



Fakultät Elektrotechnik Feinwerktechnik Informationstechnik efi
Projekt MatRigX

Prüfungsstudienarbeit von

Christian Siemer

Matr.Nr.: 2125124

B-ME 6

Bestätigung gemäß § 35 (7) RaPO

Studienarbeiten /Prüfungsstudienarbeiten sind mit einer Erklärung des Studierenden zu versehen, dass sie/er die Arbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet hat.

Christian Siemer [BME6]

Ich bestätige, dass ich die Prüfungsstudienarbeit mit dem Titel: **MatRigX** selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: Unterschrift:

Erwartungen der Prüfungsstudienarbeit

Seit Jahren wird in der Film- sowie Spielindustrie aufwendig und kostspielig Bullet Time Spezialeffekte eingesetzt. Die in der Praxis gebräuchlichen Verfahren sind technisch sehr aufwendig und mit hohen Kosten verbunden. Beim professionellen Einsatz wird eine Vielzahl von Kameras verwendet um eine natürliche Kamerafahrt im „zeittoten Raum“ zu simulieren. Wir haben uns das Ziel gesetzt genau an dieser Stelle anzusetzen und Bullet Time Consumer tauglich mit einem Bruchteil der Kosten zu realisieren. Die Verwendete Hardware sind acht einfache Canon PowerShot SX130 IS Kameras die überall käuflich für ca. 130€ zu erwerben sind und ein Arduino Mikrocontroller für ca. 20€. Unser Wunsch war es mit diesen Mitteln eine Kamerafahrt zu simulieren.

Das ganze Projekt war ein Forschungsauftrag bei dem kein fertiges Endprodukt sondern ein Prototyp zu erstellen war. Es sollte die Möglichkeiten Aufzeigen und einen ersten Grundbaustein für spätere vertiefende Gruppenprojekte liefern. Es war von Beginn an klar, dass es zu zahlreichen Problemen kommen wird. Diese sollten Dokumentiert und falls möglich behoben werden. Zudem musste sich in verschiedenste Bereiche eingearbeitet und informiert werden.

Meine Aufgaben haben sich hauptsächlich auf das simultane Auslösen, die Schaltung sowie die Programmierung des Mikrocontrollers beschäftigt. Zudem habe ich mich um das CHDK Scripting und der USB-Übertragung gekümmert. Der Fokus lag auf das synchrone Auslösen der acht Kameras und eine spätere Verifizierung der Genauigkeit. Hierbei gab es eine Vielzahl von Problemen und Schwierigkeiten die es zu bewältigen gab. Angefangen von Elektrotechnischen Grundlagen bis hin zur späteren C-Programmierung. Weitere Aufgaben war die Einarbeitung in OpenCV sowie die Mathematischen Grundlagen der Kalibrierung.

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 Ausrichtungsgenauigkeit		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
Dokumentation des Codes	../Code/	Kiesel/ Neubauer/ Voit
Materialbeschaffung		Neubauer/ Siemer

INHALTSVERZEICHNIS

1. PROJEKTPROFIL:	13
1.1 Kurzbeschreibung:	13
1.2 Ergebnis:	14
1.3 Lerneffekt:	15
2. PROJEKTFORTSCHRITT:	17
2.1 Aufgaben und Ziele:	17
2.2 Zeitliche Planung:	18
2.3 Aufteilung des Projekts:	21
3. PROJEKTAUFGABEN	23
3.1 CHDK	23
3.1.1 Auslösen:	24
3.1.2 USB-Übertragung:	28

3.1.3 Konsolensteuerung der Kamera:.....	31
3.1.4 Auslösen und Übertragung:	34
3.2 Schaltung:	38
3.2.1 Aufbau:	39
3.2.2 Arduino:.....	43
3.2.2.1 Technische Daten:	44
3.2.2.2 Programmierschnittstelle:	50
3.2.2.3 Code:	52
3.2.2.4 Schaltungsaufbau:.....	54
3.2.2.5 Probleme:.....	59
3.2.2.6 Problemlösung:.....	60
3.2.3 Synchronität:.....	66
3.2.3.1 Verifizierung LCD:.....	66
3.2.3.2 Verifizierung Arduino.....	70

3.3 Kalibrierung.....	78
3.3.1 Einarbeitung OpenCV.....	78
3.3.2 Mathematische Einarbeitung.....	79
3.4 Weitere Aufgaben.....	80
3.4.1 Logo Erstellung.....	81
3.4.2 Powerpoint Präsentation/Layout.....	81
3.4.3 Hardware Kalibrierung/Grid Erstellung.....	82
4. Reflexion.....	86
4.1. Soll/Ist Vergleich.....	86
4.2. Quellen.....	87
4.3. Persönliche Fazits.....	88

1. Projektprofil

1.1 Kurzbeschreibung

Das Ziel war es mit einfachen Mitteln eine Kamerafahrt im „zeittoten Raum“ zu simulieren ein sogenannten Bullet Time Effekt.

Wikipedia: „Def.: **Bullet Time** (von 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 rückwärtslaufende Zeit möglich.“

Hierzu sollten acht Kameras sowie ein Arduino Mikrocontroller verwendet werden. Mit Hilfe von CHDK (**C**anon **H**ack **D**evelopment **K**it) wurde die Grundlage für das Projekt geschaffen. Mit einem extra angefertigten Rig sowie einem auf der Kamera einstellbaren Grid sollte die Hardwarekalibrierung vorgenommen werden. Neben dem synchronen Auslösen der Kameras war ein entscheidender Punkt die Software-Feinkalibrierung. Hier sollte eine Software basierend auf OpenCV/OpenFrameworks geschrieben und mit einer professionellen Software verglichen werden. Zudem sollte aufgrund der begrenzten Kameras und der daraus resultierenden Framerate eine Interpolation angestrebt werden.

1.2 Ergebnis

Das Ergebnis unserer Arbeit ist ein Programmierter Arduino Mikrocontroller der zum einen die Aufgabe hat die Kameras auslösen und weiterhin zur Verifizierung der Synchronität eingesetzt wird. Das Ergebnis sind 8 auf die Millisekunde genaue, synchrone Bilder die von dem auf der Kamera geladenen CHDK Script initialisiert

wird. Zudem eine selbstgeschriebene Software zur Kalibrierung, die Fehler aufweist jedoch ihre Grundfunktionen erfüllt. Weiterhin wurde ein Vergleich zu der Professionellen Software und einer selbstgeschriebenen gezogen. Letztlich gehören vor allem auch die Probleme und Schwierigkeiten die in dem Projekt entstanden sind zu dem Ergebnis um künftige Arbeiten zu erleichtern und davor zu bewahren in dieselben zeitaufreibenden Probleme zu laufen.

1.3 Lerneffekt

Das Projekt bietet eine Vielzahl an unterschiedlichen Lerneffekten. Vor allem das vertiefen in unterschiedlichen Bereichen mit denen man zum Beginn keinerlei Erfahrungen hatte war ein Positiver Lerneffekt. Hierzu gehört der Elektrotechnische Aspekt der leider in unserem Studium viel zu kurz gekommen ist. Aus diesem Grund mussten Kenntnisse vom Grund auf erlangt werden wie z.B. eine Schaltung aufgebaut wird, wozu Widerstände oder Transistoren da sind usw. Zudem war es spannend, theoretisches Wissen welches im Studiengang gesammelt wurde in der Praxis anzuwenden. Ein Beispiel ist die C-

Programmierung die bei vielen auf Unverständnis stößt da heute Java und objektorientierte Sprachen sehr gefragt sind. Die Programmierung des Mikrocontrollers jedoch geschieht ausschließlich in C und ich konnte mein Wissen, auch wenn es schon ein paar Jahre her ist gut anwenden. Auch mathematischen Grundlagen wie Transformationen und Affine Abbildungen bei denen ich niemals gedacht hätte, dass ich sowas jemals brauchen könnte waren von großem Nutzen. Das Projekt bietet viele Möglichkeiten in einer späteren Projektarbeit anzusetzen und zu vertiefen. Viele Möglichkeiten konnten leider aufgrund des knappen Zeitmanagements nicht verwirklicht werden. Ein weiterer persönlicher Lerneffekt war wie das richtige Projektmanagement ist. Es wurden einfach zu viele Fehler gerade in Sachen Kommunikation gemacht die das Ergebnis in bestimmten Bereichen eindeutig beeinflusst haben.

2. PROJEKTFORTSCHRITT

2.1 AUFGABEN UND ZIELE:

1. Recherche vorhandener Bullet Time Projekte
2. Kostenvoranschlag für Rig
3. Recherche CHDK
4. Recherche bzgl. After Effekts und Interpolation
5. Organisatorische Abwicklungen (Dropbox)
6. Erste Interpolierung mittels AE
7. Konzeption und Realisierung des Rigs
8. Grobstruktur des Lastenheftes
9. Test der Kamera mit CHDK
10. Finale Version: Lastenheft
11. **Meilenstein:** Triggerung

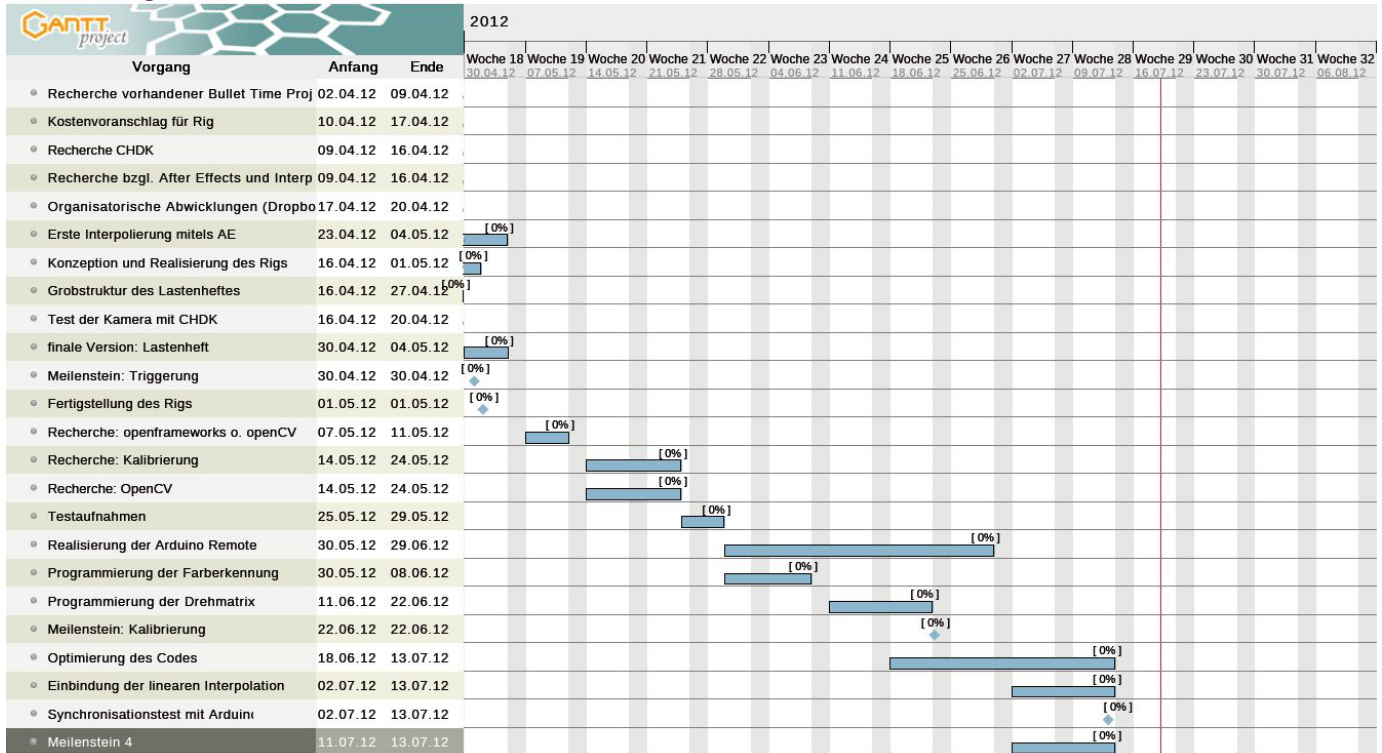
12. Fertigstellung des Rigs
13. Recherche: Openframeworks o. OpenCV
14. Recherche: Kalibrierung
15. Recherche: OpenCV
16. Testaufnahmen
17. Realisierung der Arduino Remote
18. Programmierung der Farberkennung
19. Programmierung der Drehmatrix
20. **Meilenstein:** Kalibrierung
21. Optimierung des Codes
22. Einbindung der linearen Interpolation
23. Synchronisationstest mit Arduino

2.2 Zeitliche Planungen

Nach den ersten Projektbesprechungen wurde das in der Abbildung dargestellte Gantt-Diagramm für den Projektablauf festgelegt. Es war uns von Anfang an klar, dass es sich dabei nur um einen ersten Grobentwurf handeln konnte, da

wesentliche Parameter noch unbekannt waren. In vielen Bereichen gab es einfach keine Vorkenntnisse die zu einer genaueren Planung hätten beitragen können. Zudem gab es Verzögerungen bei der Hardware so dass einige Planungen zu Beginn noch offen waren. So wurde das Diagramm fortlaufend angepasst. Viele Aufgaben wurden unterschätzt bzw. nicht genügend in der Zeit Planung berücksichtigt. Im Ganzen kann man sagen das viel Zeit oder zu viel Zeit für theoretisches Einarbeiten in die verschiedenen Materien geflossen ist. Der Grund ist das einfach die Aufgaben sehr komplex und zu wenig Vorkenntnisse vorhanden waren. Für das Gantt-Diagramm war unser Projektleiter zuständig. Er hat sich um die Planung gekümmert. Die genaue Reihenfolge kann ich nicht wirklich nachvollziehen und erschließt sich mir nicht.

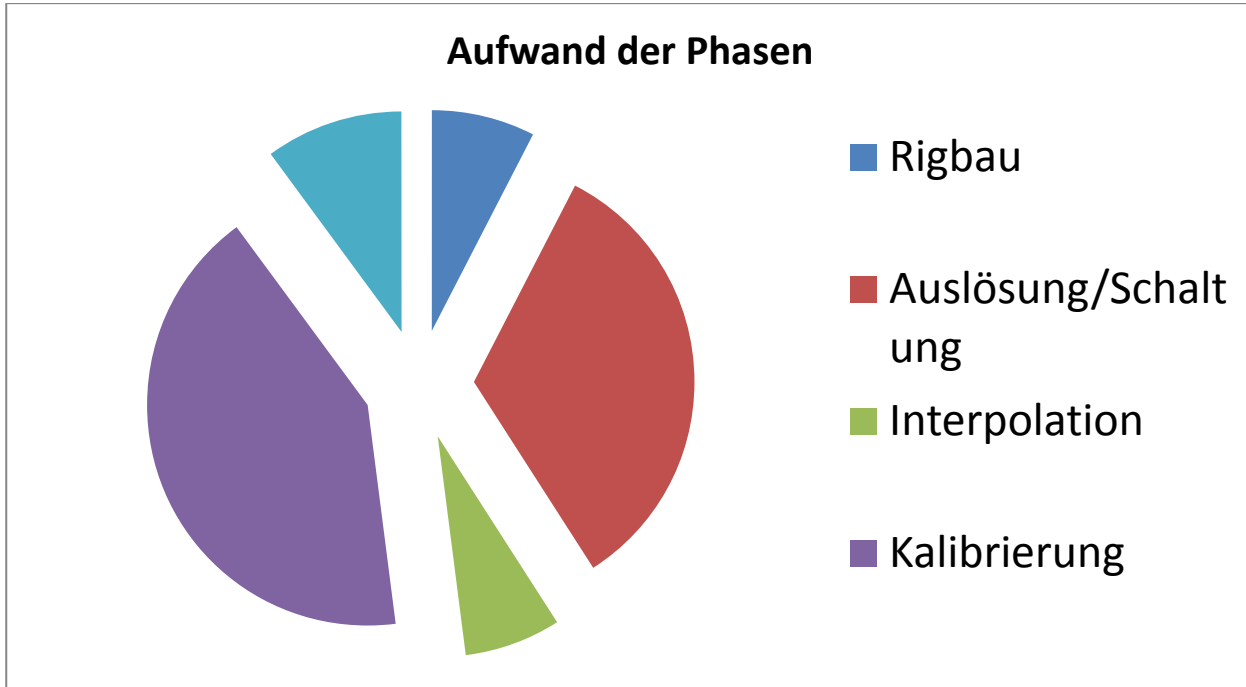
Gantt-Diagramm



2.3 Aufteilungen des Projekts

Nachdem die Aufgabenstellung definiert war, wurde das Projekt in 5 Phasen eingeteilt. **Phase 1:** Kamerarigbau, **Phase 2:** Auslösen/ Übertragung, **Phase 3:** Kalibrierung, **Phase 4:** Interpolation, **Phase 5:** Nachbearbeitung.

Diese Phasen sind nicht in chronologischer Reihenfolge zu verstehen. Sie konnten zeitgleich erfolgen oder sich überschneiden. Phase 1: hat Christian Neubauer übernommen, der sich anschließend mit Armin Voit und Johannes Kisel um Phase 3 gekümmert haben. Ich habe Phase 2 übernommen. Phase 4 hat Johannes Kisel übernommen. Phase 5 wurde von allen Projektteilnehmern durchgeführt.



Das Diagramm zeigt von mir eine subjektive Einschätzung des Aufwandes.

PROJEKTAUFGABEN

3.1 CHDK

Die Abkürzung **CHDK** steht für Canon Hacker Development Kit. Dabei handelt es sich um einen unabhängigen Open Source Firmware für digitale Kompaktkameras der Firma Canon. Durch diese Software wird der Funktionsumfang der Kamera erheblich erweitert. Die wichtigste Eigenschaft von CHDK ist, dass die Firmware der Kamera unangetastet bleibt. Die originale Firmware wird weder ersetzt noch verändert und so geht man kein Risiko oder sogar Garantiegefährdung ein. Zusatzfunktionen sind z.B. Aufnahme von RAW/DNG-Bildern, Schnelle Belichtungsreihen mit frei einstellbaren

Belichtungswerten und natürlich die für unser Projekt entscheidende Fernauslösung mit Hilfe einer Fernbedienung. Es gibt zwei Programmiersprachen uBasic und Lua. Mit Ihnen ist es möglich Skripte zu schreiben und so die Kamera zu programmieren wie z.B. die Kamera auszulösen bei Bewegung im Bild. Der Kreativität sind mit Programmierkenntnissen kaum Grenzen gesetzt. Besonders hervorzuheben ist die große Community die dieses Open Source Projekt am Laufen hält und aktiv weiterentwickelt.

3.1.1 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 meine Aufgabe mich mit den Sprachen vertraut machen und CHDK erstmals richtig verstehen. Also welche Möglichkeiten habe ich mit CHDK, wie funktioniert es, wie läuft die Installation ab und wie spiele ich die Skripte auf meine Kamera. An welche

Grenzen stoße ich, gibt es vielleicht Erfahrungen von Personen mit ähnlichen Projekten. Dies waren alles Fragen die ich mir zu Beginn stellen musste. 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

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
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 wird 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 wäre 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.

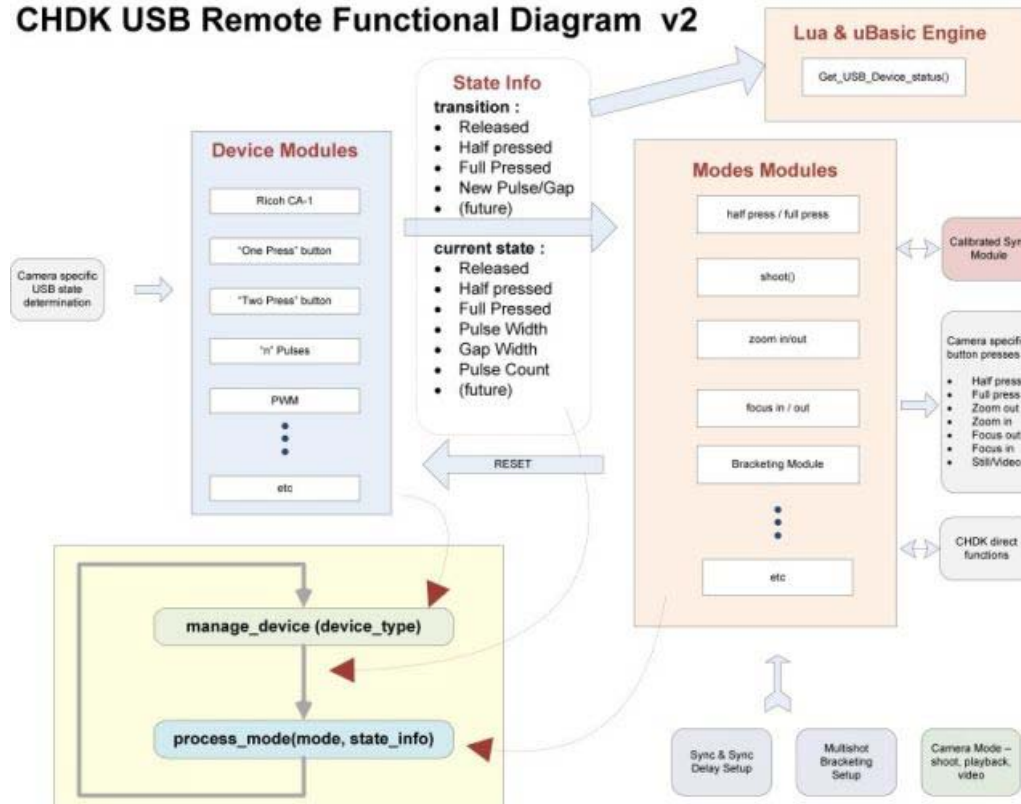
3.1.2 USB-Übertragung:

Nachdem die Grundfunktionen geklärt war und das laden der Skripte funktionierte war die große Frage „wie komme ich an meine Bilder? Wie lese ich die Kamera aus und wie wird dies programmiert?“

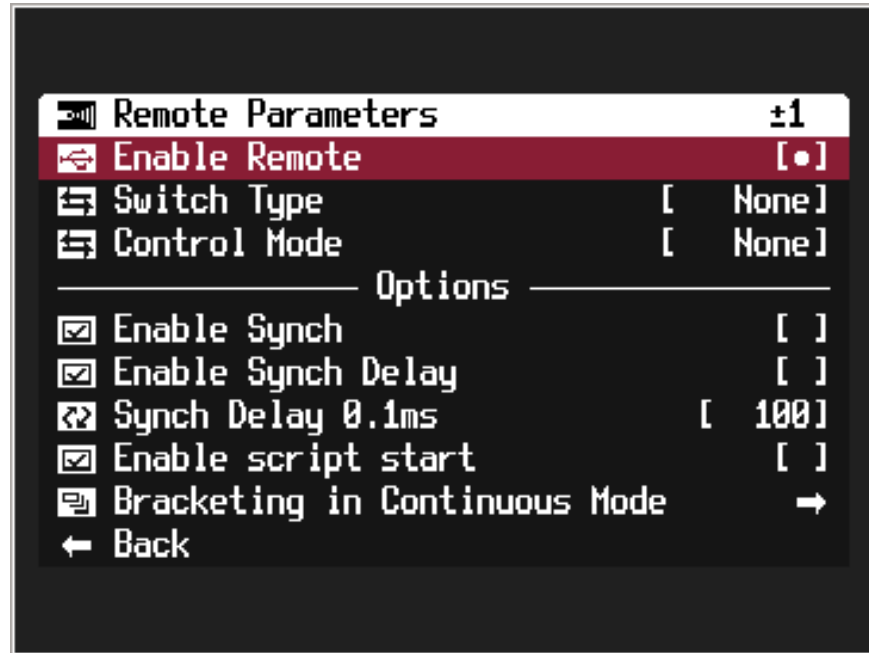
Auch für die USB-Übertragung kann man ein Script schreiben welches die Bilder der Speicherkarten an den Rechner sendet. Hierzu muss anders als bei dem Kamera Auslöse Skript die interne CHDK Einstellung „USB-Remote Operation“

deaktiviert werden. Nur so kann ein Zugriff auf die Speicherkarte erfolgen. Dies wird sich später noch als der große Knackpunkt herausstellen. Beim Auslösen muss diese Funktion aktiviert werden. Auf der nächsten Seite ist ein Funktion Diagramm gezeigt das den Ablauf von USB-Remote zeigt.

CHDK USB Remote Functional Diagram v2



Auf diesem Bild kann man einen Screenshot des CHDK-Menüs mit der Remote-Steuerung sehen. Auf dem Bild ist die Fernbedienung aktiviert, und so ist das Auslesen sowie PTP-Operationen nicht möglich, jedoch das synchrone Auslösen der Kameras.



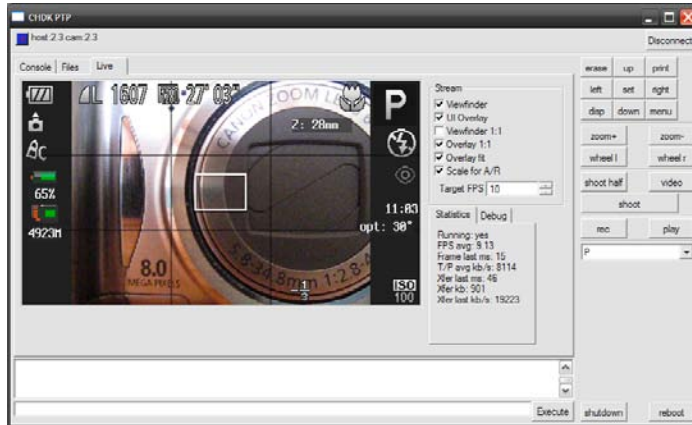
3.1.3

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. Ich traue dem PTP-Klienten eine Menge zu und denke eine Steuerung von mehreren Kameras wäre in einem zukünftigen Projekt zu realisieren und eine elegante Lösung für die Steuerung. Die Hauptprobleme dürften die Treiber sein, die bisher leider nur eine Kamera erkennen.

3.1.4 Auslösen und Übertragung

Von Beginn an stand außer Frage, dass wir CHDK nutzen wollten um alle 8 Kameras anzusprechen und synchron auszulösen. Dies war das oberste Ziel und musste sich in jedem Fall umsetzen lassen. Leider konnte ich erst relativ spät mit zwei Kameras testen und so die Schwachpunkte herausfinden. Nach Erhalt der zweiten Kamera habe ich die ersten Tests Richtung synchrones Auslösen gemacht und über einen USB-Hub, welches direkt an den Rechner angeschlossen war, die CHDK-Skripte ausgelöst. Bei Strom sollten die Kameras fokussieren und bei abgehender Flanke, also Strom **aus** die Kameras auslösen. Diese Erkenntnis hat sich aus diversen Tests ergeben, bei denen z.B. das Fokussieren zu kurz und der „shot“, also das Auslösen der Kameras nicht durchgeführt wurde. Hier kommt es zwar zu dem Fokussieren, jedoch bleibt die Auslösung der Kameras aus. Es hat sich als optimale Zeitspanne zum Fokussieren 2ms herausgestellt. Zu Beginn musste das Kabel manuell entfernt werden, dies sollte später durch einen Taster

oder einem anderen Trigger abgelöst werden. Nach ersten Verifizierungen der Synchronität durch einen LCD-Bildschirm mit einer Stoppuhr konnten wir die



Genauigkeit auf 17ms angeben. Dies hat etwas mit der maximalen Bildübertragung bei 60 Hz des hier verwendeten Bildschirms zu tun. Mit diesem

Ergebnis waren wir zuerst zufrieden, und wollten im späteren Verlauf des Projektes bei genügend Zeit eine genauere Verifizierung realisieren.

Die optimalste Lösung wär gewesen Alle Kameras simultan auszulösen und kurz darauf ohne Zutun eine Datenübertragung der Bilder an den Rechner zu initialisieren. Leider war dieses Vorhaben mit unseren Mitteln nicht möglich. Es hat viel Zeit gekostet dieses Ergebnis zu ermitteln. Über die geladenen Skripte ist ein Auslösen und gleichzeitiges Übertragen aufgrund der Einstellung, Fernbedienung aktivieren/deaktivieren nicht möglich. Man kann nicht beide Skripte in einem großen Script zusammenfassen so bleibt es unausweichlich an der Kamera das Script sowie die Funktion „USB-Remote aktivieren/deaktivieren“ von Hand umzustellen. Der PC bzw. ein USB-Host bringt zwangsweise 5 Volt Betriebsspannung am USB-Port mit. Damit ist parallele Remote- und Datenübertragungsfunktion nicht möglich. Offensichtlich wurde bei der USB-Entwicklung vergessen, eine softwareseitige Schaltung der USB-Stromversorgung

zu integrieren. Jedenfalls habe ich bisher keine Möglichkeit gefunden, die USB-Stromversorgung mit dem PC ein- und auszuschalten.

Aus diesem Grund wurden auch im CHDK Remote-Steuerung und Datenübertragung per Option getrennt. Die Andere Möglichkeit die ich fokussiert habe als ich nur eine Kamera zur Verfügung hatte war über den PTP-Klienten. Durch eine vorher definierte Skriptfolge mit einem Delay war es mir möglich direkt in der Konsole die Kamera auslösen und das Bild, (Bildname muss vorher bekannt sein) an einen vorher definierten Ort zu speichern. So war ein auslösen sowie Speicherung möglich. Nachdem die zweite Kamera für Testzwecken eingetroffen ist, habe ich mich damit beschäftigt beide Kameras anzusteuern. Leider war dies nicht ohne weiteres Möglich. Mein Rechner konnte leider nur mit einer Kamera eine Verbindung aufbauen. Die Zweite Kamera wurde nicht erkannt. Ich habe viel Arbeit in den Versuch gesteckt dass zu Ändern jedoch ohne Erfolg. Es hätte zu viel Zeit gekostet die Treiber umzuschreiben und so eventuell mehrere

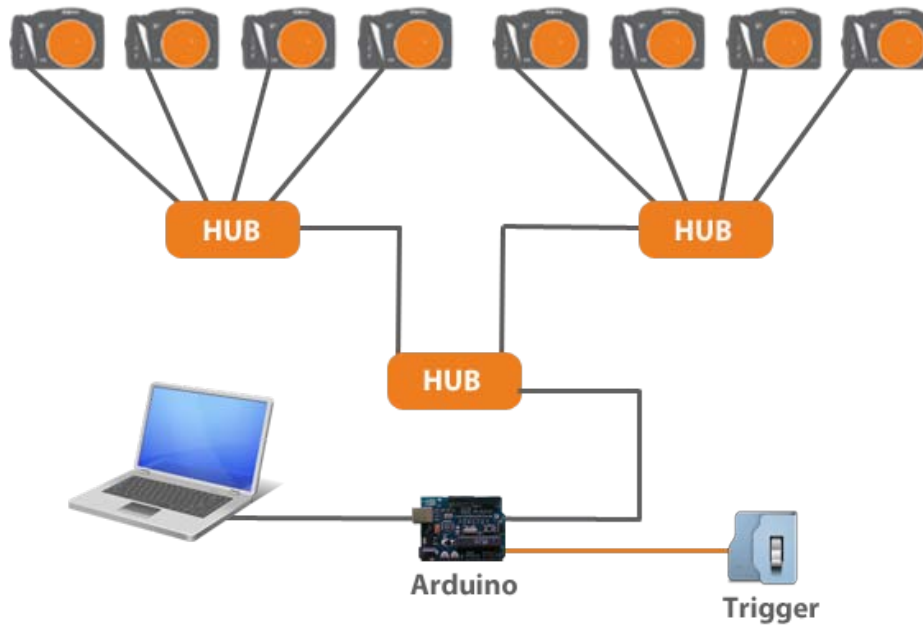
Kameras anzusprechen. Ich denke die Möglichkeit besteht und in einem weiteren Projekt könnte in diese Richtung geforscht und entwickelt werden.

3.2 Schaltung

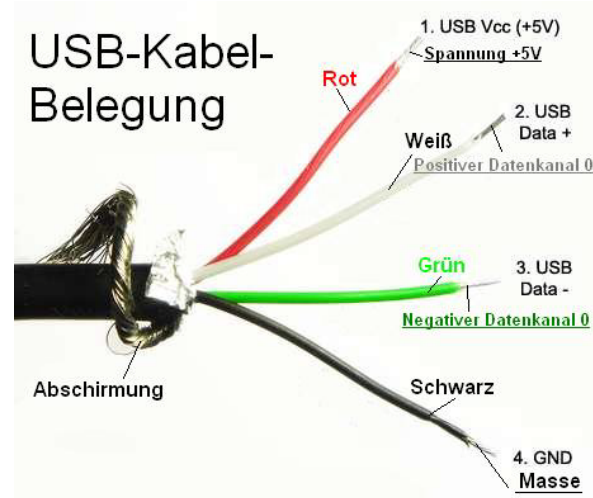
Die zweite große Aufgabe mit der mich nach dem CHDK beschäftigt habe war die Schaltung unserer Kameras. Das Budget spielte eine große Rolle und musste bei dieser Aufgabe im berücksichtigt werden. Die Schaltung sollte mit einfachen finanziellen Mitteln realisiert, jedoch die Möglichkeit für eine spätere 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.

3.2.1 Aufbau

Der Aufbau unserer Schaltung kurz illustriert.



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 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 ein Arduino-Mikrocontroller angeschlossen wird. Dies ist das zentrale Bauteil mit dem eine Vielzahl kreativer Möglichkeiten realisiert werden können. Im nächsten Abschnitt wir genauer auf diesen Mikrocontroller eingegangen. Zahlreiche Schaltungen bzw. Trigger könnten so angesteuert und ausgelöst werden. Wir haben uns für einen

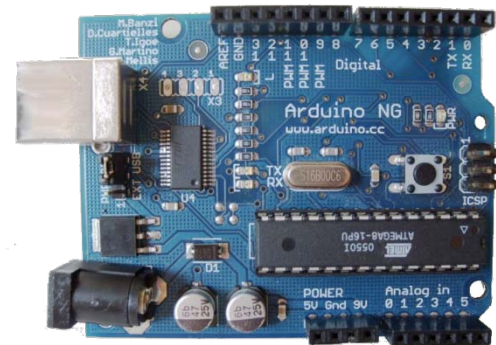
Taster entschieden da hier gezielt ohne Komplikationen oder äußere Einwirkungen ausgelöst werden kann. Als Zusatz haben wir uns vorbehalten, die Schaltung durch ein akustisches Signal wie z.B. ein Handschlag auszulösen. Der Arduino-Mikrocontroller bezieht seinen Strom aus einem Notebook wie in der Zeichnung 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 ihn 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.



3.2.2 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.

3.2.2.1 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 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.

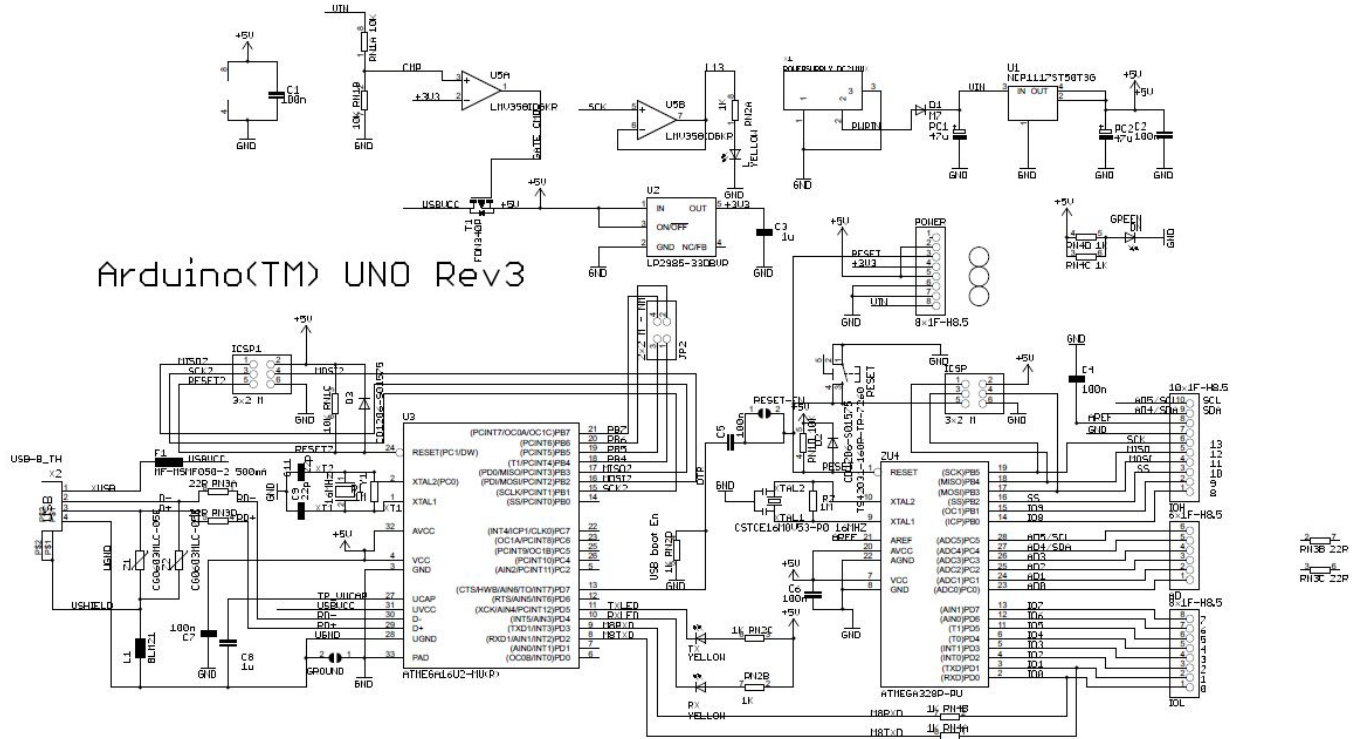
Auf dem Board befindet sich ein ATmega16U2 Mikrocontroller als USB-Seriell-Konverter. Diese neue USB Anbindung besitzt ein paar technische Neuerungen: Der ATmega16U2 lässt sich auch umprogrammieren, man hat dadurch neue Anwendungsmöglichkeiten geschaffen. Das Board kann so vom Rechner als Keyboard, Joystick, 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





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.

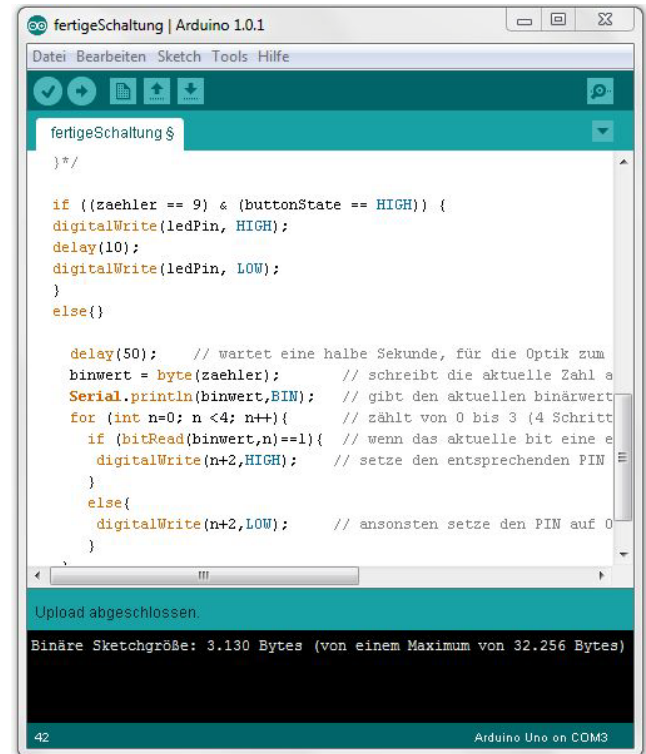
Zudem bietet der Arduino-Mikrocontroller einen USB-Überstromschutz, der den Computer vor über 500 mA schützt. Die Sicherung trennt die Verbindung bis die kurz- oder Überlast entfernt ist.

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.



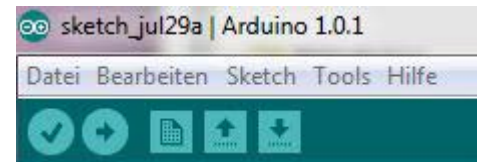
3.2.2.2 Programmierschnittstellen

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.

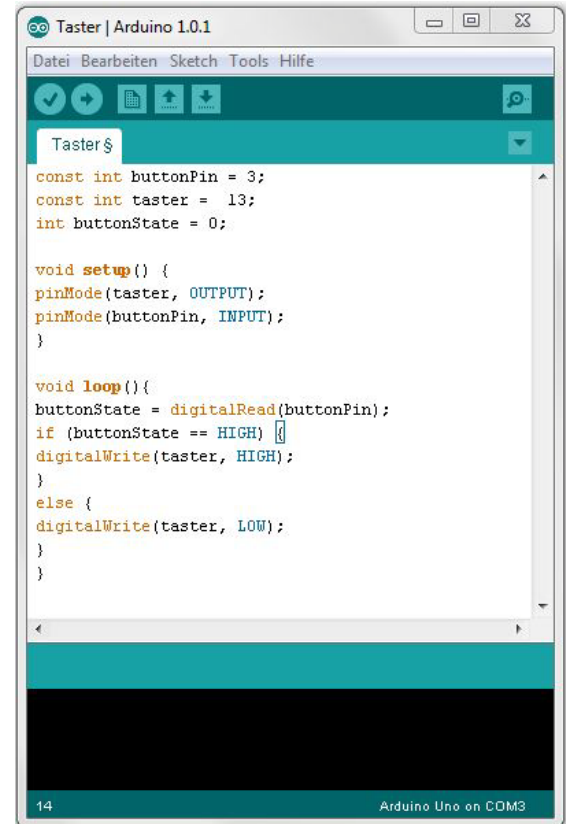
Durch den zweiten Button im Hauptmenü **Upload** (der Pfeil nach rechts) kann der programmierte Code direkt auf den Mikrocontroller übertragen werden. Dies dauert nur wenige Sekunden und wird visuell durch eine blinkende LED angezeigt. So kann relativ leicht Programmiert und sofort an dem Mikrocontroller 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.

3.2.2.3 Code:

Der in der Abbildung rechts gezeigt Code war meine 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 benötigen 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



```
Taster | Arduino 1.0.1
Datei Bearbeiten Sketch Tools Hilfe

Taster$
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);
  }
}
```

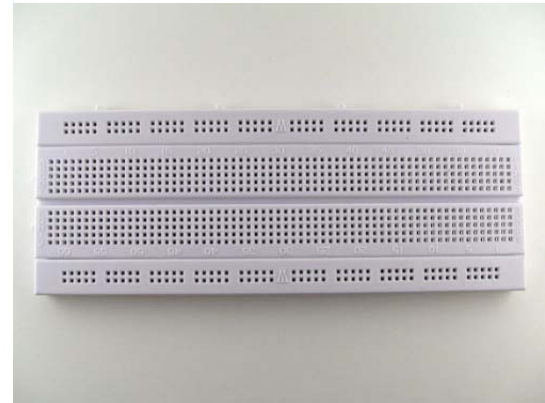
14 Arduino Uno on COM3

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.

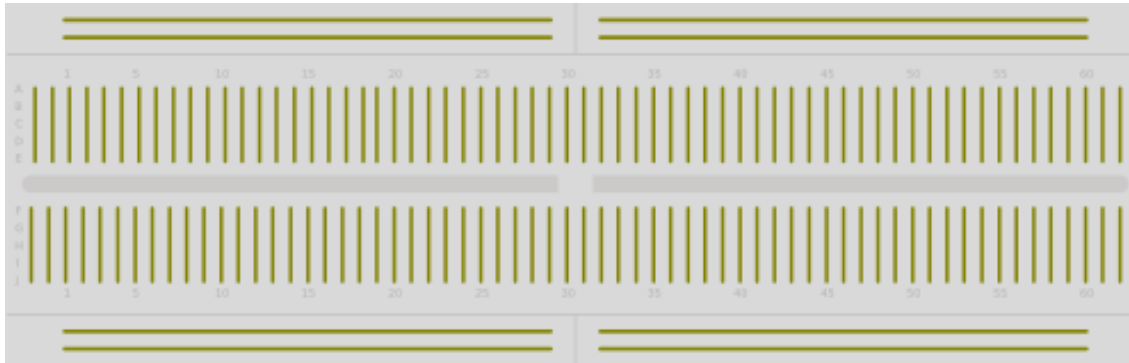
3.2.2.4 Schaltungsaufbau:

Zu Beginn habe ich leider nur den Arduino Mikrocontroller sowie das USB Kabel erhalten. Leider waren keinerlei Beschreibungen oder andere Informationen enthalten. Aus diesem Grund musste ich alle Informationen im Netz suchen und Teile Vorort im Geschäft kaufen. Für den ersten Test habe ich eine Sammlung unterschiedlicher LEDs gekauft um einen Eindruck über das Funktionsprinzip zu bekommen. Durch die ersten gesammelten Erfahrungen und Wissen wurde mir klar 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.

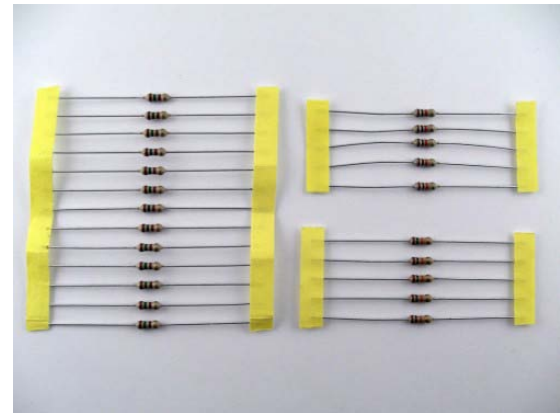
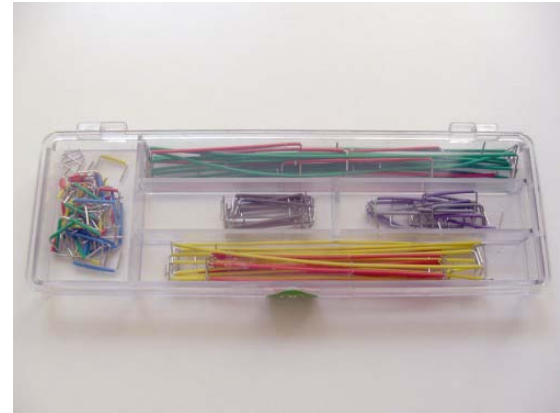
Die Steckplatine ist die Grundlage für alle Schaltungen, die man mit dem Arduino aufbauen kann. Ohne zu löten kann man so schnell beliebige Schaltungen zusammenstecken und problemlos verändern oder erweitern. Die Lücke in der Mitte der



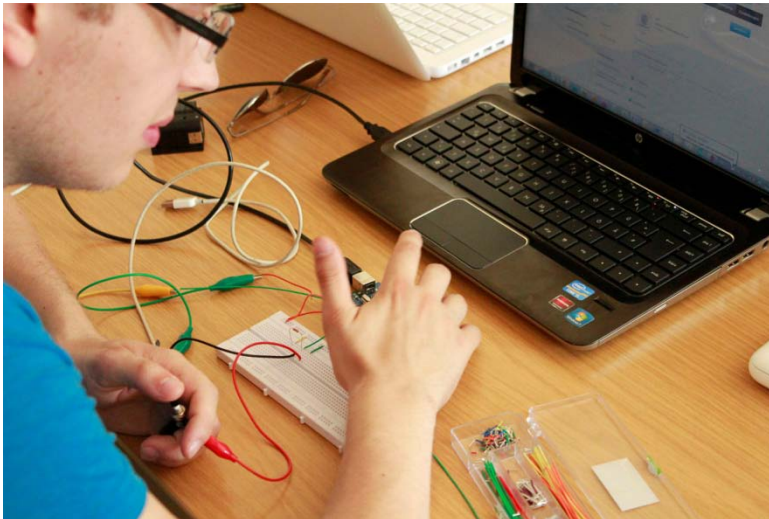
Steckplatine hat genau den passenden Abstand, um darauf ICs (Mikrochips) im sogenannten DIP-Gehäuse zu stecken, deren Anschlüsse man dann gut von Hand verkabeln kann. Da ich leider gar keine Grundkenntnisse in der Elektrotechnik besaß war das ganze Gebiet der Schaltungen neu für mich und ich musste mir Grundlagen selbst beibringen. Aus diesem Grund musste ich erstmal verstehen wie eine solche Steckplatine verbunden ist. Dies wird in der nächsten Illustration verdeutlicht.

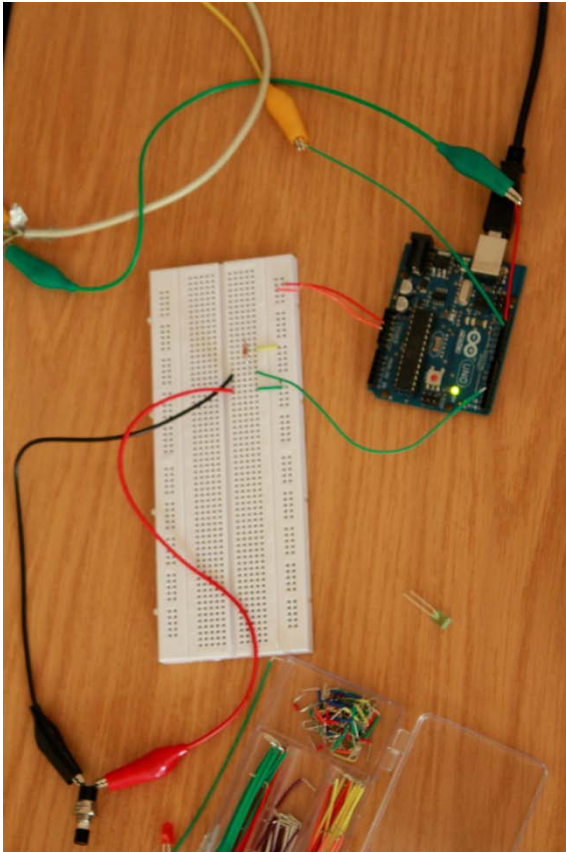


Meine zu Beginn erstellten Kabel die von Hand verlötet wurde, konnte ich durch Steckbrücken ersetzen. So war es bedeutend einfacher eine Verbindung in den kleinen Löchern der Steckplatine aufzubauen. Ein ganz anderes Themengebiet waren Widerstände. Ich wusste zwar dass es so etwas gab, nur hatte ich 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 Bauele-



mente) 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.

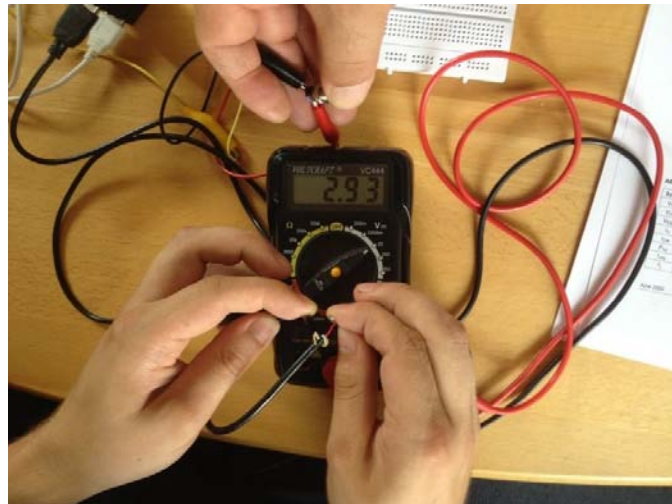




Das Bild zeigt die erste Schaltung zu dem vorigen gezeigten Code. Das mit dem Rot/Schwarzen verbundenen Kabel zeigt den Taster, der dafür zuständig ist das Signal auszulösen. Der Ausgang hier mit einem Gelben und Grünen Kabel gezeigt ist mit dem +5V Spannung und Masse eines USB Kabel verbunden. In den ersten Tests konnte nur mit einer Kamera getestet werden. Diese wurde jedoch wie gewünscht fokussiert und bei Taster loslassen ausgelöst.

3.2.2.5 Probleme:

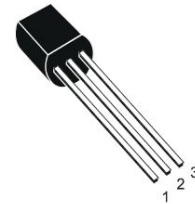
Leider musste ich feststellen, dass bei dem Versuch alle 8 Kameras auszulösen, die Kameras weder fokussiert noch ausgelöst haben. Zu Beginn stand mir kein Messgerät zur Verfügung so dass ich nur durch Versuche den Fehler lokalisieren musste. Es stellte sich heraus, dass bei Einsatz eines USB-Hubs die 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 ausreicht um das CHDK-Script zu aktivieren. Das Problem liegt an den USB-Hubs die anscheinend mehr Strom benötigen als zunächst angenommen.

3.2.2.6 Problemlösung:

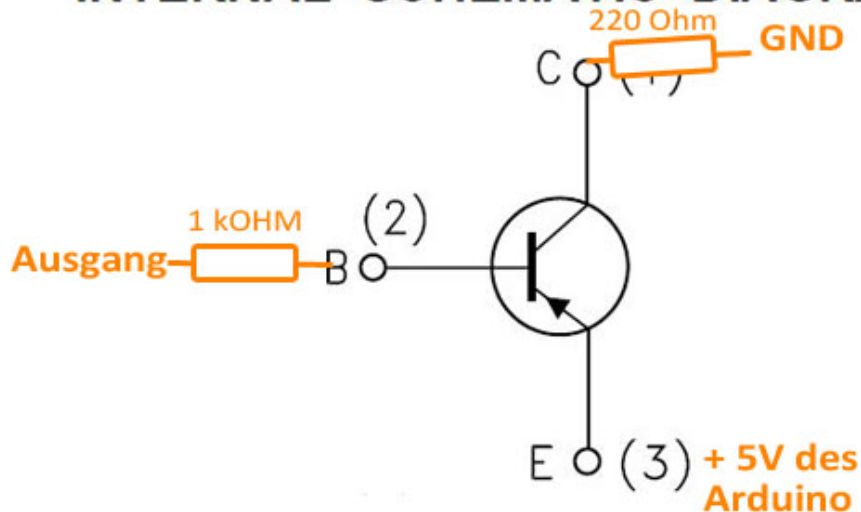
Die große Frage war jetzt wie ich das Signal verstärkt bekomme so dass alle Kameras 3-5V empfangen. Nach anfänglicher Recherche und Hilfe der Projekt betreuenden Professoren fiel der Begriff „Transistor“. Dieser verstärkt das elektrische Signal ohne dabei mechanische Bewegungen auszuführen. Es gibt eine Fülle von Transistoren, ich entschied mich für ein PNP Transistor **BC327-40** der laut Datenblatt ausreichen sollte. Zudem habe ich 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



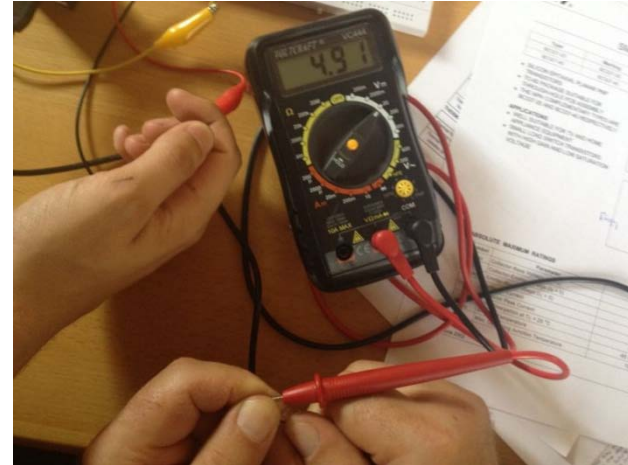
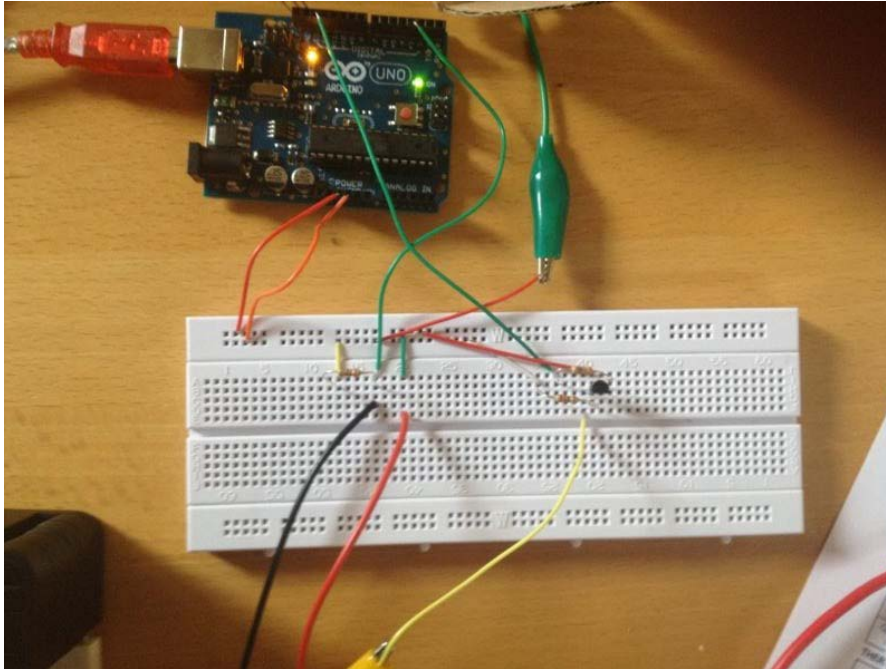
verzerrungsfrei verstärkt. Man kann sagen es ist ein Schaltungsverstärker, mit dem sich kontaktlos kleine bis mittlere Leistungen schalten lassen. Er ist jedoch mechanischen Schaltern in einigen Punkten unterlegen. Der Widerstand bei einem mechanischen Schalter beträgt 0 Ohm und es entsteht keine Verlustleistung. Anders als bei dem Transistor, der nicht verlustleistungslos 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 mich war besonders schwierig Schaltbilder zu lesen bzw. zu deuten da ich 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 Widerstände 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 Widerstände eingezeichnet.

INTERNAL SCHEMATIC DIAGRAM

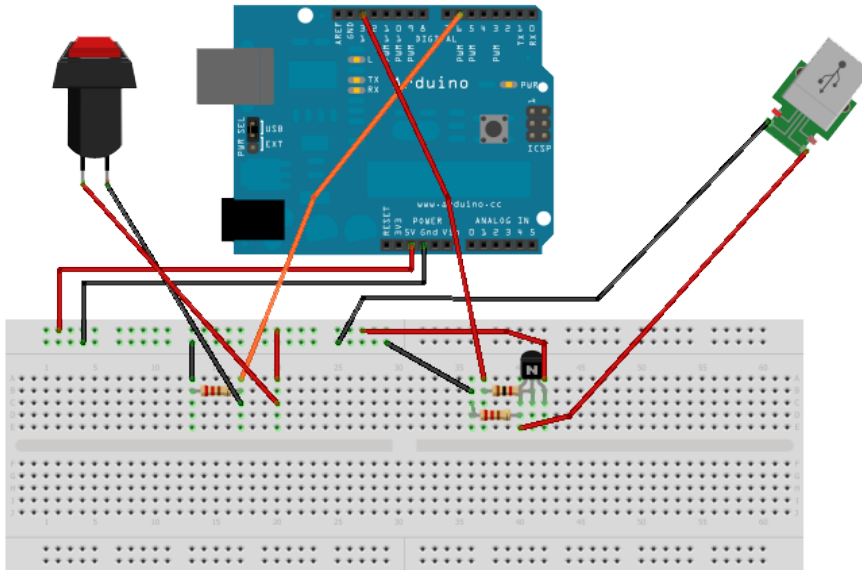


Wichtig ist die Erkenntnis, dass diese Art von Transistoren das komplette Signal invertiert. Daher muss der Code des Arduino-Mikrocontrollers angepasst bzw. geändert werden. Das dieses strukturierte Vorgehen beim Bau einer Schaltung nicht von Beginn so funktionierte ist denk ich normal. Zu Beginn musste ich mich erst einarbeiten und Fehler machen z.B. habe ich erstmals drauf losgesteckt und direkt ein Transistor das Leben genommen. Ich wollte endlich sehen dass etwas passiert. Relativ schnell merkt man, dass wenn man keine Ahnung von Widerständen 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 konnte ich im Praxistest bestätigen.

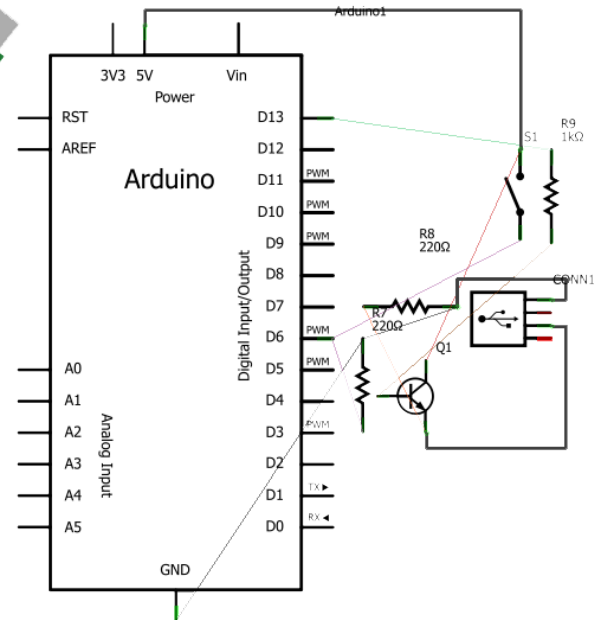


Anschließende Messung ergaben
4.91V

Arduino Steckplatine



Arduino Schaltplan

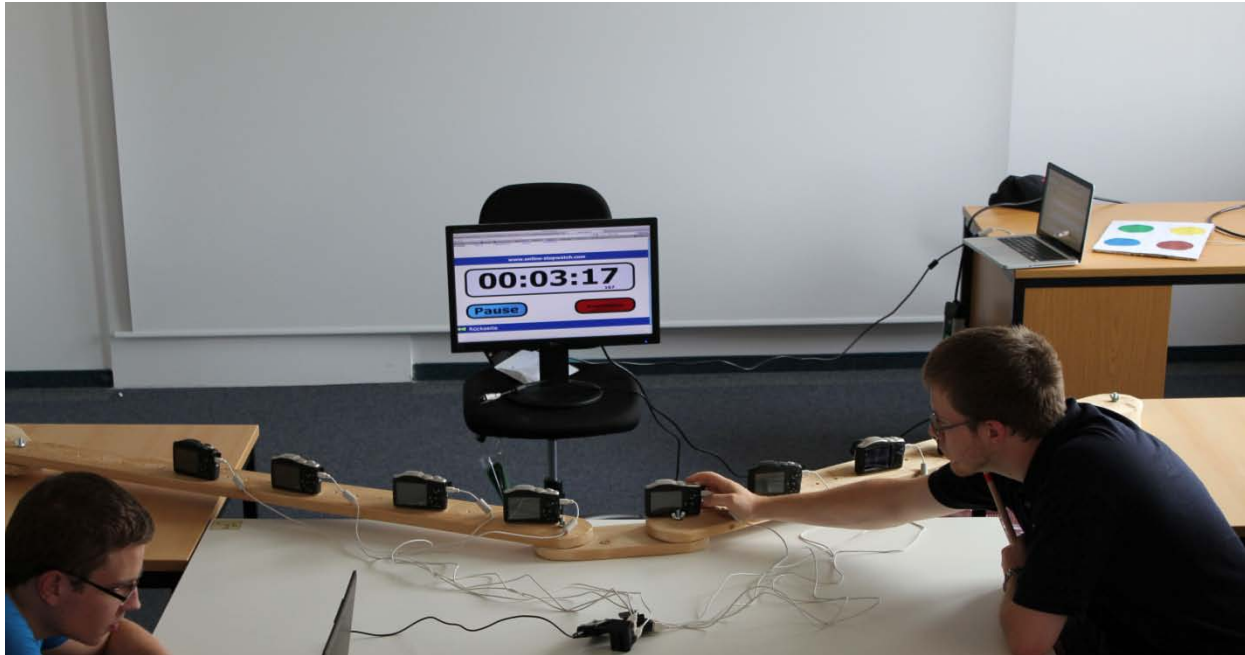


3.2.3 Synchronität:

Das nächste große Thema mit dem ich mich Beschäftigt habe 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 habe ich mich auf zwei Arten damit befasst.

3.2.3.1 Verifizierung LCD:

Dieses Vorgehen habe ich 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 wollte ich 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.



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 von mir um die eigene Achse konnte man eindeutig extreme Unterschiede erkennen. Ich habe mich 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 machte ich mich zusammen mit Herrn Voit an die Problemfindung. Es stellte sich schnell heraus, dass es etwas mit der Einstellung auf den Kameras 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 genau ist. Aus diesem habe ich mich entschlossen die Genauigkeit der Verifizierung zu Verbessern.

3.2.3.2 Verifizierung Arduino:

Die zweite Möglichkeit der Verifizierung mit der ich mich befasst habe ist mit Hilfe des Arduino. Ich dachte mir, falls wir schon die Möglichkeit haben mit einem Mikrocontroller zu arbeiten der im Stande ist jeden Pin einzeln anzusprechen, wieso dann nicht auch Nutzen. Meine 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 habe ich 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 ich genau 7 Zustände abdecken. Bei 6 LEDs wird jedoch die Steckplatine sehr unübersichtlich da jede LED einzeln angesteuert wird. Aus diesem Grund habe ich die Schaltung erweitert und einen Binärzähler gebastelt. So war es mir möglich mit nur 4 LED's, 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 mir 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. Meine 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 habe ich den Programmiercode angepasst und so programmiert das 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 100 ms anfangen 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);

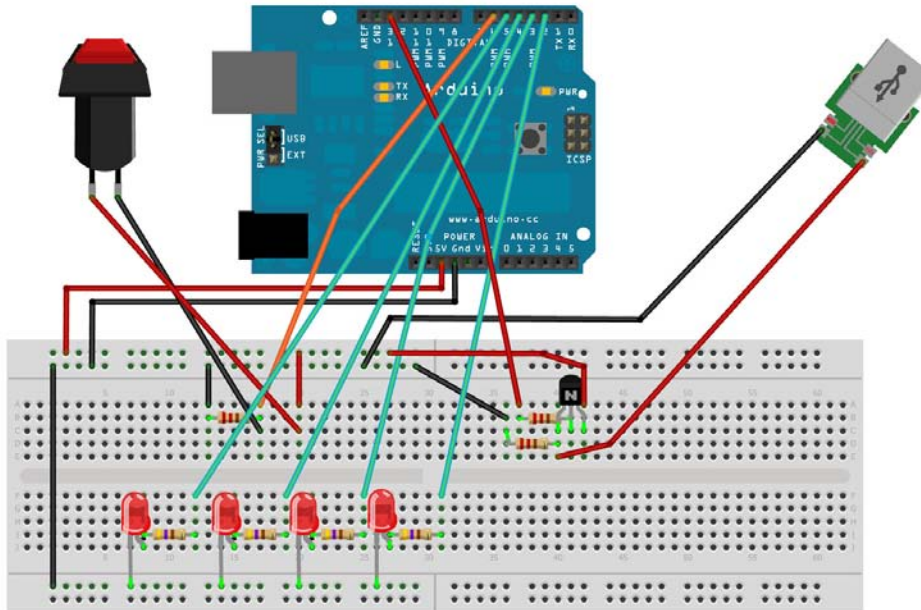
    buttonState = digitalRead(buttonPin);
```

```
/*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);  
binwert = byte(zaehler);  
Serial.println(binwert,BIN);  
for (int n=0; n <4; n++){  
    if (bitRead(binwert,n)==1){  
        digitalWrite(n+2,HIGH);  
    }  
    else{  
        digitalWrite(n+2,LOW);  
    }  
}  
}}}
```

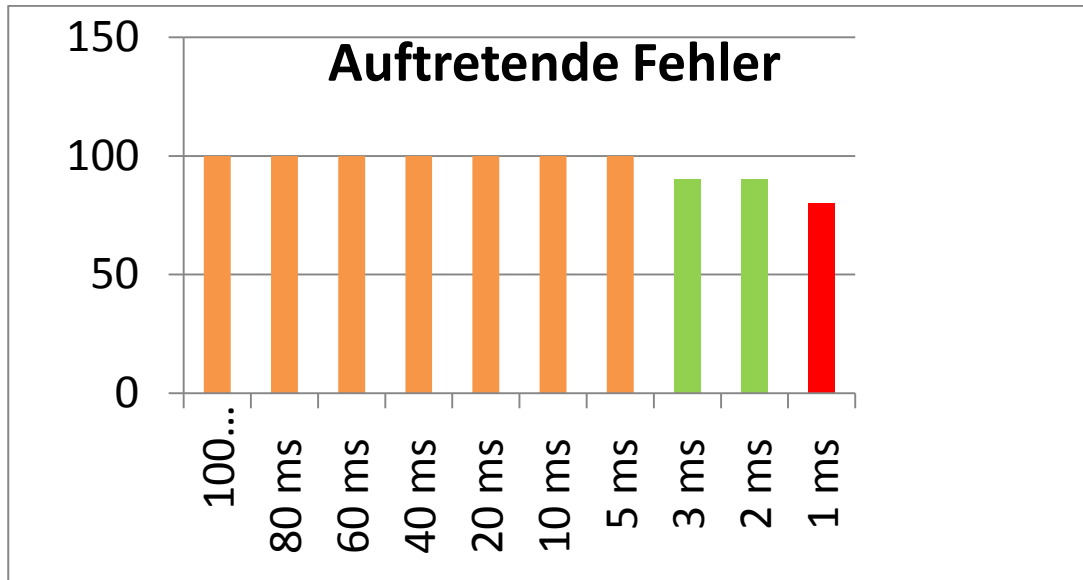
// wartet eine 50ms, für die Optik zum testen
// schreibt die aktuelle Zahl als binärwert in binwert
// gibt den aktuellen binärwert Seriell aus (nur zum debuggen)
// zählt von 0 bis 3 (4 Schritte) um jedes einzelne bit des halbbbytes abzufragen
// wenn das aktuelle bit eine eins ist
// setze den entsprechenden PIN auf 1

// ansonsten setze den PIN auf 0

Aufbau der Schaltung mit den jeweiligen Leuchtdioden.



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



Erst bei 3 ms 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 2 ms. 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.



3.3 Kalibrierung

Nachdem die Schaltung sowie das Auslösen (ohne Arduino) bereits mit zwei Kameras funktionierte und ich das erste erfolgreiche Synchronisationsergebnis vorlag habe ich mich zusammen mit den anderen Gruppenteilnehmern an die Kalibrierung gesetzt. Ich musste sowieso auf die restlichen 6 bestellten Kameras warten um mit der Schaltung und der Verifizierung fortzufahren. Zu Beginn mussten viele theoretische Grundlagen erarbeitet werden.

3.3.1 Einarbeitung OpenCV

Der erste Schritt in dieser Richtung war die Einarbeitung und Installation von OpenCV. Hierbei habe ich Visual Studio 2010 Ultimate (x86) verwendet um die Basis zur späteren Installation von OpenCV und OpenFrameworks zu schaffen. Leider war ich der einzige ohne Mac Rechner und bei Problemen die aufgetreten sind konnte mir niemand helfen. Im Verlauf dieser Phase kristallisierte sich

heraus, dass die anderen Gruppenmitglieder ausschließlich auf Ihren Mac Rechner arbeiten wollten und ich außen vor gelassen wurde. Visual Studio ist sehr komplex und Projekte werden anders aufgebaut. Schließlich lief durch zahlreiche Tutorials OpenCV auf meinem Notebook.

3.3.2 Mathematische Einarbeitung

Als nächstes musste sich in die Mathematik eingearbeitet werden. Hierzu habe ich das Mathe Skript aus dem dritten Semester Revue passieren lassen. Viele Zusammenhänge bezüglich Affine Abbildungen und Transformationen sind mir so klar geworden. Zuerst hat es viel Zeit gekostet zu verstehen was den die Kalibrierung überhaupt macht und wozu sie gut ist. Das Verständnis hat einfach gefehlt. Anschließend habe ich mitgeholfen das Konzept aufzubauen, welche Marker wir brauchen und welche Theoretischen Grundlagen dahinter stecken.

Bei Beginn der Programmierung unserer Software ist das Thema des Arduino Mikrocontroller wieder aufgekommen. Da ich leider keinen Mac besitze wurde ich für die Aufgabe bestimmt. Im späteren Verlauf des Projektes hatte ich leider keinerlei Einfluss auf die Kalibrierung. Aus diesem Grund habe ich bei der Kalibrierung nur einen geringen Anteil beigetragen.

3.4 Weitere Aufgaben

Bei dem Projekt gab es weitere Aufgaben die ich neben meiner Hauptbeschäftigung übernommen habe. Angefangen von der Idee bei der Markerkennung einen LCD einzusetzen und so die RGB Werte direkt live zu verändern bis hin zur Gestaltung. Im Folgenden möchte ich kurz auf die wichtigsten Aufgaben eingehen.

3.4.1 Logo Erstellung

Nachdem die Tage bis zur Vorpräsentation näher kamen und wir keinerlei Logo oder gestalterische Abbildungen vorzuweisen hatten, entschloss ich mich das Logo für das Projekt zu erstellen. Als Programme dienten mir Illustrator zur Vektorisierung der Kamera sowie Photoshop zur Erstellung der Schrift.



3.4.2 Powerpoint Präsentation/Layout

Desweiteren habe ich mich um Teile der Präsentation sowie das Layout gekümmert.

Ich bin der festen Überzeugung, dass auch wenn man ein eher trockenes, theoretisches Projektthema hat, trotzdem eine interessante und unterhaltende Präsentation abhalten kann. So habe ich versucht durch Illustrationen oder Animationen (Schaltung mit Binärzähler) den Stoff etwas aufzufrischen und durch Abbildungen verständlich zu machen.

3.4.3 Hardware Kalibrierung/Grid Erstellung

Ein weiteres Thema was ich zusammen mit Herrn Voit übernommen habe war die Hardwarekalibrierung anhand eines eigens dafür erstellten Grids. Ein Grid ist ein Raster oder ein Gitter welches selber in Text2Grid oder einem anderen Programm erstellt werden kann. Es muss mit dem Dateinamen **.GRD** enden und in einem CHDK / Raster / Sub-Ordner gespeichert werden. Über das OSD-Menü mit der Option „Grid Settings“ kann es verwendet werden.

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.

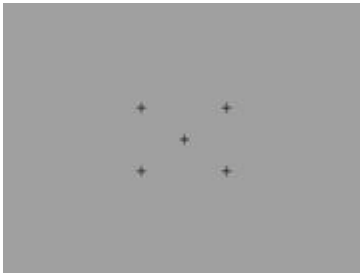
```
@ Title Goldenen Kreuz-Ratio

@ 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 aussieht.



Dieses Bild zeigt die Bildschirmansicht unserer Kamera mit dem von uns aktivierten Grid.

Durch das Fadenkreuz war es uns Möglich die Kameras relativ genau zu kalibrieren und auf einen Punkt auszurichten. Die spätere Softwarekalibrierung sollte Ungenauigkeiten beheben.



4. Reflexion

4.1 Soll/Ist Vergleich auf der Grundlage des Projektplan

SOLL	IST
Rig für min. 8 Kameras	✓
CHDK Skript	
- Auslösen	✓
- Bildübertragung	✓
Hardwarekalibrierung	
- Grid (Sucherzielkreuz)	✓
- Nachweis Synchronität	✓
Softwarekalibrierung	
- Markererkennung	✓

- Übereinanderlegen der Bilder - lineare Interpolation mit Alphablending	✓ ✓
Vergleich mit professioneller Software Twixtor	(Subjektiver Vergleich bei Nebeneinanderstellung)
optional	
Fernauslöser mit Hilfe von Arduino	✓

4.2 Quellen

- <http://forum.chdk-treff.de> Forum Austausch/Informationssuche
- <http://chdk.wikia.com> Informationssuche CHDK
- <http://chdk.bplaced.net> Forum Austausch/Informationssuche
- <http://arduino.cc> Hersteller, Datasheets
- www.mikrocontroller.net Elektrotechnische Grundlagen
- Grundlagen der Elektrotechnik, Gert Hagmann, ISBN-10: 3891047215
- Wikipedia Informationssuche

4.3 Persönliche Fazits

Zum Ende meiner Projektarbeit möchte ich gerne mein persönliches Fazit abgeben.

Zum positiven kann gesagt werden das es ein sehr lernreiches Projekt war. Dies bezieht sich auf der einen Seite auf die Arbeit die ich übernommen habe, auf der anderen Seite die Feststellung wie wichtig richtiges Projektmanagement in einem Projekt ist. Zu Beginn des Projektes war nicht vollkommen klar in welche Richtung das Projekt laufen wird. Ich habe es so verstanden dass es ein Versuchsprojekt werden soll bei dem die ersten Grundbausteine für spätere Projekte geliefert werden sollen. Ich bin von einer praktischen Aufgabe ausgegangen die sich nicht hauptsächlich mit der Programmierung beschäftigt. Ich selber bin kein Programmierer. Das ist auch nicht mein Anreiz. Ich kann ein Code nachvollziehen und mich einarbeiten. Dies kostet sehr viel Zeit und Arbeit. Anders als es vielleicht bei Informatiker bei denen die Programmierstrukturen nach der Zeit einfach

sitzen. Ich selber habe mich noch nie mit Elektrotechnik beschäftigt und hatte in dieser Richtung gar keine Vorkenntnisse. Aus diesem Grund musste ich mir alles selber durch Videos und Bücher beibringen. Im Ganzen finde ich es gut sich auch mal mit diesen Dingen zu beschäftigen. Es ist schon etwas seltsam, später Ingenieur von der Fakultät Elektrotechnik zu sein und eventuell Interaktive Präsentationen oder ähnliches zu entwerfen und keine Ahnung von einer einfachen Schaltung zu haben. Aus diesem Grund empfinde ich meine Aufgabe nicht als Schlimm oder besonders uninteressant.