Rig mit parallelen Kamera

Die zweite Variante eines Rigs ist der kreisförmige Aufbau. Hierbei wird das Rig kreisförmig um das abzulichtende Objekt aufgestellt. Die Kameras sind hier, nicht wie zuvor parallel, sondern auf das zentrale Objekt ausgerichtet. Die wahrscheinlich bekannteste Szene, die mit einem zentral gerichteten Rig erstellt wurde ist aus dem Film Matrix.



Abbildung 2: Kreisförmiges Rig aus dem Film "Matrix"

Ausgehend von der Popularität der Matrix Filme und den sensationellen Bildern, die noch ein Jeder im Gedächtnis hat, haben wir uns dafür entschieden, unser Rig als ein Kreisförmiges zu gestalten. Das Rig hatte fortan den Zweck, dass alle Kameras auf einer Ebene ausgerichtet sind

und sich in ihrer horizontalen Ausrichtung im Raum parallel zum Rig befinden. Da unser Rig aus Kostengründen aus minderwertigem Holz hergestellt wurde und zusätzlich noch in einem sehr warmen Raum gelagert wurde, konnten wir leider nicht verhindern, dass es sich mit der Zeit stark in seiner Form veränderte. Bedingt durch den aufgeheizten Raum und die starke Sonneneinstrahlung verzog sich das Rig sehr schnell und stark. Dies machte es uns leider schon nach kurzer Zeit immer schwieriger genaue Messungen am Rig durchzuführen und mit dem Rig zu arbeiten, da sich ständig die Resultate unterschieden und die Ergebnisse immer ungenauer wurden.

4.2 Rigbau

Um zu gewährleisten dass sich alle unsere Kameras in der gleichen Ebene befestigen lassen haben wir ein für unserer Zwecke angepasstes Multikamerarig entworfen und gebaut das als Halterung für unsere acht Kameras diente und einen wichtigen Faktor in der (Grob-) Hardwarekalibrierung darstellt.

Nachdem wir uns grundsätzlich auf die Form des Rigs geeinigt hatten, stellte sich für uns die Frage welche Faktoren für unser Rig wichtig sind und auf jeden Fall in der Konstruktion verinnerlicht werden mussten.

Gleich mehrere Punkte sollte das zukünftige Rig nach unserer Meinung erfüllen. Dazu zählte unter anderem, dass der Aufbau möglichst stabil und robust sein sollte, da es sowohl das Gewicht der Kameras tragen können muss als auch den geplanten Transport an verschiedene Orte möglichst unbeschadet überstehen sollte. Weiter sollte es möglichst flexibel in Auf- und Abbau sein, damit das Rig auch für den mobilen Einsatz verwendet werden kann.

Und der dritte und der für uns wichtigste Punkt, sollte es kostengünstig in der Herstellung sein, da dies ein No-Budget-Projekt war.

Folgendes Bild stellt einen Schnappschuss des von mir in Google SketchUp7 (Jetzt Trimble SketchUp) erstellten 3D Modells unseres geplanten Rigaufbaus.

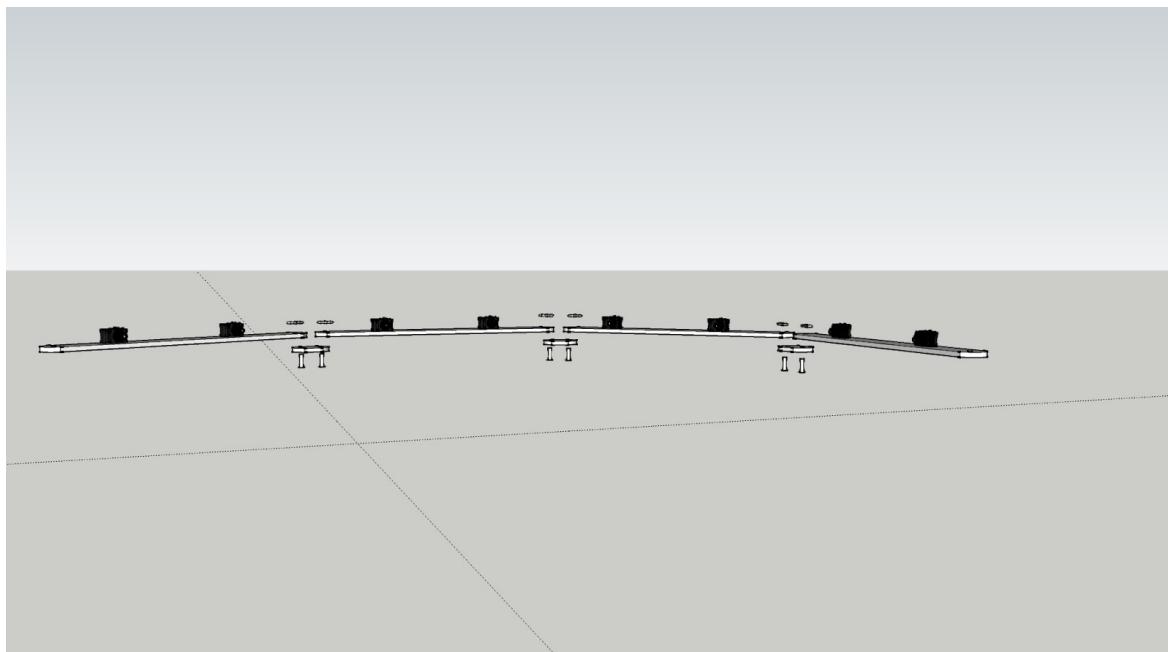


Abbildung 3: Konzept des Rigs

Der Entwurf sieht vor das Rig in vier Elemente zu teilen die jeweils aus einem Brett von 1350 x 150 x 20 mm bestehen die auf an den Enden halbkreisförmig abgerundet sind. Im Zentrum dieses Kreises befindet sich jeweils ein Loch durch das die Elemente untereinander verbunden werden können.

Zwischen befindet sich jeweils eine "Schiene" aus regelmäßigen Bohrungen in jeweils 5cm Abständen wodurch wir die Kameras nahezu frei zueinander positionieren können um verschiedene Zwischenabstände zu nutzen oder um später mehr als die jetzigen 8 montieren zu können.

Als Verbundstücke sind 3 300 x 150 x 20mm Bretter vorgesehen die ebenfalls an beiden Enden halbkreisförmig abgerundet sind und ebenfalls

mit einem Bohrloch im Zentrum des jeweiligen Halbkreises versehen wurden.

Verbunden werden diese Stücke mit 80mm M8 Schrauben die von oben mit passenden Flügelmuttern befestigt werden um ein leichtes Zerlegen zu gewährleisten.

Durch diese Art der Verbindung und der Länge der Bretter erreichen wir eine Gesamtspannweite von etwas mehr als 5m was bei dem von uns Anfangs anvisierten Objekt, einem Kletterer, einen möglichen Umspannwinkel von ~270 Grad bedeutet. natürlich ist durch die Gelenke jede Art von Anordnung möglich abhängig davon wie groß das Objekt ist. Bei kleinen Gegenständen Könnte man von 4 Seiten gleichzeitig bei großen Objekten nur aus einem kleinen Winkel fotografieren. Ebenso wäre es möglich den zuvor beschriebenen linearen Aufbau mit diesem Rig einzusetzen wenn man das wünscht.

Als Befestigung für die Kameras waren 45mm x ¼"-Schrauben vorgesehen die von unten in die gebohrten Löcher gedreht werden und dadurch die nötige Stabilität erhalten. ¼" ist das Standardstativgewinde an nahezu allen gängigen Kameras.

Wie in der Planung vorgesehen habe ich für den Bau des Rigs Holz verwendet dieses konnte aus Restbeständen gewonnen werden wodurch wir unsere Kosten minimieren konnten. Dieses habe ich nach den zuvor gefertigten Plänen zugeschnitten und danach, da es sich um unbearbeitetes, ungeschliffenes Bauholz handelte mit einem Winkelschleifer glatt geschliffen um Verletzungen bei der Handhabung vorzubeugen und eine bessere Verschiebbarkeit der Gelenkteile zueinander herzustellen.



Abbildung 4: Fertiges Rig

Das linke Bild zeigt das Rig in seinem zusammengeklappten, platzsparenden und portablen Zustand. Das Rechte zeigt eine Detailansicht eines Gelenks.



Abbildung 5: Nahaufnahme des Rigs

4.3 Grid

Unser fertiges Rig war lediglich für eine horizontale Ausrichtung unserer Kameras ausgelegt. Damit die Kameras das zu aufzunehmende Objekt möglichst alle im gleichen Bildbereich haben – in unserem Fall stets der Bildmittelpunkt - um ein späteres Springen bei der Bilderfolge zu verhindern, haben wir uns dazu entschieden ein Grid auf der Kamera anzeigen zu lassen. Ein Grid ist ein Raster, das den Bildschirm in einzelne Partitionen unterteilt. Die bekanntesten Grids, oder auch Raster, sind der goldene Schnitt¹¹ und das 3x3 Grid¹². Da wir für unsere Zwecke einen möglichst konkreten Punkt des aufzunehmenden Objektes anvisieren und nicht nur das aufzunehmende Objekt in einen bestimmten Bereich auf dem Display einordnen wollten, da uns dies zu ungenau erschien, entschieden wir uns deshalb, ein Grid zu erstellen, welches einem Fadenkreuz gleicht. Die Auswahl und Erstellung des Grids, welches wir letztlich verwendeten wurde von uns vollzogen. Das „Fadenkreuz“ unserer Kameras bestand aus einer horizontalen und einer vertikalen Linie und hatte seinen Schnittpunkt in der Bildmitte.



Abbildung 6: Grid auf dem Display einer Kamera

¹¹ Definition: Unter Goldener Schnitt versteht man das Teilungsverhältnis einer Strecke, bei der die größere Teilstrecke zur gesamten Strecke im gleichen Verhältnis steht, wie die kleinere Teilstrecke zur Größeren.

¹² Definition: Aufteilung des Bildschirms in 9 gleich große Rechtecke.

So konnten wir nun einen konkreten Punkt in unserem Objekt mit allen Kameras anvisieren und somit erzielen, dass das Objekt bei allen Kameras in gleichen Bildschirmbereich auftritt.

Durch Befehle mit unterschiedlichen Koordinaten kann das Raster gezeichnet werden. Es erinnert an ein Canvas wie bei anderen Programmiersprachen.

```
@ Title <text in Menu> zeigen  
@ Line x0, y0, x1, y1, lineColor  
@ Rect x0, y0, x1, y1, borderColor  
@ Rectf x0, y0, x1, y1, borderColor, fillColor  
@ ELPs x0, y0, rx, ry, borderColor  
@ Elpsf x0, y0, rx, ry, fillColor
```

Für einen besseren Einblick kann man sich z.B. den Code von einem Fadenkreuz genauer anschauen.

```
@ Title Goldenen Kreuz-Ratio  
  
@ Line 133, 92, 141, 92, 0xFF  
@ Line 218, 92, 226, 92, 0xFF  
@ Linie 133, 148, 141, 148, 0xFF  
@ Linie 218, 148, 226, 148, 0xFF  
  
@ Line 137, 88, 137, 96, 0xFF  
@ Linie 137, 144, 137, 152, 0xFF  
@ Line 222, 88, 222, 96, 0xFF  
@ Linie 222, 144, 222, 152, 0xFF  
  
@ Linie 176, 120, 184, 120, 0xFF  
@ Linie 180, 116, 180, 124, 0xFF
```

Das Ergebnis ist ein Fadenkreuz was folgendermaßen ausschaut.

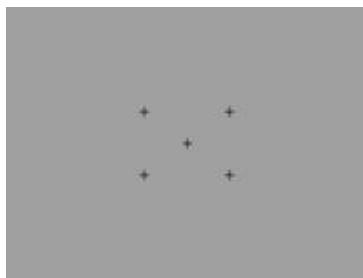


Abbildung 7: Beispiel Grid

4.4 Verifikation der Ausrichtungsgenauigkeit des Rigs

Nachdem unser Rig gebaut und unsere Kameras mit Grids versehen wurden, konnten wir eine quantitative Verifikation der Ausrichtungsgenauigkeit unserer Konstruktion vornehmen. Hierfür nahmen wir zusammen eine Szene mitsamt dem Rig und allen acht Kameras auf. Objekt der Szene war ein Lot, welches in der Mitte einen blauen Marker hatte und orthogonal zum Boden hing. Mit Hilfe der Grids visierten wir diesen blauen Marker mit jeder Kamera an. Nachdem wir die Fotos aufgenommen hatten, hatten wir also acht Einzelbilder von verschiedenen Perspektiven mit jeweils dem gleichen Motiv. Daraufhin bearbeiteten wir diese Bilder so, dass sie alle mit halber Transparenz übereinander lagen und ein neues Bild erzeugten.

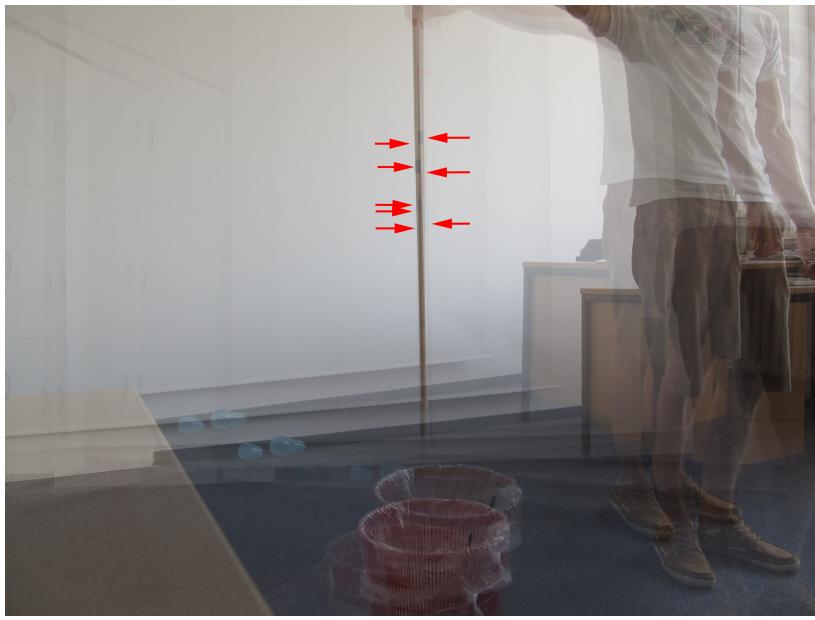


Abbildung 8: Verifikation der Ausrichtungsgenauigkeit des Rigs

Wie in der oberen Abbildung deutlich zu sehen, trat bei den Kameras ein deutlicher Unterschied auf, wo der Marker im Bild zu sehen ist. Dies hatte seine Folgen, in der zuvor beschriebenen Ungenauigkeit des Rigs. Dadurch dass sich das Rig extrem verformt hatte, standen die Kameras leider alle in einem leicht anderen Winkel auf dem Rig. Die roten Pfeile geben dabei die Lage des Markers wieder. Ein Nachmessen des Rig mit einer Wasserwaage hat ergeben, dass die Kameras einen maximalen Winkelunterschied von 5° Grad aufweisen. Dies ergibt bei einem Abstand der Kameras vom Objekt, hier das Markierungslot, von 2,50 Meter einen maximalen Höhenunterschied der Marker von knapp 22cm. Berechnet wird dies über den Tangens. Da wir den Abstand von Kamera zu Objekt wissen, sowie den Winkelunterschied der Kameras zueinander können wir die Gleichung des Tangens nach der gesuchten Gegenkathete auflösen und erhalten so das Ergebnis von ca. 22cm maximaler Höhenunterschied bei den Markern. Die nächste Abbildung soll dies veranschaulichen.

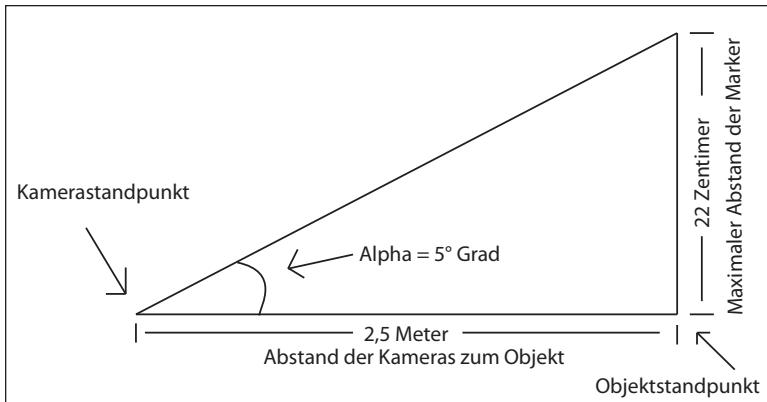


Abbildung 9: Berechnung der Gegenkathete

5. Canon Hack Development Kit (CHDK)

Das Canon Hack Development Kit ist ein Firmware-Aufsatz für eine große Anzahl an Canon Kameras, welcher auf Open Source Basis entstand. CHDK wird nicht direkt auf die Kamera, sondern auf eine einführbare Speicherkarte geschrieben, um so die Original Firmware unangetastet zu lassen. Durch die Verwendung von CHDK sind enorme Funktionserweiterungen bei den verwendeten Canon Kameras möglich.¹³ So ist es z.B. bei Digitalkameras möglich auf die sonst für den Benutzer nicht zur Verfügung stehenden .raw-Dateien zuzugreifen, anstatt auf die üblichen Bilder im .jpg Format. Für unsere Zwecke war jedoch eine andere Eigenschaft von CHDK von besonderem Nützen. Und zwar haben wir uns die Möglichkeit zu eigen gemacht, mittels CHDK unsere Kameras fernzusteuern. Dazu gleich mehr in Punkt 5.2 Simultane Triggerung.

¹³ http://chdk.wikia.com/wiki/CHDK_in_Brief.

5.1 Sprachen

Wird CHDK auf dem Kamera installiert, so sind schon vorgeschriebene Programme in den Skriptsprachen uBasic und Lua vorhanden. Auch wenn man nun selbst etwas programmieren möchte, hat man die Möglichkeit zwischen beiden Sprachen zu unterscheiden. uBasic war die Skriptsprache, die zusammen mit CHDK eingeführt wurde. Nach einiger Zeit, als die Skripte immer komplexer wurden und die Programmiermöglichkeiten an ihre Grenzen der sehr simplen Sprache uBasic gestoßen sind, fand im April 2008 die Integration von Lua in CHDK statt.¹⁴ Lua ist eine Skriptsprache, die deutliche Vorteile zu uBasic liefert. Dazu zählt z.B., dass nun auch Fehlermeldungen auf dem Display ausgegeben werden können. Dies war mit uBasic bisher nicht möglich. Des Weiteren ist die Syntax konsistenter geworden und erstmals kann man auch eigene Funktionen definieren. Der Nachteil von Lua kommt mit seiner Komplexität und seiner späten Einführung in CHDK. Viele Nutzer von CHDK sind keine versierten Programmierer. Dies ist auch keineswegs notwendig, solange man sich nur mit uBasic beschäftigt. Die Syntax bei uBasic ist auch für einen Nicht-Programmierer verständlich. Jedoch sollten sich Neulinge in CHDK von Anfang an mit der Syntax in Lua anfreunden. Auch wenn es anfangs sehr frustrierend erscheinen mag, eigene Skripte zu erstellen, so sind doch die Möglichkeiten, wenn man erst einmal den Grundaufbau verstanden hat, um einiges größer als mit uBasic.

5.2 Installation

Um CHDK auf unseren Canon Powershot SX 130 IS installieren zu können, reichte es nicht aus lediglich den Namen der Kamera zu wissen, sondern

¹⁴ <http://chdk.setepontos.com/index.php/topic,1194.0.html> (Stand 04/2008).

man benötigte die konkrete Versionsnummer der darauf gespielten Firmware. Die Versionsnummer ist für den Endkunden eigentlich nicht einsehbar. Um an diese Nummer zu kommen, gibt es verschiedene Möglichkeiten. Wir werden im Folgenden nur die Methode beschreiben, die wir gewählt haben, um unsere Kameras mit der korrekten CHDK Version zu bespielen.

Wir haben mich dazu entschieden das Programm ACID¹⁵ zu benutzen. ACID ist eine plattformunabhängige Java Applikation, die entwickelt wurde, um auf einfache Art herauszufinden, welche Firmware-Version auf einer Canon Kamera installiert ist.¹⁶ Dabei geht man wie folgt vor:

Zuerst wird ein Bild mit der entsprechenden Kamera gemacht von der man die Firmware-Version erfahren möchte. Darauf hin zieht man das Foto via Drag 'n Drop in das Programm, welches nun das Bild analysiert und aus den Exif¹⁷ Daten des Bildes Metadaten zum Kameramodel und dessen Firmware-Version ausliest. Folgend ein Beispielbild zur Version 1.09 von ACID.

¹⁵ ACID: **A**utomatic **C**amera **I**entifier and **D**ownloader.

¹⁶ <http://chdk.wikia.com/wiki/ACID>.

¹⁷ Definition: Das **E**xchangeable **I**mage **F**ormat ist ein Standart der Japan Electronic and Information Technology Industries Association für das Dateiformat, in dem moderne Digitalkameras Informationen über die aufgenommenen Bilder (Metadaten) speichern.

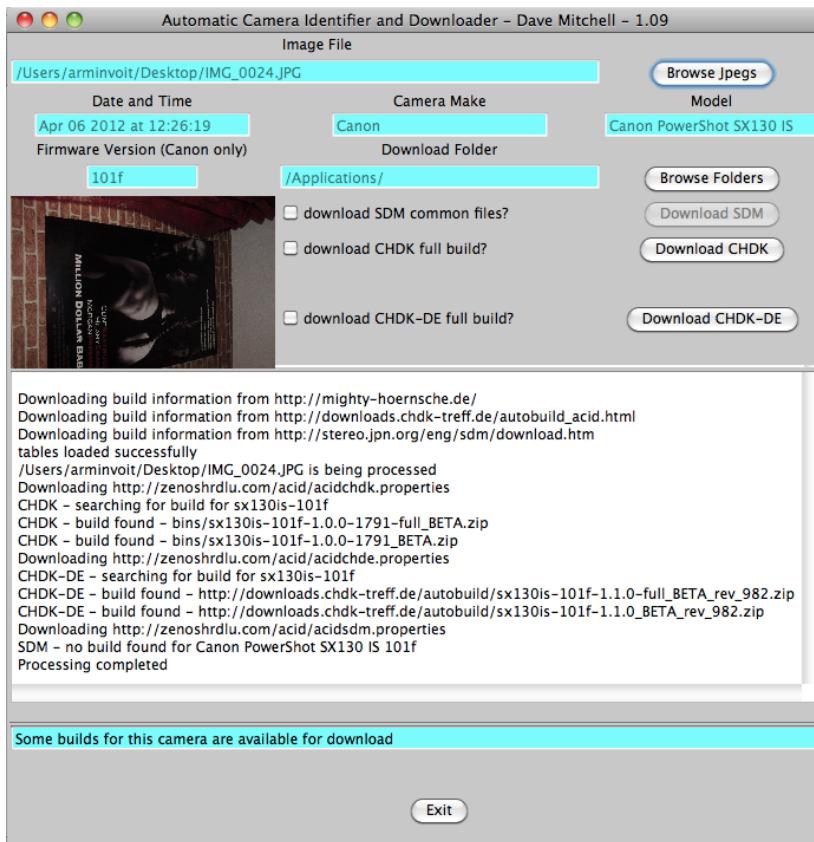


Abbildung 10: ACID nach dem Auslesen der Exif-Daten

Ausgehend von den nun gewonnenen Informationen – im konkreten Fall hat die Kamera die Firmware-Version 101f – konnte die entsprechende CHDK Version heruntergeladen werden und auf die SD Karte gespielt. Jedoch ist das alleinige Kopieren der korrekten CHDK Version auf die SD Karte nicht ausreichend. Damit die Kamera in der Lage ist, den Firmware Aufsatz zu laden, ist es zuvor nötig eine bootfähige SD Karte zu erzeugen. Auch hierfür gibt es mehrere Möglichkeiten. Da ich auf Mac OS gearbeitet habe, habe ich mich hierbei für das Programm SDM Install¹⁸ entschieden. Mit diesem Programm ist es möglich gleichzeitig eine bootfähige Partition auf der SD Karte zu erstellen und CHDK auf der SD Karte zu installieren, so dass hierdurch CHDK auf der Kamera gebootet werden kann. Folgend ein Screenshot der Version 1.24.

¹⁸ SDM: Stereo Data Maker

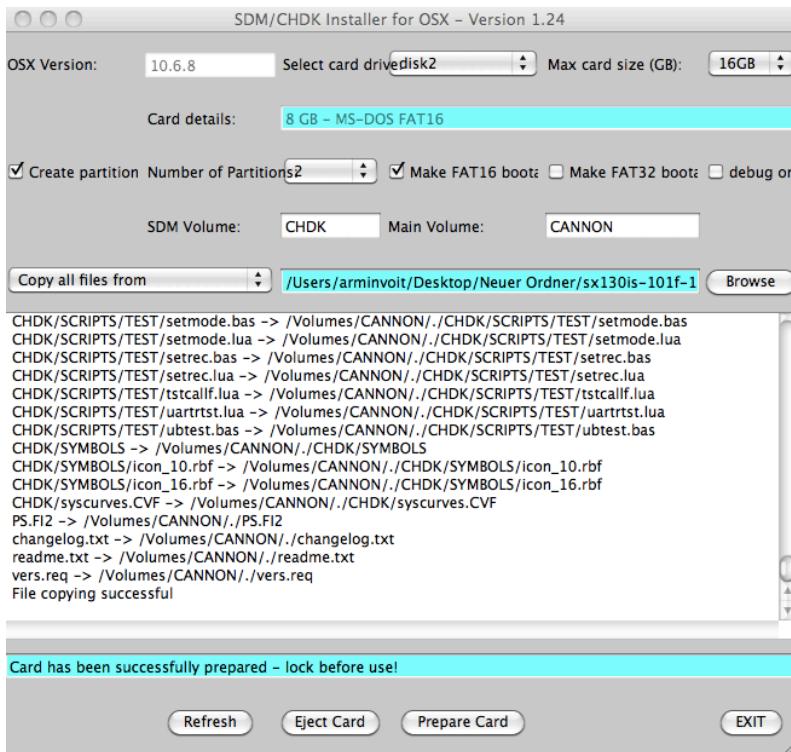


Abbildung 11: SDM/CHDK Installer für OS X

Nach diesen beiden Schritten ist die Kamera nun in der Lage CHDK selbstständig zu booten und anzuwenden. Möchte man CHDK nicht verwenden, es aber auch nicht von der SD Karte löschen, so kann man einfach den SD Lock auf „unlocked“ stellen und die Kamera ist nicht mehr in der Lage CHDK zu laden.

5.3 Auslösen

Die Hauptfunktion und für uns entscheidende Methode ist das Auslösen der Kamera durch ein Script. Dies kann auf mehreren Wegen mit beiden Programmiersprachen uBasic und Lua geschehen. Zu Beginn war die Aufgabe sich mit den Sprachen vertraut machen und CHDK erstmals richtig verstehen. Bei den zahlreichen Funktionen ist es schwierig die richtigen zu finden und so kostet die Einarbeitung viel Zeit und viele

Stunden lesen in Foren und Wikis. Alleine das CHDK Handbuch umfasst 150 Seiten und zeigt eine Fülle von Funktionen.

Es gibt 5 Methoden die Kamera auszulösen. Anbei die verschiedenen Methoden.

Methode 1

```
1 shoot()
```

Methode 2

```
1 click("shoot_full")
```

Methode 3

```
1 press "shoot_half"  
2 sleep 1000  
3 press "shoot_full"  
4 release "shoot_full"  
5 release "shoot_half"  
6 sleep 1000
```

Methode 4

```
1 press("shoot_half")  
2 repeat  
3 until get_shooting() == true  
4 press("shoot_full")  
5 release("shoot_full")  
6 release("shoot_half")  
7 repeat
```

Methode 5

```
1 press("shoot_half")  
2 repeat  
3 until get_shooting() == true  
4  
5 for i=1 , 10 do  
6  
7 click("shoot_full_only") -- Befehl nur in CHDK-DE verfügbar  
8 sleep(2000)  
9  
10 end  
11
```

Methode 5

```
12 release("shoot_half")
13 repeat
14   until get_shooting() ~= true
```

Die erste und zweite Methode unterscheiden sich kaum, außer das Methode 2 mit 10 ms etwas schneller ist bei uBasic. Bei beiden Methoden wir eine komplette Auslösung über einmessen, fokussieren bis hin zum auslösen durchgeführt. Die dritte Methode eignet sich zum Einfügen von Befehlen nach dem Fokussieren. Messergebnisse können berücksichtigt werden! Ebenso eignet sich Methode vier nur das hier solange gewartet wird, bis die Kamera mit dem Messen und Fokussieren sowie dem Shooting fertig ist. Bei Methode 3 wurde dies durch eine Wartezeit gelöst. Dies wär Fehleranfälliger da sich Probleme z.B. durch zu kurze oder zu lange Wartezeiten ergeben könnten. Die gezeigte fünfte Methode zeigt einen Sonderfall, eine sogenannte Intervall-Aufnahme. Diese eignet sich besonders falls eine hohe Verarbeitungsgeschwindigkeit gewünscht ist da zwischen den sehr kurzen Intervallabständen nicht erneut eingemessen werden muss.

Wichtig ist das man berücksichtigt, dass die komplette Auslösung einfach etwas Zeit beansprucht und erst wenn das „Shooting“ abgeschlossen wurde man weitere Befehle im Zusammenhang mit der Auslösung abschließen kann. Dies wird unter Methode vier vollständig und bei Methode 3 bedingt beachtet. Einstellungen wie z.B. die ISO-Werte, Blende oder Verschlusszeit können durch diverse Möglichkeiten für die Methoden beeinflusst werden. Jedoch muss hier berücksichtigt werden, dass nicht jeder Wert bei jeder Bedingung einsetzbar ist. Es gibt zwei Möglichkeiten Einfluss zu nehmen, vor und während des „Shootings“. Vor dem Shooting alles festzulegen ist einfach und unkompliziert, anders als während des Shootings. Hier ist der Vorteil, dass von der Kamera gemessene Werte beeinflusst werden können. Jedoch kann es auch zu Fehlern in bestimmten Situationen kommen.

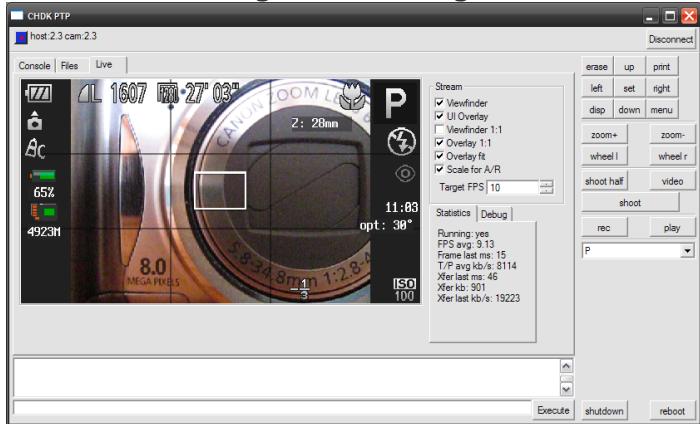
5.4 Konsolensteuerung der Kamera

Eine weitere Möglichkeit neben dem Laden von Scripts auf der Kamera ist eine Steuerung über einen PTP-Klienten. Nach einer Installation von einem alternativen Treiber (Download unter: <http://sourceforge.net/projects/libusb-win32/>) ist es möglich in einem PTP-Konsolen-Fenster Verbindung zu einer Kamera aufzubauen. Eine Statuszeile liefert die Information ob eine Verbindung besteht. Die normalen USB-Funktionen sind nach der Installation der alternativen Treiber nicht mehr möglich. Die Kamera wird nicht erkannt! In der Konsole können ausschließlich Lua Befehle verwendet werden und ermöglichen so eine Fernsteuerung der Kamera. Durch eine grafische Oberfläche können Lua-Codes direkt in der Konsole eingegeben werden. Es ist daher möglich eine Kamera individuell anzusprechen und ist nicht von dem installieren eines vorgefertigten Scripts abhängig. Dies hat zu Folge, dass man bedeutend flexibler ist und so auf bestimmte Situationen wie z.B. Änderung der Lichtverhältnisse sofort reagieren kann. In dem Bereich der PTP-Klienten ist momentan ziemlich Bewegung und viele interessante Projekte sind gerade dabei sich zu entwickeln. Hauptsächlich von Usern des größten deutschen CHDK-Forums „<http://forum.chdk-treff.de/>“.

Vor wenigen Wochen wurde eine neuer PTP-Klient veröffentlicht bei dem Live-Bild-Übertragung möglich ist. Die Entwicklungsarbeiten sind mittlerweile so weit fortgeschritten, dass dieses Angebot für die meisten CHDK-unterstützten Kameras zur Verfügung steht. Das Programm ist als grafische Benutzeroberfläche und als Kommandozeileninterpreter für Windows und Linux erhältlich. Es handelt sich bei diesen Programmen meist um Hobby Projekte die teilweise noch zahlreiche Bugs aufweisen. Wir traune dem PTP-Klienten eine Menge zu und denken eine Steuerung

von mehreren Kameras wäre in einem zukünftigen Projekt zu realisieren und eine elegante Lösung für die Steuerung. Die Hauprobleme dürften die

Treiber sein, die bisher leider nur eine Kamera erkennen.



5.5 Simultane Triggerung

Abbildung 12: PTP Client

Hauptgrund für die Verwendung von CHDK für unser Projekt war das Erreichen einer simultanen Triggerung, sprich einem simultanen Auslösen aller Kameras. Um dies zu bewerkstelligen, mussten wir alle Kameras mit dem gleichen Skript ausstatten. Unser CHDK Skript, welches wir letztlich auch bei der Projektpräsentation verwendet haben, ist folgendermaßen aufgebaut.

Zu Beginn werden alle Kameras und CHDK gestartet und in den Aufnahmemodus geschaltet. In diesem Zustand warten die Kameras darauf, das, vorher manuell an der Kamera angewählte, Skript zu starten. Alle Kameras sind über USB Kabel und USB Hubs an einer Stromquelle, in unserem Fall der USB Port des PCs, verbunden, um so gleichzeitig ein Stromimpuls zu erhalten. Wird nun das USB Kabel, welches mit allen Kameras verbunden ist, in den PC eingesteckt. Somit bekommen die Kameras einen Stromimpuls. Mit dieser positiven Spannungsflanke startet das CHDK Skript. Die Kameras beginnen nun alle zu fokussieren und verharren daraufhin in dieser Position. Da der Fokus bei allen Kameras unterschiedlich schnell funktioniert, sollte man hier mindestens 2

Sekunden warten. Wird nun das USB Kabel wieder herausgenommen, führt das dazu, dass der Stromimpuls wieder verschwindet und eine negative Flanke bei den Kameras erscheint. Diese negative Flanke bringt die Kameras in ihrem Skript zum nächsten Schritt und bewirkt, dass alle Kameras auslösen. Da die negative Flanke bei allen Kameras simultan auftritt, wird auch das Foto simultan ausgelöst.

5.6 Synchronitätsnachweis (ohne Arduino)

Nachdem wir das zeitgleiche Auslösen der Kameras realisiert hatten, haben wir uns nun an die Arbeit gemacht, nachzuweisen, wie genau unsere simultane Triggerung wirklich funktioniert. Dazu haben wir erst einen Versuchsaufbau mit zwei Kameras und anschließend einen mit allen Kameras durchgeführt.

Beim ersten Versuch wollten wir herausfinden, ob die Kameras prinzipiell synchron sind. Dazu waren zwei Kameras ausreichend. Denn wären hier schon Probleme aufgetreten, so wäre das Ergebnis mit acht Kameras nicht anders verlaufen. Wir haben nun also zwei Kameras so aufgestellt, dass sie beide auf eine Stoppuhr¹⁹ gerichtet waren. Das, mit beiden Kameras, geschossene Foto würde nun beweisen, ob die Kameras synchron sind oder nicht. Die beiden Kameras zeigten das exakt gleiche Bild auf die hundertstel Sekunde genau an und somit war bewiesen, dass die Kameras synchron ausgelöst wurden. Jedoch hatten wir eine Sache nicht berücksichtigt. Das Handydisplay arbeitet lediglich mit einer Bildwiederholungsrate von 30 Hertz. Somit war unser Messergebnis lediglich auf ~34ms genau.

Das langte uns allerdings noch nicht und somit suchten wir nach Möglichkeiten unser Messergebnis noch genauer darzustellen. Bei der

¹⁹ Darstellung auf einem Samsung Galaxy S2

zweiten Durchführung, die Synchronität nachzuweisen, besorgten wir uns einen Desktopmonitor, der mit einer Bildwiederholungsrate von 60 Hertz arbeitete. Für diesen Test verwendeten wir alle acht Kameras. Auch hier richteten wir wieder die Kameras auf die Stoppuhr²⁰. Das Ergebnis war das gleiche wie zuvor mit den zwei Kameras. Auch hier zeigten alle Kameras die exakt gleiche Zeit auf der Stoppuhr an. Somit hatten wir nachgewiesen, dass unsere Kameras mindestens auf 17ms genau auslöst.

Spätere Test bis hin zu 1ms Genauigkeit, wurden dann zusätzlich von Christian Siemer mittels Arduino im späteren Projektverlauf durchgeführt.

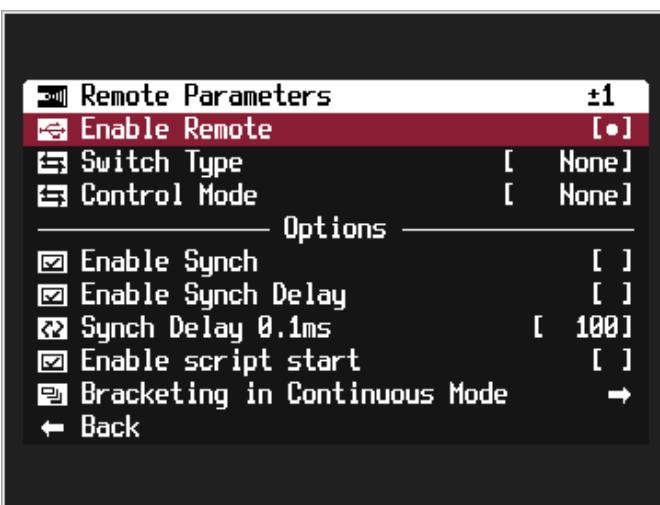
5.7 Datenübertragung

Das Thema Datenübertragung haben wir ebenfalls bearbeitet. Anfänglich war unser Ziel, einen automatisierten Ablauf mit den Kameras mit Hilfe eines Skripts zu erzeugen. Nach vielen Recherchen im Internet und zahlreichen Skriptvarianten, haben sich zwei entscheidende Probleme herauskristallisiert. Das erste Problem war, beim Versuch alle acht Kameras an den PC anzuschließen, dass dieser stets sobald er eine Kamera erkannt hat, keine weiteren mehr gesucht hat und sich quasi mit einer einzigen zufrieden gegeben hat. Um dieses Problem zu lösen, hätte man Änderungen am USB Treiber vornehmen müssen, da es prinzipiell möglich ist alle acht Kameras gleichzeitig zu erkennen, jedoch hätte eine solche Treibermodifizierung den zeitlichen Rahmen bei Weitem gesprengt. Das zweite Problem, welches sich aufgetan hat war, dass es zwar prinzipiell möglich war ein Skript zum Auslösen der Kameras und ein Skript zur Bildübertragung in einem gemeinsamen Skript zu kombinieren, jedoch gab es einen entscheidenden Faktor der sich zu diesem Vorhaben antagonistisch verhielt. Eine einzelne Einstellung in der Kamera verhindert

²⁰ Darstellung auf einem Fujitsu Siemens Monitor

nämlich, dass das Auslösen und Übertragen in einem „Rutsch“ möglich ist. Möchte man die Kameras auslösen, so muss man direkt in den CHDK Einstellungen auf der Kamera unter dem Punkt *Skripteinstellungen* den Punkt *Fernbedienung* deaktivieren. Möchte man nun das geschossene Bild auf seinen Rechner ziehen, so ist dies nur möglich, wenn man den eben beschriebenen Punkt aktiviert.

Allein über das Skript hat man keine Möglichkeit auf diesen Punkt zuzugreifen. Aus diesen Gründen habe wir uns dazu entschlossen, die Bildübertragung, nach dem Schießen der Fotos semiautomatisch ablaufen zu lassen. Dabei hatten wir sukzessiv für jede Kamera den Punkt *Fernbedienung* aktiviert und waren in der Lage die Bilder der Kameras herunterzuladen ohne dabei ein uneffektives Entnehmen der Speicherkarte vorzunehmen.



6. Schaltung

Abbildung 13: CHDK Menü

Eine wichtige Aufgabe für ein erfolgreiches Projekt war die Schaltung unserer Kameras. Das Budget spielte eine große Rolle und musste bei dieser Aufgabe im berücksichtigt werden. Die Schaltung sollte mit einfachen finanziellen Mitteln realisiert, jedoch die Möglichkeit für eine

spätere Weiterentwicklung besitzen. Bei Schaltungen gibt es beliebig viele Möglichkeiten. Wichtig für uns war, dass die Kameras einen Micro-USB-Anschluss besitzen und sich daher USB anbietet unseren 3-5V Impuls, den wir zum auslösen der Kameras benötigen, zu übertragen.

6.1 Aufbau

Der Aufbau unserer Schaltung kurz illustriert.

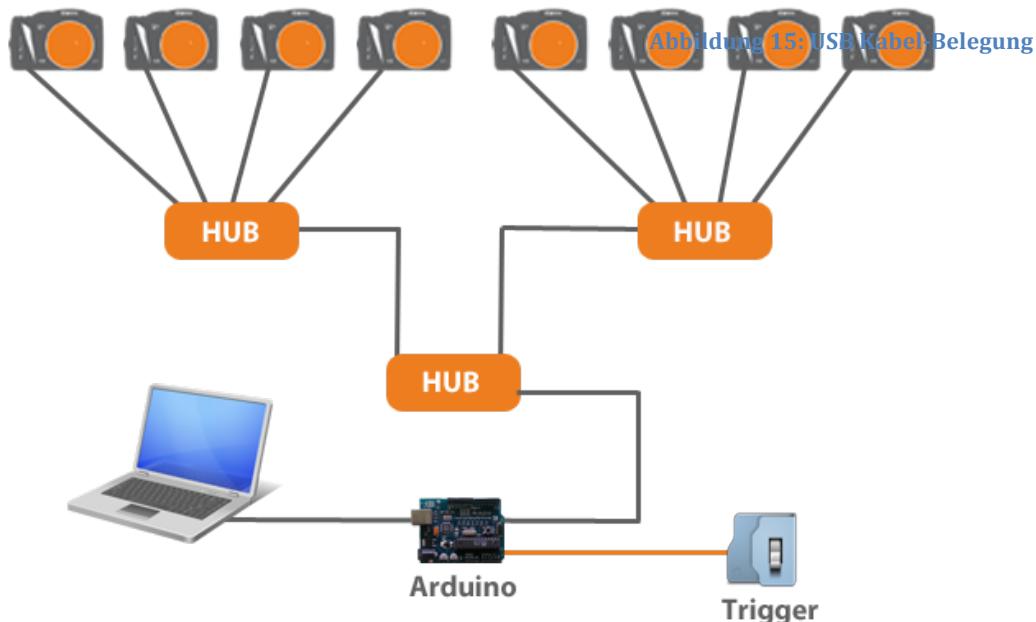
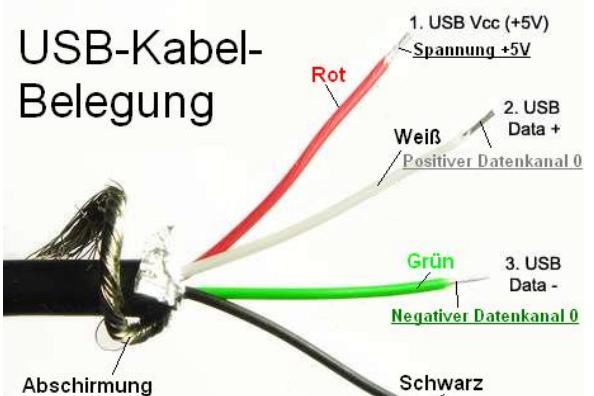


Abbildung 14: Schaltungsaufbau

Nach Anfänglichen Überlegungen haben wir uns dazu entschieden jeweils vier Kameras über ein USB-Hub zu verbinden. Bei dieser Lösung haben wir den Vorteil, dass die Schaltung sich leicht aufbauen und demontieren lässt. So können alle Teile auch weiterhin benutzt werden und das löten entfällt. Zudem muss



berücksichtigt werden, dass wir nicht nur die 3-5V mit der Schaltung übertragen sondern auch Daten der Kameras übermitteln wollen. Hierzu ist ein grundsätzliches Verstehen von USB Notwendig. Es gibt insgesamt 4 Leiter, wobei 2 Datenkabel (weiß + und grün -) sowie rot das die 5V Spannung überträgt und Schwarz die Masse. USB-Kabel können (je nach Bauart) mit 100 mA bzw. 500 mA belastet werden.

Geräte mit einer Leistung von bis zu 2,5 W können per USB-Anschluss betrieben werden. Ein USB-Hub verteilt ein USB-Signal an mehrere Ports. Passive Hubs (Bus-Powered-Hubs) können ihren Strom aus dem Bus selbst beziehen, dies gilt für die meisten handelsüblich USB-Hubs mit bis zu sieben Downstream-Ports.

Aktive Hubs (Self-Powered-Hubs) verfügen über eine eigene Stromversorgung z.B. über ein Netzteil und **jedes** daran angeschlossene Gerät kann die 500 mA Strom beziehen. Passive Hubs können maximal an alle

Geräte **zusammen** nur 500 mA



Abbildung 16: Versuchsaufbau

übertragen. Dies ist ein wichtiger Faktor der bei der Planung der Schaltung berücksichtigt werden muss. Wir verwenden hybride Self – und Bus-Powered-Hubs die ohne externe Stromversorgung passiv sonst aktiv verwendet werden können. Jeweils 4 Kameras sind an ein hybrides Hub angeschlossen, eine Verbindung von beiden Hubs geht in ein weiteres Hub, welches an einen Arduino-Mikrocontroller angeschlossen wird. Dies ist das zentrale Bauteil mit dem eine Vielzahl kreativer Möglichkeiten realisiert werden können. Im nächsten Abschnitt wird genauer auf diesen Mikrocontroller eingegangen. Zahlreiche Schaltungen bzw. Trigger könnten so angesteuert und ausgelöst werden. Wir haben uns für einen Taster entschieden da hier gezielt ohne Komplikationen oder äußere Einwirkungen ausgelöst werden kann. Als Zusatz haben wir uns vorbehalten, die Schaltung durch ein akustisches Signal wie z.B. ein Handschlag auszulösen. Der Arduino-Mikrocontroller bezieht seinen Strom aus einem Notebook wie in der Zeichnung angedeutet. Es könnten auch andere Stromquellen wie z.B. Batterien oder ein externes Netzteil verwendet werden. Da wir den Arduino durch ein Notebook programmieren, bietet sich an Ihnen auch direkt als Stromquelle zu nutzen. Die gesamte Schaltung lässt sich in kürzester Zeit auf- und abbauen und

für die mobile Nutzung verpacken. Die gesamten Kosten der Schaltung bleibt überschaubar mit ca. 35€, wobei der Arduino das teuerste Bauteil mit ca. 20€ zu Buche schlägt.

6.2 Arduino:

Unter Arduino versteht man sowohl die Open Source quelloffene Software wie auch die Hardware. Die Hardware besteht aus einem Mikrocontroller mit analogen sowie digitalen Ein- und Ausgängen. Die Entwicklungsumgebung beruht auf der von Processing und Wiring. Processing konnten wir bereits im dritten Semester kennen lernen. Aus diesem Grund konnte sich schnell eingearbeitet und zurechtgefunden werden.

Die Arduino-Plattform wurde speziell für Künstler, Designern, Bastlern und anderen Interessierten entwickelt um Personen die nicht besondere Erfahrung mit Elektrotechnik haben den Zugang zu Mikrocontrollern zu

erleichtern. So gibt es vielseitige Anwendungsgebiete wie z.B. zur Steuerung von interaktiven Objekten oder zur Interaktion von Softwareanwendungen mit dem Menschen. Der Kreativität sind keine Grenzen gesetzt, es gibt zahlreiche Erweiterungen wie z.B. ein Wireless-Shield, so dass auch komplexe Themen mit einem Arduino Mikrocontroller umgesetzt werden können.



6.3 Technische Daten:

Der Mikrocontroller basiert auf einen Atmel AVR-Mikrocontroller aus der MegaAVR-Serie. Bei unserem verwendetet Arduino Uno wurde ein ATmega328 verbaut. Versorgt wir die Hardware wie schon kurz angedeutet über eine externe Spannungsquelle oder oder oder über USB. Zudem verfügt der Arduino-Mikrocontroller über einen 16 MHz-Quarzoszillator. Programmiert wird dieser über eine serielle Schnittstelle. Der Mikrocontroller ist mit einem Boot-Loader vorprogrammiert, wodurch die Programmierung direkt über die serielle Schnittstelle ohne externes Programmiergerät erfolgen kann.

Unser Arduino verfügt über 14 digitale Input/ Output Pins. Davon können insgesamt 6 als PWM-Ausgänge verwendet werden. Zudem gibt es 6 analoge Eingänge, ein USB-Anschluss, ein Stromanschluss, ein ICSP-Header sowie eine Reset-Taste. Auf dem Board befindet sich ein ATmega16U2 Mikrocontroller als USB-Seriell-Konverter. Diese neue USB Anbindung besitzt ein paar technische Neuerungen:

[Abbildung 19: Arduino Chip \(schräg\)](#)

Der ATmega16U2 lässt sich auch umprogrammieren, man hat dadurch neue Anwendungsmöglichkeiten geschaffen. Das Board kann so vom Rechner als Keyboard, Joystick, MIDI Interface erkannt werden. Außerdem ist kein Treiber mehr notwendig und das Board wird vom Rechner als "Arduino" erkannt, da die Boards jetzt eine eigene USB ID bekommen

haben.

Zusammenfassung des Arduino UNO

Mikrocontroller

ATmega328

Betriebsspannung

5V



Eingangsspannung (empfohlen) 7-12V

Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Eingänge	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB
SRAM	2 KB
EEPROM	1 KB
Quartzfrequenz	16 MHz

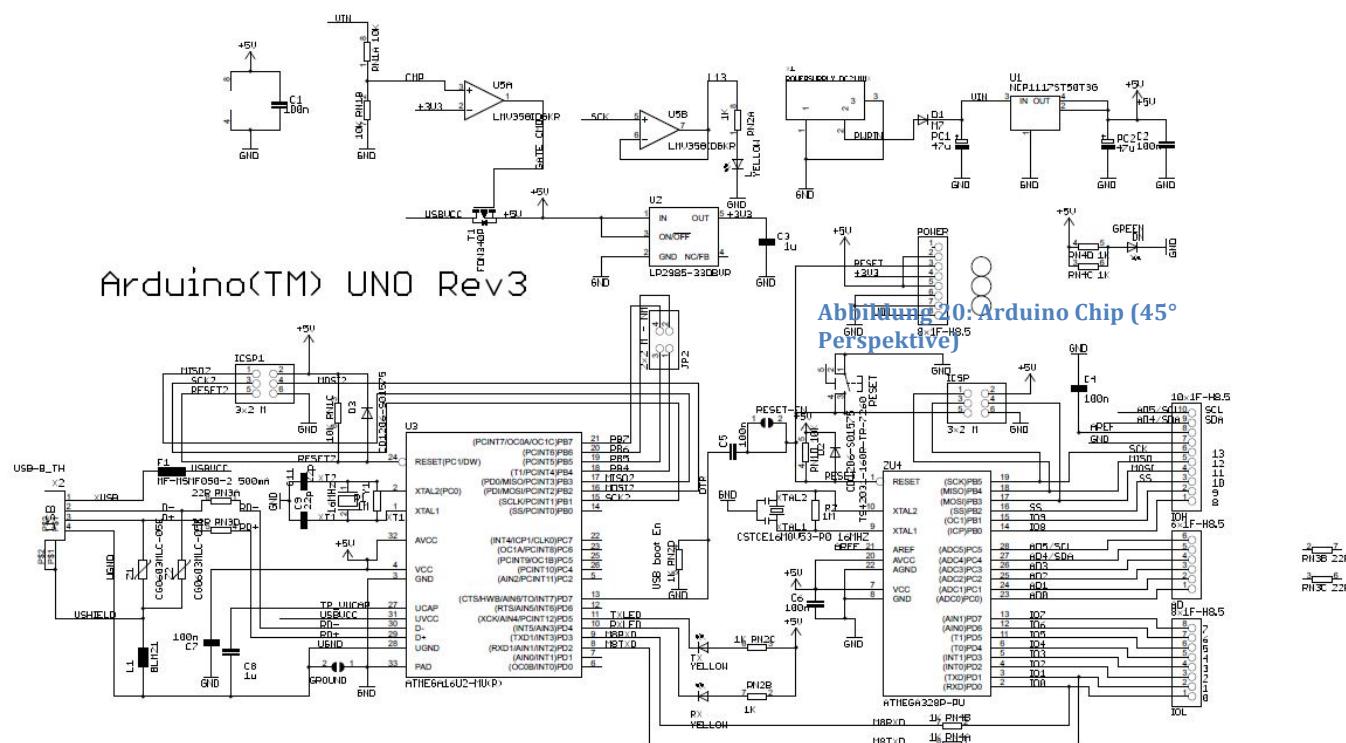


Abbildung 21: Arduino (TM) UNO Rev3

Das Board sollte über eine externe Spannungsversorgung mit 6 bis 20 Volt betrieben werden, der empfohlene Bereich liegt zwischen 7-12V. Sollte die Versorgung weniger als 7 Volt sein, kann es dazu führen dass der 5V Pin weniger als 5V liefert und so das Arduino-Board instabil wird. Bei über 12V kann es passieren das der Spannungsregler überhitzt und so das Board

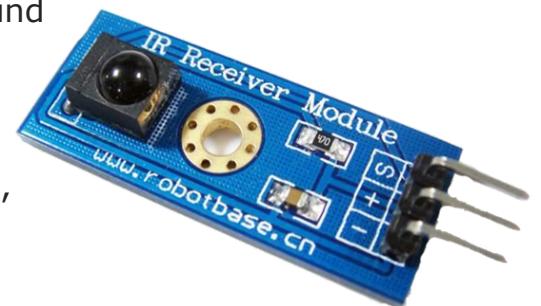
beschädigt

wird.

Für unser Vorhaben ist besonders der **5V Pin** wichtig. Dieser Pin gibt eine geregelte 5V Spannung aus dem Regler des Arduino, falls das Board über eine externe Stromquelle oder USB mit Strom versorgt wird. **GND** ist die Masse und wird im Regelfall mit dem Potential 0 Volt definiert. Jedes der 14 Pins kann als Eingang oder Ausgang benutzt werden. Durch `pinMode ()`, `digitalWrite ()`, und `digitalRead ()`-Funktionen können diese direkt angesprochen werden. Jeder Pin arbeitet mit 5V und liefert oder erhält maximal 40 mA. Zudem verfügen sie über einen internen Pull-up Widerstand von 20-50 kOhm. Einige Pins haben zusätzliche Funktionen z.B. ist an Pin 13 eine eingebaute LED. Daher eignet sich dieser um eine Schaltung zu testen da man sofort ein visuelles Feedback erhält.

Zudem bietet der Arduino-Mikrocontroller einen USB-Überstromschutz, der den Computer vor über 500 mA schützt. Die Sicherung trennt die Verbindung bis die kurz- oder Überlast entfernt ist.

Ein weiterer Vorteil des Arduino ist seine schnelle und unkomplizierte Erweiterungsfunktion. Es lassen sich zahlreiche Trigger und Sensoren anschließend wie z.B. Schallsensoren, IR-Sensoren, Laser Schranken, Wireless Module usw.



The screenshot shows the Arduino IDE interface. The title bar says "fertigeSchaltung | Arduino 1.0.1". The code editor contains the following sketch:

```
if ((zaehler == 9) & (buttonState == HIGH)) {  
    digitalWrite(ledPin, HIGH);  
    delay(10);  
    digitalWrite(ledPin, LOW);  
}  
else{}  
  
delay(50); // wartet eine halbe Sekunde, für die Optik zum  
binwert = byte(zaehler); // schreibt die aktuelle Zahl in  
Serial.println(binwert,BIN); // gibt den aktuellen binärwert  
for (int n=0; n <4; n++){ // zählt von 0 bis 3 (4 Schritte)  
    if (bitRead(binwert,n)==1){ // wenn das aktuelle bit eine 1 ist  
        digitalWrite(n+2,HIGH); // setze den entsprechenden PIN  
    }  
    else{  
        digitalWrite(n+2,LOW); // ansonsten setze den PIN auf 0  
    }  
}
```

The status bar at the bottom right shows "Upload abgeschlossen." and "Binäre Sketchgröße: 3.130 Bytes (von einem Maximum von 32.256 Bytes)".

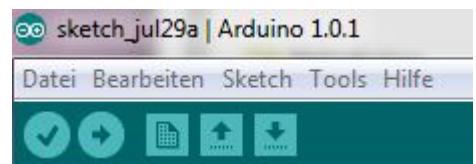
6.4 Programmierschnittstellen

Programmiert wird der Mikrocontroller über eine integrierte Entwicklungsumgebung in Form einer

plattformunabhängigen Java-Anwendung. Die Entwicklungsumgebung basiert auf Processing und bringt einen Code-Editor mit. Als Compiler wird gcc eingebunden, sowie eine große Anzahl Arduino-Libraries und die avr-gcc-Library. Programmiersprachen sind C sowie C++ die durch den Gebrauch der Libraries vereinfacht wird. Auf der Hauptseite www.arduino.cc kann jeweils für Windows, MacOS X sowie Linux die passende Software herunterladen werden. Nach dem Entpacken des Installationsarchivs wird der USB-Treiber installiert. Anschließend kann die Arduino-IDE (Entwicklungsumgebung) gestartet werden. Durch Getting Started Tutorials wird die Installation sowie die Einrichtung erklärt. Vorgefertigte Beispiele zeigen dir ersten Programme und wie man die Programmierung beginnt. Für ein funktionstüchtiges Programm reichen zwei Methoden. **Setup()** wird beim Start des Programms einmalig aufgerufen, nützlich um z.B. Pins als Eingang oder Ausgang zu definieren. **Loop()** – wird durchgehend aufgerufen, solange bis das Arduino-Board ausgeschaltet wird. Hier werden die Funktionen aufgerufen z.B. das Auslösen mit Hilfe eines Tasters.

[Abbildung 24: Arduino Setup](#)

Durch den zweiten Button im Hauptmenü **Upload** (der Pfeil nach rechts) kann der programmierte Code direkt auf den Mikrocontroller übertragen werden. Dies dauert nur wenige Sekunden und wird visuell durch eine blinkende LED angezeigt. So kann relativ leicht Programmiert und sofort an dem Mikrocontroller übertragen werden. Durch den ersten Button (Kreis mit Haken) kann der Code überprüft bzw. debugged werden ohne ihn auf den Arduino zu übertragen. Die restlichen Buttons dienen ein neues leeres Dokument zu öffnen, ein vorhandenes zu öffnen und der Speicherung.



 A screenshot of the Arduino IDE showing the code editor. The title bar says 'Taster | Arduino 1.0.1'. The code editor displays the same code as the previous screenshot, but the title bar and some UI elements are different. The code is:


```
const int buttonPin = 3;
const int taster = 13;
int buttonState = 0;

void setup() {
pinMode(taster, OUTPUT);
pinMode(buttonPin, INPUT);
}

void loop() {
buttonState = digitalRead(buttonPin);
if (buttonState == HIGH) {
digitalWrite(taster, HIGH);
}
else {
digitalWrite(taster, LOW);
}
}
```

6.5 Code:

Der in der Abbildung rechts gezeigt Code war die programmierte Anfangsbasis für die Auslösung über einen Taster. Zu Beginn werden die Variablen deklariert. Mein Taster Ausgang liegt hier auf Pin 13 und der Button der gedrückt wird auf 3. In der Setup() Methode wird kurz beschrieben welches Signal ein Ausgangssignal und welches ein Eingangssignal ist. Wir benötige beide da wir auf der einen Seite etwas auslösen und dann etwas übertragen wollen und zwar unseren 3-5V Impuls. In der loop() Methode wird mit einer if-else Abfrage gearbeitet. Falls der Taster gedrückt wird soll der das Ausgangssignal auf HIGH gesetzt werden, ansonsten LOW also aus. So wird gewährleistet dass nur wenn der Taster wirklich gedrückt wird die 5V übertragen werden. Zu Beginn musste ich mich erstmals in die Programmierung einarbeiten. Den Taster habe ich erst außen vor gelassen und Versucht eine LED auf Pin 13 anzusprechen. Über ein Delay kann diese zum Blinken gebracht werden. So habe ich einen ersten Eindruck bekommen wie die Funktionsweise mit OUTPUT, INPUT, LOW,HIGH und das Programmieren des Arduino ablaufen könnten.

6.6 Schaltungsaufbau:

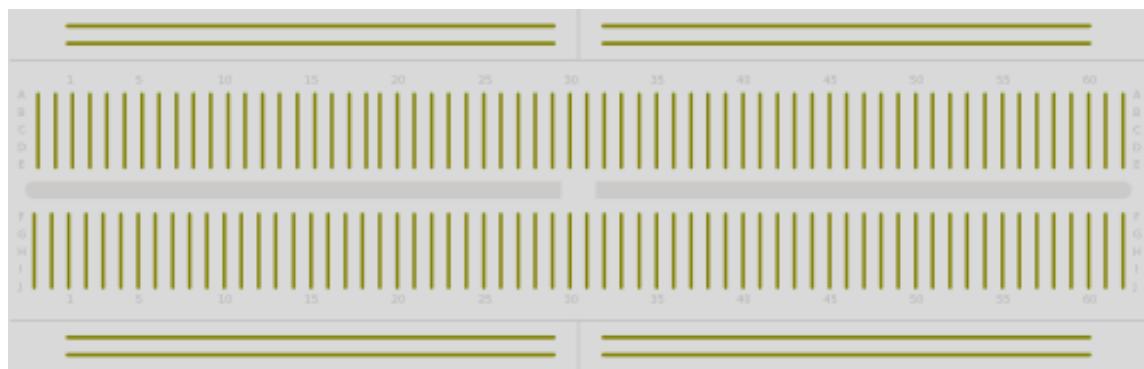
Abbildung 26: Steckplatine

Zu Beginn haben wir leider nur den Arduino Mikrocontroller sowie das USB Kabel erhalten. Leider waren keinerlei Beschreibungen oder andere Informationen enthalten. Aus diesem Grund mussten wir alle Informationen im



Netz suchen und Teile Vorort im Geschäft kaufen. Für den ersten Test haben wir eine Sammlung unterschiedlicher LEDs gekauft um einen Eindruck über das Funktionsprinzip zu bekommen. Durch die ersten gesammelten Erfahrungen und Wissen wurde uns klar, dass der Arduino nicht dazu gedacht ist um direkt an die Platine zu löten sondern über Steckverbindungen die Schaltung aufzubauen. Aus diesem Grund habe ich mir eine Steckplatine sowie Steckbrücken besorgt. Die Steckplatine ist die Grundlage für alle Schaltungen, die man mit dem Arduino aufbauen kann. Ohne zu löten kann man so schnell beliebige Schaltungen zusammenstecken und problemlos verändern oder erweitern. Die Lücke in der Mitte der Steckplatine hat genau den passenden Abstand, um darauf ICs (Mikrochips) im sogenannten DIP-Gehäuse zu stecken, deren Anschlüsse man dann gut von Hand verkabeln kann. Da wir leider gar keine Grundkenntnisse in der Elektrotechnik besaßen, war das ganze Gebiet der Schaltungen neu für uns und wir mussten mir Grundlagen selbst beibringen. Aus diesem Grund mussten wir erstmal verstehen wie eine solche Steckplatine verbunden ist. Dies wird in der nächsten

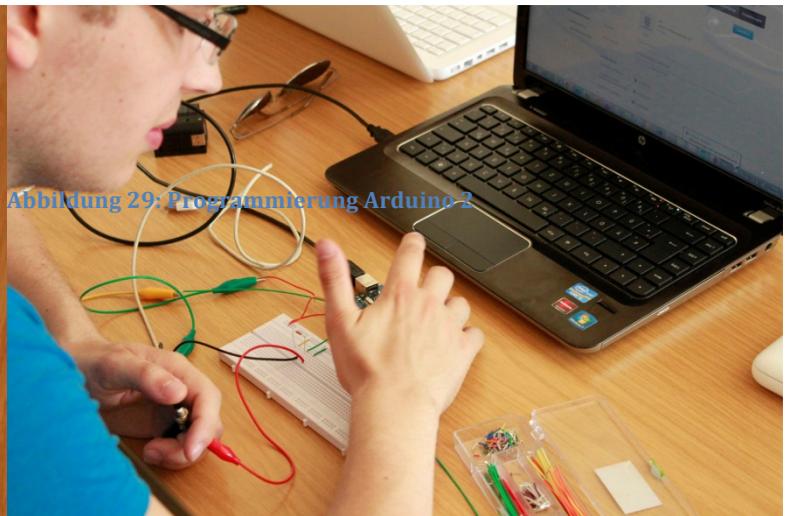
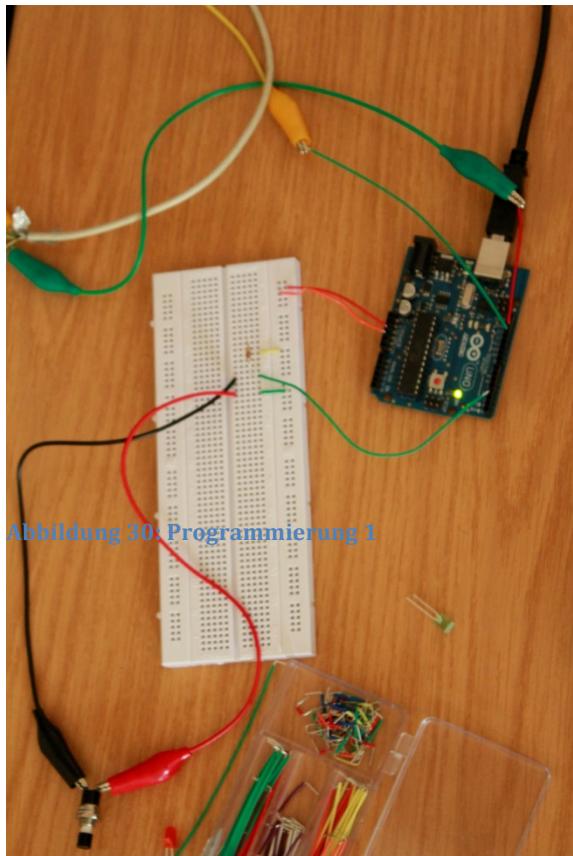
[Abbildung 27](#) verdeutlicht.



Die zu Beginn erstellten Kabel die von Hand verlötet wurde, konnte wir durch Steckbrücken ersetzen. So war es bedeutend einfacher eine Verbindung in den kleinen Löchern der Steckplatine aufzubauen.



Ein ganz anderes Themengebiet waren Widerstände. Wir wussten zwar dass es so etwas gab, nur hatte wir keine Ahnung wie und wann man diese verwendet. Wie man berechnet ob der Widerstand ausreicht oder nicht. Mit einem Strommessgerät ist zudem ein einfaches Nachmessen möglich. Alle Widerstände weisen eine bestimmte Markierung auf, um diese auseinanderzuhalten. Die Anschlüsse einer Leuchtdiode (und anderer gepolter Bauelemente) nennt man Anode und Kathode. Die Kathode wird mit dem „Minuspol“ der Stromquelle verbunden, die Anode mit dem „Pluspol“, beim Arduino entspricht das den Anschlüssen „GND“ (Ground, Masse) und „5V“ (5 Volt) bzw. einem Digitalausgang des Arduinos, der einen entsprechenden Pegel hat. Der Vorwiderstand kann entweder in der Verbindung zur Masse oder zum Pluspol liegen. Ist die Leuchtdiode falsch herum angeschlossen, geht sie nicht kaputt, sondern leuchtet nur einfach nicht.



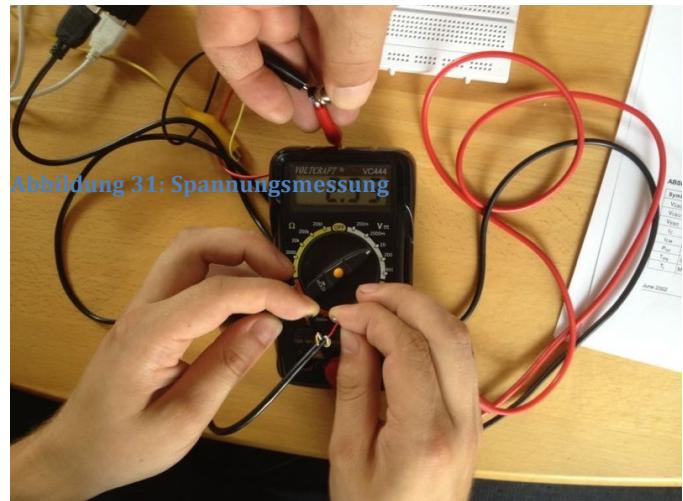
Das Bild zeigt die erste Schaltung zu dem vorigen gezeigten Code. Das mit dem Rot/Schwarzen verbundenen

Kabel zeigt den Taster, der dafür zuständig ist das Signal auszulösen. Der Ausgang hier mit einem Gelben und Grünen Kabel gezeigt ist mit dem +5V Spannung und Masse eines USB Kabel verbunden. In den ersten Tests konnte nur mit einer Kamera getestet werden. Diese wurde jedoch wie gewünscht fokussiert und bei Taster loslassen ausgelöst.

6.7 Probleme:

Leider mussten wir feststellen, dass bei dem Versuch alle 8 Kameras auszulösen, die Kameras weder fokussiert noch ausgelöst haben. Zu Beginn standen uns kein Messgerät zur Verfügung, so dass wir nur durch

Versuche den Fehler lokalisieren musste. Es stellte sich heraus, dass bei Einsatz eines USB-Hubs die Spannung nicht mehr ausreicht bzw. die 3-5V nicht mehr an die Kameras weitergegeben werden. Bei einer späteren Messung mit einem offenen USB Kabel und einem Messgerät hat sich dieser Fehler bestätigt. Es wurden nur exakt 2.93V übertragen, dies hat leider nicht mehr ausgereicht um das CHDK-Script zu aktivieren. Das Problem liegt an den USB-Hubs die anscheinend mehr Strom benötigen als zunächst angenommen.



6.8 Problemlösung:

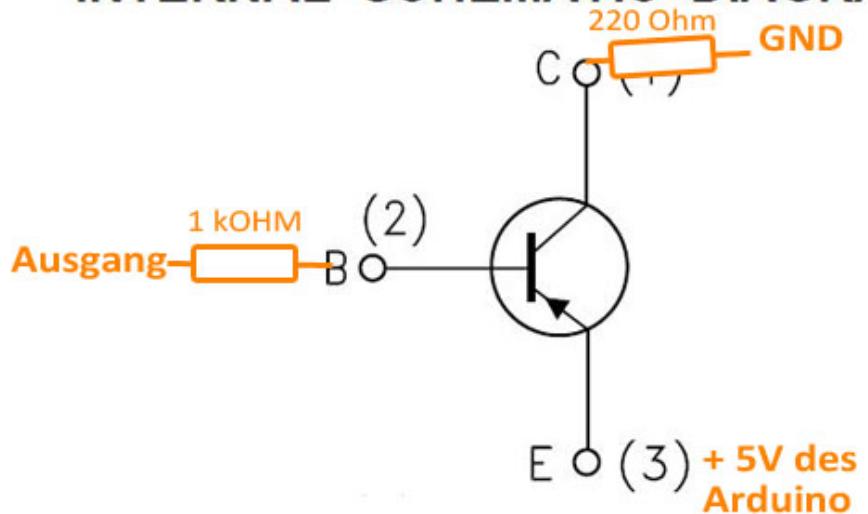
Die große Frage war jetzt wie ich das Signal verstärkt bekomme so dass alle Kameras 3-5V empfangen. Nach anfänglicher Recherche und Hilfe der Projekt betreuenden Professoren



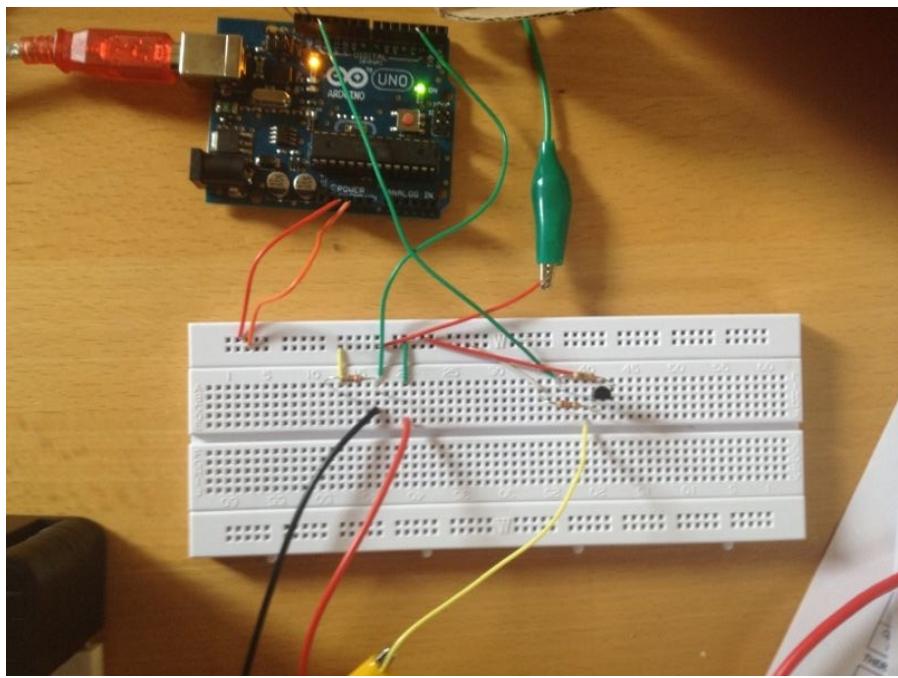
fiel der Begriff „Transistor“. Dieser verstärkt das elektrische Signal ohne dabei mechanische Bewegungen auszuführen. Es gibt eine Fülle von Transistoren, ich entschieden wir uns für ein PNP Transistor **BC327-40** der laut Datenblatt ausreichen sollte. Zudem haben wir noch einen BC557 gekauft der jedoch nicht ideal für mein Vorhaben ist. Vor dem Einbau war eine grundlegende Einarbeitung in das Thema Transistor notwendig. Durch ein Transistor wird das Eingangssignal weitgehend linear und verzerrungsfrei verstärkt. Man kann sagen, es ist ein Schaltungsverstärker, mit dem sich kontaktlos kleine bis mittlere Leistungen schalten lassen. Er ist jedoch mechanischen Schaltern in einigen Punkten unterlegen. Der Widerstand bei einem mechanischen Schalter beträgt 0 Ohm und es entsteht keine Verlustleistung. Anders als bei dem Transistor, der nicht verlustleistunglos schaltet. Als Vorteile kann beim Transistor sagen, dass kein Verschleiß wie bei einem mechanischen Schalter auftritt und die Schaltgeschwindigkeit oder Schaltfrequenz bis in den MHz-Bereich reicht. Für uns war besonders schwierig Schaltbilder zu lesen bzw. zu deuten da wir keinerlei Erfahrungen mit ihnen hatte. Gerade das Schaltbild des Arduino-Mikrocontroller war sehr wichtig und es hat viel Zeit zum nachzuvollziehen gekostet. Für jeden Transistor kann man im Netz ein Datasheet mit allen relevanten Informationen wie z.B. Absolute Maximum Ratings, Schaltbilder usw. finden. So ist es möglich die Schaltung nachzuvollziehen bevor sie überhaupt gesteckt und Wiederstände berechnen sind. Der Transistor hat drei Beinchen, bevor man diese Steckt sollte man wissen was welches tut, um einen Kurzschluss auszuschließen. In einem schematischen Diagramm habe ich die relevanten Ausgänge sowie verwendete Wiederstände eingezeichnet.

Abbildung 32:
Transistor

INTERNAL SCHEMATIC DIAGRAM

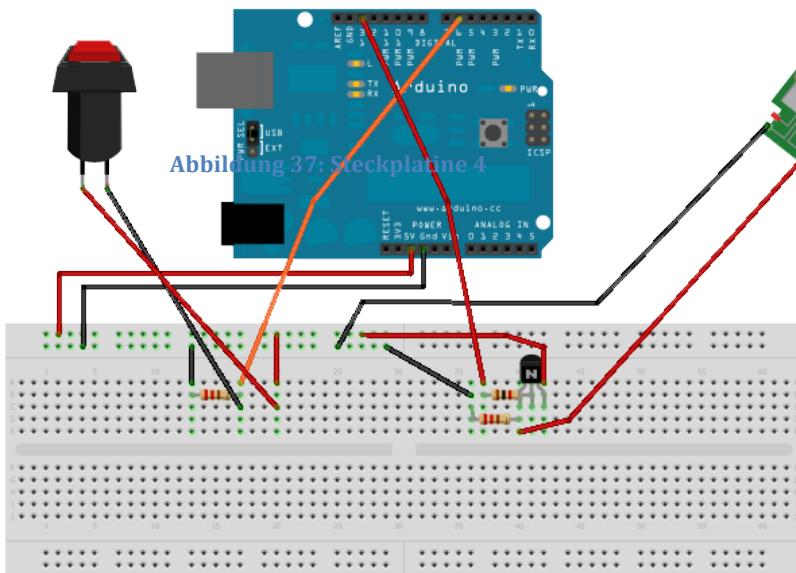


Wichtig ist die Erkenntnis, dass diese Art von Transistoren das komplette Signal invertiert. Daher muss der Code des Arduino-Mikrocontrollers angepasst bzw. geändert werden. Das dieses strukturierte Vorgehen beim Bau einer Schaltung nicht von Beginn so funktionierte ist denk ich normal. Zu Beginn mussten wir uns erst einarbeiten und Fehler machen z.B. haben wir erstmals drauf losgesteckt und direkt ein Transistor das Leben genommen. Wir wollten endlich sehen dass etwas passiert. Relativ schnell merkt man, dass wenn man keine Ahnung von Wiederständen und Co hat und einfach drauf los steckt keinen Erfolg hat. Es passiert einfach nichts und kommt nicht weiter. Nachdem der Transistor richtig eingebaut wurde sieht meine Steckplatine folgendermaßen aus (siehe nächste Seiten). Nach ersten Funktionstests und Messungen nach dem USB-Hub konnte man eine erfolgreiche Übertragung von 4.91V feststellen. Somit ist die Spannung ausreichend um bei allen 8 Kameras das darauf befindliche CHDK Script auszulösen. Dieses Ergebnis konnten wir im Praxistest bestätigen.



A close-up photograph showing a person's hands holding a digital multimeter and a component. The multimeter is black with a digital display showing "4.9" and various analog scales. The person is using red and black test leads to connect the multimeter to the component. A yellow lead is also visible. The background shows a wooden desk with some papers.

Arduino Steckplatine



Arduino Schaltplan

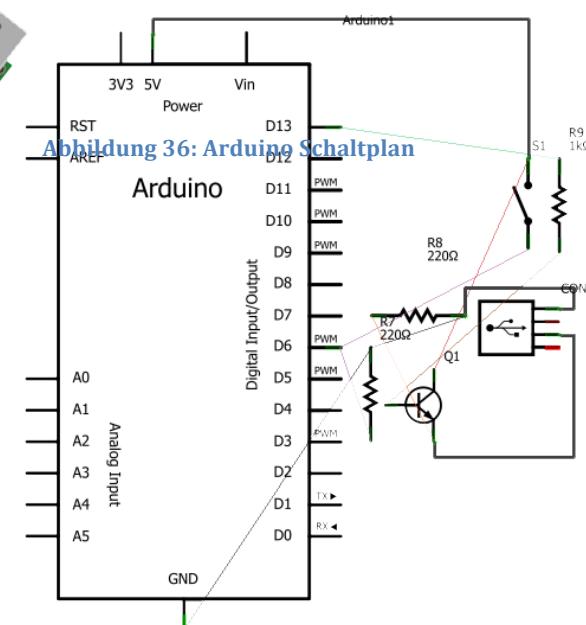


Abbildung 34: Messung der Spannung

Abbildung 35: Steckplatine 3

6.9 Synchronität:

Das nächste große Thema mit dem wir uns beschäftigt haben war die Verifizierung der Synchronität. Da für unser Projekt ein nahezu simultanes Auslösen gewünscht war, ist es zwingend erforderlich die Genauigkeit zu bestimmen und festzuhalten. Aus diesem Grund und der hohen Priorität haben wir uns auf zwei Arten damit befasst.

6.9.1 Verifizierung LCD:

Dieses Vorgehen habne wir schon relativ am Anfang mit den damals zwei vorhandenen Kameras durchgeführt. Nachdem die Schaltung mit dem Arduino-Mikrocontroller einsatzbereit und alle 8 Kameras vorhanden waren wollten wir nun die Verifizierung im endgültigen Stadium ausführen. Hierbei wurden alle 8 Kameras auf dem Rig auf einen Bildschirm ausgerichtet. Dieser Bildschirm zeigt eine Stoppuhr mit einer Millisekunden Anzeige. Ausgelöst sollte über den Taster am Arduino werden.

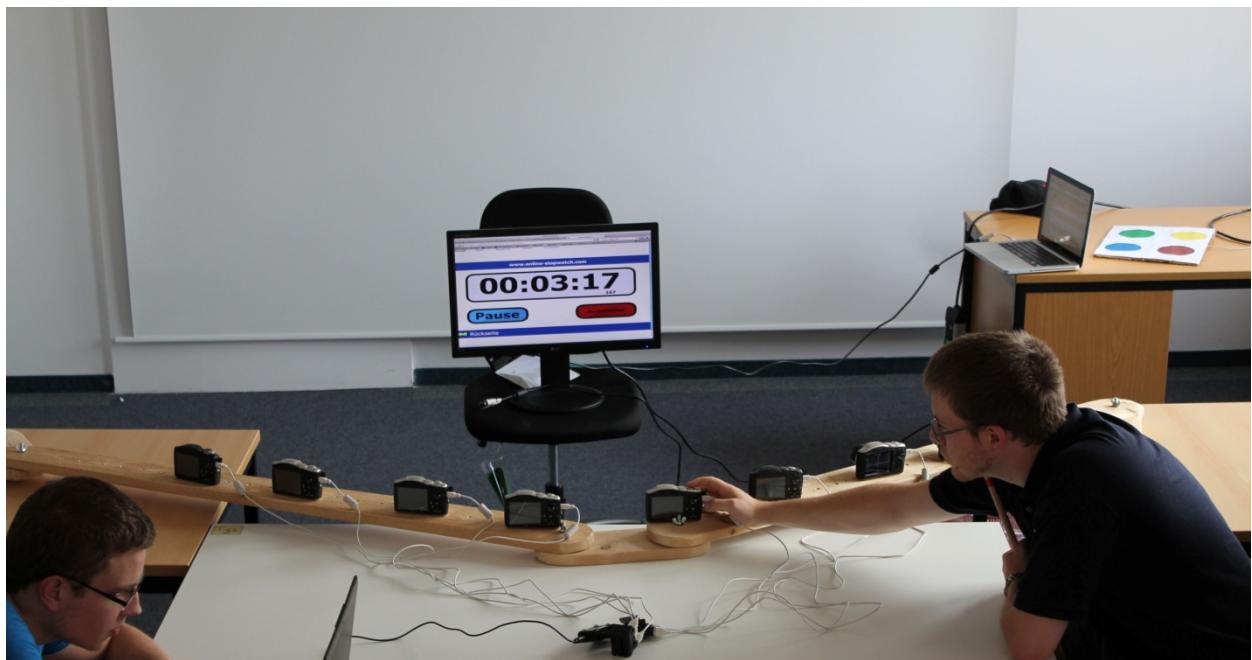


Abbildung 38: Desktopmonitor mit Stoppuhr

Falls bei allen 8 Kameras das gleiche Bild angezeigt wird, können wir die Genauigkeit auf ca. 17ms Genauigkeit aufgrund der 60Hz Bildwiedergabe des Bildschirmes angeben. Zu Beginn hatten wir jedoch ein Problem mit der Genauigkeit. Bei einer schnellen Bewegung, hier eine Drehung um die



Abbildung 39: Asynchronität

eigene Achse konnte man eindeutig extreme Unterschiede erkennen. Wir haben uns auf den unterschiedlichen Bildern um ca. 60 Grad gedreht. Es konnte von keiner synchronen Auslösung gesprochen werden. Nachdem der erste Schock von dem schlechten Ergebnis überwunden war machten wir uns an die Problemfindung. Es stellte sich schnell heraus, dass es etwas mit der Einstellung auf den Kamera zu tun haben muss. Nach einer Recherche wurde klar, dass eine Einstellung Synchronisation-Verzögerung des CHDK auf der Kamera falsch eingestellt wurde. Es gab drei Einstellung ON/OFF/Disable für die Einstellung. Wir sind davon

ausgegangen das OFF die Funktion ausstellt, jedoch war dies nur bedingt der Fall, die richtige Einstellung ist Disable.

Nach Behebung des Problems konnte man schon anhand der Auslösegeräuschs eine merkliche Verbesserung wahrnehmen. Diese Annahme wurde durch die Aufnahmen bestätigt. In dem nächsten Bild wurden alle 8 geschossenen Bilder nebeneinander zur Veranschaulichung gelegt. Auf allen Bildern ist **01:633** ms zu erkennen. Somit kann man sagen die Kameras werden auf mindestens 17ms Genauigkeit ausgelöst. Dies ist in erster Linie ein schönes Ergebnis jedoch nicht besonders Aussagekräftig da 17ms bei einer schnellen Bewegung nicht gerade sehr



Abbildung 40: Nachweis der Synchronität

genau ist. Aus diesem Grund haben wir uns entschlossen, die Genauigkeit der Verifizierung zu verbessern.

6.9.2 Verifizierung Arduino:

Die zweite Möglichkeit der Verifizierung mit der wir uns befasst haben, ist mit Hilfe des Arduino. Wir dachten uns, falls wir schon die Möglichkeit haben mit einem Mikrocontroller zu arbeiten der im Stande ist jeden Pin einzeln anzusprechen, wieso dann nicht auch Nutzen. Die erste Idee war es ein LED-Lauflicht zu programmieren und auf der Steckplatine zu schalten. Mit dem Arduino hat man die Möglichkeit mit einem Delay jede LED auf die Millisekunde zu schalten. Hierzu ist ein grundsätzliches Verstehen einer LED notwendig. Eine LED ist eine Leuchtdiode. Fließt durch die Diode Strom in Durchlassrichtung, so leuchtet sie. Anders als bei einer Glühbirne muss kein Faden oder Draht zum glühen gebracht werden bis sie leuchtet. Bei Strom leuchtet sie Augenblicklich. Aus diesem Grund

sind LED bestens geeignet eine Verifizierung der Genauigkeit durchzuführen.

Zu Beginn haben wir ein Lauflicht programmiert mit 6 LED, diese sollten in einer Schleife nacheinander zum Leuchten gebracht und durch ein Delay gesteuert werden. Mit diesen 6 LEDs kann genau 7 Zustände abgedeckt werden. Bei 6 LEDs wird jedoch die Steckplatine sehr unübersichtlich da jede LED einzeln angesteuert wird. Aus diesem Grund haben wir die Schaltung erweitert und einen Binärzähler gebastelt. So war es uns möglich mit nur 4 LEDs, 2^4 Zustände also insgesamt 16 Zustände abzudecken. Hier war zwar ein bedeutend größerer Programmieraufwand notwendig jedoch wurde dies durch eine übersichtlicherer Schaltung sowie eine höhere Genauigkeit belohnt. Eine zusätzliche Erweiterung ist während des Praktischen Teils der Verifizierung gekommen. Es ist schwierig mit allen 8 Kameras die sehr kleinen Leuchtdioden aufzunehmen. Bei einer schlechten Perspektive wird das Leuchten nicht erkannt. Unsere verwendeten LEDs sind nur von Vorne bzw. Oben zu erkennen ob sie überhaupt leuchten. Außerdem gibt es beim Auslösen über den Taster ein minimales Delay welches ich am liebsten umgehen wollte. Aus diesem Grund haben wir den Programmiercode angepasst und so programmiert, dass wenn der Taster gedrückt wurde bei einer bestimmten vorher definierten Binäranzeige die Kameras erst auslösen. So wird gewährleistet, dass alle Kameras immer zu selben Zeitpunkt ausgelöst werden und das Delay des Schalters nicht so sehr in das Gewicht fällt. So ist es sogar möglich nicht alle Kameras direkt auf die LEDs auszurichten sondern jede Kamera einzeln. Falls bei allen Kameras dasselbe Bild mit der gleichen Binärzahl angezeigt wird so sind die Kameras synchron. Mit dem Delay kann so z.B. bei 100ms angefangen und sich langsam runter bis auf eine Millisekunde getastet werden. Im Folgenden wird kurz auf den Code des Mikrocontrollers eingegangen.

```

const int buttonPin = 7;
int buttonState = 0;
const int ledPin = 13;
boolean Ausloeser;

void setup (){
    Serial.begin(9600); //Serielle Ausgabe einleiten zum
    debuggen
    pinMode(buttonPin, INPUT);
    pinMode(ledPin, OUTPUT);
    for (int loop = 2; loop < 6; loop++){
        pinMode (loop, OUTPUT); // PINs 2-5 definieren
        digitalWrite (loop, LOW); // PINs direkt ausschalten
    }
}
byte binwert; // definiert eine Variable als byte

void loop(){
    digitalWrite(ledPin, LOW);

    for (int zaehler=0; zaehler < 16;zaehler++){ //zählt von 0 bis 15 (also 16 Schritte)
        buttonState = digitalRead(buttonPin);

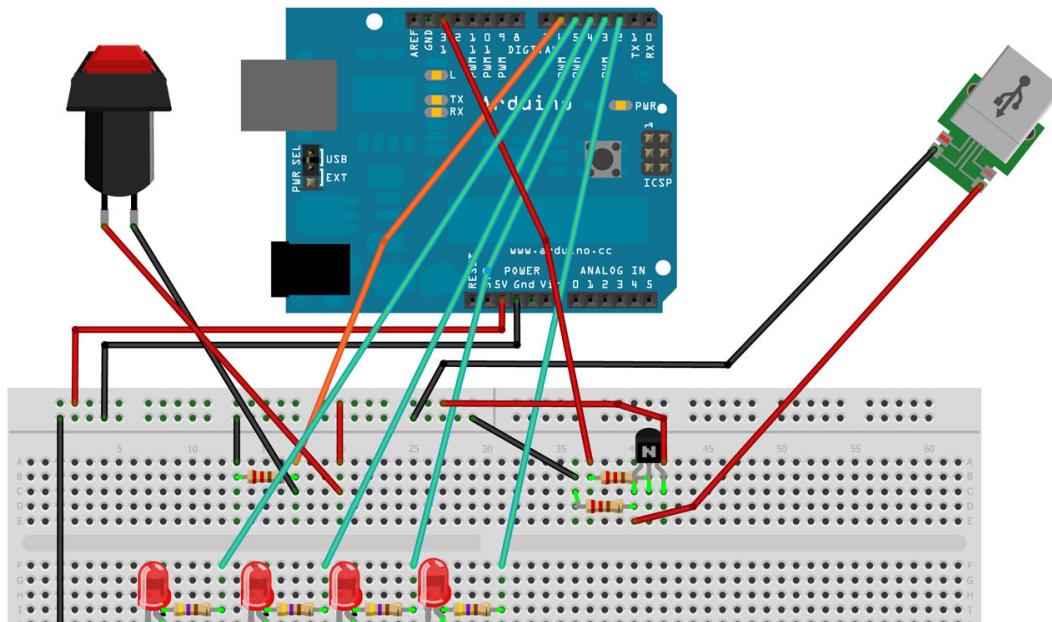
        buttonState = digitalRead(buttonPin);

        /*if (buttonState == HIGH) {
            digitalWrite(ledPin, LOW);
        }
        else {
            digitalWrite(ledPin,HIGH);
        }
    }
}

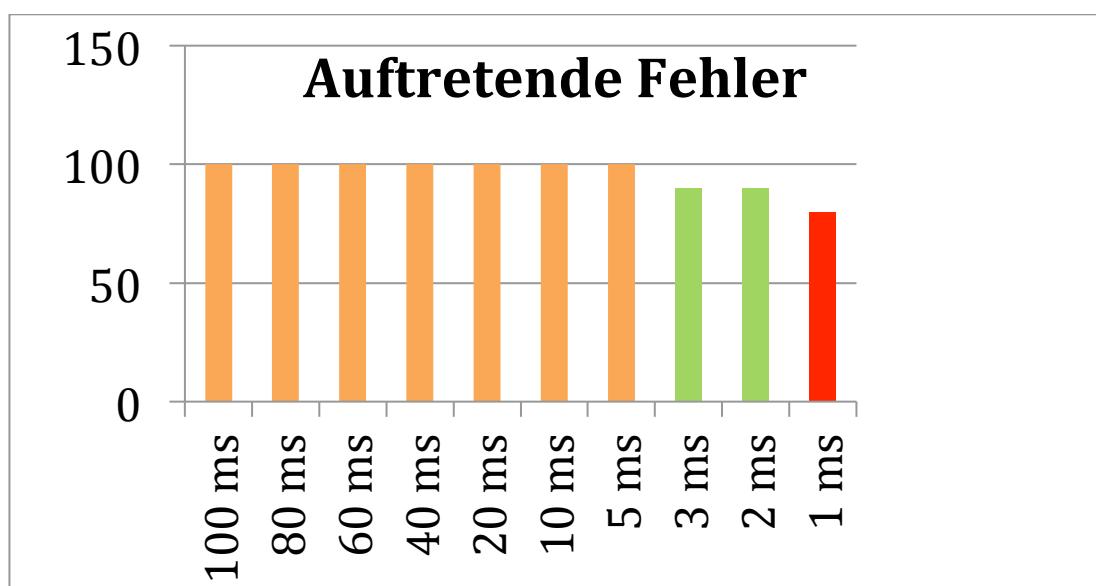
```

```
if ((zaehler == 9) & (buttonState == HIGH)) {  
    digitalWrite(ledPin, HIGH);  
    delay(10);  
    digitalWrite(ledPin, LOW);  
}  
else{}  
  
delay(50); // wartet eine 50ms, für die Optik zum testen  
binwert = byte(zaehler); // schreibt die aktuelle Zahl als binärwert in  
binwert  
Serial.println(binwert,BIN); // gibt den aktuellen binärwert Seriell aus (nur  
zum debuggen)  
for (int n=0; n < 4; n++){ // zählt von 0 bis 3 (4 Schritte) um jedes  
einzelne bit des halbbytes abzufragen  
    if (bitRead(binwert,n)==1){ // wenn das aktuelle bit eine eins ist  
        digitalWrite(n+2,HIGH); // setze den entsprechenden PIN auf 1  
    }  
    else{  
        digitalWrite(n+2,LOW); // ansonsten setze den PIN auf 0  
    }  
}  
}  
}
```

Aufbau der Schaltung mit den jeweiligen Leuchtdioden.



Das Synchronisationsergebnis haben wir durch Änderung des Delays ermittelt. Angefangen bei 100ms haben wir bis 1ms kontinuierlich jeweils 10 Messungen vorgenommen.



Erst bei 3ms ist es zu ersten Fehler gekommen. Bei jeweils 10 Messungen der Kameras war gerade einmal 1 Kamera bei einer Messung nicht 100% synchron sondern wies einen Fehler auf. Das Gleiche gilt auch für 2ms. Bei einer Millisekunde ergaben sich bei 10 Messungen bei genau zwei Kameras einen Fehler. Bei diesem Ergebnis kann von einer nahezu synchronen Auslösung bei 1ms sprechen.

Ein typischer Fehler kurz im Vergleich gezeigt. Die Kamera hätte binär eine 11 anzeigen müssen, jedoch wird eine 12 gezeigt.



7. Feinkalibrierung

Da einer unserer Projektschwerpunkte auf der Kalibrierung der aufgenommenen Bilder lag haben wir natürlich schon mit dem Rig eine Grobkalibrierung vorgenommen. Da diese nur sehr grob war, führte kein Weg daran vorbei, noch eine softwarebasierte Feinkalibrierung auf die Bilder anzuwenden. Ziel war es ein Springen des Objektes von Bild zu Bild möglichst zu vermeiden. Unser Programm sollte dabei wie folgt ablaufen. Zu Beginn wird die Szene noch ohne Objekt aber mit einem Marker abfotografiert. Dieser Marker dient dann später dazu, alle Bilder mit Objekt auf die gleiche Position zu transferieren. Im Anschluss an die Fotos mit den Markern werden die Fotos mit dem Objekt gemacht. Danach werden die Bilder von den Kameras sukzessiv auf den Rechner in einen gemeinsamen Ordner geladen. Von diesem Ordner aus, werden die Bilder automatisch von unserem Programm geladen, transformiert und in einem Video ausgegeben.

7.1 Recherche: openCV & openFrameworks

Für unser Projekt wurde von unserem Auftraggeber der Wunsch geäußert, unsere Kalibrationssoftware mit den freien Bibliotheken openCV und openFrameworks zu realisieren. Da keiner von uns bisher mit diesen Bibliotheken gearbeitet hat, hat die Recherche- und Einarbeitungsphase relativ viel Zeit in Anspruch genommen. Zu Beginn unseres Projekts war auch noch nicht klar, dass wir mit diesen Bibliotheken arbeiten werden, weshalb diese Phase auch relativ spät erst in Angriff genommen wurde.

Ich habe mir, wie meine Teammitglieder auch, sehr schwer getan bei den ersten Versuchen mit openCV und openFrameworks. Allein alle Komponenten der Bibliotheken zu installieren und diese wiederum korrekt in Xcode zu integrieren und zu verknüpfen, war ein anfangs sehr frustrierender und langwieriger Prozess. Erst nach zwei Wochen haben wir es geschafft auf allen Mac Rechnern die beiden Bibliotheken erfolgreich einzubinden. Nachdem nun erste Beispielprogramme geverviewt wurden und wir ganz langsam ein Gefühl für die C++ Syntax bekamen, konnten wir uns daran wagen, erste Komponenten unserer Software zu schreiben.

7.2 Image-load Funktion

Da wir einen komplett automatisierten Prozess innerhalb unserer Software angestrebt hatten, war es unser Ziel, dass natürlich auch das Laden der Bilder voll automatisch abläuft. Hierzu haben Johannes und ich eine Image-load Funktion geschrieben, welche in einer Schleife alle Bilder aus dem Bilderordner lädt. Hierzu laden wir die Bilder mit dem Objekt in einer ersten und die Bilder mit den Markern in einer zweiten Schleife. Damit die Bilder nicht verloren gehen, werden sie in einer Array namens *buffer_mrk[]* und *buffer_src[]* temporär gespeichert.

7.3 Markererkennung

Die Markererkennung ist der Ausgangspunkt unserer Software. Zu Beginn haben wir uns überlegt, wie wir es am Besten schaffen, dass das spätere Video keinerlei Sprünge aufweist. Dazu brauchten wir die Information, wo genau das Objekt auf jedem Bild vorkommt, um anschließend die Bilder

aufeinander anzupassen. Wir entschieden uns dafür, die gesuchten Werte mittels Farbmarker aus dem Bild auszulesen. Nachdem wir dieses Vorgehen gemeinsam entschieden hatten, machten sich Johannes Kiesel und ich an die Umsetzung einer Markererkennung.

Die Markererkennung wurde von uns dabei wie folgt konzipiert:

Zuerst wird ein, zuvor mit der Kamera aufgenommenes Bild, welches sich im RGB-Farbraum befindet, in den HSV²¹-Farbraum übersetzt. Dies hat den Vorteil, dass die Markererkennung robuster wird und weniger fehleranfällig bei unterschiedlichen Lichtverhältnissen ist, als im RGB-Farbraum. Anschließend werden unsere vier unterschiedlich farbigen Marker aus dem Bild gefiltert und jeweils in einem eigenen Vorschaufenster dargestellt. Die Darstellung erfolgt dabei als Binärbild. Dies bedeutet, dass auf den Vorschaufenstern lediglich zwei Werte (S/W) zu sehen sind. Die Marker werden weiß dargestellt und alles andere, was nicht zum Marker gehört schwarz. Diese Kontrolle hat den Vorteil, dass man im Vorschaufenster genau erkennen kann, welcher Marker von der Software gut erkannt wird und welcher nicht.

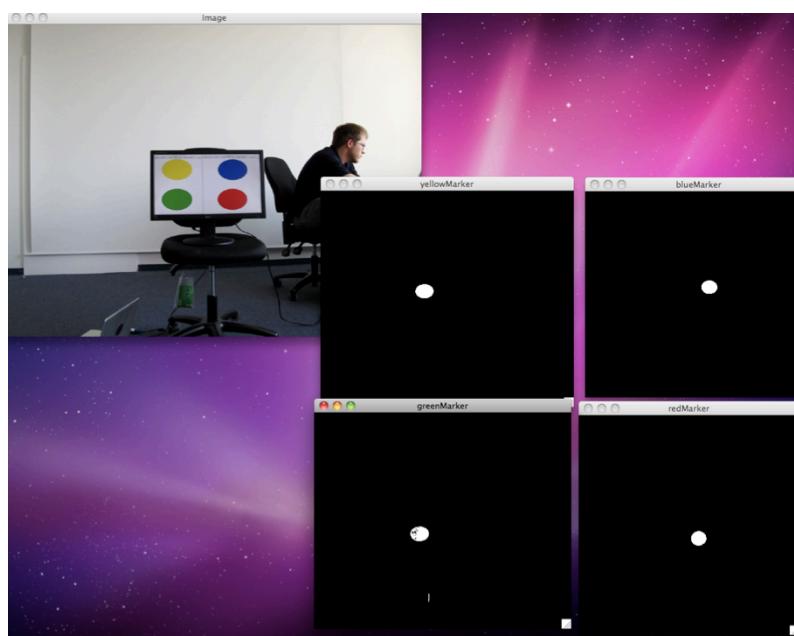


Abbildung 41: Markererkennung

²¹ HSV: Hue, Saturation, Value

Wie im obigen Bild deutlich zu erkennen, werden die vier Marker sehr gut von unserer Software ausgelesen. Anfangs haben wir noch mit drei Markern auf einem Stab gearbeitet. Dies hat sich jedoch als ungeeignet für die Berechnung der späteren Matrix herausgestellt, da uns so eine Dimension gefehlt hat.

7.4 Morphologische Operationen

Obwohl unsere Markererkennung schon sehr gut funktionierte, war das Ergebnis immer noch verbesserungsfähig. So tauchten in den binären Vorschaubildschirmen der einzelnen Marker vereinzelt Störpixel auf, welche nicht zu den Markern gehörten. Dies lag daran, dass sich der Farbton der Marker teilweise auch in anderen Elementen, die auf dem Bild im Hintergrund zu sehen waren, wiederspiegeln.

Um diese Störpixel zu entfernen, bedienten wir uns der morphologischen Operatoren `cvErode()` und `cvDilate()`. Diese Operatoren sind standartmäßig 3x3-Masken.

Bei der Funktion `cvErode()` fährt nun die Maske über jeden Pixel im Bild. Mit dem mittleren Pixel in der Maske wird der aktuelle Pixel im Bild ausgewählt. Sind nun alle anderen acht Pixel der Maske mit weißen Pixeln belegt, so bleibt der ausgewählte Pixel weiß. Ist dieser schwarz, so wird er auf weiß gesetzt. Ist auch nur ein Pixel der äußeren acht Pixel auf der Maske schwarz, also nicht belegt, so wird der ausgewählte Pixel im Maskeninneren auf schwarz gesetzt, sofern er weiß war. War er schon schwarz, so ändert sich nichts.

Die Funktion `cvDilate()` funktioniert genau entgegengesetzt wie `cvErode()`. Auch hier fährt die 3x3-Maske über jeden einzelnen Pixel im Bild. Mit dem mittleren Pixel in der Maske wird wieder der aktuelle Pixel im Bild ausgewählt. Ist nun mindestens ein Pixel der äußeren acht Pixel der

Maske auf weiß gesetzt, so wird der ausgewählte Pixel ebenfalls auf weiß gesetzt. Ist kein Pixel der äußeren acht Pixel der Maske mit weiß belegt, so wird der ausgewählte Pixel in der Mitte der Maske auf schwarz gesetzt. Durch eine Verbindung dieser zwei Operatoren lässt sich sicherstellen, dass kleinere Störpixel beseitigt, jedoch der eigentlich Marker nicht wesentlich verändert wird.

7.5 Transformationsmatrix

Nachdem wir die Markererkennung realisiert hatten, galt es nun die daraus gewonnenen Informationen auf unsere Kalibrierung anzuwenden. Die Transformationsmatrix wurde dabei auf folgende Weise gewonnen: Zuerst berechneten wir den Schwerpunkt aus den gefilterten Markern. Mit diesen Schwerpunkten hatten wir nun in jedem der acht Bilder vier fixe Werte. Um eine Matrix zu gewinnen, mussten wir nun diese Fixwerte in Relation zueinander stellen. Dies geschah, indem wir die vier Schwerpunkte der Marker eines Bildes mittels der Funktion *getAffineTransform()* auf die vier Schwerpunkte der Marker des vorherigen Bildes²² transferierten. Dabei erfuhren die veränderten Schwerpunkte eine Translation, eine Rotation, eine Skalierung und eine Scherung.

Diese Operationen wurden dann in einer Matrix, in unserem Fall *map_matrix1* gespeichert und so auf jeden einzelnen Bildpunkt des Bildes angewendet. Dieser Vorgang wurde dann bei jedem Bild wiederholt. Da das erste Bild in der Bildabfolge keinen Vorgänger hat, wurde auf dieses Bild keine Matrix angewendet, sprich es blieb auch bei der Vorschau völlig unverändert.

²² „Vorherig“ bezieht sich hierbei auf die aneinander gereihte finale Abfolge der Bilder als Film

7.6 Implementierung des openCV Codes in openFrameworks

Da es eine unserer Vorgaben war unser Programm innerhalb von openFrameworks zu verwenden war es eine meiner Aufgaben eine Lösung zu finden wie dies Bewerkstelligt werden konnte. Nach einiger Zeit in der wir verschiedene Ansätze verfolgte hat uns Hr. Johannes Brendel über den genauen Unterschied zwischen openCV und openFrameworks und über die genaue Arbeitsweise mit openFrameworks, so vermeidet man hier die grundsätzliche Struktur von ofx vorgegebenen Beispiele zu verändern, aufgeklärt, sodass wir eine Lösung finden konnten. Das Problem das wir erkennen mussten war das openCV in C und openFrameworks objektorientiert in C++ geschrieben wird. So musste man den ursprünglichen Code soweit verändern, dass er in dieses Modell passt. Begonnen haben wir damit, dass wir den openCV - Code in eine Headerfile und eine Sourcefile zu trennen. In der Headerfile definieren wir ein Klasse Calibration mit privaten und öffentlichen Funktionen und Variablen um einen Zugriff von außen aus der TestApp-Klasse von openFramework zu gewährleisten. In dieser Headerfile definieren wir außerdem alle im Code verwendeten Funktionen und Variablen.

In der Sourcefile bleibt soweit alles unangetastet außer das die Mainfunction umbenannt werden musste.

7.7 Lienare Interpolation mit Alpha Blending

Meine letzte aber zum Abschluss wichtigste Aufgabe war das Erstellen einer Vorschau in der die kalibrierten Bilder in einem animierten Video abgespielt werden. Dafür sollten die Bilder durch Alphablending linear interpoliert werden.

Um das zu erreichen haben wir die Funktion `showPreview()` erstellt. In dieser wird zuerst ein neues Fenster "Preview" erzeugt in dem die Vorschau angezeigt werden soll. Dieses wird auf die position 100,100 verschoben um vollständig angezeigt zu werden.

Um die Bilder nacheinander anzuzeigen haben wir zwei verschachtelte for-Schleifen vorgesehen, wobei die äußere dafür sorgt die Bilder nachzuladen und die innere für die Überblendung zuständig ist.

Um einen Crossfade-Effekt mit Alphablending zu erzeugen ist es wichtig nach und nach den Alphawert des einen Bildes zu verringern wobei gleichzeitig der des zweiten Bildes zunimmt. Hier verhält es sich so, dass ein Alphawert von 1 eine volle Deckkraft darstellt und ein Alphawert von 0 eine 100%ige Transparenz bedeutet.

Dafür verändern wir in der inneren for-Schleife stufenweise den Alphawert der beiden Bilder.

Für die Darstellung nutze ich 2 Regions of Interest (ROI) von gleicher Größe die ich mit Hilfe der `cvAddWeighted` Funktion die Bilder mit den zugehörigen Alphawerten übereinanderlege.

Die Funktion `cvWaitKey()` setzt für uns die Zeit wie lang ein Frame angezeigt wird.

7.8 Codedokumentation

Eine weitere Aufgabe bestand darin, unseren Code ordnungsgemäß zu dokumentieren. Prof. Dr. Ralph Lano sagte einst: „Codedokumentation müsse sich wie Prosa lesen. Auch jemand, der keine Ahnung vom Programmieren hat, müsse anhand der Dokumentation verstehen, was in dem Programm, wann vor sich geht.“

Daran orientierend, haben wir unseren Code so gut und ausführlich wie möglich dokumentiert, um auch für zukünftige Projektteams, die eventuell

auf unserem Code aufbauen oder auch nur Teile davon verwenden möchten, einen leichten Einstieg in unsere Software zu ermöglichen.

8. Professionelle Software

Um unser selbst erzeugtes Video in Vergleich mit einem professionell erzeugten Video zu zeigen, musste natürlich zuvor ein Video mit professioneller Software erzeugt werden. Hierfür wurde Adobe After Effects und Twixtor verwendet.

8.1 Adobe After Effects

After Effects ist eine von Adobe entwickelte Film und Media Production Software, die speziell für Filmschnitt und Animation oft verwendet wird. Wie auch Photoshop, Illustrator und alle anderen Adobe Programme ist sie sehr mächtig und überwältigend. Die genauen Prozesse rauszufinden, die man benötigt um spezielle Effekte oder gewünschte Aktionen auszuführen, bedingt viel Zeit und Geduld. Wenn man dann jedoch die Prinzipien verstanden hat, sind einem fast keine kreativen Grenzen gesetzt.

Speziell auch in unserem Fall war dieses Programm perfekt. Das eigentliche Programm After Effects übernimmt schon per Tracking Aktion das Kalibrieren der Frames. Hier wird ein Punkt von hohem Kontrast gewählt und dann angeben, in was für einem Radius er im nächsten Frame nach diesem Punkt suchen soll. Die Frames werden dann aufeinander abgeglichen und angepasst, so dass das Verwackeln der Bilder minimal ist oder gänzlich wegfällt. Um die Affine Transformation 'Rotation' mit einzubeziehen, werden zwei Punkte im Bild definiert, sodass

AE den Winkel der Verdrehung rausrechnen kann. Wenn man schon im vornherein eine gute Hardwarekalibrierung vornimmt, erreicht man ein so gut wie perfektes Ergebnis.

Im folgenden Bild kann man diese Tracking Aktion sehr gut erkennen.

Auf diesem Bild sind zwei verschiedene Bühnen dargestellt. Auf der linken Seite, die Bühne mit der Tracker Position und auf der rechten Seite die Bühne, welche die Transformation der Frames auf dem Clip anzeigen.

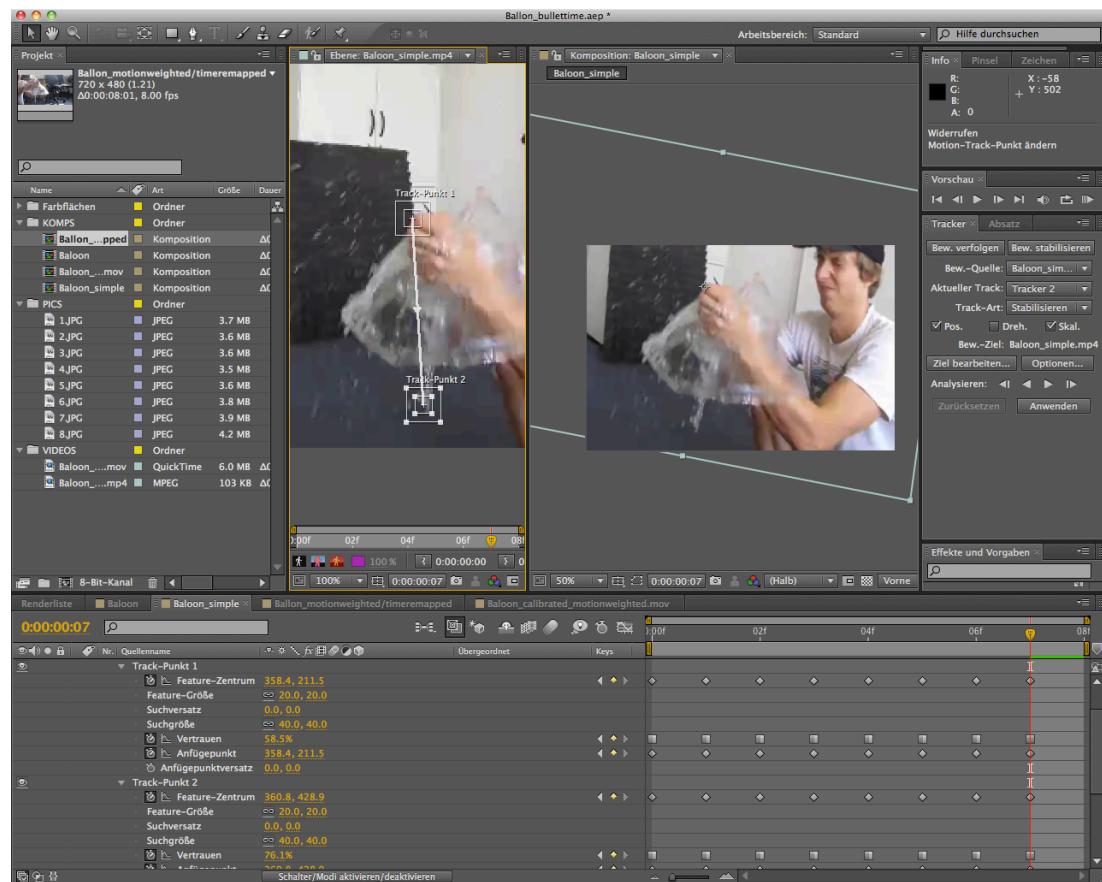


Abbildung 42: Ausschnitt aus Adobe After Effects

8.2 Twixtor

Twixtor ist ein von RE:Vision Effects, Inc. entwickeltes PlugIn für After Effects und ein paar andere, ähnliche, Software Programme. Mit diesem unheimlich mächtigen PlugIn lässt sich sie Motion Interpolation fast

spielerisch auf ein Clip anwenden. Der herkömmliche Gebrauch dieses PlugIns liegt in dem Slow Motion Effekt, den diese Interpolation ermöglicht. Ausgelegt auf mindestens 59.96 fps, kann es bis auf mehrere hundert Frames interpolieren und einen sehr sauberen Slow Motion Effekt erzeugen. Slow Motion ist in unserem Fall unter einer anderen Anwendung bekannt. Normalerweise verbindet man Slow Motion mit einem Video Clip, der dann zeitlich langsamer abläuft. Aber wie es das Wort schon verrät, muss Zeit in diesem Fall keine Rolle spielen. Motion ist hier das Stichwort. Das einzige was Slow Motion beinhaltet ist das Verlangsamen der Bewegung, ob im Zeittotenraum oder nicht. Diese motion-weighted Interpolation kann man sich natürlich auch zu nutzen machen, wenn man eine Kamerafahrt im Zeittotenraum nachahmen möchte. Hierbei werden zwischen den schon existierenden Frames weitere 'interpolierte' Frames hineingerechnet und man kann sich mal schneller und mal langsamer um das stillstehende Objekt drehen, soweit ein Kreisförmiges Rig gewählt wurde.

Wie auch After Effects, hat Twixtor hunderte von feinen Einstellungen, deren Beschreibung alleine den Umfang eines Projekts ergeben würde. Die einfachste Variante, die auch wir letztendlich gewählt hatten, war das Reduzieren der Speedramp unter motion-weighted Blend. Hier kann man den Clip stark verlangsamen, und das motion-weighted Blend übernimmt die Interpolation. Nach der Interpolation muss man den Clip timermappen, um die volle Länge des neuen Clips darstellen zu können. Da wir jedoch nur mit 8 fps arbeiten konnten, war das Resultat relativ enttäuschend da auch Twixtor mit nur so wenig fps keine ordentliche Interpolation zu Stande bekommt. In dem folgenden Bildern kann man zunächst das PlugIn sehen und dann einen Screenshot von dem gerenderten Video.

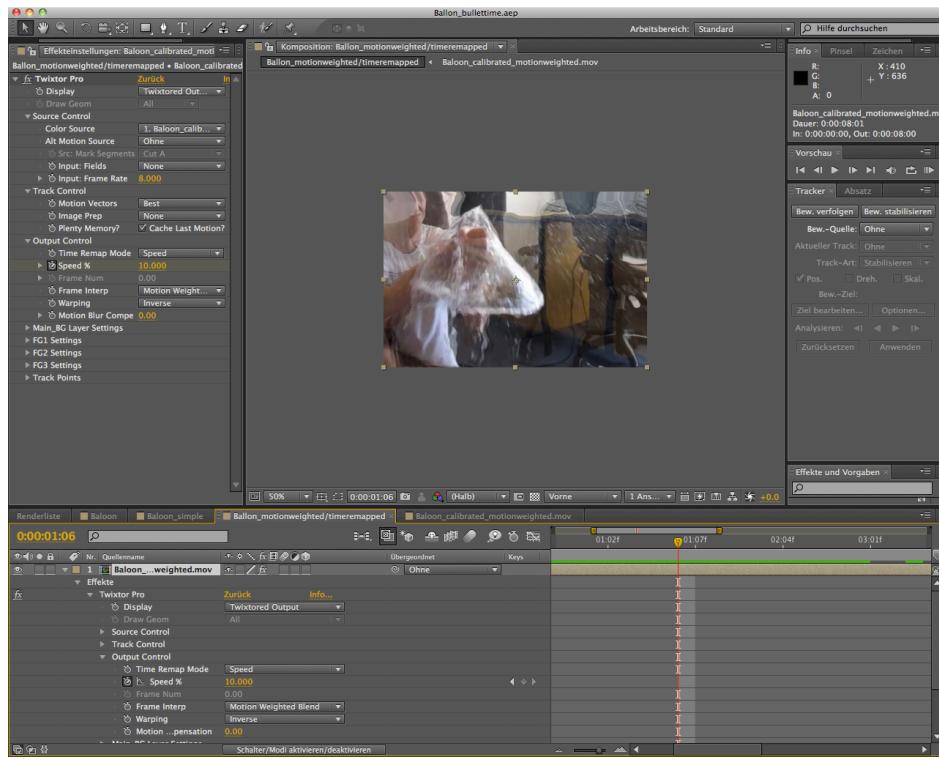


Abbildung 43: PlugIn Twixtor



Abbildung 44: Screenshot von gerendertem Video

9. Soll/Ist-Vergleich

SOLL	IST
Rig für min. 8 Kameras	✓
CHDK Skript - Auslösen - Bildübertragung	✓ ✓
Hardware-Kalibrierung - Grid (Sucherzielkreuz) - Nachweis Synchronität	✓ ✓
Software-Kalibrierung - Markererkennung - Übereinanderlegen der Bilder - lineare Interpolation mit Alphablending	✓ ✓ ✓
Qualitativer Vergleich mit professioneller Software	✓
Quantitativer Vergleich mit professioneller Software	✗
Affine Transformation	✓
Perspektivische Transformation	✗
optional	
Fernauslöser mit Hilfe von Arduino	✓

Ausgehend vom Projektplan, haben wir alle bis auf die Anwendung der perspektivischen Transformation, sowie dem quantifizierten Vergleich des professionellen Videos mit unserem, durch unsere Software produziertem, Video, alle unsere gesteckten Projektziele erreicht.

10. Resümee

Dieses Resümee soll versuchen, das komplette Projekt, mit all seinen Facetten und besonders unsere Erfahrungen bezüglich des Projekts zu reflektieren.

Schon unser Projektstart verlief ziemlich zäh und sollte sich auch über das komplette Projekt hinweg erstrecken. Das Semester begann Mitte März, jedoch wurde ein erstes Treffen mit den betreuenden Professoren, zur Ideenfindung und -vorstellung erst Ende März einberufen. Damals noch mit anderen Ideen und möglichen Projektgruppen, wurden unsere Vorschläge jedoch schnell als nicht projekttauglich abgestuft. Nachdem kein Vorschlag unsererseits von den Professoren akzeptiert wurde, haben wir einen vielversprechenden Projektvorschlag von Herrn Röttger, der ursprünglich in stark abgeänderter Version von Benjamin Kuckuk stammte, als zu bearbeitendes Projekt ausgewählt.

Da das Projekt als solches nicht von uns kam, hatten wir auch diesbezüglich keine konkreten Projektziele vor Augen zu diesem Zeitpunkt. Auch zwei Wochen später, bis zum ersten Treffen mit unserem Betreuer, war nicht wirklich klar, ob das Projekt wirklich umgesetzt wird und werden kann. Zwischenzeitlich haben wir uns sogar schon Gedanken gemacht über alternative Projektideen, da wir eben bis zu diesem Zeitpunkt nicht wussten, was wir bei diesem Projekt erreichen sollen und ob diesbezüglich überhaupt Equipment vorrätig sein wird.

Nach dem ersten Treffen wurden uns dann erste Projektziele definiert und versprochen, dass in kürzester Zeit acht Kameras für unser Projekt bereitstehen. Dies war Anfang April.

Mitte April bekamen wir dann die erste Kamera. Nach einem zähen Start konnten wir also leider auch nicht schneller voranschreiten, da uns das Equipment fehlte. Wir machten die Arbeiten, die mit einer Kamera möglich waren, wie das Auslösen und die Datenübertragung. Als Mitte April die zweite Kamera eintraf, konnten wir erst mit den Tests beginnen, ob

unsere Kameras synchron auslösen. In der Zwischenzeit hatten wir unser Rig gebaut und alle Formalitäten wie Lastenheft, Zeitplan, Recherchen etc. abwickeln können. Jedoch verschaffte uns das Warten auf unsere Projektutensilien einen immer größer werdenden Abstand im Projektfortschritt zu den anderen Gruppen. Des Weiteren schwand die Motivation stetig bei uns, aufgrund der Tatsache, dass wir einerseits Ergebnisse erzielen wollten, andererseits diese Ergebnisse nur theoretisch vorbereiten und nicht praktisch ermitteln konnten. Erst Mitte Juni, also drei Monate nach dem offiziellen Semesterstart, trafen die restlichen sechs Kameras ein. Für Mitte Juli war die Abschlusspräsentation angesetzt. Dies hatte zu bedeuten, dass wir lediglich einen Monat Zeit hatten, unser Rig, unsere Software, unserer Arduino Controller und die synchrone Auslösung gleichzeitig mit allen acht Kameras zu testen. Natürlich tauchten in dieser Phase viele Fehler auf, die speziell mit allen Kameras erst entdeckt werden konnten, auf die wir dann aufgrund der stark fortgeschrittenen Zeit schnellstens reagieren mussten. So war z.B. der Arduino Controller mit zwei Kameras bestens gelaufen, für acht Kameras, mit insgesamt drei zwischengeschalteten USB Hubs ist jedoch nicht mehr genug Strom an den Kameras angekommen, um hier eine entsprechende positive Flanke zu erzeugen. Auch konnten erste Aufnahmen von aussagekräftigen Motiven erst zu diesem Zeitpunkt gestartet werden, da man mit zwei Kameras im Grunde nur statische Objekte aufnehmen kann, um einen Bullet Time Effekt zu erzielen. Durch diese Fehler, von denen wir überhaupt erst sehr spät erfahren konnten, schoss unser Arbeitspensum natürlich in die Höhe. Der Aufwand bei der Fehlersuche, die wir in der Mitte des Semesters eingeplant hatten und für die wir zu diesem späten Zeitpunkt eigentlich keine Zeit hatten, ließ unser Projekt nur sehr langsam voranschreiten. Zu diesem Zeitpunkt war die gesamte Stimmung im Team sehr schlecht. Die schier unendliche Arbeit, die sich im letzten Monat des Projekts aufgetan hat und die sich bis zum Schluss aufstaute, da sich der Liefertermin der Kameras immer wieder nach hinten verschob, bis kurz vor den Prüfungszeitraum, ließ unser aller Motivation rapide sinken.

Nichtsdestotrotz haben wir uns den sehr widrigen äußersten Umständen gestellt, ihnen getrotzt und unser Projekt noch zu einem durchaus vertretbaren und respektablen Ende geführt.

Rückblickend kann man sagen, dass das Projekt sehr schleppend angefangen hat, bedingt durch mehrere Faktoren. Im Laufe des Semesters hat es etwas stagniert. Die Ursachen hierfür wurden bereits ausführlich beschrieben. Die Arbeitsleistung schoss jedoch gegen Ende gleich einer exponentiellen Leistungskurve nach oben. Gemittelt über das ganze Semester ergab dies weit über die, für ein solches Projekt angedachten, 300 Mannstunden Arbeit. Der Aufwand, den wir für dieses Projekt aufgewendet haben, mag, verglichen mit den anderen Projektteams, anfangs noch hinter denen gelegen haben, jedoch gegen Ende haben wir unseren Rückstand wieder aufgeholt. Jeder bei uns im Team hat durchgehend mitgearbeitet. Von unseren gesteckten Zielen haben wir den Großteil erreicht.

Als diese Faktoren sollten auch bei einer objektiven Beurteilung dieser Projektarbeit ins Gewicht fallen.

Literaturverzeichnis

Bradski, Gary/ Kaehler, Adrian: Learning openCV, Sebastopol 2008,
O'Reilly Media

Internetquellen

http://de.wikipedia.org/wiki/Bullet_Time (Stand: 13.07.2012).

<http://wiki.ohm-hochschule.de/roettger/index.php/Project-R3DS/Projektorganisation> (Stand: 02/2012)

<http://wiki.ohm-hochschule.de/roettger/index.php/Project-R3DS/Projektorganisation> (Stand: 02/2012)

<http://wiki.ohm-hochschule.de/roettger/index.php/Project-R3DS/Aufgabenverteilung> (Stand 02/2012)

<http://wiki.ohm-hochschule.de/roettger/index.php/Project-R3DS/Meetings> (Stand: 02/2012).

<http://schorsch.efi.fh-nuernberg.de/mewiki/index.php/Project-BME-2012-07-MatRigX/Blog> (Stand: 07/2012).

<http://schorsch.efi.fh-nuernberg.de/mewiki/index.php/Project-BME-2012-07-MatRigX/Dateiverwaltung> (Stand: 07/2012)

<http://de.wikipedia.org/wiki/GitHub> (Stand: 18.07.2012)

<http://schorsch.efi.fh-nuernberg.de/mewiki/index.php/Projects/BME-2012-07-MatRigX> (Stand: 07/2012)

http://chdk.wikia.com/wiki/CHDK_in_Brief. (Stand: k. A.)

<http://chdk.setepontos.com/index.php/topic,1194.0.html> (Stand 04/2008).

<http://chdk.wikia.com/wiki/ACID>. (Stand: k. A.)

Abbildungsverzeichnis

Abbildung 1: Geraedliniges Rig mit parallelen Kameras	19
Abbildung 2: Kreisförmiges Rig aus dem Film "Matrix"	19
Abbildung 3: Konzept des Rigs	21
Abbildung 4: Fertiges Rig	23
Abbildung 5: Nahaufnahme des Rigs	23
Abbildung 6: Grid auf dem Display einer Kamera	24
Abbildung 7: Beispiel Grid	26
Abbildung 8: Verifikation der Ausrichtungsgenauigkeit des Rigs	27
Abbildung 9: Berechnung der Gegenkathete.....	28
Abbildung 10: ACID nach dem Auslesen der Exif-Daten	31
Abbildung 11: SDM/CHDK Installer für OS X	32
Abbildung 12: PTP-Klient.....	36
Abbildung 13: CHDK Menü.....	39
Abbildung 14: Schaltungsaufbau	40
Abbildung 15: USB Kabel-Belegung.....	40
Abbildung 16: Versuchsaufbau	42
Abbildung 17: Arduino Logo.....	43
Abbildung 18: Arduino Chip	43
Abbildung 19: Arduino Chip (schräg).....	44
Abbildung 20: Arduino Chip (45° Perspektive).....	44
Abbildung 21: Arduino (TM) UNO Rev3	45
Abbildung 22: IR Sensor	46
Abbildung 23: Entwicklungsumgebung	46
Abbildung 24: Arduino Setup	47
Abbildung 25: Arduino Programm	47
Abbildung 26: Steckplatine	48
Abbildung 27: Steckplatine 2	49
Abbildung 28: Steckbrücken und Kabel	49
Abbildung 29: Programmierung Arduino 2	50
Abbildung 30: Programmierung 1	50

Abbildung 31: Spannungsmessung	51
Abbildung 32: Transistor	51
Abbildung 33: Schematisches Diagramm	53
Abbildung 34: Messung der Spannung.....	54
Abbildung 35: Steckplatine 3	54
Abbildung 36: Arduino Schaltplan	54
Abbildung 37: Steckplatine 4	54
Abbildung 38: Desktopmonitor mit Stoppuhr.....	56
Abbildung 39: Asynchronität	56
Abbildung 40: Nachweis der Synchronität	56
Abbildung 41: Markererkennung.....	64
Abbildung 42: Ausschnitt aus Adobe After Effects	70
Abbildung 43: PlugIn Twixtor	72
Abbildung 44: Screenshot von gerendertem Video	72