



Fakultät Elektrotechnik Feinwerktechnik Informationstechnik efi
Projekt MatRigx

Prüfungsstudienarbeit von

Christian Neubauer
Matr.Nr.: 2118116
B-ME 6

Projekt MatRigX – Bullettime Rig

Sommersemester 2012

Bestätigung gemäß § 35(7) RaPO

Neubauer, Christian

Ich bestätige, dass ich die Prüfungsstudienarbeit mit dem Titel:

Projekt MatRigX – Bullettime Rig

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 habe

Datum: 10.02.2012

Christian Neubauer

Abstract

In meiner Projektstudienarbeit möchte ich auf unser Projekt MatRigX zurückblicken und den Verlauf sowie meinen Beitrag zu diesem Projekt darstellen.

Zuerst werde ich tabellarisch die Projektaktivitäten zusammenfassen.

Anschließend umreiße ich die Ausgangssituation und die Projektidee und erkläre die wichtigen Punkte wie die von uns genutzten Software und Hardwarekomponenten bevor ich explizit auf meine Hauptaufgaben , den Rigbau, und die Realisierung in Openframeworks und OpenCV eingehen.

Besonders möchte ich hierbei auf die Realisierung von linearer Interpolation mit Alphablending eingehen.

Weiter werde ich auf den gesamten Projektablauf, die Projektplanung und die Zusammenarbeit mit den anderen Teammitgliedern eingehen, sowie verworfene Ansätze darstellen.

Abschließen möchte ich diesen Bericht mit einem Fazit das den aktuellen Stand mit unseren ursprünglichen Plänen abgleicht und mögliche Erweiterungen oder Fortführungen erörtern.

Projekt - Roadmap

After Effects

Tätigkeit	Dokument	Beteiligung
Video Rendering der Aufnahmen	../Bilder/ ../Video/	Kiesel
Tracking und Kalibrierung der 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 etc.	../Code/	Kiesel/ Neubauer/ Voit
Laden der Bilder in openCV	../Code/	Kiesel/ Voit
Farberkennung/ Markererkennung	../Code/	Kiesel/ Neubauer/ Voit
Morphologische Transformation 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		Voit
Dokumentation des Codes/Code/	Kiesel/ Neubauer/ Voit
Materialbeschaffung		Neubauer/ Siemer

Inhaltsverzeichnis

[Projekt MatRigX – Bullettime Rig](#)

[Bestätigung gemäß § 35\(7\) RaPO](#)

[Abstract](#)

[Projekt - Roadmap](#)

[After Effects](#)

[Auslösen der Kameras](#)

[Kalibrierung/Hardware](#)

[Kalibrierung/Software](#)

[Präsentation des Projekts](#)

[Projektorganisation](#)

[Inhaltsverzeichnis](#)

[Bericht](#)

[Einleitung - Ausgangssituation & Projektidee](#)

[Erklärung: Bullettime](#)

[Einschub: Canon Powershot SX 130 IS](#)

[Einschub: CHDK \(Canon Hack Developers Kit\)](#)

[Einschub: Rig](#)

[Phase 1: Kamerarigbau](#)

[Phase 2: Auslöser/Fernbedienung](#)

[Einschub: Arduino](#)

[Phase 3: Kalibrierung](#)

[Einschub: openCV](#)

[Phase 4: Interpolation und Ausgabe](#)

[Einschub: openFrameworks](#)

[Meine Aufgaben:](#)

[Rigbau](#)

[Markererkennung und Transformation](#)

[Implementierung des openCV Codes in OpenFrameworks](#)

[Vorschau - Lineare Interpolation mit Alphablending](#)

[Weitere Aufgaben](#)

[Reflexion Projektablauf](#)

[Quellen- und Literaturverzeichnis:](#)

[Anhang](#)

[Soll/Ist Vergleich auf der Grundlage des Projektplans](#)

Bericht

Einleitung - Ausgangssituation & Projektidee

Die Ausgangssituation für unser Projekt, dem wir den Namen "MatRigX" gegeben haben, war die Projektarbeit die im 6. Semester des Studiengangs Media Engineering, an der Georg-Simon-Ohm Hochschule vorgesehene ist.

Das Ziel einer solchen Projektarbeit ist die Konzeptionierung und die Realisierung einer Idee innerhalb einer Gruppe.

Die Projektarbeit erstreckte sich über das komplette Semester.

Die Mitglieder unserer Projektgruppe waren Johannes Kiesel, Christian Neubauer, Christian Siemer und Armin Voit.

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

Die Grundidee des Projekts war das Nachbilden der besonders aus den Matrix-Filmen bekannten Kamerafahrt im zeittoten Raum, die in "Fachkreisen" auch als Bulletpoint bezeichnet wird.

Erklärung: Bulletpoint

Der Begriff "**Bullet Time**" bezeichnet in der Filmkunst einen Spezialeffekt, bei dem der Eindruck einer **Kamerafahrt** um ein in der Zeit eingefrorenes Objekt (zeitot) 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. Neben der Verwendung in Actionfilmen, bekanntestes Beispiel hier wohl sicher die Matrix Filmreihe, wird Bullet Time auch als Spielelement in Videospielen, wie z.b. der Max Payne-Reihe, verwendet.

Ein ähnliches Verfahren, aber mit Ultrakurzzeit-Belichtung und Zwischenbildberechnung, ist die Frozen Reality.

Der Ausdruck Bullet Time ist ein eingetragenes Warenzeichen von Warner Bros., dem Distributor des Kinofilms Matrix. Vorher war es ein Warenzeichen von 3D Realms, dem Produzenten der Max-Payne-Spiele.

Man bezeichnet diese deshalb als "zeittot" da aufgrund der angewandten Technik während der Kamerafahrt die Zeit auf den Bildern stillsteht. Um dies zu bewerkstelligen benötigt man mehrere Kameras, die alle exakt zeitgleich auslösen müssen.

Von der Hochschule wurden uns zu diesem Zweck acht Kameras vom Typ Canon Powershot SX 130 IS zur Verfügung gestellt die wir unter Verwendung des Canon Hack Developers Kit (CHDK) über ein Skript auslösen konnten.

[**Einschub: Canon Powershot SX 130 IS**](#)

Bei der **Canon Powershot SX 130 IS** handelt es sich um eine von der Firma Canon vertriebenen digitale Kompaktkamera der Consumer Klasse. Sie ist ca. 113,3 x 73,2 x 45,8 mm groß und wiegt Ca. 308 g (einschließlich Akku und Speicherkarte). Betrieben wird die Kamera von 2 Mignonzellen: Alkali-Batterien (Alkali-Batterien im Lieferumfang enthalten) die Strom für ca. 130 Aufnahmen liefern.

An technischen Merkmalen bietet Die SX130 IS ein 12fach Weitwinkelobjektiv mit 28mm Brennweite und optischem Bildstabilisator und einem 12,1 Megapixel-Sensor. Als Sucher dient ein 7,5 cm (3,0 Zoll) TFT mit ca. 230.000 Bildpunkten

[**Einschub: CHDK \(Canon Hack Developers Kit\)**](#)

Das **Canon Hack Development Kit**, kurz CHDK, ist ein Funktionssatz speziell für Canon Kameras. Es ermöglicht einen enormen Funktionsgewinn, der weit über die Standardfunktionen der Canon Kameras hinausgeht. So ist es z.B. möglich bei einer handelsüblichen Digitalkamera gezielt deren RAW-Dateien auszulesen, welche dem Nutzer sonst nicht zugänglich wären. Weitere Beispiele wäre eine gezielte Fernsteuerung oder Automatisierung von Kameras, wie z.B. als Bewegungsmelder oder bei Zeitrafferaufnahmen.

Die Funktionsmöglichkeiten scheinen fast grenzenlos. Die wichtigste Eigenschaft jedoch ist, dass CHDK lediglich ein Softwareaufsatz ist und die eigentliche Firmware der Kamera unangetastet lässt, da CHDK nicht direkt auf die Kamera gespielt wird, sondern ausschließlich auf eine SD Karte, welche dann in die Kamera gesteckt wird.

Damit eine möglichst gleichmäßige Ausrichtung der Kameras sichergestellt werden kann positioniert man diese in der Regel auf einem Rig dessen Konzeption und Bau ich noch näher darstellen werde.

Einschub: Rig

Der Ausdruck **Rig** stammt ursprünglich aus der Bühnentechnik und bedeutet in etwa soviel wie Aufbau. In unserem Fall bezieht sich der Ausdruck auf das Multikamerarig das wir nutzen um alle Kameras in einer Ebene zu positionieren.

Um einen leichteren Ablauf zu gewährleisten haben wir unser Projekt in 4 Phasen aufgeteilt die wir nicht chronologisch sondern teilweise simultan abgearbeitet haben:

- Phase 1: Kamerarigbau

In Phase 1 haben wir Informationen zum Thema Rigbau eingeholt, verglichen und bewertet und anschließend daraus unser eigenes Rig konzipiert und konstruiert.

- **Phase 2: Auslöser/Fernbedienung**

In der zweiten Phase haben wir einen Auslöser gebaut mit dem ein gleichzeitiges Auslösen aller Kameras ermöglicht wurde. Dafür haben wir alle Kameras über USB-Hubs mit einem Arduino-Mikrocontroller verbunden, der wiederum mit dem Rechner verbunden wurde.

Einschub: Arduino

Die **Arduino-Platform** ist eine im Sinne von Open-Source quelloffene Physical-Computing-Plattform. Die Hardware besteht aus einem I/O Board mit einem Mikrocontroller mit analogen und digitalen Ein- und Ausgängen.

Die Entwicklungsumgebung basiert auf der von Processing und Wiring die besonders unter Designern, Bastlern und Künstlern sehr beliebt ist. Die zu verwendende Programmiersprache ist C/C++.

Der Controller ist ein sehr verbreitetes Element um interaktive Systeme zu steuern.

Zusätzlich zum eigentlichen Arduino Controller gibt es eine ganze Reihe an Erweiterungen wie z.B. einen Soundsensor, um auf Schallereignisse zu reagieren, IR-Sensoren, Laserschranken u.v.m..

- **Phase 3: Kalibrierung**

Die dritte Phase bildete den Schwerpunkt unseres Projektes. Die Kalibrierung erfolgte durch die von uns eigens für dieses geschriebene Software, welche eine Markererkennung durchführt, eine Transformationsmatrix errechnet und eine Affine Transformation anwendet um die Bilder übereinander zu legen. Die Kalibrierung haben wir in openCV und openFrameworks realisiert.

Einschub: openCV

OpenCV (*Open Source Computer Vision Library*) ist eine freie Programmzbibliothek mit Algorithmen für die Bildverarbeitung und maschinelles Sehen. Sie ist für die Programmiersprachen **C** und **C++** geschrieben und steht als freie Software unter den Bedingungen der BSD-Lizenz . Das „CV“ im Namen steht für englisch „Computer Vision“. Die Entwicklung der Bibliothek wurde von Intel initiiert und wird heute hauptsächlich von “Willow Garage” gepflegt.

Im September 2006 wurde die Version 1.0 herausgegeben, Ende September 2009 folgte nach längerer Pause die Version 2.0.0, welche die Bezeichnung „Gold“ trägt. Die Stärke von OpenCV liegt in ihrer Geschwindigkeit und in der großen Menge der Algorithmen aus neuesten Forschungsergebnissen.

Die Bibliothek umfasst unter anderem Algorithmen für Gesichtsdetektion, 3D-Funktionalität, Haar-Klassifikatoren, verschiedene sehr schnelle Filter(Sobel,Canny,Gauß) und Funktionen für die Kamerakalibrierung.

- **Phase 4: Interpolation und Ausgabe**

in Phase 4 haben wir eine Vorschau mithilfe von Linearer Interpolation erstellt. Diese Variante ist auch als Alpha Blending bekannt. Für die Ausgabe des Video, welches mit unserer Software erstellt wurde, wurde eine Darstellung in openFrameworks gewählt.

Einschub: openFrameworks

openFrameworks ist ein open source toolkit das für “creative coding” entwickelt wurde. OpenFrameworks ist in **C++** geschrieben und läuft auf Windows, Mac OS X, Linux, iOS und Android. Es wird von Zachary Lieberman, Theo Watson und Arturo Castro gepflegt die auf Beiträgen aus der openFrameworks community aufbauen.

Meine Aufgaben:

Während des Projekts verfolgte ich verschiedene Aufgaben u.a. waren das die Konstruktion und Bau des Multikamerarigs, die Mithilfe bei Markererkennung und Transformation und die Implementierung des openCV-Codes innerhalb des openFrameworks. Ebenso habe ich mich mit der Umsetzung der linearen Interpolation und der Ausgabe als Vorschauvideo beschäftigt.

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 unserere acht Kameras diente und einen wichtigen Faktor in der (Grob-) Hardwarekalibrierung darstellt.

Bevor wir unser Rig konzipieren und realisieren konnten, mussten wir uns erst einmal darüber klar werden, wie ein Rig beschaffen sein muss, um für unsere Zwecke die bestmöglichen Ergebnisse zu liefern.

Grundsätzlich kann man hier zwischen zwei Varianten des Rigaufbaus unterscheiden.

Die erste Variante ist ein geradliniger Rigaufbau bei dem die Kameras auf einer Geraden positioniert und in die gleiche Blickrichtung parallel zueinander ausgerichtet werden. Der Effekt der durch diesen Aufbau erreicht wird ist eine geradlinige Vorbeifahrt an dem aufgenommenen Objekt.

Alternativ dazu ist ein Rigaufbau bei dem die Kameras alle auf einen zentralen Punkt ausgerichtet sind und auf einem runden Aufbau um das "Ziel" herum stehen. Berühmtestes Beispiel hierfür ist der Aufbau aus dem Film Matrix. Dabei sind die Kameras spiralförmig um den Protagonisten angeordnet und auf selbigen ausgerichtet. Der Vorteil dieses Aufbaus ist

das man das aufgenommene Objekt am Ende aus verschiedenen Sichten/Blickwinkeln betrachten kann, wodurch eine Rundumansicht möglich ist.

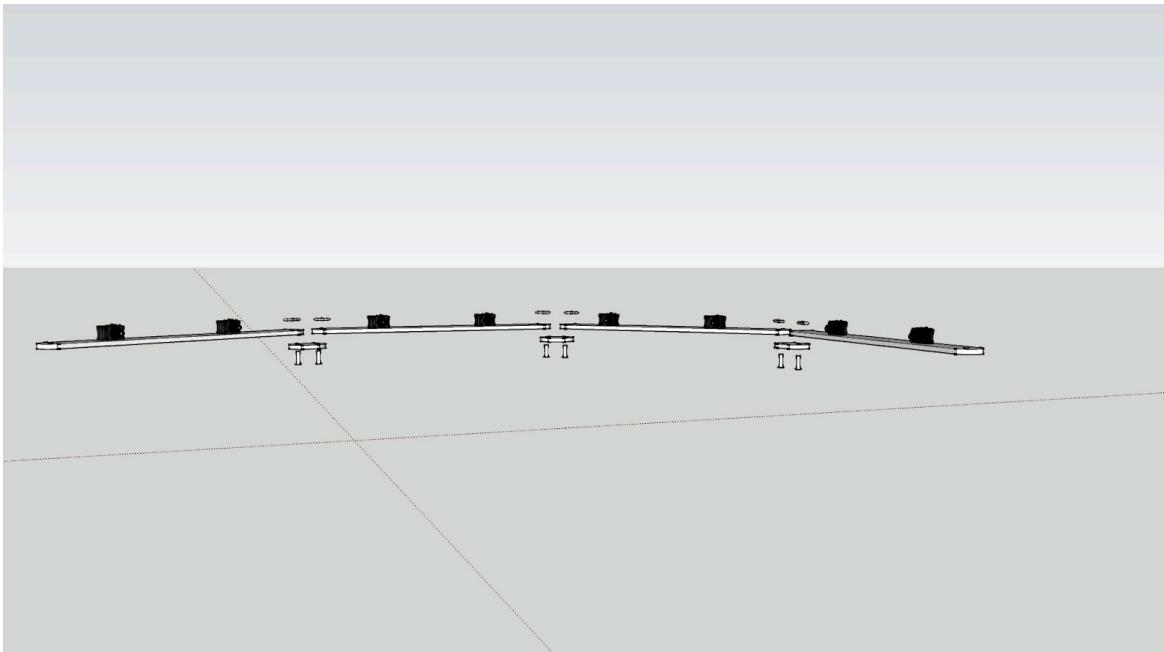
In einem unserer Treffen haben wir uns in der Gruppe für die zweite Variante entschieden da diese eben durch dieses Beispiel einen enormen Bekanntheitsgrad besitzt und wir damit den Effekt erzielen den wir uns als Endprodukt vorgestellt haben.

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.



Mein 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. Da ich keinen Sinn darin gesehen habe, habe ich mich dagegen entschieden das Holz zu behandeln oder zu streichen.



Das erste Bild zeigt hierbei das fertige Rig in seiner vollen Länge.

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



Probleme mit dem Rig:

Im Laufe des Projekts haben sich verschiedene Probleme mit dem Rig herausgestellt, so war es im Endeffekt für unseres acht Kameras viel zu lang da wir diese für ein möglichst gutes Ergebnis möglichst eng aneinander positionieren mussten so das wir am Ende nur 2 der 4 Elemente genutzt haben. Weiter hat sich das Rig aufgrund der natürlichen Eigenschaften von Holz in Verbindung mit der unvorteilhafte Lagerung des Rigs so stark verformt das wir bereits auf 1m Abstand vom Ziel einen Höhenversatz von bis zu 30cm feststellen konnten. Dies führte zu einem weiteren Problem mit den Schrauben die zur Befestigung der Kameras vorgesehen waren. Aufgrund ihrer Länge verstärkten sie diesen Effekt nur noch so das wir am Ende bei den Aufnahmen darauf zurückgriffen die Kameras frei auf dem Rig zu positionieren.

Markererkennung und Transformation

Der eigentliche Code für Markererkennung wurde von meinen Teammitgliedern Johannes Kiesel und Armin Voit entwickelt ich habe mich hier nur zu einem späteren Zeitpunkt mit eingebracht als es darum ging Fehler in dem von ihnen erdachten System zu finden und Feineinstellungen zu machen.

Die Transformation haben wir in einer gemeinsamen Sitzung entworfen in der die grundlegende Matrixberechnung und Anwendung der Transformation erdacht wurde. Der Code zu diesem Punkt wurde dann ebenfalls von Johannes Kiesel fertig geschrieben.

Meine Aufgabe bei der Transformation war es dann diese so umzuschreiben das sie für eine Echtzeitdarstellung der Transformation in unserem Vorschauvideo verwendet werden konnte.

Hierfür war es zunächst nötig sich zu überlegen in welcher Reihenfolge die Bilder verarbeitet werden sollten, also welches Bild auf welches gemappt werden sollte.

Am Ende sind wir zu dem Schluss gekommen das wir auf 2 ImageArrays zurückgreifen von dem eines unsere Marker-Bilder und das zweite die zu verarbeitenden Bilder enthält.

```
//□□□□□DEFINE IMAGEARRAY AND LOAD CALIBRATION IMAGES
for (int i=1, j=1; i<=counter; i++, j++) {
    char buffer_mrk[33];
    sprintf(buffer_mrk,"/Users/arminvoit/Documents/openframeworks/apps/Stand_22.07_sp
    ät/Matrigx/bin/MarkerPics/src%d.jpg", j);
    markerImages[i-1] = cvLoadImage(buffer_mrk);
}
for (int i=1, j=1; i<=counter; i++, j++){
    char buffer_src[33];
    sprintf(buffer_src,"/Users/arminvoit/Documents/openframeworks/apps/Stand_22.07_spä
    t/Matrigx/bin/Pics/%ds.jpg", j);
    images[i-1]= cvLoadImage(buffer_src);
}
```

Aus den Markerbildern berechnen wir anschließend aus den darauf erkannten Markern die affine Transformationsmatrix, die wir am Ende auf die Zielbilder anwenden. Dafür definieren wir innerhalb einer for-Schleife jeweils ein Destination- und ein SourceImage aus den Markern und ein PreviewImage aus den zu verarbeitenden Bildern

```
//Calculate all Matrices
for (int i=1; i< counter; i++){
    //Load Destination Image and define srcimage
    desimage = markerImages[i-1];
    //Load Images for mapping
    srcimage = markerImages[i];
    previewimage = images[i-1];
```

Berechnen dann die Markerwerte aus Destination und Source und schreiben diese in ein SrcPoint und DesPoint Pointarray aus denen dann die Matrix berechnet wird die wir dann in map_matrix1 speichern.

```
// Des-Points with marker-values
// Uncomment for using markers
setDesPoints(posXyellowDes, posYyellowDes, posXblueDes, posYblueDes, posXredDes,
posYredDes, posXgreenDes, posYgreenDes);

// Src-Points with marker-values
// Uncomment for using markers
setSrcPoints(posXyellowSrc, posYyellowSrc, posXblueSrc, posYblueSrc, posXredSrc,
posYredSrc, posXgreenSrc, posYgreenSrc);

// generate map_Matrix
cvGetAffineTransform(desPoints, srcPoints, map_matrix1);
```

Diese map_matrix1 wenden wir dann auf das previewimage an das wir dann in ein weiteres imageArray previewImages speichern das wir dann später für die Lineare Interpolation benutzen

```
//Map source to destination pic
cvWarpAffine(previewimage, previewimage, map_matrix1, CV_WARP_FILL_OUTLIERS);
previewImages[i]=previewimage;
```

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 ich 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 das er in dieses Modell passt. Begonnen habe ich damit das ich den openCV - Code in eine Headerfile und eine Sourcefile zu trennen. In der Headerfile definiere ich ein Klasse Calibration mit privaten und öffentlichen functionen und Variablen um einen Zugriff von aussen aus der TestApp-Klasse von openFramework zu gewährleisten. In dieser Headerfile definiere ich ausserdem alle im Code verwendeten Funktionen und Variablen.

In der Sourcefile bleibt soweit alles unaufgetastet ausser das die Mainfunction umbenannt werden musste.

Vorschau - Lineare Interpolation mit Alphablending

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 habe ich 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 habe ich 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 verändere ich 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.

```
//-----
// Our PreviewPlayer
//-----
void Calibration::showPreview(){

    IplImage *src1, *src2;

    int x = 0;
    int y = 0;
    int width = 800;
    int height = 600;
    double alpha;
    double beta;

    cvNamedWindow("Preview");
    cvMoveWindow("Preview", 100,100);
    char c;

    for (int i=0; i<=counter; i++) {
        src1= previewImages[i];
        src2= previewImages[i+1];

        for (int j=1; j<10; j++) {
```

```
alpha = 1.0 - j*0.1;  
beta = (1-alpha);  
  
cvSetImageROI(src1, cvRect(x,y,width,height));  
cvSetImageROI(src2, cvRect(0,0,width,height));  
cvAddWeighted(src1, alpha, src2, beta,0.0,src1);  
cvResetImageROI(src1);  
cvShowImage("Preview", src1);  
cvWaitKey(150);  
}  
}  
cvDestroyWindow("Preview");  
}
```

Die Programmieraufgaben habe ich größtenteils zusammen mit Armin Voit im pair programming bearbeitet, so konnten viele Probleme sehr effektiv gelöst werden.

Weitere Aufgaben

Neben meinen Hauptaufgaben habe ich noch weitere kleiner Arbeiten übernommen.

So habe ich mich zusammen mit Johannes Kiesel und Armin Voit um die Codedokumentation gekümmert, habe Recherchen in verschiedenen Bereichen durchgeführt und mich bei der Gestaltung von Zwischen und Abschlusspräsentation eingebracht. Nebenbei habe ich wichtige Arbeitsschritte durch Fotoaufnahmen dokumentiert.

Reflexion Projektablauf

Zum Projektablauf muss ich sagen dass es phasenweise sehr unterschiedlich gelaufen ist. Anfangs sind wir sehr langsam in Fahrt gekommen, wir haben sehr lange gebraucht uns darüber klar zu werden wo wir hinwollen und was unser genaues Ziel am Ende sein soll.

Als wir dann endlich unsere Ziele festgemacht hatten, mussten wir erst einmal sehr viel Recherchearbeit leisten um einen Plan davon zu bekommen was alles nötig ist und wie die vielen Aufgaben zu realisieren sind. In dieser Zeit haben wir hauptsächlich hardwarespezifische Fortschritte gemacht, so entstand in dieser Zeit unser Rig und wir konnten Erfolge in der Nutzung von CHDK und dem getriggerten Auslösen der Kamera (in der ersten Zeit stand uns nur eine Kamera zur Verfügung).

Da es einige Zeit dauerte bis wir eine zweite Kamera nutzen konnten und wir aus privaten Beständen (Versionsprobleme) keinen Ersatz nutzen konnten steckte der praktische Teil einige Wochen fest. Erst mit dem Erhalt der zweiten Kamera konnten wir Versuche zum simultanen Auslösen und anderer nötiger Funktionen machen.

Das Warten auf die restlichen Kameras resultierte in einem Schwund an Motivation bei allen Teammitgliedern so das gerade zur Semestermitte besonders wenig vonstatten ging.

In dieser Zeit kämpften wir ebenfalls mit einigen Problemen in der Software besonders der Installation von openFrameworks und openCV und der Vereinbarkeit dieser beiden was einige nervenaufreibende Stunden an Fehlschlägen beinhaltete.

Erst mit dem Eintreffen der restlichen Kameras etwa Mitte Juni machten wir wieder größere Fortschritte. In dieser Zeit entstanden eine ganze Reihe von Testaufnahmen mit denen wir intensiv die Markererkennung und die Transformationen erproben konnten.

Die produktivste Phase begann für uns Ende Juni Anfang Juli als es darum ging alles zusammenzufügen und einen funktionierenden Prototypen zu schaffen.

Alles in Allem kann man sagen, dass sich das Projekt das wir ursprünglich als ein Forschungsprojekt angetreten haben aufgrund seiner Vielschichtigkeit und der vielen Probleme an allen möglichen Stellen zu einem ganz schönen Brocken entwickelt hat den wir unter den gegeben Bedingungen zu einem guten Abschluss bringen konnten.

Quellen- und Literaturverzeichnis:

- [Learning OpenCV: Computer Vision with the OpenCV Library](#) von Mike Loukides, Robert Romano, Gary Bradski und Adrian Kaehlervon O'Reilly Media (Taschenbuch - 3. Oktober 2008)
- **OpenCV 2.1 Cheat Sheet (C++)** - <http://opencv.willowgarage.com/documentation/cpp/> .

Anhang

Soll/Ist Vergleich auf der Grundlage des Projektplans

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	✓

Weitere Anhänge