



IFS

INSTITUTE FOR
SOFTWARE

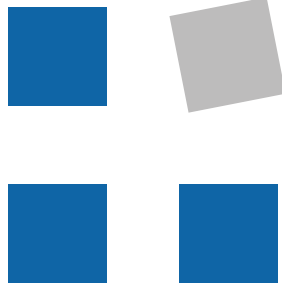
Bachelorarbeit

Presentation Writer

Whiteboard Funktionalität für Beamer

Renato Bosshart, Josua Schmid

13. Juni 2013



Hochschule für Technik Rapperswil
Institut für Software
Oliver Augenstein

Experte:

Dr. Peter Rost, NEC Labs Europe

Gegenleser:

Hansjörg Huser

Danksagungen

Die Autoren möchten sich bei Oliver Augenstein für die kompetente und wohlwollende Betreuung bedanken. Weiterer Dank geht an Benjamin Hug, Stefan Kienzel, Edith Schmid und Pascal Spörri für ihre kritischen Kommentare.

Abstract

Es wird eine neue digitale Whiteboard-Technologie beschrieben. Während herkömmliche digitale Whiteboards viel Geld kosten, verspricht das in dieser Arbeit entwickelte System eine günstige Alternative. Es wird gezeigt, dass Standardhardware und einfache Algorithmen mit der Genauigkeit von herkömmlichen Produkten konkurrenzieren können. Mit einem selbst entwickelten günstigen Stift emuliert die Software „Presentation Writer“ Mausklicks auf einer normalen Beamerleinwand - und das auf 4 Pixel genau.

Inhaltsverzeichnis

1	Einleitung	6
1.1	Problemstellung	8
1.1.1	Aufnahmegerät	8
1.1.2	Präsentationsgerät	9
1.1.3	Positionierung	10
1.1.4	Leinwand	12
1.1.5	Laptop	12
1.1.6	Präsentationsweise	12
1.2	Ergebnis	13
2	Analyse	16
2.1	Erkennen von Vierecken	20
2.1.1	Differenzbildansatz	20
2.1.2	Histogramm-Analyse	21
2.2	Kalibrierung: Referenzpunkte	21
2.2.1	Dies Erkennen der Bildschirmgrenze	22
2.2.2	Ordnung kleiner Vierecke	22
2.2.3	K-Means Clustering	23
2.2.4	OpenCV	23
2.3	Kalibrierung: Abbildung & Interpolation	24
2.3.1	Linear	24
2.3.2	Baryzentrische Koordinaten	25
2.3.3	Zweidimensional per Integration	27
2.3.4	Homogene Transformation	33
2.4	Stift Erkennen	34
2.4.1	Präsenz durch Helligkeit	34
2.4.2	Präsenz durch Kardinalität	34
2.4.3	Position durch Schwerpunkt	35
2.4.4	Performance durch gezieltes Suchen	36
2.4.5	Stabilität durch Differenzbilder	36
3	Umsetzung	38
3.1	Technologien	38
3.1.1	Programmiersprache	38
3.1.2	Framework	38
3.2	Architektur	39
3.2.1	Domain	39
3.2.2	Hauptsequenz	40
3.3	Visualisierung	42
3.3.1	Umsetzungen	43
3.4	Kalibrierung	44
3.4.1	Schachbrett-Differenzbilder	48

3.5	Stifterkennung	48
3.5.1	Verfahren	48
3.5.2	Umsetzung	54
3.5.3	Einschränkungen	54
3.5.4	Wahl des Stiftes	55
3.6	Input Emulation	56
3.7	Erkenntnisse	56
4	Ausblick	60
4.1	Kamera & Stift	60
4.2	Kalibrierung	60
4.3	Input Emulation	61
5	Anhang	62
5.1	Kalibrierung: Bilderkennung	62
5.1.1	Eckdetektion mit Differenzbildern und zufällig verteilten Rechtecken.	62
5.1.2	Verwendung des AForge-Blob-Detectors	63
5.1.3	Einfache Differenzbild-Kalibrierung	64
5.1.4	Histogramm-Analyse	65
5.1.5	Erweitertes Differenzbild	66
5.2	Kalibrierung: Zuordnung von Punkten	68
5.2.1	Rekursiv	68
5.2.2	Nutzung der Interpolation	70
5.2.3	K-Means Cluster-Zuordnung mit steigender Genauigkeit	70
5.3	Interpolation	71
5.3.1	Lineare Interpolation	71
5.3.2	Baryzentrische Koordinaten	74
5.4	Versuche zur Umsetzung von Touch	79
5.5	Quellenverzeichnis	86

1 Einleitung

Die in dieser Arbeit entworfenen und in einem Prototyp eingesetzten Algorithmen sollen einer Person ermöglichen, interaktive Präsentationen in einem kleinen oder mittelgrossen Sitzungszimmer zu halten. Die Person kann ihren eigenen Laptop dafür verwenden. Der Laptop wird am lokal vorhandenen Beamer angeschlossen. Nach einer kurzen Kalibrierung kann die Person die Präsentation starten. Dabei schreibt sie, falls gewünscht, mit einem virtuellen Stift an die Leinwand. Das Geschriebene wird aufgezeichnet und wieder auf das Beamerbild projiziert.

Aufbau In einem normalen Sitzungszimmer soll ein Vortrag gehalten werden können. Dazu sollen Standardausstattung wie Laptop, integrierte Kamera und Beamer benutzt werden können. Als Addendum wird ein spezieller Stift mitgeliefert. Der Stift soll günstig sein. Der Aufbau unterliegt zu definierenden Beschränkungen wie Winkel und Raumbeleuchtung.

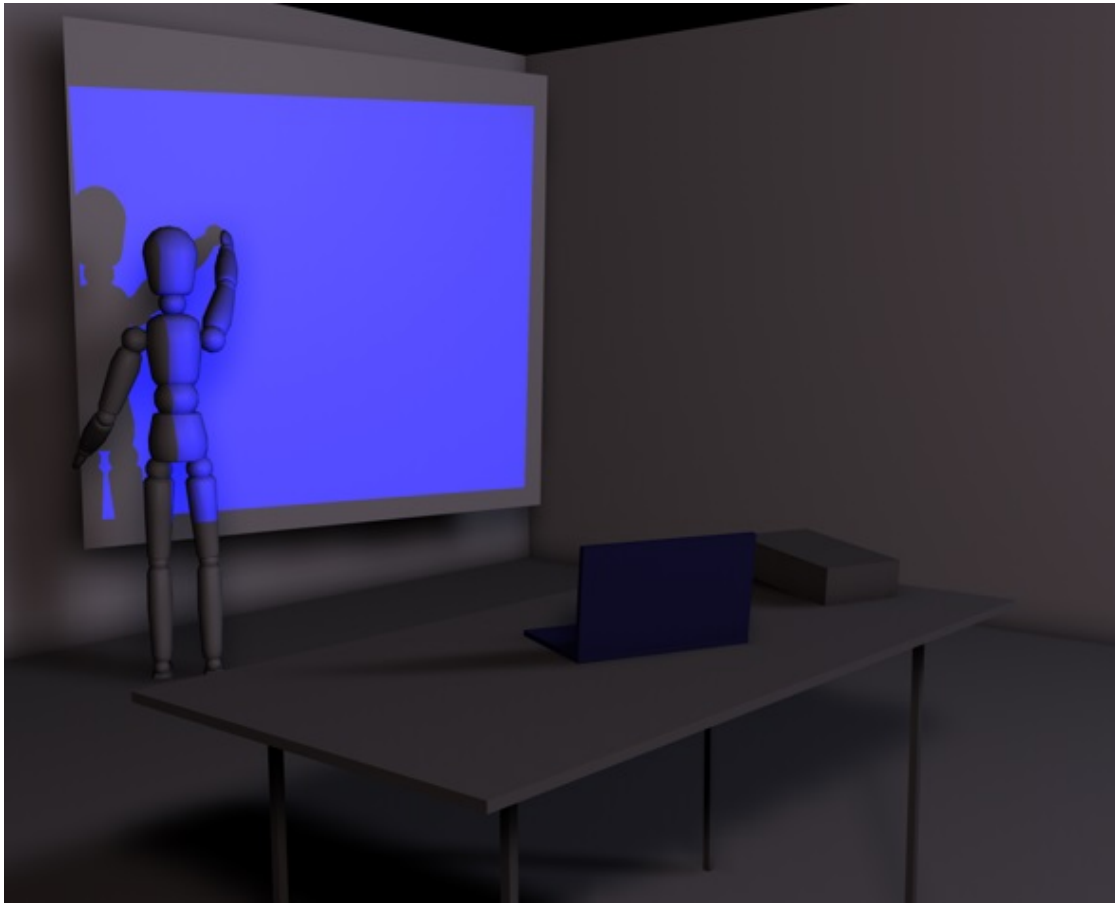


Abb. 1: Beispiel eines Aufbaus in der Perspektive

Kalibrierung Bevor ein Stift auf der Leinwand gefunden werden kann, muss die Erkennungssoftware kalibriert werden. Dazu werden folgende Schritte vorgenommen:

1. Die Software „Presentation Writer“ starten
2. Den Laptop so ausrichten, dass die ganze Leinwand im Blickfeld der integrierten Webcam ist.
3. Auf den Kalibrierungsknopf drücken. Wichtig ist hier, dass die Kamera die Leinwand sieht.
4. Kameraoptionen wie automatische Belichtungskorrektur einstellen
5. Pen Tracking aktivieren

Nun sollte das projizierte Bild erkannt werden. Alle Kamerabildkoordinaten sollten nun auf das Beamerbild umgerechnet werden.

Präsentation Nach der erfolgreichen Kalibrierung kann der Benutzer mit einem speziellen Stift an die Leinwand zeichnen. Vorläufig ist dieser Stift eine einfache weisse LED. Die Person muss darauf achten, dass der Stift für die Kamera sichtbar ist. Das erfordert Disziplin. Deshalb gilt abzuklären, ob und wie das Präsentationssystem akzeptiert wird. Es ist einem Benutzer vermutlich zumutbar, nur dann schreiben zu können, wenn ihn das Publikum sieht.

1.1 Problemstellung

Diese Arbeit stellt sich verschiedenen Herausforderungen. Es soll ein günstiger Stift von einer Standardkamera mittels eines Standardbeamers gefunden und erkannt werden können. Sowohl bei der dazu verwendeten Hardware als auch bei der Art und Weise wie die hier entwickelte Software benutzt werden kann, gibt es Einschränkungen. Obwohl das erarbeitete Präsentationssystem zum Ziel hat, einschneidende Beschränkungen zu vermeiden, gibt es doch verschiedene Grenzen, die von Anfang an definiert sind und als nicht überwindbar gelten. Grundsätzlich sind sie physischer und physikalischer Art. Im Folgenden werden die Probleme kurz diskutiert um zu zeigen, welche Themengebiete überhaupt einer detaillierten Analyse bedürfen. Für einen besseren Überblick wird zum Teil kurz angeschnitten, wie das jeweilige Problem gelöst werden soll.

1.1.1 Aufnahmegerät

Das in dieser Arbeit verwendete Aufnahmegerät ist eine integrierte Laptopkamera eines Thinkpad T430s mit einer Auflösung von 640x480 Pixel und 30 Bildern pro Sekunde. Es traten technische Probleme mit der automatischen Belichtungskorrektur auf, welche sich durch einen im Treiber integrierten Konfigurationsdialog manuell minimieren lassen. Allgemein formuliert ist die Bildauflösung der verwendeten Kamera weniger wichtig als eine korrekte Belichtung. Das liegt daran, dass die präsentierende Person immer in Bewegung ist, mal mehr beleuchtet, mal weniger, und somit ein irritierendes Ungleichgewicht für die Kamera schafft. Übertriebene Änderungen der Beleuchtung seitens der Kamera verwechseln den Stift. Folglich ist er nicht mehr der hellste Punkt der Aufnahme und wird nicht mehr erkannt, bzw. gefunden.

Beispiel: Abbildung 2 zeigt eine Beamerprojektion einer weissen Fläche. Der Raum ist wenig abgedunkelt, damit die Kamera, welche die Aufnahme gemacht hat, eine weniger starke automatische Belichtungskorrektur vornimmt. Nichtsdestotrotz beträgt die Anzahl völlig weisser Pixel in diesem Bild mehr als 11'000. Das macht es unmöglich, optisch einen hellen Punkt (Stift) zu finden. Eine stärker abgedunkelte Umgebung erhöht den Kontrast und verstärkt somit auch das Problem.

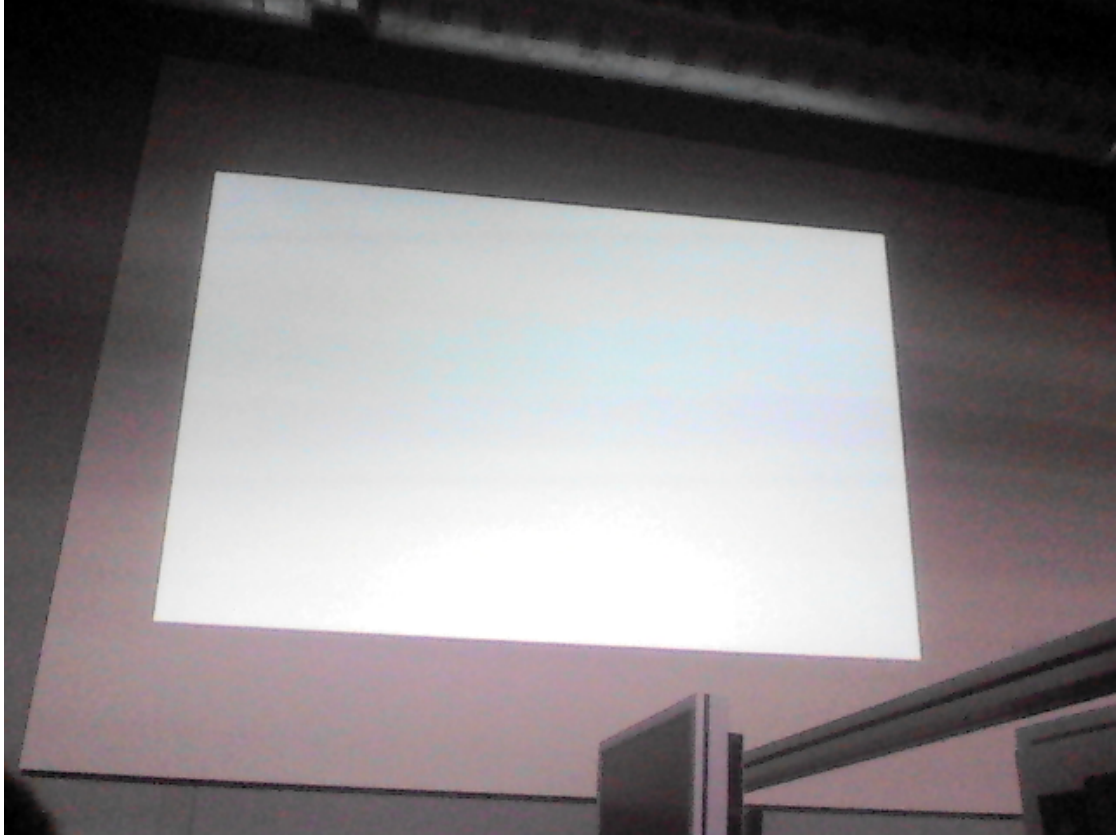


Abb. 2: überbelichtetes Kamerabild

Fazit: Die automatische Belichtungskorrektur kann über den Hardwaretreiber abgestellt werden¹, was aber sehr kompliziert und zeitintensiv ist. Zudem ist dieser Prozess bei Windows-Betriebssystemen deutlich aufwändiger als bei Linux. Die verwendeten Geräte müssten genau bekannt sein, damit sich ihre Treiber konfigurieren zu lassen. Wahrscheinlich stellt dieser Aufwand ein Hindernis dar. Dies würde eher durch Mitliefern einer eigenen Infrarot-Kamera überwunden. Das hätte den Vorteil, dass sowohl Präsentator als auch Zuschauer nicht von sichtbarem Licht abgelenkt würden. In dieser Arbeit wird der Einfachheit halber lediglich auf das Finden einer von Menschen sichtbaren Lichtquelle eingegangen. Es wird angenommen, dass sich alle Konzepte auf nicht sichtbares Licht übertragen lassen.

1.1.2 Präsentationsgerät

Ein weniger wichtige Rolle als die Kamera spielt das Präsentationsgerät. Weder die Auflösung noch die Bildwiederholrate sind für die Qualität des Präsentationssystems von nennenswerter Bedeutung. Die Helligkeitseinstellung ist jedoch wichtig. Der in dieser

¹in dieser Arbeit geschieht dies manuell über ein DirectShow Konfiguraitonsdialog

Arbeit verwendete Beamer beleuchtet die Leinwand sehr stark und verstärkt dadurch das Problem der automatischen Belichtungskorrektur der Kamera. Zur Minimierung des Problems wurde die Helligkeit auf ein schwächeres Niveau reduziert. Es wurde darauf geachtet, dass die Einstellung als „für eine üblich Präsentation“ wahrgenommen wird.

1.1.3 Positionierung

Der Laptop wird so aufgestellt, dass das Sichtfeld der integrierten Kamera den Projektionsbereich auf der Leinwand voll umfasst.

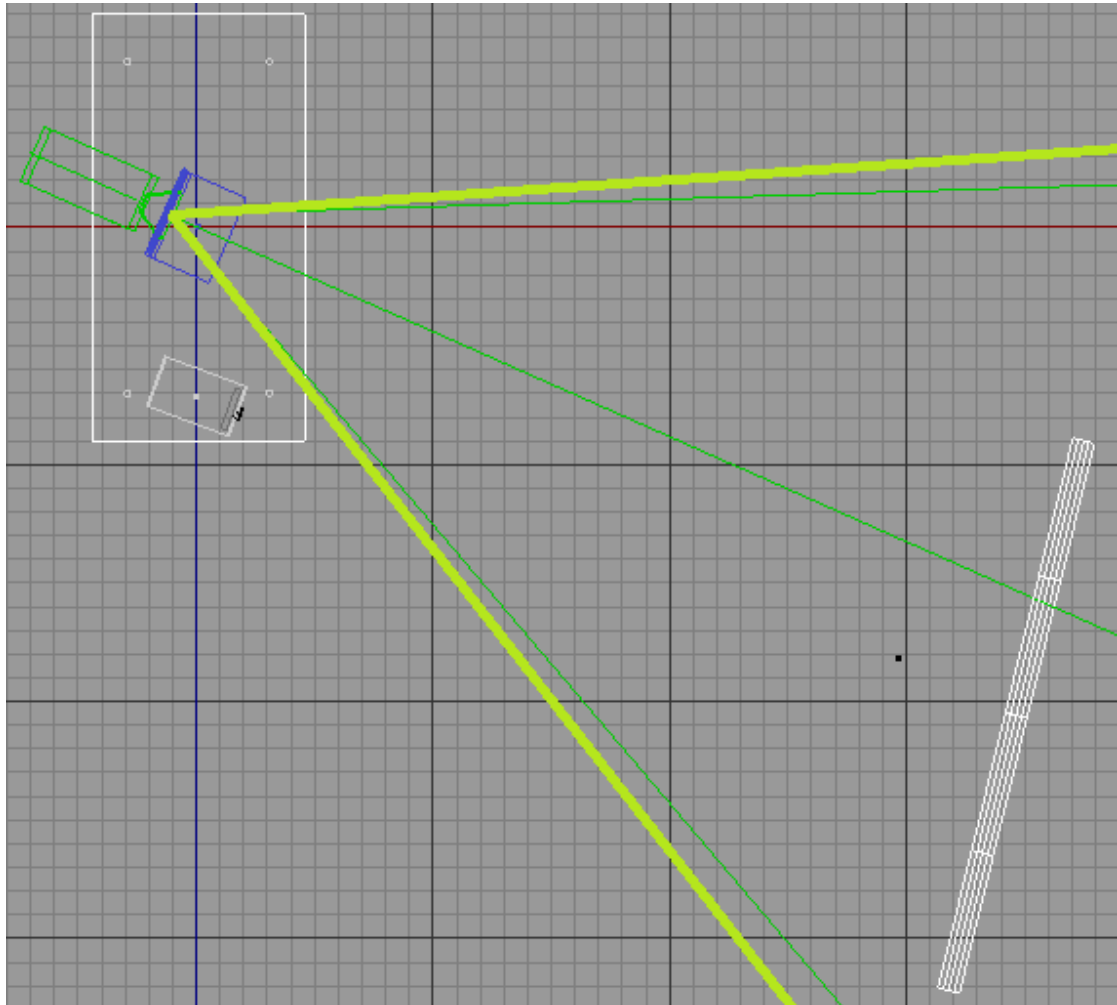


Abb. 3: Beispiel eines Aufbaus von oben

Der Winkel zwischen Projektionsfläche und Laptopkamera hat Auswirkungen auf die Genauigkeit des Präsentationssystems. Je grösser der Winkel zwischen Kamera und Projektionsfläche ist, desto stärker verzerrt nimmt die Kamera das projizierte Bild wahr.

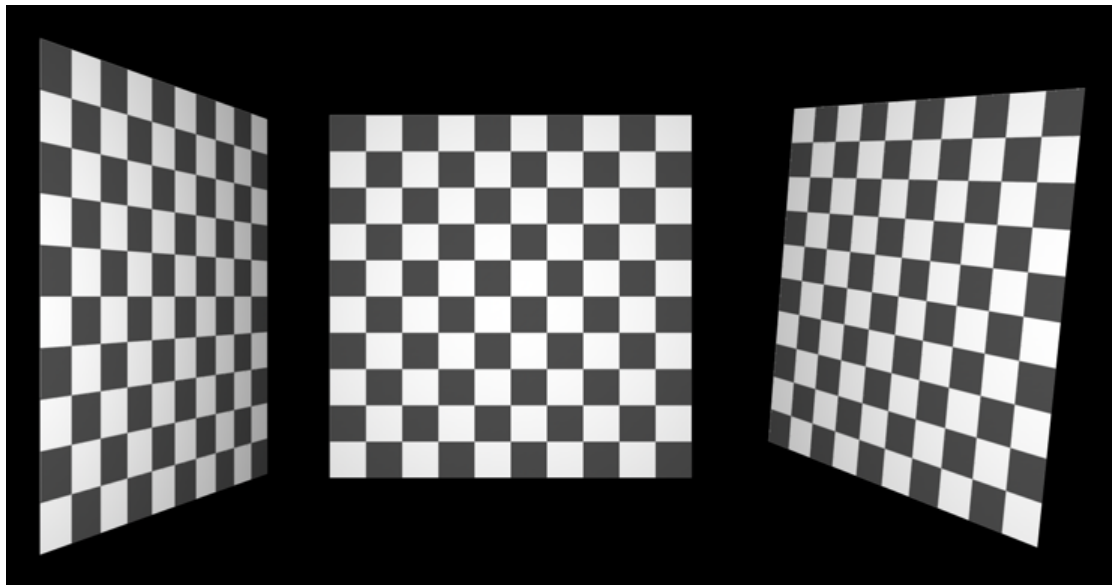


Abb. 4: Leinwand aus drei verschiedenen Blickwinkeln

Abb. 4 zeigt die Leinwand aus drei verschiedenen Winkel mit den daraus resultierenden einfachen perspektivischen Verzerrungen.

- Die linke Abbildung zeigt den angenäherten Normalfall²: Beamerbild und Laptopkamera liegen auf zwei parallelen Ebenen. Es wird nur in X-Richtung verzerrt. Das Bild ist weder winkel- noch flächentreu.
- In der Mitte ist der Optimalfall dargestellt: Die Kamera sieht direkt von vorne auf das Beamerbild, d.h. die perspektivische Verzerrung ist minimal. Das Bild ist Winkel- und Flächentreu.
- Rechts ist das Bild in alle Richtungen verzerrt und stellt somit die grössten Anforderungen an die Kalibrierung. Hierbei ist weder Winkeltreue noch Fläche gegeben.

Die Kalibrierung kann die perspektivische Verzerrung nur begrenzt ausgleichen. Deshalb kann die Position eines Stiftes nicht immer gut oder in allen Bereichen der Leinwand gleich präzise gefunden werden. Zur Vereinfachung des zu lösenden Problems wird im Folgenden jeweils angenommen, dass die perspektivische Verzerrung lediglich auf der Horizontalen (X-Achse) der aufgenommenen Bilder verläuft (linke Abbildung). Messungen zur Genauigkeit unseres verwendeten Abbildungsverfahrens (s.u.) zeigen, dass die Verzerrung auf der Vertikalen (Y-Achse) vernachlässigt werden kann. Die Dokumentation der Messresultate wird weiter unten für jedes wichtige Verfahren erbracht. Die für die Umsetzung relevanten sind in den Abb. 20 und 21 dargestellt.

²In dieser Arbeit wird angenommen, dass sich Beamer und Laptop auf gleicher Höhe befinden. Dies ist das Setup in einem üblichen kleineren Sitzungsraum.

1.1.4 Leinwand

Es ergeben sich Beschränkungen aus dem Untergrund, auf welchen präsentiert wird. Unebene oder glänzende Texturen sind ungeeignete Projektionsflächen, da sowohl Kalibrierung als auch Stifterkennung darunter leiden können. Grosse Probleme können Reflexionen der Beamerlampe auf der Leinwand verursachen. Dies kann passieren, wenn Beamer und Laptop in einem ungünstigen Winkel zueinander stehen.

1.1.5 Laptop

Bildverarbeitung braucht in der Regel viel Rechenleistung. Die in dieser Arbeit entwickelte Software berücksichtigt dies, indem zum Teil Kompromisse eingegangen werden um den Leistungsanforderungen gerecht zu werden. Um eine reibungslose Erkennung zu gewährleisten, sollten 2 GHz Doppelkernprozessoren oder ähnliche verwendet werden.

1.1.6 Präsentationsweise

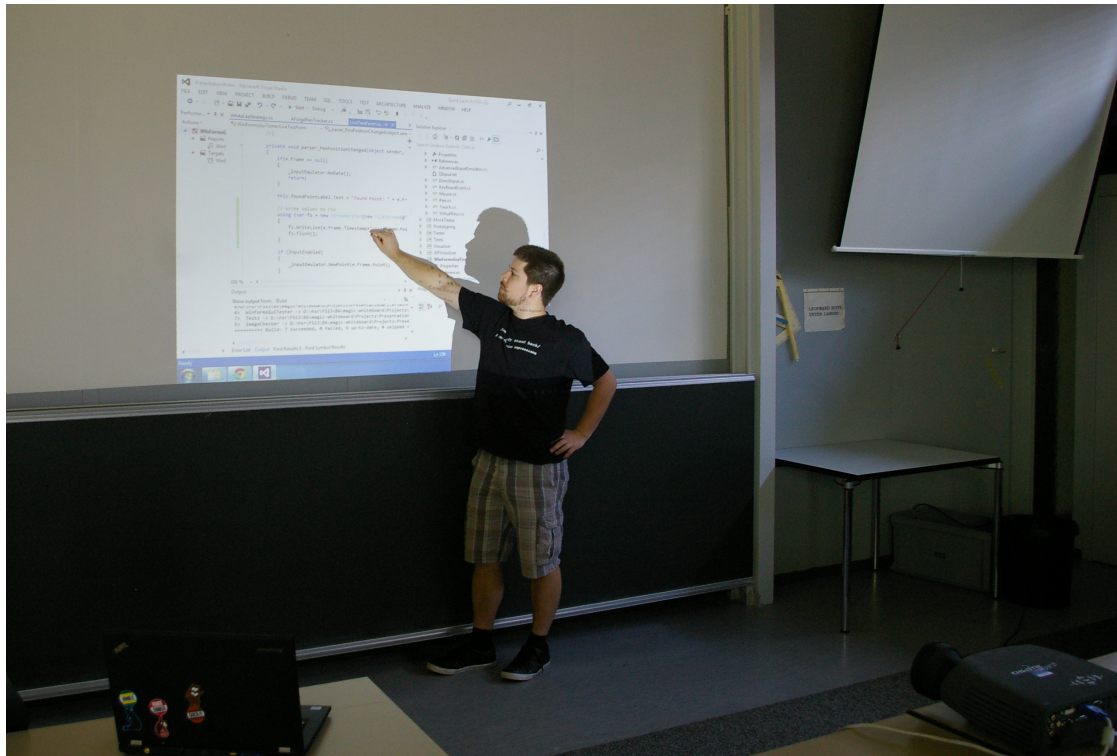


Abb. 5: Gute Präsentationsweise - Die zeigende Hand ist von Laptopkamera (links unten) und Publikum (rechts unten) aus sichtbar.

Da der Stift immer von der Laptopkamera gesehen werden muss, ist es am Präsentator, die richtige Haltung beim Schreiben anzunehmen. Eine gute Art zu präsentieren ist,

wenn das Publikum sieht, wo der Präsentator hinzeigt. Dies hat den Vorteil, dass auch die Laptopkamera sieht, wo auf die Leinwand geschrieben werden soll. Falls sich der Präsentator zwischen Kamera und Leinwand stellt, kann nicht mehr geschrieben werden. Da in diesem Fall meist auch das Publikum nichts mehr sieht, wird diese Einschränkung ohne weiteren Diskurs in Kauf genommen.

1.2 Ergebnis

Die in dieser Arbeit entwickelte Software „Presentation Writer“ kann gewinnbringend für Vorträge eingesetzt werden. Die tiefe Auflösung der Kamera von 640x480 Pixel und ihre Sensibilität gegenüber Helligkeitsveränderungen haben schlussendlich wenig Einfluss auf die Genauigkeit des Resultates. Eine erfolgreiche Kalibrierung der Kamera, gute Einstellungen der Bildfilter und genügend gute Abbildungsalgorithmen mit Interpolation machen dies möglich. Optimierungen, wie eine hochauflösende Kamera, würden noch genauere und bessere Resultate liefern. Aus Zeitgründen wurde in der Software zum Teil auf Optimierungen verzichtet. Jene die erschlossen wurden, sind jedoch in der schriftlichen Arbeit jeweils kurz erwähnt.

Im Folgenden dieser Arbeit wird genauer darauf eingegangen, wie alle diese Zwischenschritte, vom Kamerabild bis zum gezeichneten Bildpunkt, auf dem Beamer analysiert und gelöst wurden. Zum einen werden Ideen zum Finden eines Stiftes in einem Kamerabild diskutiert, zum anderen werden verschiedene Ansätze erarbeitet, die eine näherungsweise korrekte Abbildung von Kamera- zu Beamerkoordinaten ermöglichen. Im Anhang werden erweiterte oder fehlgeschlagene Ideen genauer erläutert. Zudem wird dort auch die Vision vom virtuellen Touchpanel ausgearbeitet. Das ist eine Idee, die eventuell auch mit einer optischen Kamera, aber ohne Stift lösbar ist.

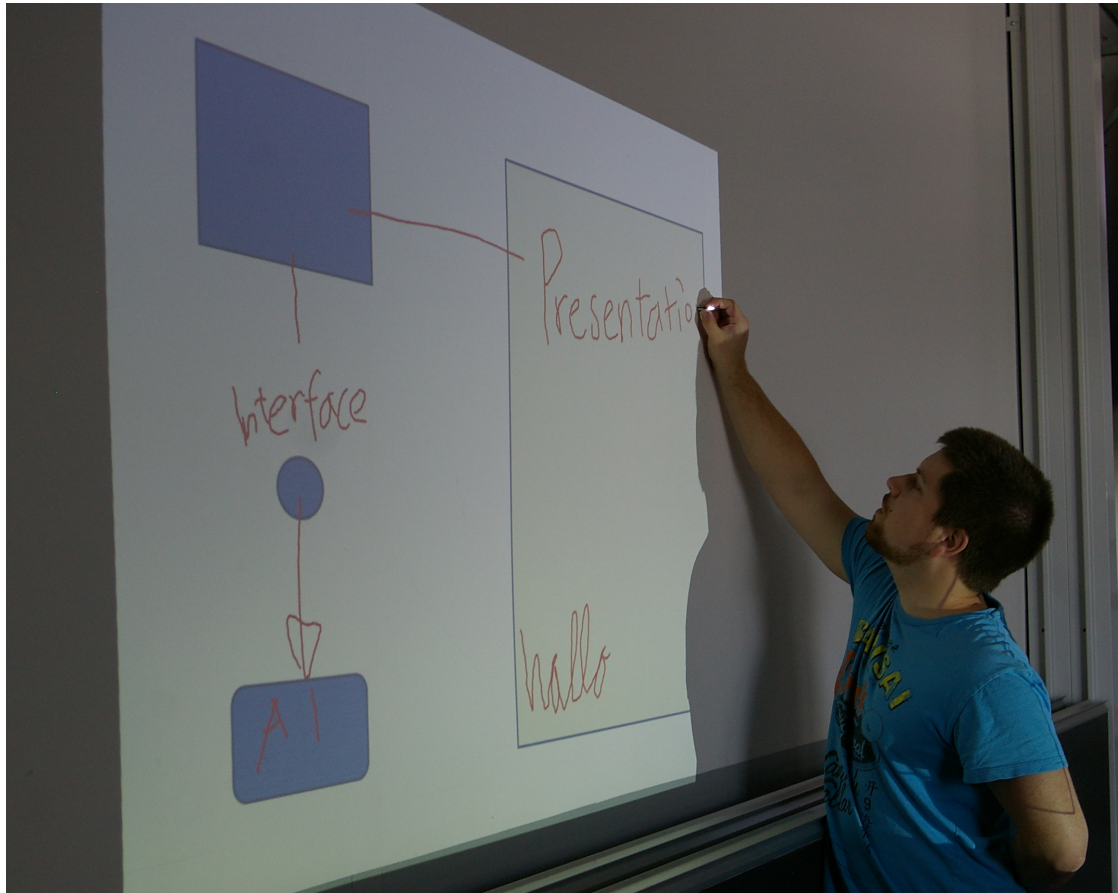


Abb. 6: Presentation Writer - Unterstützung bei Vorträgen

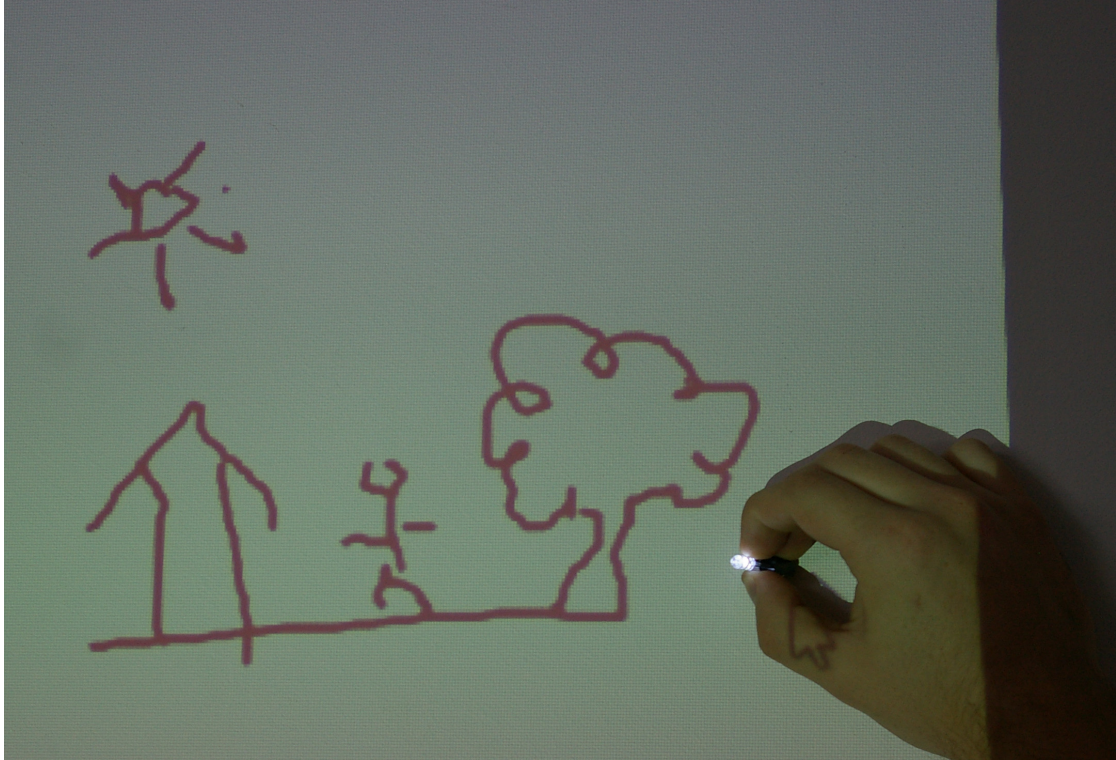


Abb. 7: Kleines Gekritzelt macht Spass - Trotz der tiefen Tracking-Auflösung von 640x480 Pixel.

2 Analyse

Im Folgenden werden Überlegungen zur Beziehung Beamer-Kamera und zu den konkreten Eigenschaften eines Stiftes gemacht. Es gilt zu analysieren, wie und ob die oben postulierten Probleme gelöst werden können. Es wird auf Kamerakalibrierung und Bildverarbeitungsschritte eingegangen. Dazu werden eigene Algorithmen beschrieben und analysiert. Zuerst werden die für die Kamerakalibrierung nötigen oder/und möglichen Schritte diskutiert. Danach folgt ein Abschnitt über das Finden eines Stiftes in einem Kamerabild.

Begriffe Folgend wird Beamerbild und Kamerabild definiert. Das Beamerbild beschreibt das ursprüngliche Bild, welches projiziert wird. Es besitzt die Auflösung des Beamers. Das Kamerabild ist das von der Kamera aufgenommene Bild. Es besitzt die Auflösung der Kamera. Auf dem Kamerabild ist die Projektion des Beamerbildes verzerrt sichtbar. Ein Stift soll auf diesem verzerrten Bild gefunden und auf das rücktransformierte Beamerbild abgebildet werden. Diese Abbildung wird jeweils beschrieben durch $(x_{Kamera}, y_{Kamera}) \rightarrow (x_{Beamer}, y_{Beamer})$ oder kürzer durch $(x_k, y_k) \rightarrow (x_b, y_b)$. Die X-Koordinate läuft im Bild von links nach rechts. Die Y-Koordinate läuft von oben nach unten.

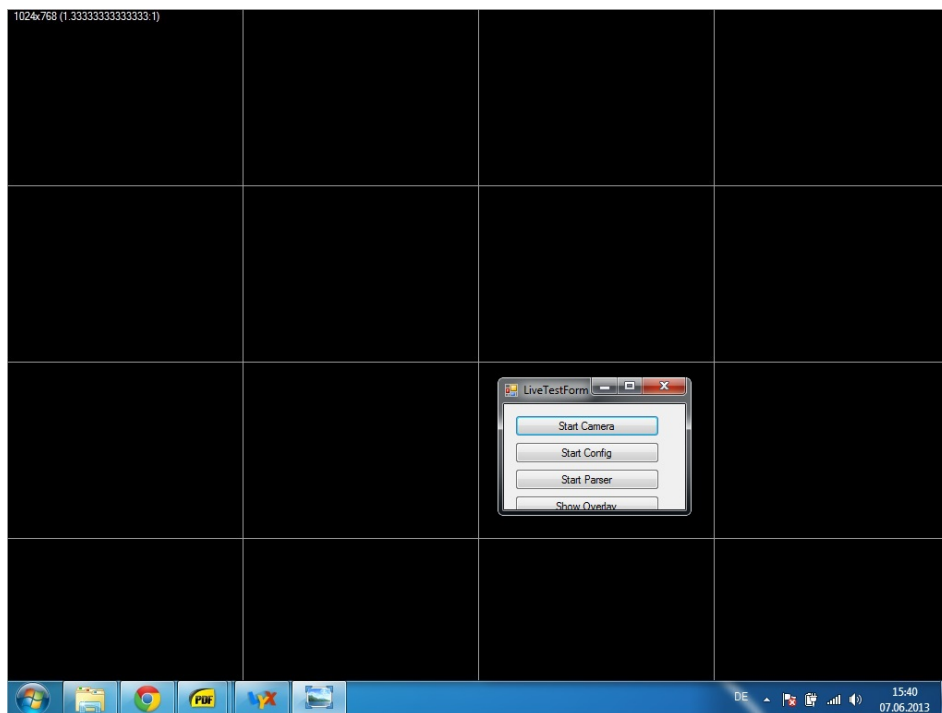


Abb. 8: Zu projizierendes Beamerbild

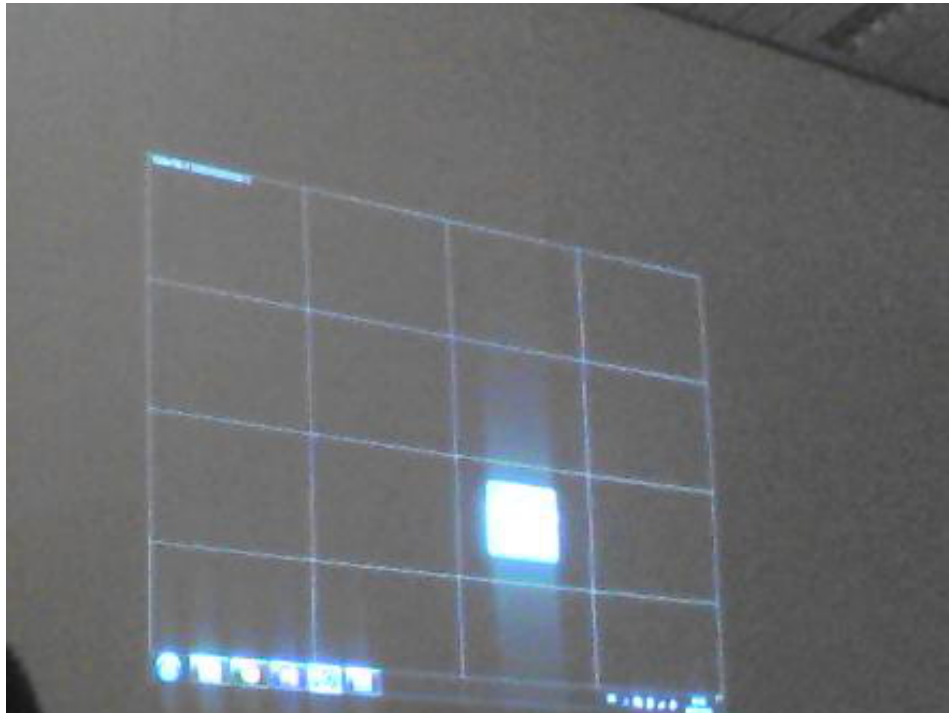


Abb. 9: Das Kamerabild enthält das verzerrte Beamerbild.

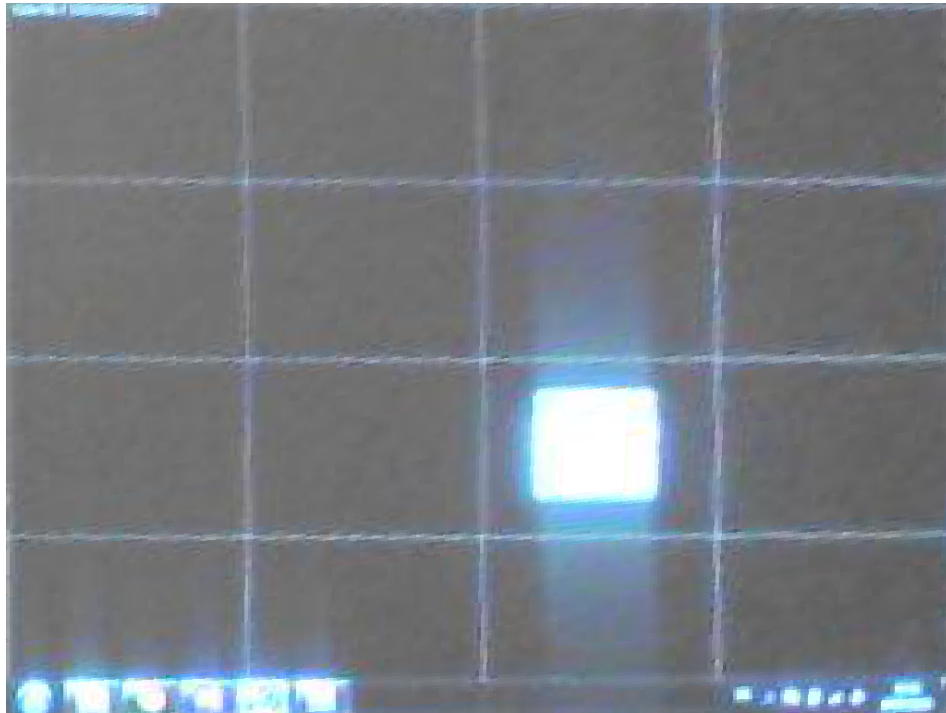


Abb. 10: Das im Kamerabild gefundene und auf die Beamerauflösung rücktransformierte entzerrte Beamerbild

Projizieren und Messen Die Lichtverhältnisse sind entscheidend für erfolgreiche Messungen. Für die Kalibrierung projizierte Referenzbilder haben Einfluss auf die automatische Belichtungskorrektur der Kamera. Grosse Helligkeitswerte im projizierten Bild verursachen schnelles Abdunkeln des Kamerabildes, tiefe Helligkeitswerte das Gegenteil. Es ist schwierig, vorauszusagen, wie die Kamera auf Helligkeitsunterschiede reagiert. In jedem Fall hat dies Einfluss auf in der Kalibrierung verwendete Differenzbilder. Der hier beschriebene Effekt kann minimiert werden, wenn nur Bilder mit im Mittel ähnlichen Helligkeitswerten verwendet werden. Die schlussendlich eingesetzte Lösung verwendet zwei Muster, deren Durchschnittshelligkeiten beinahe gleich sind. Die Kamera passt die Belichtungskorrektur kaum an. Dies wird unter 3.4.1 genauer beschrieben.

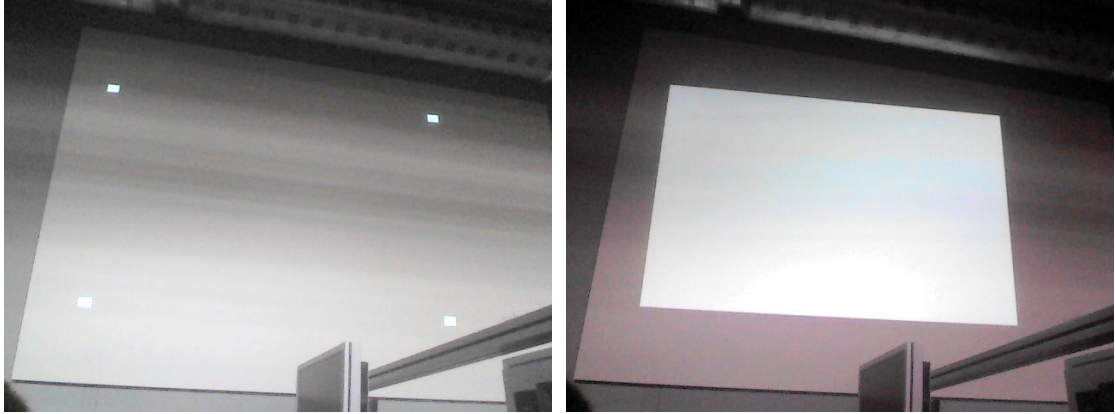


Abb. 11: Zwei schlecht gewählte Referenzbilder. Sie haben zu grosse Helligkeitsunterschiede.



Abb. 12: Das aus Abb. 11 resultierende Differenzbild ist für die Leinwanddetektion kaum brauchbar.

2.1 Erkennen von Vierecken

Im folgenden Abschnitt wird beschrieben, wie projizierte Rechtecke auf dem Kamerabild wiedererkannt werden können. Die Grösse ist sehr unterschiedlich. Je nach Verfahren beträgt der Unterschied die ganze Bildschirmgrösse oder nur wenige Pixel auf dem Kamerabild. Die Hauptziele dieses Verfahrens sind sowohl Korrektheit als auch Genauigkeit, wobei verschiedene Interpolationsverfahren auch andere Ansprüche haben. Die Korrektheit beinhaltet, dass alle Rechtecke erkannt werden und gleichzeitig das Rauschen auf ein Mass gesenkt wird, damit diese mit Grösse- und Helligkeitsfilter klar unterschieden werden können. Zudem darf ein Rechteck keinesfalls so aufgeteilt werden, dass es doppelt erkannt wird. Die Korrektheit ist in diesem Fall die Genauigkeit der Kanten an den Ecken. Diese sind relativ wichtig für die genaue Erkennung der Eckpunkte. Wenn die Kanten an den Ecken nur leicht abgerundet sind, kann das einen resultierenden Eckpunkt relativ stark beeinflussen.

2.1.1 Differenzbildansatz

Bei einem Differenzbild wird auf zwei Bildern für jeden Pixel die Differenz gebildet. Somit entsteht ein Bild auf dem alle statischen Regionen entfernt sind. Dazu gehört zum Beispiel der Hintergrund eines Bildes. Er ist eine Region, die sich auf zwei Bildern nicht ändert und darum verschwindet.

Eckdetektion mit Differenzbildern Bei diesem Verfahren werden auf Differenzbildern entlang von diagonalen Scanlinien Eckpunkte gesucht und so Rechtecke gefunden. Der detaillierte Ablauf ist im Anhang im Kapitel 5.1.1 genau beschrieben. Dieses Verfahren kann auch auf Farbkanäle erweitert werden. Es funktioniert relativ gut auf Bildern mit konstanter Belichtungskorrektur und stark unterschiedlichen Helligkeitsverteilungen. Diese zwei Faktoren sind in der Problemstellung dieser Arbeit jedoch nicht positiv zu beantworten. Deshalb wurde eine Weiterentwicklung dieser Lösung verworfen.

Einfacher Differenzbildansatz Bei den nun folgenden Verfahren wird ein universeller Differenzbild-Hintergrund verwendet. Damit kann der statische Hintergrund des Bildes gut entfernt werden und es muss nur ein zusätzliches Bild projiziert werden. Somit wird nur wenig zusätzliche Zeit für die Kalibrierung benötigt. Dieser Ansatz hat dafür Probleme, wenn die Kamera die Belichtungskorrektur ändert. Dies kommt leider sehr häufig vor, wenn sich die Projektion auf der Leinwand selbst ändert. Da die Änderung der Folie der Sinn jeder Präsentationssoftware ist, wurde dieser Ansatz verworfen. Der genaue Ablauf des Verfahrens ist im Anhang im Kapitel 5.1.3 beschrieben.

Individuelle Differenzbilder Zu jedem Referenzbild, das mit der Kamera gemessen werden soll, werden zwei Bilder generiert, deren Differenzbild genau einem gesuchten Musterbild entspricht. Diese Teilbilder müssen eine ähnliche Helligkeitsverteilung aufweisen. Das verdoppelt die Kalibrierungszeit dadurch, dass die Zahl der anzuzeigenden Bilder

verdoppelt wird. Es resultieren sehr gute Differenzbilder mit genauen Kanten. Die Kalibrierungszeit liegt dabei immer noch im Rahmen des zumutbaren. Das Verfahren entspricht trotzdem den von uns gestellten Anforderungen, da eventuell zusätzlich noch nutzbare Information auf das Wesentliche reduziert wird.

2.1.2 Histogramm-Analyse

Bei der Histogramm-Analyse werden keine Differenzbilder verwendet. Stattdessen wird mit den aktuellen Kamerabildern gearbeitet. Darauf wird die durchschnittliche Helligkeitsverteilung und die Standardabweichung analysiert. Alles was heller ist, wird akzeptiert. Dadurch müssen bei der Kalibrierung keine zusätzlichen Bilder angezeigt werden. Analysen haben jedoch gezeigt, dass der Helligkeitsunterschied zwischen verschiedenen Bildteilen (z.B. oberer und unterer Bildhälfte) viel grösser ist als der Projektionsrand. Daraus ist die Idee entstanden, das Bild für die Analyse aufzuteilen.

Optimierung: Histogramm-Analyse auf Bildausschnitten Wenn das Bild am Anfang in kleine Ausschnitte aufgeteilt wird, kann der Hauptnachteil des obigen Verfahrens eliminiert werden. Die Resultate sind relativ gut, mit dem Nachteil, dass keine klaren Kanten resultieren, wie dies bei Differenzbildern der Fall ist. Eine weitere Konsequenz sind uneinheitliche Übergänge an den Bildausschnitten, die jedoch für die Eckdetektion keine Rolle spielen. Das Rauschen, welches bei diesem Verfahren relativ stark ist, befindet sich zum grössten Teil ausserhalb der Leinwand und kann gut herausgefiltert werden. Dieses Verfahren funktioniert ziemlich gut, aber die Einschränkung der ungenauen Kanten stört relativ stark. Genaue Kanten wären wichtig für die Kalibrierung.

2.2 Kalibrierung: Referenzpunkte

Der Beginn der Kalibrierung bildet das Finden von Referenzpunkten. Diese Punkte bilden die Orientierung für darauf aufbauende Abbildungen. Je nach Genauigkeit der Abbildung sind mehr oder weniger gemessene Referenzpunkte nötig. Die Abbildung vom Kamerabild zum ursprünglich projizierten Beamerbild ist eine Transformation $(x_k, y_k) \rightarrow (x_b, y_b)$ und lässt sich durch eine Transformationsmatrix mit 11 Freiheitsgraden korrekt beschreiben³, wenn man Faktoren wie den Algorithmus des Beamers für die Trapezkorrektur und dessen Linsenverzerrung ausser Acht lässt und annimmt, dass der Beamer das Bild korrekt projiziert, was nicht immer der Fall ist. Dies ist im Paper [1] beschrieben. Die Trapezkorrektur des Beamers fällt einem Zuschauer nicht auf, ist aber für die folgenden Algorithmen ziemlich wichtig. Falls zusätzlich noch die Brennweite der Kamera⁴ ausser Acht gelassen und mit einer bekannten Bildskalierung gearbeitet wird, hat die Transformation lediglich 8 Freiheitsgrade, welche durch 4 Messpunkte gebunden werden können. Die Implementationen des linearen- und des Integralansatzes benötigen nur die vier gemessenen Eckpunkte der Projektion und lassen somit Linsen und andere Effekte ausser

³3 für die Translation, 3 für die Rotation, 2 für den Kameranullpunkt, 1 für die Brennweite, 2 für die Bildskalierung

⁴Diese resultiert in einer Linsenverzerrung

Acht. Messungen haben ergeben, dass auftretende Ungenauigkeiten regelmässig sind und sich grösstenteils auf die horizontale Achse beschränken. Durch eine einfache Korrekturfunktion lassen sich gewisse Effekte vermindern, dass ein gutes Resultat erreicht werden kann.

Bei dieser Arbeit wurden folgende Möglichkeiten, Referenzpunkte zu erhalten, evaluiert und getestet. Dabei ist die Zahl der erhaltenen Referenzpunkte sehr unterschiedlich. Die Ansprüche an die Verfahren sind relativ hoch, da ein einzelner falsch zugeordneter Referenzpunkt die Interpolation auf einer grossen Fläche stark beeinträchtigen oder unmöglichen kann. Dafür ist die Quantität nicht besonders wichtig sobald die für das Verfahren minimal erforderliche Anzahl korrekt zugeordnet wurde. Wenn dann einzelne Punkte fehlen, beeinträchtigt das die Qualität der Interpolation, was aber nur geringe Auswirkungen hat, die kaum bemerkbar sind.

2.2.1 Dies Erkennen der Bildschirmgrenze

Mit dem nachfolgend beschriebenen Verfahren werden lediglich die Ecken des Bildschirms erkannt. Dafür gibt es mehrere Varianten, die im nächsten Abschnitt evaluiert werden. Anhand von vier Eckpunkten kann die perspektivische Verzerrung schon korrigiert werden. Aufgrund der Einfachheit, ist dieses Verfahren sehr stabil und fehlertolerant. Ein weiterer Vorteil besteht darin, dass der Benutzer selber verifizieren kann, anhand des Kamerabildes, ob die Ecken richtig erkannt wurden. Ein Nachteil dieses Verfahrens besteht darin, dass es keine zusätzlichen Referenzpunkte gibt, um die Erkennung zu verbessern. Es wurde jedoch trotzdem im Prototyp eingesetzt, da es gute Genauigkeit und Fehlertoleranz mitbringt.

2.2.2 Ordnung kleiner Vierecke

In den folgenden Algorithmen werden mehrfarbige Schachbrettmuster analysiert. Dies hat den Vorteil, dass die Farbkanäle getrennt sind, und so ein Gitter mit Quadraten resultiert, die nur einen direkten Nachbarn haben. Dieser Ansatz ist im Anhang bei Kapitel 52 noch genauer beschrieben. Die folgend beschriebenen Varianten können generell für das Finden von Vierecken verwendet werden und haben keinen direkten Zusammenhang mit der Bildschirmdetektion per Schachbrettmuster.

Rekursives Traversieren von Vierecken Auf einem mehrfarbigen Schachbrettmuster werden Quadrate erkannt. Alle Quadrate werden vom Rand her durch rekursive Iteration erkannt. Die Iteration hat die Struktur eines Baumes. Das Verfahren ist in der Theorie sehr stabil und kann sehr fein angepasst werden. Der genaue Ablauf dieses Verfahrens ist im Anhang 5.2.1 beschrieben. Bei fehlenden Erkennungen kann der Algorithmus sogar um eine Lücke herumgehen. Bei Versuchen wurde jedoch festgestellt, dass dieses Verfahren sehr fehleranfällig ist. Es sind nur 90% Korrektheit möglich. Verbesserungen lassen sich zwar umsetzen, jedoch führen diese zu einer stark erhöhten Komplexität und tragen nur sehr wenig zu einer Verbesserung bei. Weitere Probleme bereiten diesem Algorithmus nicht erkannte Quadrate, da dann die Diagonale als direkter Nachbar erkannt wird, was

verheerende Folgen hat. Zudem ist keine Fehlerdetektion möglich weil eine falsche Annahme getroffen wird, die später weder korrigiert noch detektiert werden kann, da jede weitere Detektion von ihr abhängt.

Nutzung der Interpolation Hierbei werden die Rechtecke anhand einer Interpolation zu den Eckpunkten zugeordnet. Dieses Verfahren ist sehr einfach⁵ und es können Fehler mit einer Abweichung einer halben Seitenlänge eines Rechtecks korrigiert und zusätzlich eine ganze Seitenlänge detektiert werden. Zudem ist dieser Algorithmus sehr performant. Er ist allerdings auch von einem guten Interpolationsverfahren abhängig, was wiederum Referenzpunkte erübrigt, da diese nicht mehr benötigt werden. Dieser Algorithmus kann durch eine Vergrößerung der Rechtecke stabilisiert werden, da die Flächen vergrößert werden, was wiederum zu einer geringeren Fehlerwahrscheinlichkeit führt. Dies resultiert aber wiederum in weniger Referenzpunkten, da weniger angezeigt werden können. Mit einem schlechten Interpolationsverfahren kann es zu gravierenden Fehlerkennungen kommen.

2.2.3 K-Means Clustering

Hiermit werden auf dem Bildschirm immer feiner aufgeteilte Cluster erstellt. Dieses Verfahren bietet gute Möglichkeiten zur Fehlerdetektion und liefert gute Ergebnisse. Bilder und eine detaillierte Beschreibung ist im Anhang unter Kapitel 5.2.3 zu finden. Die Autoren empfehlen dazu das Paper [5]. Mit der Fehlerdetektion können alle üblichen Fehlerkennungen ausgeschlossen werden. Allerdings ist sie ziemlich intolerant gegenüber Fehlerkennungen wie z.B. Rauschen oder fehlende Quadrate. Zudem stellt sie ziemlich hohe Anforderungen an das Kamerabild. Wenn das Bild auch nach mehreren Versuchen nicht korrekt erkannt werden kann, scheitert die Kalibrierung. Danach müsste der Benutzer die Beleuchtung ändern oder seine Webcam umpositionieren. Nicht alle Webcams liefern Bilder, die auch bei Optimalbedingungen den Qualitätsansprüchen dieses Ansatzes genügen. Ebenso ist die Anzahl der nutzbaren Referenzpunkte von der Kameraauflösung abhängig. Dieses Verfahren ist sehr genau und stabil, jedoch stellt es erhöhte Anforderungen, die dazu führen, dass die Kalibrierung nicht immer klappt.

2.2.4 OpenCV

OpenCV bietet die Möglichkeit, anhand verschiedener Aufnahmen eines Referenzbildes einen ganzen dreidimensionalen Raum zu rekonstruieren. Dafür wird für jede erkannte Ebene eine Transformationsmatrix erstellt. Jene werden anschliessend so kombiniert dass die resultierende Matrix eine Rücktransformation auf den Raum erlaubt. Da Abbildungen sich in dieser Arbeit jeweils auf eine einzelne Ebene beziehen, hätte der OpenCV-Algorithmus stark angepasst werden müssen. Dies war jedoch aufgrund der enormen Komplexität dieses Codes nicht einfach möglich. In einem weiteren Versuch wurde mit verschiedenen Testbildern versucht, einzelne Transformationsmatrizen zu erhalten. OpenCV hat jedoch zu keinem einzigen Testbild ein Resultat geliefert, obwohl die spezifizierten

⁵Detaillierte Beschreibung im Anhang unter 5.2.2

Referenzbilder verwendet wurden. Zudem liefert OpenCV keine Fehlerbeschreibungen, die auf die Möglichkeit von Anpassungen hingewiesen hätten. Ein weiterer Nachteil besteht darin, dass das komplette 600 MB grosse Framework mitgeliefert werden müsste, auch wenn nur ein Teil davon verwendet wird.

2.3 Kalibrierung: Abbildung & Interpolation

Um einen Stift zu tracken ist es notwendig, dass im Kamerabild gefundene Punkte auf ursprüngliche Punkte im projizierten Bild (Beamerbild) abgebildet werden können (*Kamerakoordinaten*→*Beamerkoordinaten*). In der Praxis wird dies aufgrund von Beschränkungen wie der tiefen Kameraauflösung und der Grösse des Stiftes nicht perfekt möglich sein. Es gilt eine Funktion zu finden, welche diese Abbildung möglichst gut annähert. Dazu wurden verschiedene Ansätze analysiert, welche im Folgenden beschrieben werden.

2.3.1 Linear

Der lineare Ansatz berücksichtigt weder perspektivische Verzerrung noch Linseneffekte. Es wird davon ausgegangen, dass die Verhältnisse der Punkte zu den Rändern in beiden Abbildungen identisch sind. Wenn keine perspektivische Verzerrung vorhanden ist, funktioniert dieses Verfahren korrekt. Das heisst, dass die Genauigkeit dieses Verfahrens abhängig von der Distanz zwischen der Linse des Beamers und der Kamera ist. Wenn diese nebeneinanderstehen funktioniert dieses Verfahren sehr gut. Bei diesem Verfahren werden nur die Eckpunkte für die Berechnung benötigt. Da dieses Verfahren nur in eine Richtung funktioniert, müssen einmalig alle möglichen Werte berechnet werden, dafür sind nachher keine Berechnungen mehr nötig, ein einfacher Lookup reicht dafür. Die komplette Formel befindet sich im Anhang unter Kapitel 5.3.1.

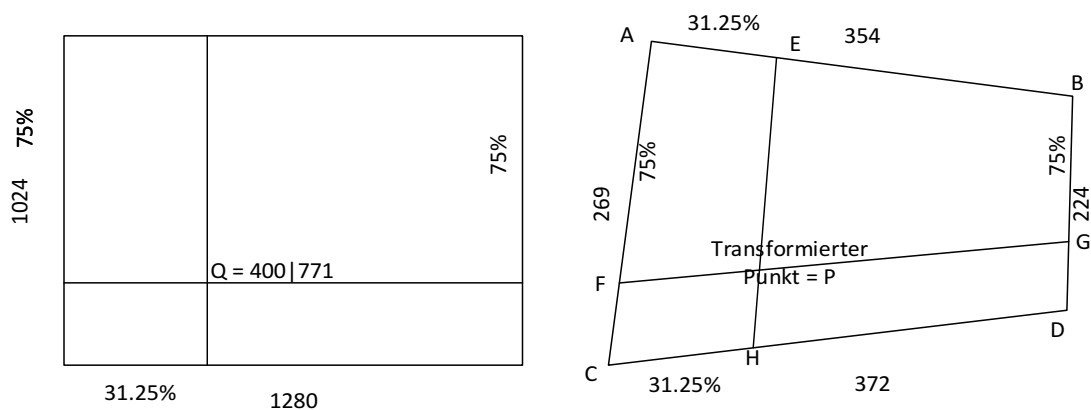


Abb. 13: Lineare Interpolation

Problem Der lineare Ansatz wird mathematisch durch eine Lochkamera und den Strahlensatz beschrieben. Der Strahlensatz funktioniert nur, wenn Kamerabild und Leinwand zueinander parallel sind. In einem üblichen Setup von Beamer und Laptop ist dies nicht der Fall. In Abb. 14 dies ersichtlich. Die Messung dazu befindet sich im Anhang in Abb. 55. Sie zeigt, dass der lineare Ansatz sehr gute Resultate liefert, wenn Beamer und Kamera nahe beieinander stehen.

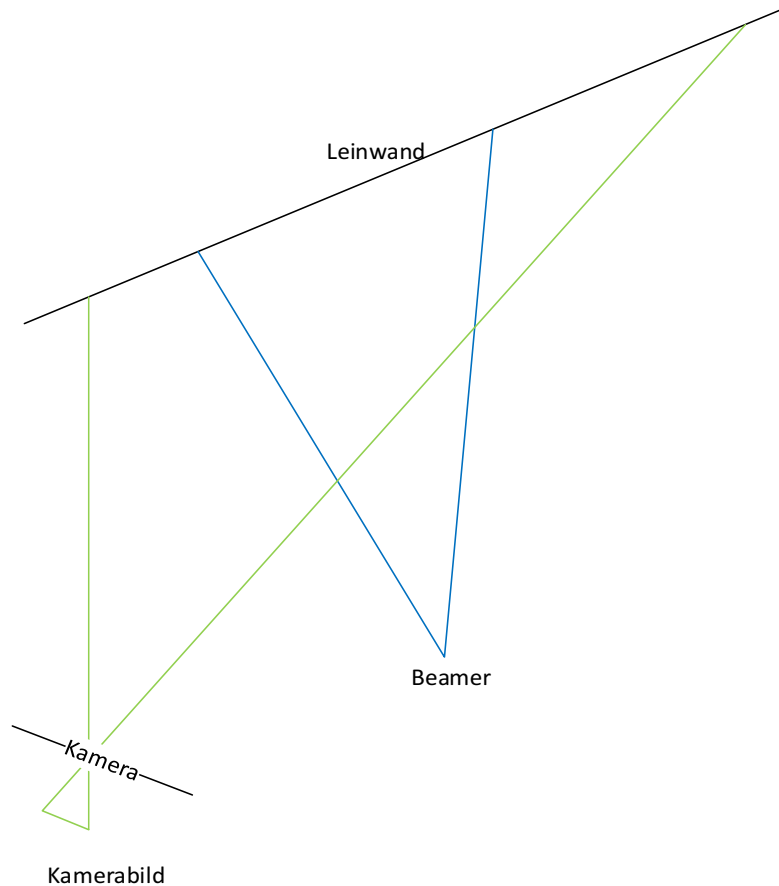


Abb. 14: Lochkameramodell - Leinwand und Kamerabild sind nicht parallel.

2.3.2 Baryzentrische Koordinaten

Hierbei werden die baryzentrischen Koordinaten⁶ aus verschiedenen Dreiecken berechnet, zurückgerechnet und anschliessend alle Resultate gemittelt. Dadurch erhält man eine Genauigkeit, die proportional zur Anzahl Referenzpunkte steigt. Zudem ist der Algorithmus relativ einfach verständlich und gut testbar. Die Berechnung mit baryzentrischen Koordinaten berücksichtigt allerdings nicht, dass die Abbildung nicht Winkeltreu ist, was zu

⁶Baryzentrische Koordinaten sind unter 5.3.2 kurz erklärt. Ausführliche Literatur zu diesem Thema findet man im Internet oder in einem der Paper: [6, 8, 4, 7, 3]

Fehlern und Ungenauigkeiten führt. Für die Mittelung ist leider nur eine begrenzte Gewichtung möglich, da keine Abstandsfunktion definiert ist. Durch Übergänge und die mangelnde Gewichtung erhält man an den Übergängen unter Umständen Knicke in geraden Linien, was der Benutzer als störend empfindet. Wenn viele Dreiecke verwendet werden, führt dies zu einer besseren Korrektur von lokalen Ungenauigkeiten und etwas sanfteren Übergängen, allerdings wird dadurch auch der Rechenaufwand erhöht.

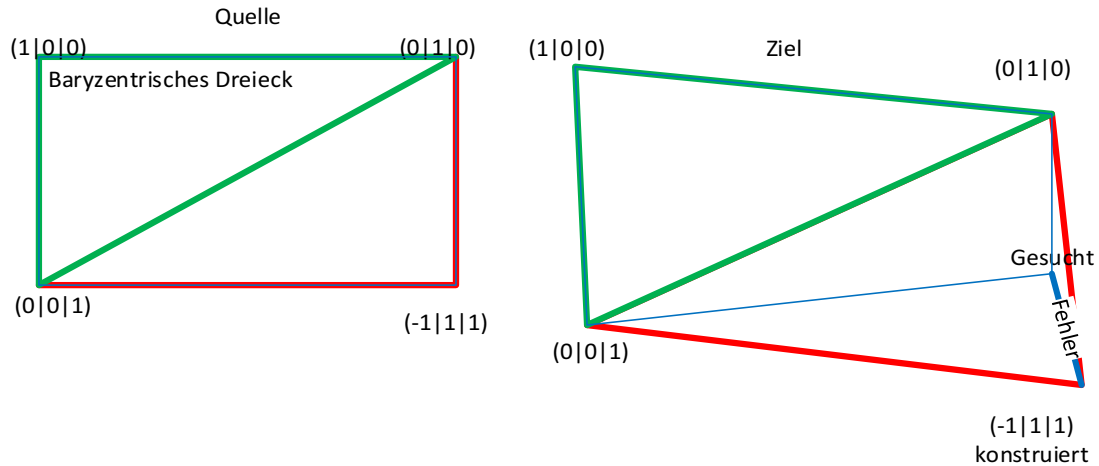


Abb. 15: Fehler bei nicht winkeltreuen Abbildungen

Optimierung: Baryzentrische Vierecke Durch eine eigene Erweiterung der baryzentrischen Koordinaten konnte berücksichtigt werden, dass Vierecke nicht winkeltreu sind. Dies geschieht durch einen vierten Punkt im Koordinatensystem und führt zu einer viel genaueren Interpolation. Der Algorithmus benötigt jedoch eine strukturierte Ordnung der Referenzpunkte, was ihn geringfügig komplizierter macht. Allerdings ist durch die Ordnung auch eine Gewichtung möglich. Dieser Algorithmus führt zu einem guten Resultat, allerdings stellt er hohe Anforderung an die Referenzpunkte. Diese Ansprüche können allerdings mithilfe des K-Means Clusterings erfüllt werden.

Algorithmus 1 Auf 4 Dimensionen erweiterte baryzentrische Koordinaten

Es wurde zu den bestehenden Gleichungen der baryzentrischen Dreieckskoordinaten noch eine vierte Gleichung hinzugefügt. Diese hat das Ziel, nahe Eckpunkte stärker zu gewichten.

P_{TL} := obere linke Ecke

P_{TR} := obere rechte Ecke

P_{BL} := untere linke Ecke

P_{BR} := untere rechte Ecke

$P := (P_X, P_Y)$: zu transformierender Punkt

$T := (T_X, T_Y)$: transformierter Punkt

$$T = P_{TL}(1 - P_x)(1 - P_y) + P_{BL}P_y(1 - P_x) + P_{TR}(1 - P_y)P_x + P_{BR}P_xP_y \quad (1)$$

Diese Gleichung formuliert eine Transformation vom Beamer zum Kamerabild. Um die Berechnung in Richtung Kamerabild \rightarrow Beamerbeamer durchführen zu können, muss zur Umkehrung die Determinante konstruiert werden:

$$Det \begin{bmatrix} (1-x)P_{TL} + xP_{TR} - P & (1-x)P_{BL} + xP - P \\ (1-y)P_{TL} + yP_{TR} - P & (1-y)P_{BL} + yP_{BR} - P \end{bmatrix} \quad (2)$$

Daraus resultieren zwei Gleichungen. Diese können an den Nullstellen nach x und y aufgelöst werden:

$$Solve(Formel(2) = 0, \{x, y\}) \quad (3)$$

Aus dem Resultat können die Beamerkoordinaten aus Kamerakoordinaten berechnet werden. Aufgrund der Länge der resultierenden Gleichung wird hier nicht weiter auf das Auflösen der Gleichung eingegangen.

Bemerkung Es ist zu beachten, dass dieser Algorithmus lediglich eine geometrisch gleichmässige Verzerrung ausgleicht. Perspektivische Verzerrungen bleiben bestehen. Zudem funktioniert er auch in der näheren Umgebung ausserhalb des Vierecks noch.

2.3.3 Zweidimensional per Integration

Der schlussendlich in der Software verwendete Ansatz zur Punktabbildung berücksichtigt die perspektivische Verzerrung in X-Richtung. Dafür wird ein vereinfachtes zweidimensionales Lochkameramodel verwendet. Der leichtere Einstieg bietet die Abbildung von Beamer zu Kamera. Die mathematische Umkehrung wird erst nach der nun folgenden Herleitung gemacht.

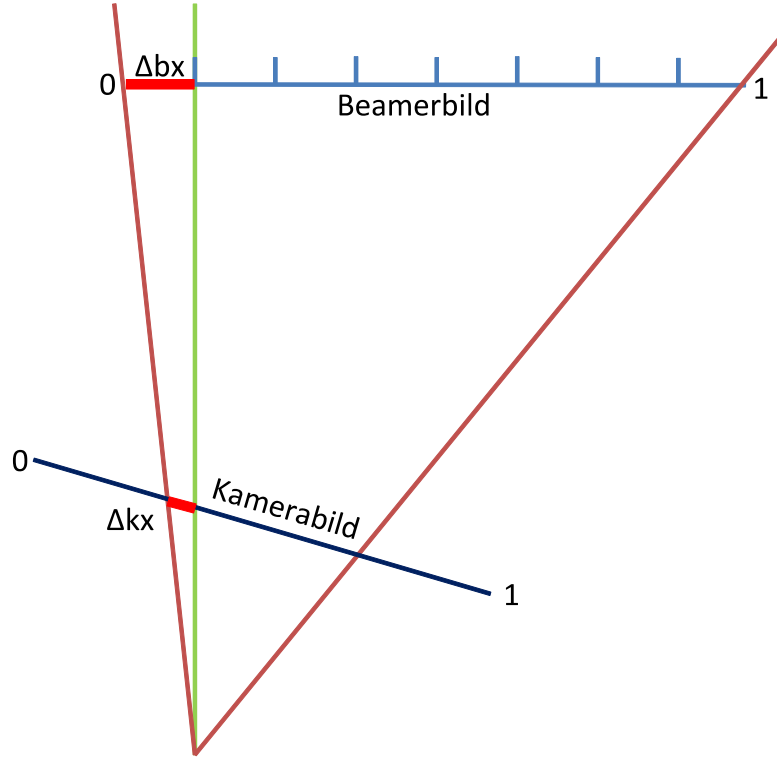


Abb. 16: Vereinfachtes Lochkameramodell - Es wird über die Strecke des Beamerbild in X-Richtung integriert.

Die Abbildungsfunktion $Beamer \rightarrow Kamera$ benutzt die Tatsache, dass Strecken von Beamerbild zu Kamerabild verschieden aber herleitbar verschieden verzerrt sind. Konkret wird benutzt, dass eine Strecke links auf dem Beamerbild um einen Faktor a gestreckt oder gestaucht auf das Kamerabild abgebildet wird. Auf der rechten Seite des Beamerbildes ist dies ein anderer Faktor b . Über die X-Koordinaten des Beamerbildes kann nun ein Integral definiert werden, welches diese Abbildung von links nach recht durchläuft und all diese Streckungen/Stauchungen infinitesimal⁷ aufsummiert bis zur Beamerposition x_b , die gesucht ist.

$$x_k(x_b) = \int_0^x a(1 - x_b) + bx_b \delta x_b = \frac{1}{2}(b - a)x_b^2 + ax_b \quad (4)$$

Die Mathematische Umkehrfunktion ($Kamera \rightarrow Beamer$) bildet jedes x_k auf ein x_b ab und ist durch das Lösen der quadratischen Gleichung (4) herleitbar.

$$x_b(x_k) = \frac{\sqrt{a^2 + 2x_k(b - a)} - a}{b - a} \quad (5)$$

⁷in unendlich kleinen Schritten

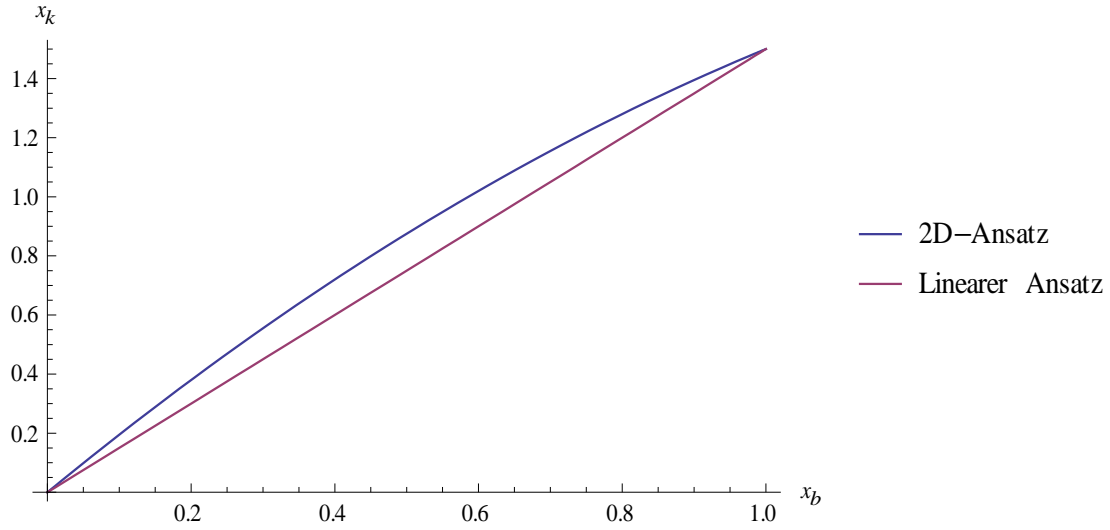


Abb. 17: Graph der Gleichung (5) für $a = 2$, $b = 1$ (blau): In der Mitte weicht die perspektivische Verzerrung in X-Richtung mehr vom linearen Ansatz (violett) ab als am Rand.

Zu beachten gilt, dass der Ursprung des verzerrten Beamerbildes nicht unbedingt auf den Ursprung des Kamerabildes zu liegen kommt. x_b und x_k müssen also um einen gemessenen Offset angepasst werden, bevor die jeweilige Abbildung ausgerechnet wird. In der hier beschriebenen Lösung wird nicht der korrekte Offset verwendet (Schnittpunkt der zwei zur Gerade verlängerten Abschnitte Kamerabild und Beamerbild). Es wird jeweils nur der Offset zum Kamerabildanfang verwendet. Dies hat zur Folge, dass resultierende X-Koordinaten gleichmässig zu lang oder zu kurz sind (je nach Abbildungsrichtung). Damit der Schnittpunkt der zwei Ebenen (Kamera und Beamer) nicht berechnet werden muss, wird in dieser Arbeit jeweils die berechnete X-Strecke auf die berechnete Maximalstrecke gegenüber der gemessenen Maximalstrecke normiert (Stauchung s).

$$s = \frac{x_{maxmeasure}}{x_{maxcalc}} \quad (6)$$

Für die Abbildung der Y-Koordinaten wird ein von X abhängiger linearer Ansatz verwendet. Wieder spielt dabei der Gedanke eine Rolle, dass die rechte und die linke Seite des verzerrten Kamerabildes etwa parallel sind. y_k wird durch eine von y_b abhängige Gerade definiert - sie wird mit x_b durchlaufen. Die Gleichung (7) beschreibt die Abbildung (*Beamer*→*Kamera*). Dabei ist y_{k0} der vertikale Offset zwischen linker und rechter Bildlinie. Als Bildlinie wird der Rand des Bildes bezeichnet. Hier sind die betrachteten Bildlinien der linke und der rechte Rand des verzerrten projizierten Beamerbildes.

$$y_k(x_b) = (ay_b + y_{k0})(1 - x_b) + by_b x_b \quad (7)$$

Die Mathematische Umkehrfunktion (*Kamera*→*Beamer*) bildet jedes y_k auf ein y_b ab

und ist mit Gleichung (8) beschrieben. Steigung und Offset der Geraden wird dabei durch x_b kontrolliert.

$$y_b(y_k) = \frac{x_b y_{k0} + y_k - y_{k0}}{a(-x_b) + a + b x_b} \quad (8)$$

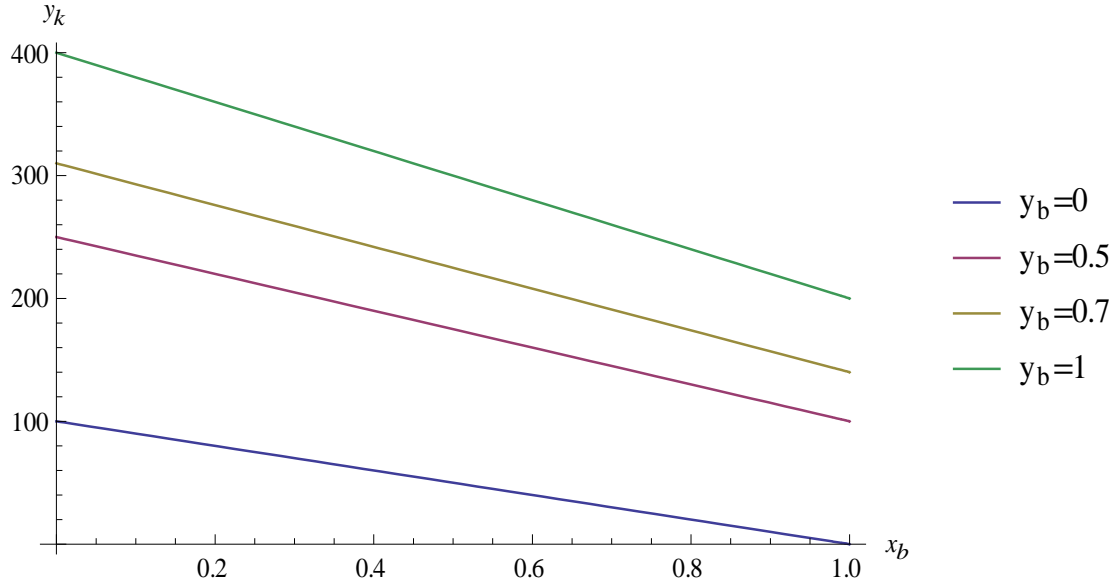


Abb. 18: Samples der Gleichung (7). Der gezeigte Ausschnitt bildet genau das verzerrte Beamerbild nach, wobei x_b auf einen Wert zwischen 0 und 1 normiert ist. y_k ist der Pixelwert in Y-Richtung auf dem Kamerabild.

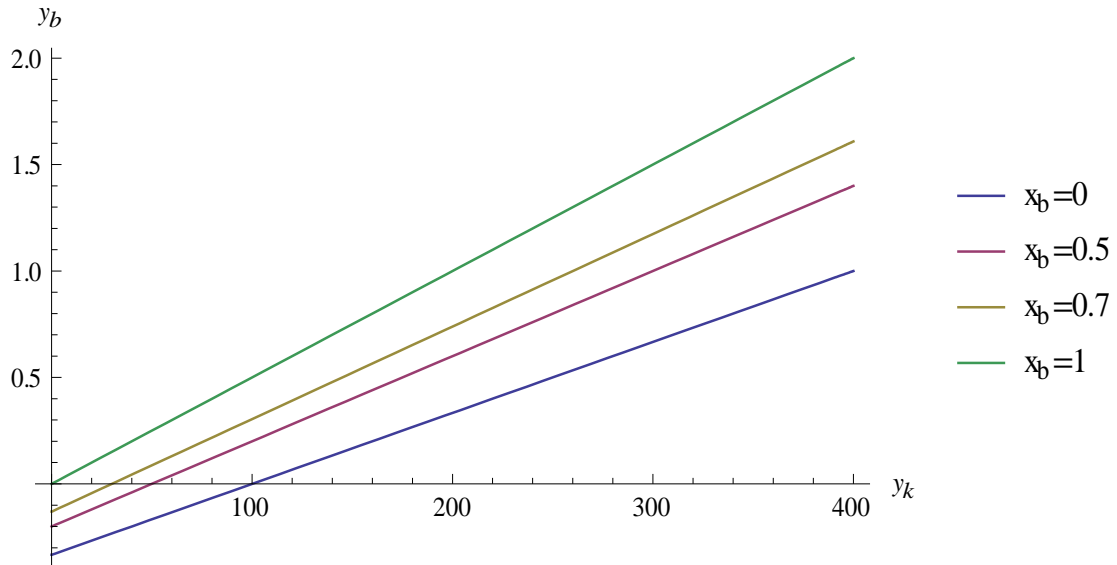


Abb. 19: Samples der Gleichung (8). Werte zwischen 0 und 400 werden auf das Beamerbild rückprojiziert. Das Beamerbild wird jedoch nur getroffen, wenn y_b zwischen 0 und 1 resultiert.

Beispiel Die Güte dieses Ansatzes wurde gemessen. Dazu wurde ein Bild an eine Leinwand projiziert und mit einer einfachen Laptopkamera aufgenommen. Die verzerrten projizierten Punkte des Beamerbildes wurden manuell auf dem Kamerabild gemessen und den ursprünglichen Punktpositionen auf dem Beamerbild zugewiesen. Die gemessenen Punkte des Kamerabildes wurden mit den in den Gleichungen (5) und (8) beschriebenen Ansätzen auf Beamerkoordinaten umgerechnet. In Abb. 20 ist das Resultat ersichtlich. Die Schräge im rechten Teil der Abbildung rührt davon, dass die linke und die rechte Seite des abgebildeten Beamerbildes nicht genau parallel sind. Die Abb. 21 zeigt die Rückberechnung der abgebildeten auf die abzubildenden Koordinaten. Im Weiteren ist anzumerken, dass als linke und rechte Bildlinien jeweils genau senkrecht verlaufende gemittelte Geraden verwendet wurden.

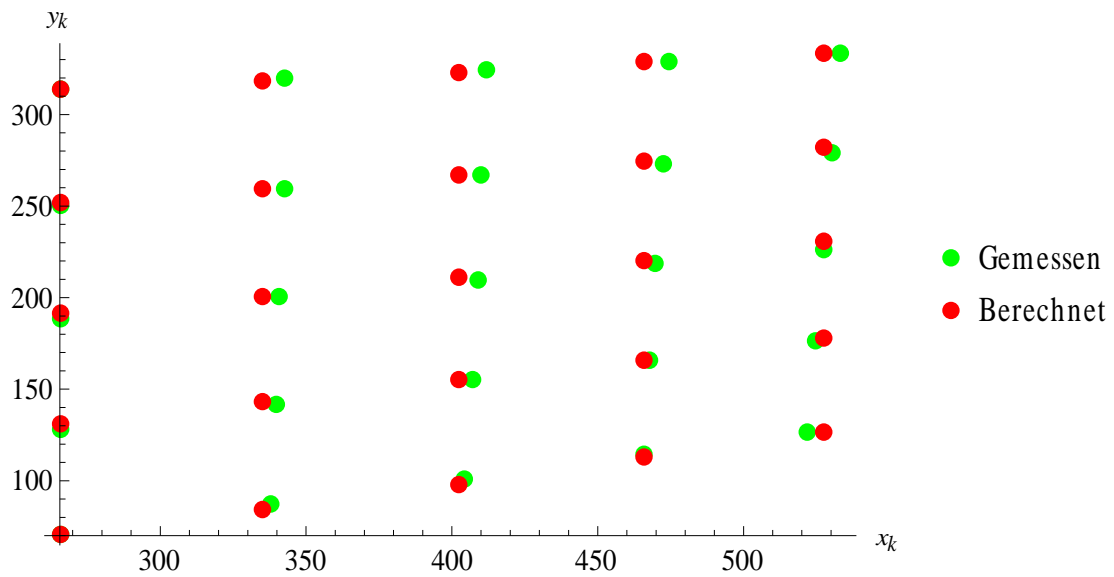


Abb. 20: Die gemessenen Punkte weichen von den geschätzten (berechneten) ab.

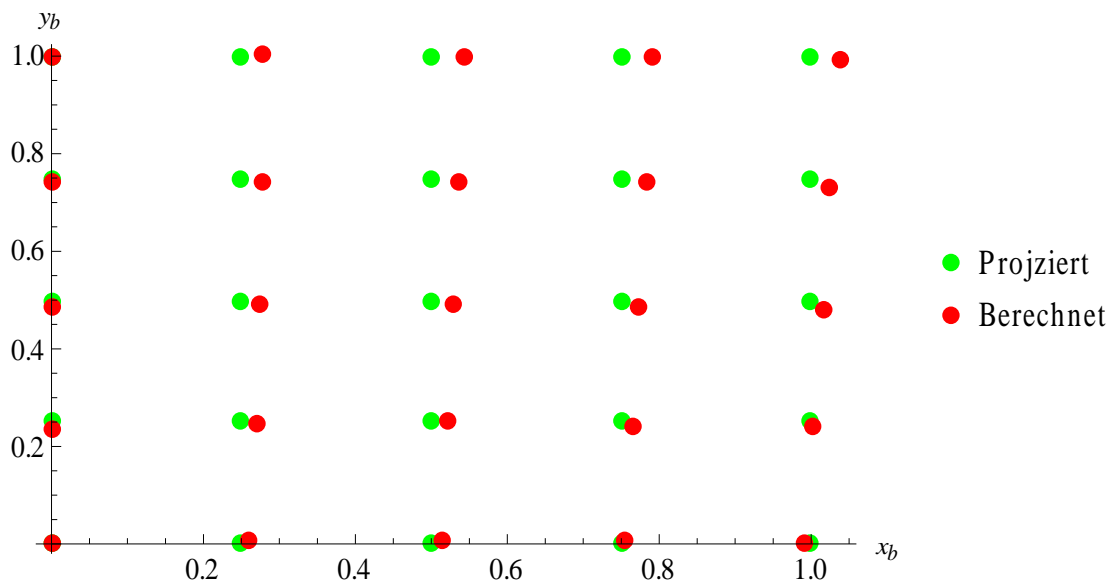


Abb. 21: Vergleich zwischen Projektion und Berechnung. Die Schräge der linken und rechten Bildlinien ergibt Abweichungen.

Baryzentrische Viereck-Korrektur Die im Beispiel ersichtlichen Abweichungen lassen sich zumindest an den Ecken mit einer baryzentrischen Viereck-Korrektur beheben. Dieses Verfahren „zieht“ alle Punkte in Richtung der Referenzpunkte eines vierdimensionalen

baryzentrischen Koordinatensystems. Das Resultat ist in Abb. 22 blau markiert. Es wird an den meisten Orten besser. Sehr viel besser wird es an den Rändern. Dieses Verfahren kann auch bei der Rückabbildung (*Kamera*→*Beamer*) angewendet werden. Das genaue Verfahren zur baryzentrischen Viereckskorrektur wird unter Algorithmus (1) beschrieben.

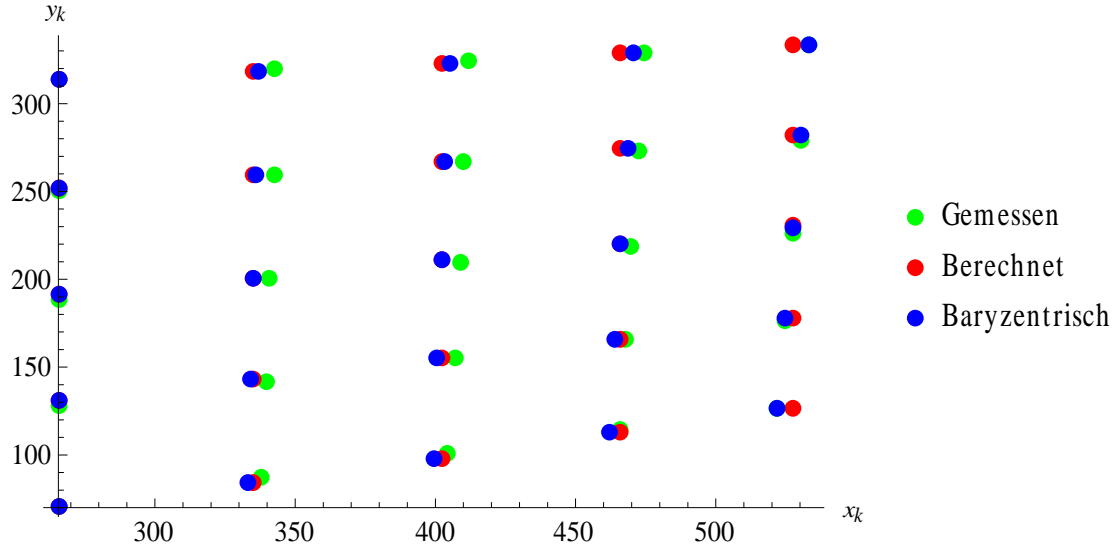


Abb. 22: Transformation (*Beamer*→*Kamera*): Baryzentrische Korrektur ergibt beinahe überall ein besseres Resultat.

Pragmatische Korrektur Tests in der Praxis haben ergeben, dass nicht nur die perspektivische Verzerrung eine Rolle spielt. Vor allem wenn der Winkel zwischen Kamera und Beamer vergrößert wird, sind systematisch stärkere Abweichungen sichtbar. Linseneffekte und Trapezkorrektur des Beamers wurden in dieser Arbeit nicht weiter modelliert. Stattdessen wird für alle zusätzlichen Abweichungen eine quadratische Korrekturfunktion über die X-Richtung gelegt. Sie soll nicht ausgearbeitete Abweichungen symmetrisch ausgleichen. Die Gleichung (9) beschreibt die zur Korrektur verwendete Funktion. Sie geht durch 0 und 1 und hat ein durch h festgelegtes Maximum bei 0.5. Wie h zu wählen ist, hängt von der bei 0.5 zu korrigierenden Abweichung ab.

$$y_{corr}(x) = -4hx^2 + 4hx \quad (9)$$

2.3.4 Homogene Transformation

Die Abbildung von Punkten zwischen Kamera und Beamer mittels homogener Transformation wurde nur kurz angetestet. Eine sauber erarbeitete Transformationsmatrix verspricht sehr genaue Resultate. Die für das traditionelle Lochkameramodell benötigte Matrix wird durch 11 Freiheitsgrade beschrieben⁸. Es reichen also 5.5 gemessene Refe-

⁸3 Translation, 3 Rotation, 2 Kameranullpunkt, 1 Brennweite, 2 Bildskalierung

renzungspunktpaare. Da dieses Vorgehen erweiterte mathematische Kenntnisse voraussetzt, wurde es nicht genauer untersucht. Programmbibliotheken wie OpenCV unterstützen diese Art der Transformation nativ und schnell⁹.

Externe Bibliotheken Es gibt die Möglichkeit, eine Kamerakalibrierung mit vollständiger Transformationsmatrix (11 Freiheitsgraden) mit OpenCV vollautomatisch vornehmen zu lassen. Diese Funktionalität konnte mit OpenCV nicht erfolgreich getestet werden. Aufgrund dessen wurde schliesslich vollständig auf OpenCV verzichtet.

2.4 Stift Erkennen

Als Ziel dieser Arbeit soll ein Stift an einer Leinwand gefunden werden. Dies soll das virtuelle Schreiben auf ihr ermöglichen. Da Bildverarbeitung auch heute noch viel Rechenleistung benötigt, gilt es genau zu wissen, was man wann und wieso tut. Das Finden eines Stiftes besteht aus verschiedenen Teilproblemen, die verschieden zu lösen sind. So sind zum Beispiel Präsenz- und Positionsaspekt des Stiftes zu unterscheiden. Es ist teilweise nicht unbedingt notwendig zu wissen, wo der Stift auf dem Bild ist. Falls es einfach ist, die Präsenz des Stiftes festzustellen, nicht aber die Position, kann erstere Information verwendet werden um komplizierte und rechenintensive Schritte erst anzustossen wenn nötig. Neben den verschiedenen vorzunehmenden Schritten, ist auch Kontextwissen wichtig. Ein Bild kann gezielter nach Präsenz und Position des Stiftes durchsucht werden, wenn bekannt ist, wie er sich zeitlich bewegt.

2.4.1 Präsenz durch Helligkeit

Generell ist es nicht einfach, herauszufinden, ob sich ein Objekt in einem Bild befindet. Ansätze aus der digitalen Bildverarbeitung wie die Diskrete Fourier-Transformation¹⁰ oder der Gradientenabstieg mittels dynamischer Bildskalierung¹¹ sind kompliziert und benötigen viel Rechenleistung. Die Komplexität des Problems wird immens reduziert, wenn die Eigenschaften des Stiftes klarer definiert sind. Es wäre möglich, Form, Farbe, Helligkeit, Farbtemperatur, Geschwindigkeit, Bewegung, Puls, etc. auf einem oder mehreren Kamerabildern zu analysieren. Der Einfachheit halber wird der Stift auf seine Helligkeit reduziert. Er ist der hellste Punkt in einem Bild. Zur Umsetzung könnte ein optisches Licht (ev. Infrarot) als Stift verwendet werden.

2.4.2 Präsenz durch Kardinalität

Der hellste Punkt auf dem Kamerabild bildet den Stift ab. Es gibt verschiedene Möglichkeiten den Fall, dass mehrere Spot auftreten, zu behandeln.

⁹Der interessierte Leser möge folgenden Artikel über die benötigten Transformationsmatrizen konsultieren: http://de.wikipedia.org/wiki/Homogene_Koordinaten#Wichtige_elementare_homogene_Transformationsmatrizen

¹⁰Die Diskrete Fourier-Transformation erhält die Eigenschaften Verschiebung und Skalierung in Zeit und Frequenz.

¹¹Literaturhinweis [2]

- Es könnte lediglich der hellste Punkt gewertet werden.
- Es könnte eine genaue Anzahl Punkte aus dem Kontext gewählt werden.
- Es könnten alle Punkte verworfen werden.

Um Fehler zu vermeiden, scheint es sinnvoll, alle undefinierten Zustände zu verwerfen, Punkte also nur zu werten, wenn sie alleine in ihrem definierten Zustand auftreten. Es befindet sich also ein Stift auf dem Kamerabild, wenn ein gewisser Helligkeitswert überschritten wird. Diese Entscheidung ist nur sinnvoll, wenn die Fehlerrate (zu viele Punkte erkannt) nicht zu hoch und die Framerate nicht zu tief ist.

2.4.3 Position durch Schwerpunkt

Die Position des Stiftes hängt vorzugsweise von der Präsenz eines Stiftes ab. So muss die Position des Stiftes nur dann gesucht werden wenn er präsent ist. Dies spart Rechenzeit. Die oben beschriebenen Kriterien zur Stiftpräsenz definieren, ob sich ein Stift im Bild befindet.

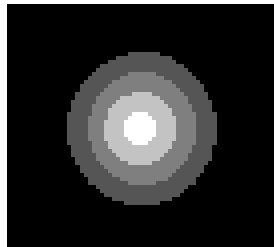


Abb. 23: Stehender Lichtpunkt in der Theorie

Die genaue Position des Stiftes befindet sich in der Mittel am hellsten Punkt. In der Theorie ist das der Mittelwert aller hellen umgebenden Pixel oder der Schwerpunkt.

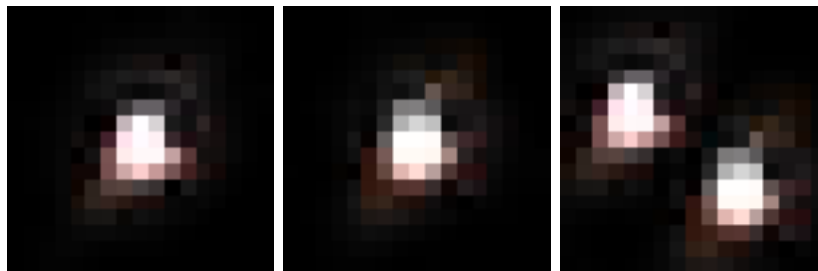


Abb. 24: Frame x, Frame x+1, Differenz - Der Punkt bewegt sich nach rechts unten.

Messungen zeigen, dass ein bewegter Lichtpunkt in der Praxis nicht rund ist, sondern in der Bewegungsrichtung gestaucht. Die Stauchung wird durch die Belichtung verursacht. In der Bewegungsrichtung erfährt der CCD-Sensor der Kamera eine starke Intensitätsänderung, während sich die Intensität am Rand der Laufrichtung weniger stark ändert.

Der CCD trägt der starken Änderung mehr Rechnung als der schwachen. Im Weiteren ist die Helligkeit des Lichtpunktes vom Zentrum her nicht weich abfallend, sonder eher scharfkantig. Dies rührt von Reflexionen auf der dahinter liegenden Leinwand.

Um der Stabilität Willen wird vorgeschlagen, die Position der Stiftposition nicht zu genau zu bestimmen. Für eine Anwendung, die lediglich auf eine Leinwand zeichnen kann, sollte die Mitte einer Umrandung der hellsten Werte mittels fixem Grenzwert reichen.

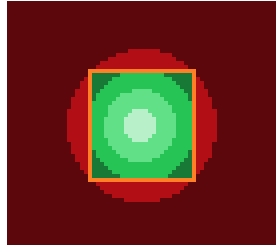


Abb. 25: Per Grenzwert wird der Lichtpunkt in das Zentrum eines Vierecks gelegt.

2.4.4 Performance durch gezieltes Suchen

Ein Algorithmus zum Finden der Stiftposition sollte den erarbeiteten Kontext berücksichtigen. In einem Bild kann zum Beispiel in der Nähe eines im vorherigen Bild gefundenen Punktes gesucht werden. So reduziert sich der Aufwand des Algorithmus von $O(cn^2)$ auf $O\left(c\left(\frac{n}{x}\right)^2\right)$ wobei x die Seitenlänge n des Kamerabildes in ein kleineres Suchfenster mit Seitenlänge $\frac{n}{x}$ teilt¹². Im Fall, dass entweder keine oder keine brauchbaren Kontextinformationen vorhanden sind, kann die Suche auf ein grösseres oder das ganze Suchfenster ausgeweitet werden. Dies erhöht die Worst-Case Laufzeit auf $O\left(c\left(\frac{n}{x}\right)^2 + c\left(y\frac{n}{x}\right)^2\right)$, wobei y der Faktor der Vergrößerung ist. Es sollte reichen, $y := x$ zu setzen, wenn das erste Suchfenster gut gewählt ist, da die Wahrscheinlichkeit gross ist, dass entweder gar kein Punkt gefunden wird oder dass sich der Punkte entgegen dem Kontext verhält¹³.

2.4.5 Stabilität durch Differenzbilder

Reduktion des Kontextes muss nicht unbedingt wie oben beschrieben zu Instabilität und mehr Fallunterscheidungen führen. Zum Beispiel können relevante Informationen durch Differenzbilder hervorgehoben und Rauschen unterdrückt werden. Sogar Kriterien, die mit den Eigenschaften eines Stiftes konkurrieren, wie zum Beispiel helle statische Reflexionen auf der Leinwand können so unterdrückt werden. Falls Differenzbilder verwendet werden, müssen weitere Überlegungen zur Kardinalität der Lichtpunkte gemacht werden. Was nun ein oder zwei resultierende Punkte auf dem Bild bedeuten ist nicht ganz klar (s.u.).

¹²Asymptotisch wird angenommen, dass das Kamerabild quadratisch ist.

¹³Dies würde den kontextabhängigen Algorithmus per se unbrauchbar machen. Ein Fallback, der alle vorhandenen Informationen neu in die Suche mit einbezieht ist also gerechtfertigt.



Abb. 26: Auf dem Differenzbild (rechts) ist der Laserpointer Punkt viel besser sichtbar als auf dem Quellbild (links).

3 Umsetzung

Im diesem Artikel wird beschrieben, welche Verfahren zur Umsetzung der oben beschriebenen Analysen verwendet wurden. Zuerst wird auf technische Details wie den Aufbau des Programms und verwendete Bibliotheken eingegangen. Danach wird erläutert wie die oben beschriebenen Kalibrierungsverfahren eingesetzt wurden und wie das Finden eines Stiftes im Softwareprototyp vonstatten geht. Die Umsetzung der Punktabbildung von Kamera zu Beamer wurde gänzlich ausgelassen, da die oben beschriebene Theorie zum Integral-Ansatz genau nach Modell eingesetzt werden konnte.

3.1 Technologien

3.1.1 Programmiersprache

Für die Entwicklung der Software wurde die Programmiersprache C# eingesetzt. Dies wurde aufgrund der Erfahrung entschieden, dass das .NET-Framework sehr pragmatische Ansätze vertritt. Diese beschleunigen die Entwicklung und ermöglichen das Nutzen von Systemfunktionen wie einer Mausemulation oder Webcam-Ansteuerung in einfacher Weise. In Java wäre das nicht so einfach möglich gewesen. C++ hätte die Einbindung von OpenCV zwar erleichtert, aber wesentlich mehr Zeit für das Erarbeiten einfachster Konzepte benötigt. Die getroffene Entscheidung für C# ist auch hinsichtlich der Portabilität des Codes interessant. Mit Mono steht auch für Linux eine Ausführungsplattform für die hier entwickelte Software zur Verfügung.

3.1.2 Framework

Beim Framework für die Bildverarbeitungsroutinen wurde AForge eingesetzt. AForge ist ein relativ neues Projekt, welches nativ in C# geschrieben und sehr gut dokumentiert ist. Wie OpenCV ist auch AForge OpenSource. Da die Schnelligkeit des Entwicklungsvorgangs im Vordergrund stand, wurde bewusst auf OpenCV verzichtet. Es wurde in Kauf genommen, dass die mit AForge zur Verfügung stehenden Algorithmen nicht gleich vielfältig und nicht gleich performant sind. Da lediglich ein Prototyp mit den oben beschriebenen Algorithmen umgesetzt wurde, scheint dies akzeptabel. Für ein Produkt würde wahrscheinlich doch OpenCV benutzt.

OpenCV OpenCV ist ein riesiges Produkt, das alle Möglichkeiten zur Bildanalyse bietet. Es gibt einen Wrapper für C#, jedoch hat sich dieser in Versuchen als nicht besonders einfach in der Benutzung herausgestellt. Die Dokumentation ist sehr spärlich gehalten. Nicht zu unterschätzen ist die grosse Community, die oft weiterhelfen kann.

Der Quellcode ist auf Leistung optimiert und aufgrund dessen wenig verständlich und das Anpassen an eigene Bedürfnisse schwierig. Ein weiterer Nachteil ist, dass man zur Nutzung das ganze Framework einbinden muss. Es ist ca. 600 MB gross. Da der Aufwand für die Einarbeitung zu gross erschien, wurde OpenCV nur kurz angetestet und nicht für den Prototyp verwendet.

AForge AForge funktioniert weniger mathematisch als OpenCV. Der pragmatische und einfache Ansatz steht hier vor der akademischen und schnellen Lösung. AForge bietet leicht benutzbare Filter und eine stabile Blob-Detektion. Blobs sind zusammenhängende Pixelhaufen. Die Blob-Detektion kann diese über verschiedenen Kriterien in einem Bild finden. Ein häufig benutztes Kriterium ist die Maximalhelligkeit.

Die Konfiguration der verwendeten Module ist einfach gehalten und intuitiv. Performance und Möglichkeiten sind begrenzt. Für einen Prototypen ist die Nutzung von AForge jedoch ideal, da es einfach zu benutzen und gut dokumentiert ist. Der Code enthält leider immer noch einige Bugs, die jedoch aufgrund der guten Verständlichkeit selber behoben werden können.

3.2 Architektur

3.2.1 Domain

Es wurde darauf geachtet, dass alle Komponenten der Software möglichst gut austauschbar und erweiterbar sind. So werden die zwei Interfaces *IPictureProvider* und *IVisualizer* benutzt um eine beliebige Schnittstelle von aussen (Input) und nach aussen (Output) zu implementieren. Ein *PictureProvider* ist ein beliebiges Device, welches Bilder liefert. Dies kann zum Beispiel eine Videokamera oder ein Dateisystemordner sein. Ein *Visualizer* stellt Bilder dar. Wo und wie diese Darstellung stattfindet ist für Parsing- und Kalibrierungsalgorithmen nicht von Belang - auch nicht wo die Bilder herkommen.

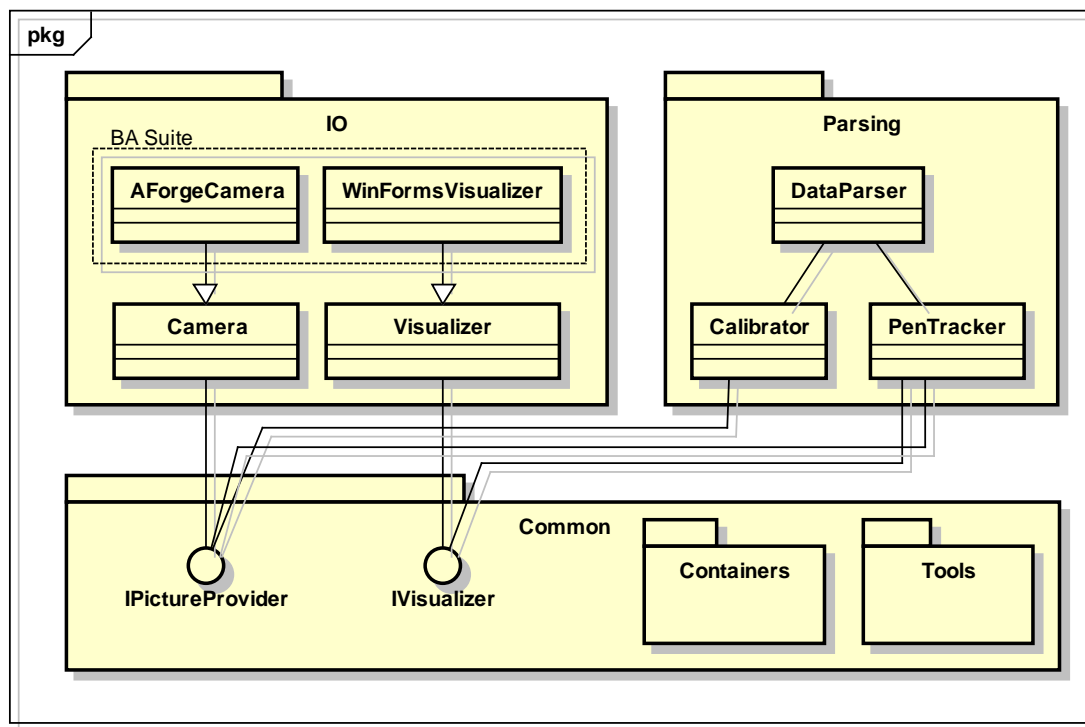


Abb. 27: Die Organisation der Klassen berücksichtigt modulare Erweiterungen.

3.2.2 Hauptsequenz

Der Ablauf des Programms orientiert sich stark am C# Eventmodell. Dabei benachrichtigen sich Komponenten gegenseitig über ihren aktuellen Status oder über Berechnungsergebnisse. In dieser Software werden Schnittstellen zwischen Input, Berechnung und Output immer über Events abgebildet. Dies hat den Vorteil, dass das GoF-Pattern Information Expert gut eingehalten werden kann. In Abb. 28 wird die Kette der Informationsflüsse und Hierarchien als Sequenzdiagramm dargestellt. Dabei entspricht die Applikation der Hauptapplikation, welche dem Benutzer ein GUI bereitstellt und Zugriff auf Kamera und Visualisierung hat. Der folgende Standardablauf wird zum besseren Verständnis der Funktionsweise der Software beschrieben:

1. Zuerst wählt der Benutzer im GUI die Kamera
 - a) Der Nutzer kriegt eine Auswahl aller gefundenen Kameras und entscheidet sich für eine.
 - b) Anhand der Auswahl wird die Kamera angelegt und gestartet.
2. Das Programm sucht einen passenden Visualisierer (WPF, WinForms, Dateisystemablage, etc.) und legt diesen an.
3. Ein DataParser wird angelegt.

4. Die Kalibrierung kann nun automatisch oder nach einer Interaktion des Benutzers gestartet werden.
5. Die Kalibrierung läuft durch und projiziert Messbilder über den Beamer an die Leinwand. Details dazu sind in Abschnitt 3.4 genauer erläutert.
6. Falls die Kalibrierung erfolgreich abgeschlossen wird, startet das Pen Tracking. Wenn ein Stift auf dem Inputbild gefunden wird, wird das an die Hauptapplikation per Event zurückgemeldet. Details über das Finden eines Stiftes finden sich unter Abschnitt 3.5.

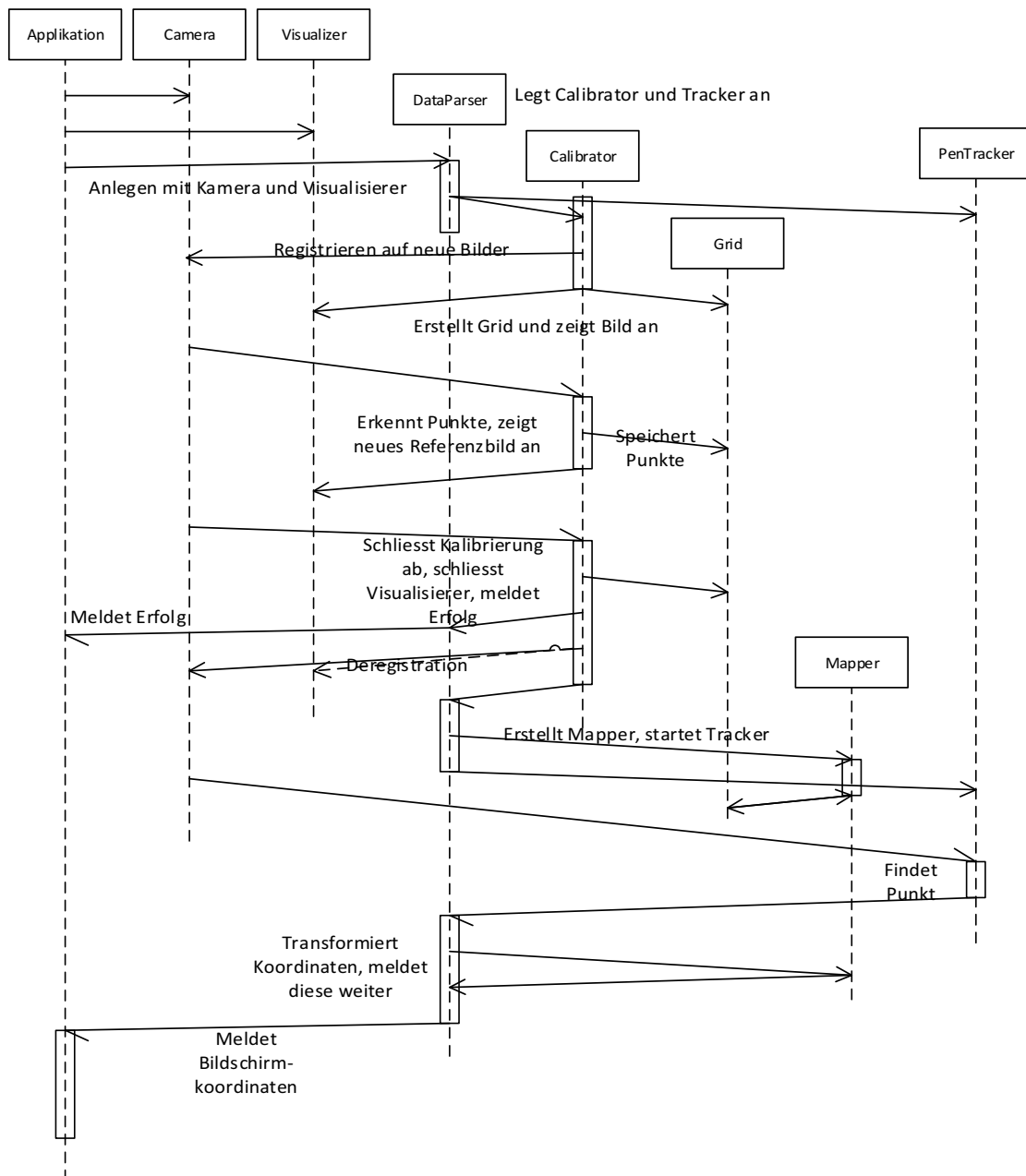


Abb. 28: Standardsequenz: Kalibrieren und Punkte auf Kamerabild finden

3.3 Visualisierung

Für die Kalibrierung ist es notwendig, dass Messmuster auf den Bildschirm gezeichnet und mit dem Beamer projiziert werden können. In dieser Software wird dies über einen sogenannten Visualisierer realisiert. Seine konkrete Implementation ist jeweils sehr stark

vom verwendeten Betriebssystem und dessen Anzeigekonzept abhängig. Es ist für die Portabilität sehr wichtig, dass die restliche Software unabhängig von der Anzeigemethode ist. Dies war zum Teil nicht ganz einfach umzusetzen. So ist z.B. nicht klar in welcher Umgebung die PenTracking-Library verwendet wird. Dies betrifft das Thread-Scheduling und die daraus entstehenden Zeitabstände für Kalibrierung und PenTracking. Mechanismen wie das Sicherstellen, dass Referenzbilder für die Kalibrierung auch wirklich von der Kamera abgefilmt wurden, sind in diesem Prototypen zurückgestellt worden und müssten noch beschrieben werden.

Ebenfalls für die Visualisierung wichtig ist, dass jeweils ein eigenes vom restlichen Programm unabhängiges GUI gestartet werden muss. Die Verwendung eines Visualisierers soll nicht an das Anzeigekonzept des Hauptprogramms gebunden sein. So soll es möglich sein, in einer WPF Applikation einen WinForms-Visualisierer einzusetzen und umgekehrt. Dies führt aber zu Problemen, wie dass jedes GUI in einem eigenen Thread mit ThreadSTA-Attribut laufen muss, oder dass der Garbage Collector den zusätzlichen GUI Thread abräumt, obwohl er in Zukunft noch vom Visualisierer verwendet werden müsste.

Im Folgenden wird konkret auf die Vor- und Nachteile der Verwendung von Windows Presentation Foundation (WPF) und WinForms eingegangen.

3.3.1 Umsetzungen

WPF Sämtliche Interaktionen mit dem GUI müssen über einen Dispatcher geschehen. Andernfalls wird sofort eine Exception geworfen. Dadurch ist es möglich, dass parallele Vorgänge im aufrufenden Code sequentiell abgearbeitet werden. Dies ist unerwünscht aber für GUI-Interaktionen nötig. Das Verhalten des WPF-Visualisierers ist abhängig vom Kontext, aus welchem er gestartet wurde. Hierbei wurden folgende drei Hauptfälle abgedeckt:

1. Wenn die Hauptapplikation von WPF gestartet wird, ist bereits ein WPF-Applikationsobjekt vorhanden. Gemäss Spezifikation darf in diesem Fall kein weiteres erstellt werden. Dafür haben alle Threads im Thread-Pool schon das STA-Attribut gesetzt.
2. Die Hauptapplikation wird von WinForms gestartet: Hierbei ist kein WPF-Applikationsobjekt vorhanden und muss zwingend erstellt werden, damit ein Fenster angezeigt werden kann. Ein Teil der Threads hat u. U. das STA-Attribut. Für Threads aus anderen Thread-Pools müssen die Interaktionen in einen Thread ausgelagert werden bei welchem dieses Attribut gesetzt ist.
3. Beim Starten aus einer Konsolenapplikation ist kein WPF-Applikationsobjekt vorhanden und muss erstellt werden. Da kein Thread das STA-Attribut gesetzt hat, müssen zwingend alle Interaktionen in einem neuen Thread abgearbeitet werden bei welchem dieses Attribut gesetzt ist. Dies ist nötig, da zur Laufzeit keine Thread-Attribute geändert werden können.

Da so bei gewissen Fällen viele zusätzliche Threads erstellt werden müssen ist Implementation eines WPF-Visualisierers mit einem gewissen Overhead verbunden, der aus der

reinen Applikationssicht nicht nötig wäre.

WinForms Mit dem *AllowUnsafeThreadExecution* Attribut kann die ganze Thread-Security abgeschaltet werden. Dafür müssen alle Grafikelemente vor dem Bearbeiten gelockt werden. Der aufrufende Code ist verantwortlich dafür, dass die Aufrufreihenfolge und der Ablauf stimmt. Durch Double-Buffering kann die Performance erhöht werden, wenn mehrere Objekte gezeichnet werden. Diese Art mit dem GUI zu interagieren widerspricht gängigen Architekturprinzipien und sollte nicht in produktivem Code verwendet werden.

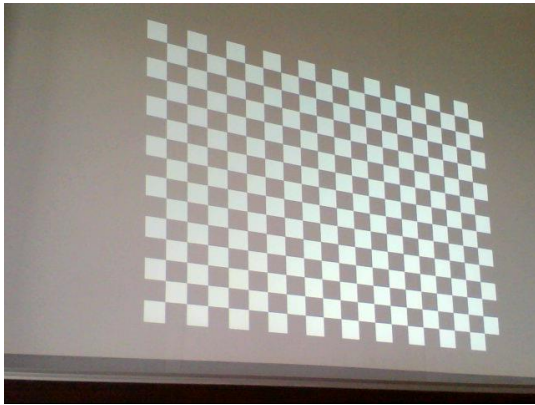
3.4 Kalibrierung

Für die Kalibrierung wird in dieser Umsetzung im Prototyp der Ansatz mit den individuellen Differenzbildern verwendet. Es werden damit nur die Ecken des Bildschirms detektiert. Diese reichen für den gewählten Interpolationsansatz.

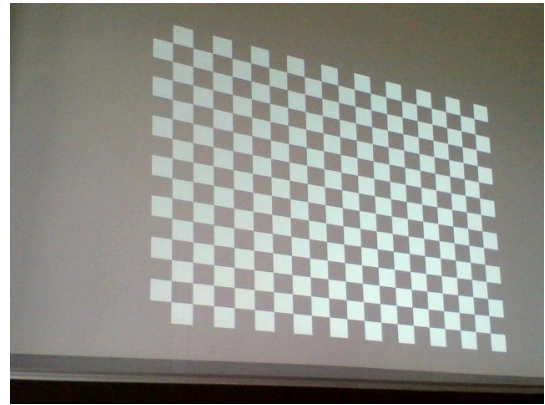
Ablauf

1. Im ersten Bild der Kamera wird die Auflösung erkannt. Damit lassen sich nun gewisse Komponenten initialisieren. Zudem wird ein Schachbrett mit $n \times m$ Quadraten projiziert. Das Verhältnis von n zu m sollte dem Bildschirmverhältnis entsprechen und die Quadrate sollten noch gut auf dem Kamerabild sichtbar sein (15-30 Pixel Seitenlänge). Diese Werte können problemlos statisch hinterlegt werden.
2. Das Schachbrett wird erneut projiziert, diesmal mit invertierten Farben. Auf dem Differenzbild sollte jetzt die Leinwand klar zu erkennen sein. Da an den Kanten der Quadrate dünne schwarze Pixelreihen¹⁴ auf dem Differenzbild resultieren, müssen diese z.B. mit einem Gaussfilter mit einer kleinen Kernel-Size eliminiert werden. Sonst kann es bei der Bloberkennung zu Fehlerkennungen kommen. Jetzt können die Ecken des Bildschirms sehr genau erkannt werden.
3. Durch die Bedingung, dass alle Bildschirmecken mindestens 5 Pixel Abstand zu jedem Kamerarand haben müssen, können zudem sehr viele Fehlerkennungen herausgefiltert werden.

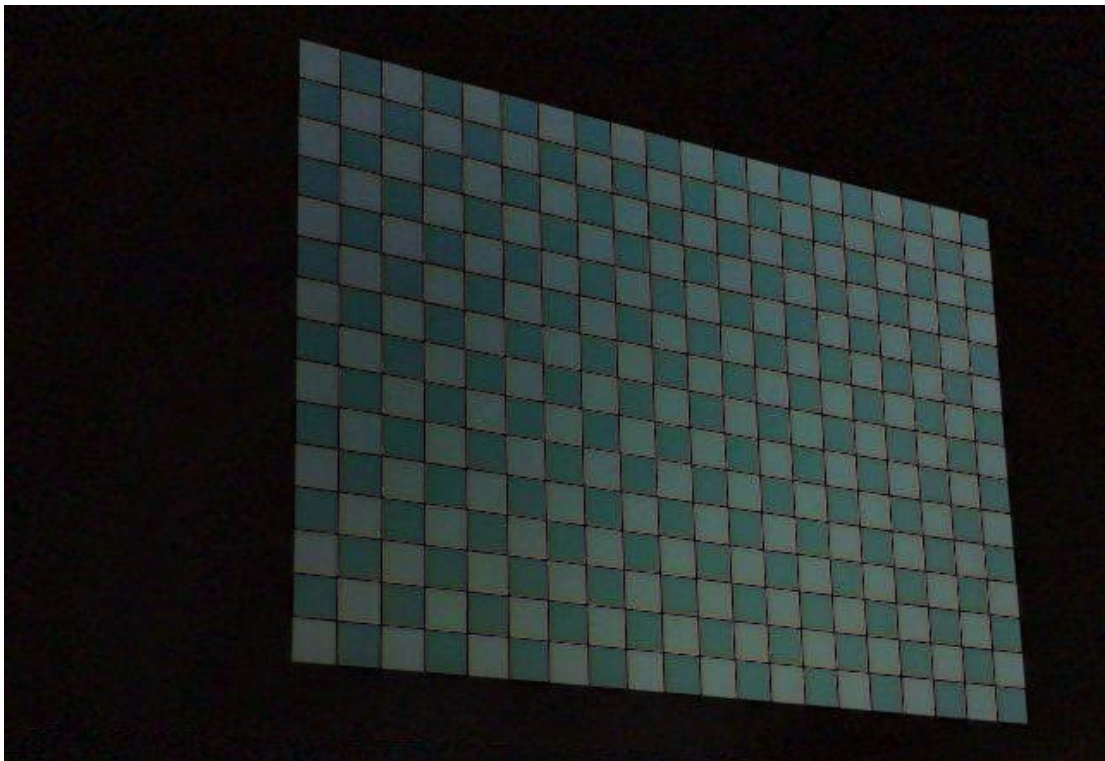
¹⁴Das sind diejenigen Pixelreihen, die von der Kamera auf beiden Bildern überschritten als beleuchtet erkannt wurden.



1) Frame x



2) Frame $x + t$



3) Differenzbild von 1) und 2)

Abb. 29: Erkennung des Bildschirms. Dass sich auch im Differenzbild noch ein Schachbrettmuster ergibt liegt an der automatischen Belichtungskorrektur der Kamera.

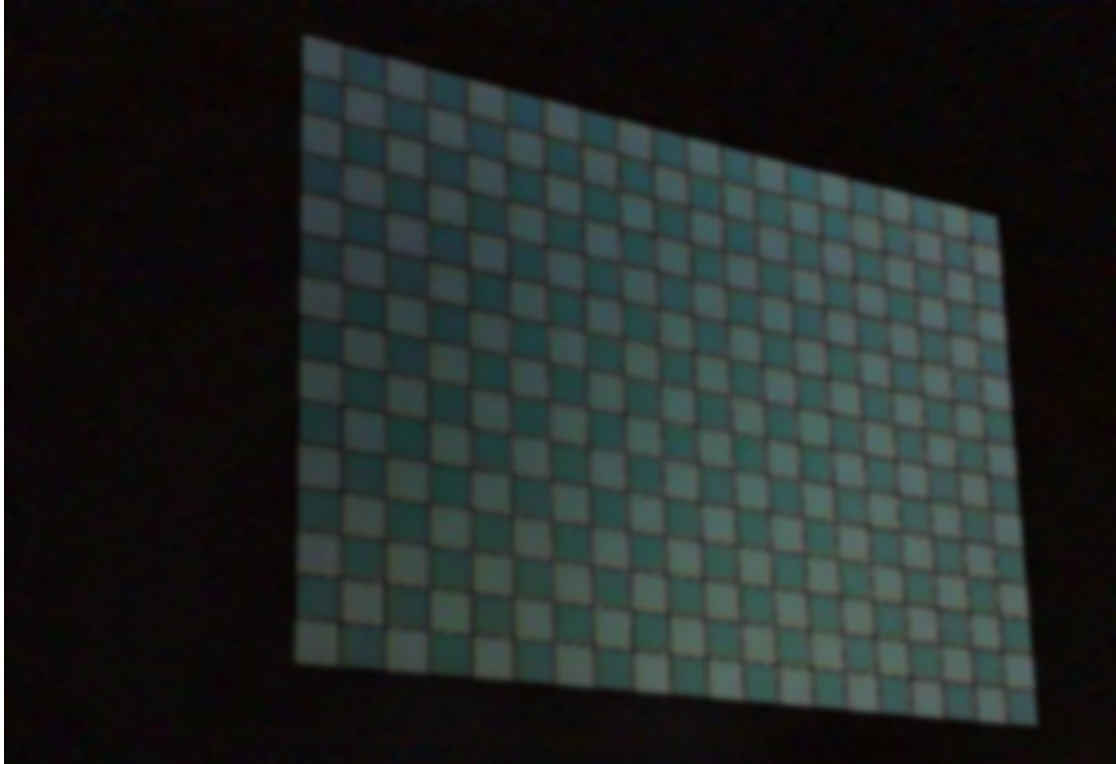


Abb. 30: Mit Gaussfilter geglättetes Differenzbild (Radius 3)

Einschränkungen Damit die Kalibrierung erfolgreich abgeschlossen werden kann, müssen ein paar technische Limitationen eingehalten werden. Die Belichtungskorrektur der Kamera reagiert erst etwa ein bis zwei Frames später. Die zu messende Referenzbilder müssen also länger angezeigt werden. In diesem Ansatz wird 500ms¹⁵ gewartet damit die Darstellung sicherlich gewährleistet ist.

¹⁵empirisch gewählter Wert - ist vom eingesetzten Visualisierer abhängig

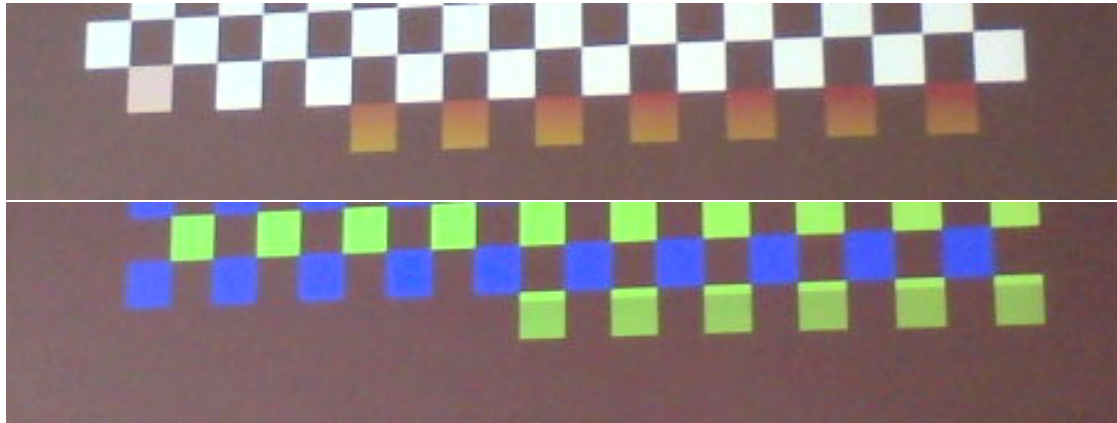


Abb. 31: Verzögerung der Darstellung

Bildverarbeitung Gewisse Webcams geben das Bild gespiegelt zurück. Deshalb muss dem User die Möglichkeit gegeben werden das Bild zu spiegeln. Weiter muss der Benutzer die Möglichkeit haben, die Kalibrierung nochmals zu starten, da keines der hier beschriebenen Verfahren erkennen kann, wenn ein Teil der Projektion verdeckt ist. Dafür ist es nötig, das Resultat der Kalibrierung visuell anzuzeigen. Dies geschieht durch einzeichnen der detektierten Eckpunkte auf dem Kamerabild.

Weiterhin muss beachtet werden, dass Bitmaps in C# sehr inperformant sind, daher müssen diese am Anfang mit Hilfe von Lock-Bits ausgelesen werden und in eine performantere Datenstruktur überführt werden, wie z.B. Bytearrays. Falls sie nochmals als Bitmap benötigt werden, kann man diese mit dem gleichen Verfahren wieder zurückschreiben.

Parallelität Die Kamera produziert regelmässig alle 40ms ein Bild. Die Bildverarbeitung für die Kalibrierung benötigt unterschiedlich lange Rechenzeit. Dies kann zu grossen Problemen führen:

- Eine sequentielle Bearbeitung der Daten führt dazu, dass mit veralteten (und somit falschen) Daten gearbeitet wird.
- Eine parallele Bearbeitung der jeweils eingehenden Bilder kann zu unkontrolliertem Spawnen von Tasks führen, die das System verlangsamen und schliesslich ein Ablaufen verunmöglichen. Falls das Auftritt, gibt es im besten Fall Probleme mit alten Daten, im Worst-Case reagiert das System nicht mehr.

Deshalb wurde entschieden, einen Supervisor einzusetzen, der die Arbeit überwacht. Dieser entscheidet bei jedem ankommenden Bild, ob ein Worker aus dem Pool frei ist. Da sich die Kalibrierung nicht parallelisieren lässt, ist in diesem Pool nur ein Worker vorhanden. Falls er keinen freien Worker findet, verwirft er das Bild. Somit ist sichergestellt dass alle Threads aktuelle Daten erhalten und das System nicht überlastet werden kann.

3.4.1 Schachbrett-Differenzbilder

Um Differenzbilder nutzen zu können, sich aber nicht durch die Belichtungskorrektur beeinflussen zu lassen, muss darauf geachtet werden, dass jedes projizierte Bild eine identische Helligkeitsverteilung zum Referenzbild besitzt. Das bedeutet, dass für jedes Bild, das analysiert werden soll, zwei Bilder projiziert werden müssen. Da diese Art der Erkennung viel genauer ist und nur ein Referenzbild benötigt wird, kann dieser Overhead in Kauf genommen werden.

3.5 Stifterkennung

3.5.1 Verfahren

Wie oben beschrieben gilt der Ansatz, dass der Stift das hellste sichtbare Licht auf dem Kamerabild darstellt. Der hellste Punkt sollte also die aktuelle Stiftposition sein. Es gibt zwei naheliegende Ansätze um diesen Punkt zu finden.

- Finde den hellsten Punkt auf dem Bild. Das hat Schwächen. Insbesondere wenn das Bild viel Rauschen beinhaltet.
- Finde den hellsten Punkt der sich bewegt.

In der Software „Presentation Writer“ wird für guten Input den Ansatz ohne Differenzbilder und für schlechten Input (wenig Licht, viel Rauschen) den stabileren Ansatz mit Differenzbildern verwendet.

Beispiel Im folgenden Beispiel wurde der automatische Weissabgleich der Kamera auf Rot konzentriert und die Überbeleuchtung drastisch reduziert. Resultierend sieht man die hellsten Punkte auf dem Beamerbild. Der alleinstehende Punkt im ersten Bild ist ein Laserpointer-Punkt. Im zweiten Bild befindet er sich im verrauschten Bereich. Den Punkt dort in einem einzelnen Bild per Software zu finden, kann Schwierigkeiten bereiten, da der Laserpunkt nicht der hellste Punkt sein muss.

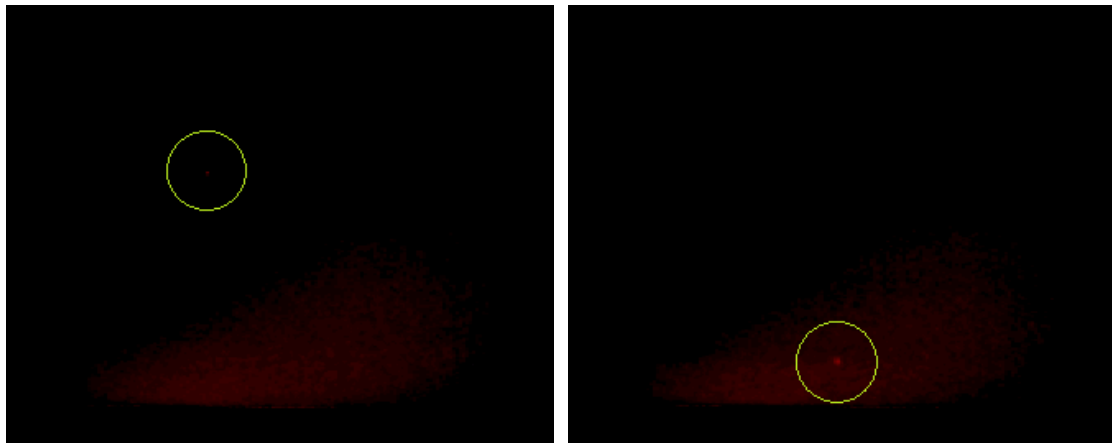


Abb. 32: Frame x : Der Laserpoint ist gut sichtbar (links), Frame $x + t$: Der Laserpointer ist kaum sichtbar (rechts)

Differenzbilder Ein stabilerer Ansatz ist, den Lichtpunkt über Differenzbilder zu finden. Das Differenzbild filtert helles Rauschen und stehende helle Lichtquellen zuverlässiger raus als eine einfache Hell/Dunkel-Schranke.

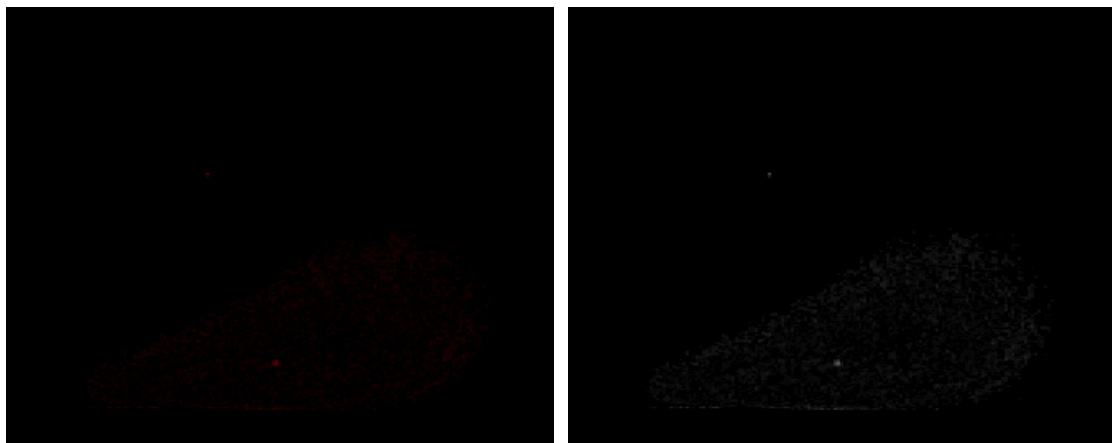


Abb. 33: Differenzbild zweier Frames mit Laserpunkten (links), Nachdem die Rotanteile zu Graustufen übersetzt wurden, sieht man die gesuchten Laserpunkte besser (rechts).

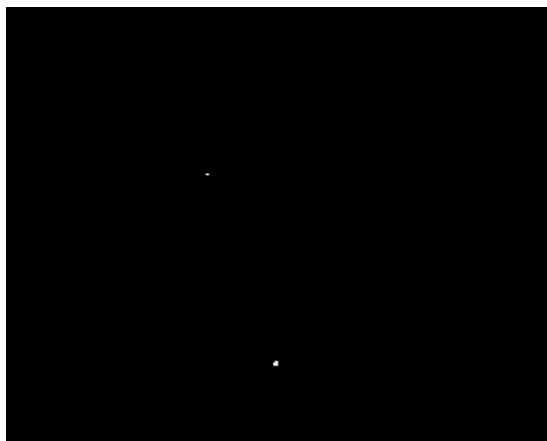


Abb. 34: Nun ergibt ein Hell/Dunkel-Filter (Threshold oder Binarize) gute Anhaltspunkte wo sich gesuchte Punkte befinden.

Kardinalität der gefundenen Punkte Alle zusammenhängenden hellen Bereiche werden Blobs genannt. Diese Blobs repräsentieren Positionen wo sich der Stift im aktuellen und im vorherigen Frame befunden hat. Es gibt verschiedene Fälle die nach dem Finden der Blobs unterschieden werden müssen.

- Es wurde kein Blob gefunden:
Für die Software ist kein Stift sichtbar. Es wird kein Punkt gefunden. Dies kann der Fall sein, wenn die präsentierende Person nicht schreibt, vor dem Stift steht und der Kamera die Sicht verdeckt oder wenn die Lichtverhältnisse nicht ideal sind und der Stift von der Kamera nicht hell genug wahrgenommen wird.
- Es wurde ein Blob gefunden:
Entweder hat sich der Stift nicht, oder zu wenig bewegt um zwei Blobs zu erzeugen oder es handelt sich um eine Fehlerkennung.

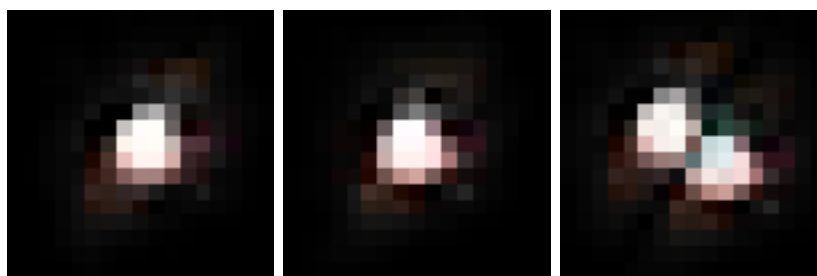


Abb. 35: Frame x, Frame x+1, Differenz

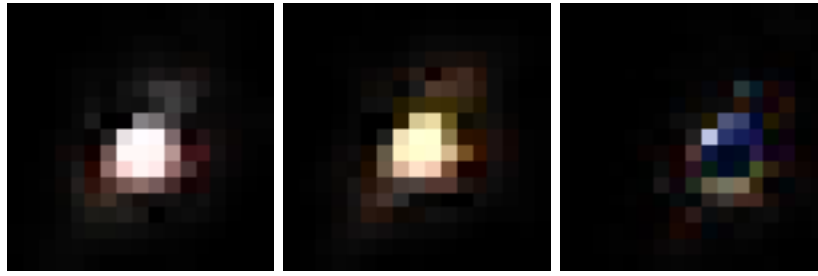


Abb. 36: Frame x , Frame $x+1$, Differenz

- Es wurden zwei Blobs gefunden:
Das ist der gewünschte Fall. Beide Punkte bilden gültige Positionen des letzten und des momentanen Frames.

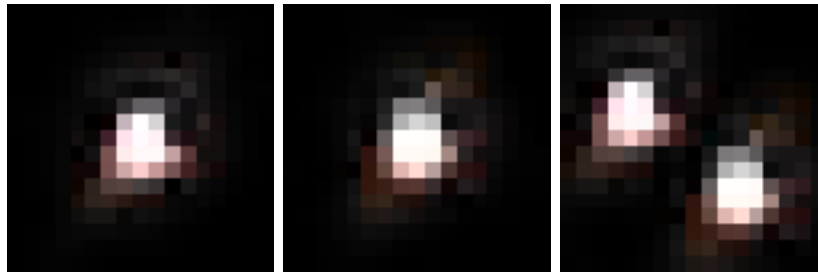


Abb. 37: Frame x , Frame $x+1$, Differenz - Die Punkte gehen

- Es wurden mehr als zwei Blobs gefunden:
Es ist hier keine sichere Erkennung möglich und alle Resultate werden verworfen. Als Optimierung in Betracht gezogen werden könnte, ob es möglich ist, die genauen Punkthelligkeit besser zu analysieren und den Punkt doch herauszufiltern. Dies könnte sich als schwierig erweisen und wurde deshalb vorerst zurückgestellt.

Generell wird angenommen, dass alle Blobs, falls deren Anzahl zwei oder kleiner ist, gültige Stiftpositionen bilden und dass Fehlerkennungen vorher gefiltert werden konnten. Gute Filter setzen die Eigenschaften eines gültigen Lichtpunktes gut um (s.u.). Ein weiterer Vorteil wenn man einen und zwei Punkte als Resultat zulässt ist die Stabilität der Software. Es kann einfach auf die Verwendung von Differenzbildern verzichtet werden, wenn die Kamera genug gute Bilder liefert.

Resultierende Punktposition Der Einfachheit halber wird nun das euklidische Zentrum der zwei gefundenen Blobs berechnet und als Resultatposition genommen. Falls nur ein Punkt gefunden wurde, wird dieser als Resultat gewählt. Dieses Verfahren beschränkt den Kontext und damit die Komplexität des Algorithmus auf das Vorhandensein von genau zwei Bildaufnahmen. Es werden weder Geschwindigkeit, noch Vorgängerpunkte

vorausgesetzt. Ein Nachteil, der dabei in Kauf genommen wird, ist, dass die gefundene Punktposition von korrekter Zeit und korrektem Ort abweichen kann. Diese Abweichung wird stärker, je schneller sich der Punkt auf der Leinwand bewegt. Mit Überlegungen aus dem WKS-Abtasttheorem wird aber klar, dass sich dieser Effekt nicht sehr stark in der mittleren Erkennungsqualität niederschlägt. Die Abtastrate vor und nach der Interpolation ist die gleiche. Die Abbildung der zwei gemessenen Punkte auf den Mittelwert resultiert in einer Position, die höchstwahrscheinlich gemessen worden wäre, wenn mit der doppelten Rate abgetastet worden wäre. Diese Behauptung gilt für unbeschleunigte gerade Bewegungen genau und für das Schreiben an eine Leinwand bei hoher Abtastrate näherungsweise.

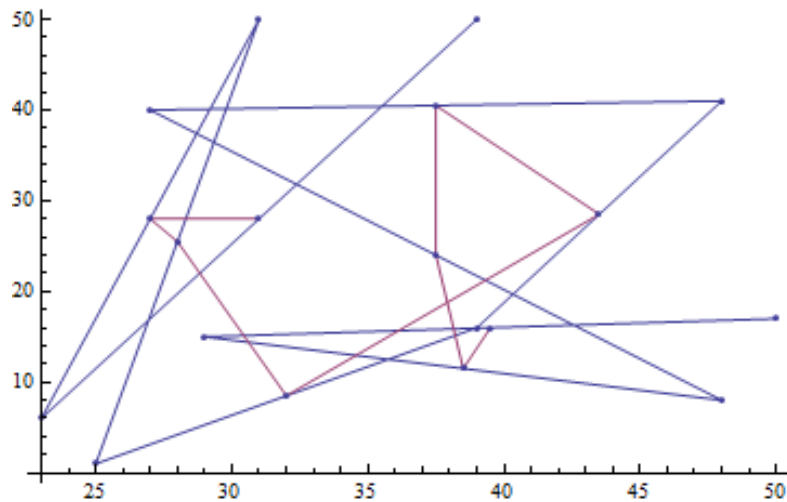


Abb. 38: Zufällig verteilte Punkte (blau) mit grossen Abständen. Die Interpolation (violett) taugt nichts.

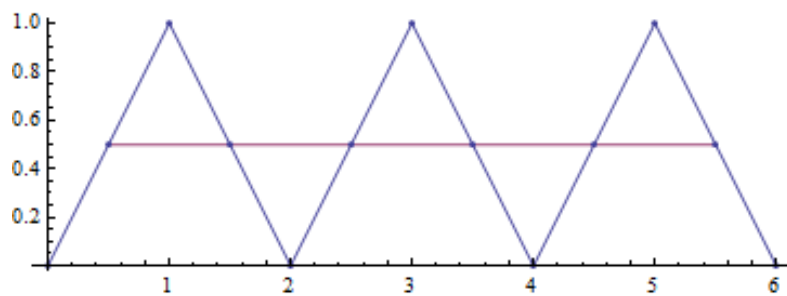


Abb. 39: Die Interpolation dient als Glättung

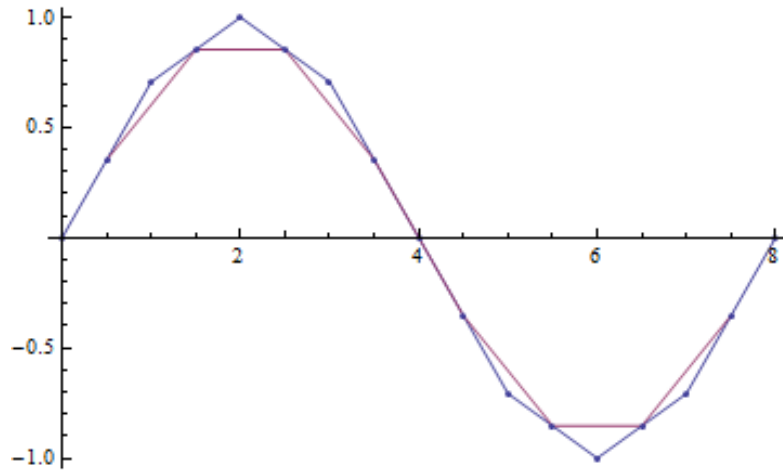


Abb. 40: Bei genug Punkten bleibt die Ursprungsfigur gut erhalten.

Zeitlich korrekte Punktposition Es wäre möglich, die genaue Punktposition zu finden, indem auf dem aktuellen Differenzbild der Vorgänger bestimmt wird. Im folgenden werden zwei Ansätze diskutiert, wie dies bewerkstelligt werden könnte. Abbildung 15 zeigt die Situation: Es wurden drei Aufnahmen gemacht, davon wiederum zwei Differenzbilder. Die zeitliche Abfolge der gefundenen Punkte ist mit den Zahlen 1 bis 3 gekennzeichnet.

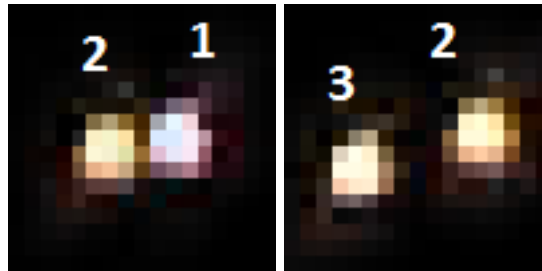


Abb. 41: Zeitliche Abfolge $Differenz(Bild_x, Bild_{x+1})$ und $Differenz(Bild_{x+1}, Bild_{x+2})$

Der erste Ansatz bezieht sich auf die im Differenzbild per Threshold¹⁶ gefundenen Blobs. Die gefundenen Positionen könnten mit dem im vorigen Differenzbild gefundenen Punkt verglichen werden. Der noch unbekannte Punkt ist der neue resultierende Punkt. Im Gegensatz zum oben genannten Interpolationsverfahren wird hier immer die aktuelle und richtige Position gefunden. Dieses Vorgehen erhöht jedoch die Komplexität des Algorithmus, da zusätzliche Fallunterscheidungen gemacht werden müssen. So muss separat unterschieden werden, was das Finden eines einzelnen Blobs nun bedeutet. Ansätze wären den Mittelpunkt oder Schwerpunkt zu berechnen oder genauere Analysen im Bild vorzunehmen. Man muss sich jedoch bewusst sein, dass Mischen von Interpolation mit Messung

¹⁶Helligkeits-Grenzwert

den Nachteil hat, dass Grössen wie Position und Geschwindigkeit verschwommen werden. Bei einem einzelnen grossen Blob wäre die Interpolation ungenau und korreliert nicht mit den Messinformationen, die aus vorherigen zwei Blobs gewonnen wurde.

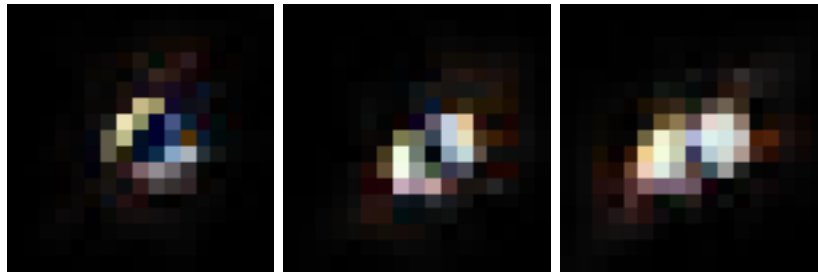


Abb. 42: Problematische Einzelfunde

Tests haben gezeigt, dass das aktuelle Interpolationsverfahren genügend genau ist. Für gute Kamerabilder (wenig Rauschen) wird der Differenzbildansatz jedoch nicht verwendet und direkt mit einem einzigen hellsten Punkt gearbeitet. Automatisch sind dann Ort und Zeit eines gefundenen Punktes besser.

3.5.2 Umsetzung

Die Umsetzung ist auf die Stabilität des Verfahrens mit Differenzbildern ausgerichtet. Sie funktioniert bei sehr schwachen Lichtpunkten wie Laserpointern und mit viel Rauschen. Nichtsdestotrotz wurde schlussendlich im Prototypen wieder auf die einfachere Version ohne Differenzbilder umgestellt. Bei guten Umgebungsbedingungen, wie guter Sichtbarkeit des Lichts und wenig Rauschen benötigt man die oben beschriebene Stabilität nicht. Zudem wird so Rechenzeit gespart.

Laufzeit Falls schon Punkte gefunden wurden, wird zuerst nur ein Bildausschnitt um den zuletzt gefundenen Punkt analysiert. Im Erfolgsfall spart das Rechenzeit. Falls man im Bildausschnitt kein Punkt findet, wird die Suche auf das ganze Bild ausgeweitet. Im Mittel wird so auf einem Intel Core Duo mit 2 Ghz ca. 4ms pro gefundenem Lichtpunkt benötigt. Gerade falls die Software auf einem schwächeren Rechner läuft, sollte sichergestellt werden, dass der Kamerainput nicht verrauscht ist. Dann kann die Punkterkennung ohne Differenzbilder ablaufen und so Rechenleistung sparen.

Als weitere Laufzeitoptimierung wird immer nur ein neuer Thread für den Bildverarbeitungsprozess gestartet, wenn kein anderer mehr läuft. Dies reduziert zwar die Erkennungsrate auf langsameren Systemen, hat aber den Vorteil, dass sich keine Frames aufstauen.

3.5.3 Einschränkungen

Die Lichtverhältnisse spielen eine wichtige Rolle. Bei jedem Setup müssen die Konfigurationsparameter wieder individuell angepasst werden - momentan manuell in einem

Treiberdialog von DirectShow. Zudem ergeben starke Reflexionen Artefakte im Input, die den Tracking-Algorithmus stören. Gerade in der Phase zwischen Kamerakalibrierung und Reduktion der automatischen Belichtungskorrektur gibt es momentan noch viele Fehlerkennungen. Dieses Problem wurde reduziert indem die Grösse des für einen Stift gültigen Blob eingeschränkt wurde. Die verbleibenden Fehlerkennungen könnten über Heuristiken eliminiert werden. So könnte gemessen werden, wie oft ein gefundener Stift in welcher Zeit im Bild umherspringt. Dafür existiert momentan noch keine Schranke.

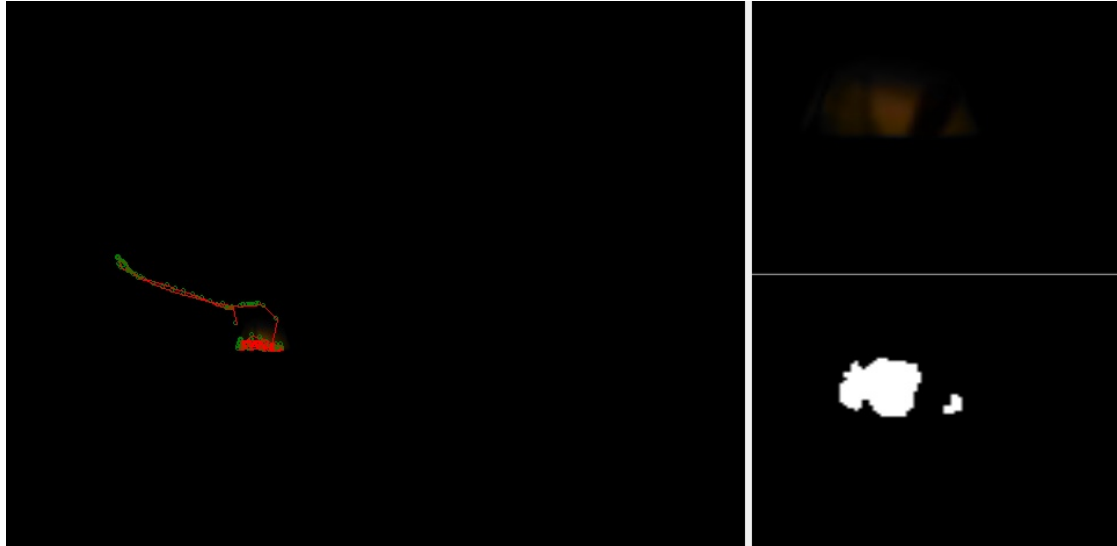


Abb. 43: Fehlerkennungen bei hellen Reflexionen nach dem Kalibrierungsvorgang

3.5.4 Wahl des Stiftes

Die Software „Presentation Writer“ ist imstande Laserpointer zu tracken. Da dies jedoch nur mit sehr genauer Anpassung der Konfiguration an die Helligkeitsbedingungen im Präsentationsraum geht, wurde schlussendlich eine LED verwendet. Es gibt noch diverse Alternativen, die stattdessen als Stift in Frage kämen. Die vielversprechendste ist eine Infrarotlampe. Sie könnte über einen $\frac{\lambda}{4}$ -Passfilter vor der normalen Kamera oder direkt über eine Infrarotkamera getrackt werden. Eine Option wäre auch ein Bandpassfilter für eine optische Laserfrequenz. Dann könnten herkömmliche Laserpointer (mit dem richtigen Lichtspektrum) für die Präsentation verwendet werden. Lediglich die Filterfolie müsste dann zur „Presentation Writer“ Software mitgeliefert werden.

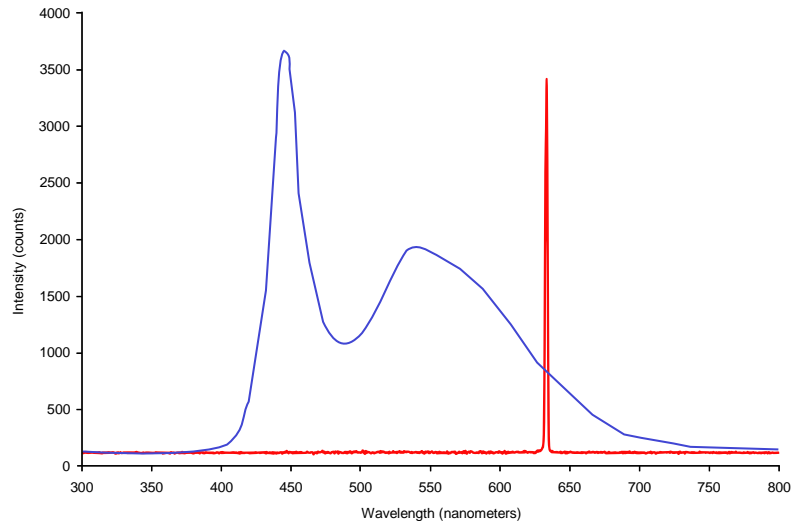


Abb. 44: Spektrum eines Helium Neon Lasers (rot) und einer weissen LED (blau).

3.6 Input Emulation

Gemäss Aufgabenstellung war lediglich eine PowerPoint-Integration gefordert. Eine komplette Mausemulation kann PowerPoint auch bedienen und bietet noch viel mehr Möglichkeiten. Zudem wurden schon Erfahrungen gesammelt, wie mit C# Mausbefehle auf Treiber-Level injiziert werden können. Diesen Weg benutzten ebenfalls Devices anderer Hersteller, wie z.B. der Logitech Presenter. Diese emulieren jedoch eine Tastatur, was für diese Arbeit nicht sinnvoll, aber optional auch möglich wäre. Unter Windows 8 ist es zudem möglich, Touch-Events zu generieren. Damit müsste keine Gestenanalyse mehr gemacht werden, da dies dann vom Betriebssystem abgenommen wird. Da jedoch auch ältere Systeme unterstützt werden sollen, wurde ein Teil der Windows 8 Touch-Funktionalität mit Mausbefehlen nachgebildet. Dazu gehört z.B. ein Rechtsklick durch Warten an einem Ort für eine gewisse Zeit.

Um den Mausinput zu glätten, wurde jeweils über die letzten sechs gefundenen Stiftpositionen gemittelt. So kann ein grosser Teil des niederfrequenten Rauschens (max. 4 Pixel um die durchschnittliche Bildschirmposition) beseitigt werden. Die resultierende Genauigkeit wirkt dadurch höher als die im Kamerabild gemessene. Eine Verzögerung durch die Mittelwertbildung wird für den User erst ab einer Mittlung über 15 Punkte bemerkbar.

3.7 Erkenntnisse

Kalibrierung

Die Kalibrierung geht ziemlich schnell vonstatten. Es wird ein Button gedrückt, dann werden zwei oder drei Bilder angezeigt und schliesslich kann das PenTracking und die Mausemulation aktiviert werden. Momentan läuft die Konfiguration der Kamera noch

über einen vordefinierten Treiber-Dialog ab. Dies ist störend und sollte bei einem Produkt irgendwie umgangen werden.

Die Kalibrierung funktioniert gut bei allen in einem Präsentationsraum üblichen Helligkeitswerten und sogar bei unüblichen Abfilmungswinkel (siehe Abb. 45).

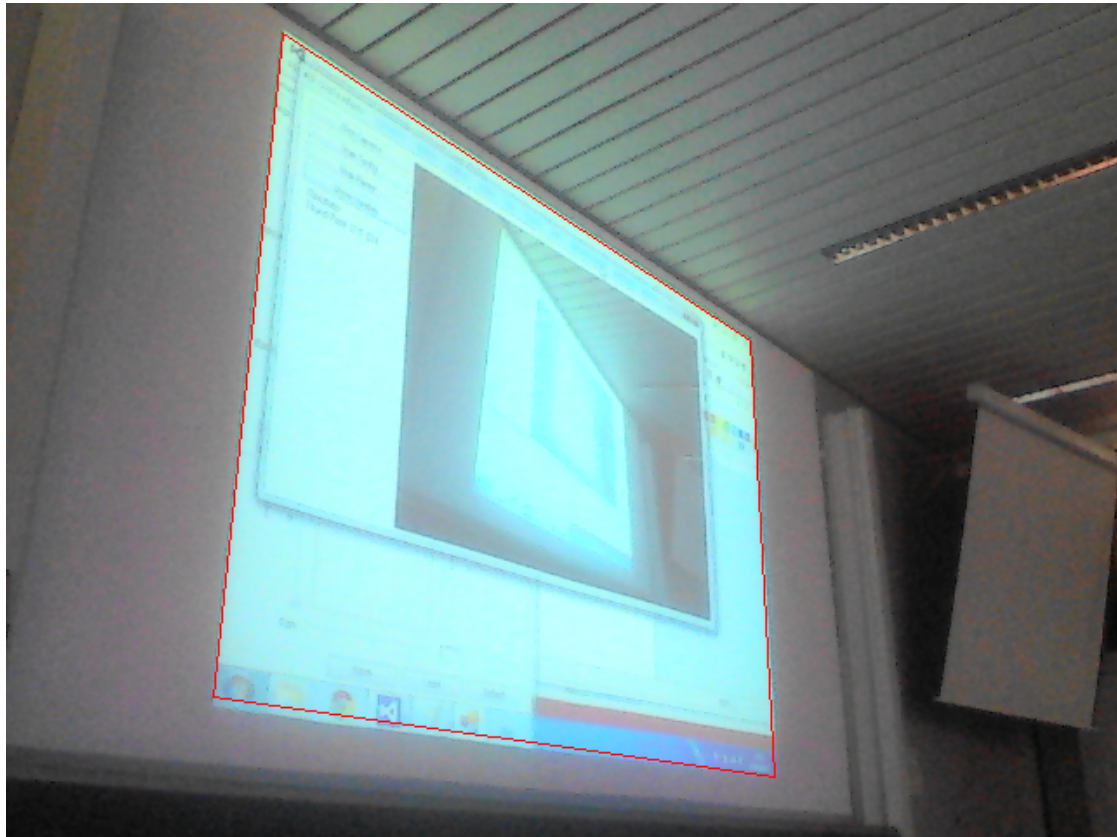


Abb. 45: Bei einem zu grossen Winkel (hier 55°) wird die Kalibrierung schwieriger

Mapping

Das Abbilden von Kamerapunkten auf Beamerpunkte geht mit dem Integralansatz sehr gut, wenn Kamera und Beamer gerade vor der Leinwand stehen. Mit zunehmender Schräge wird die Genauigkeit schlechter. Aus Abb. 46 ist ersichtlich, dass dieser Ansatz durch die pragmatische Korrekturfunktion mehr Abweichung erzeugt als nötig. Dies liegt in erster Linie daran, dass diese quadratische Ausgleichsfunktion in der Praxis die Länge der seitlichen Bildlinien a und b benutzt anstatt wie in der Theorie vorgesehen einen gemessenen Mittelpunkt des Beamerbildes ($x_b = 0.5$). Der Parameter h der Gleichung (9) wird im Programmcode durch $h = (\frac{a}{b} - 1) \frac{1}{10}$ umgesetzt. In Abb. 47 ist dargestellt, wie gut die Korrekturfunktion bei einem Winkel von 30° greift.

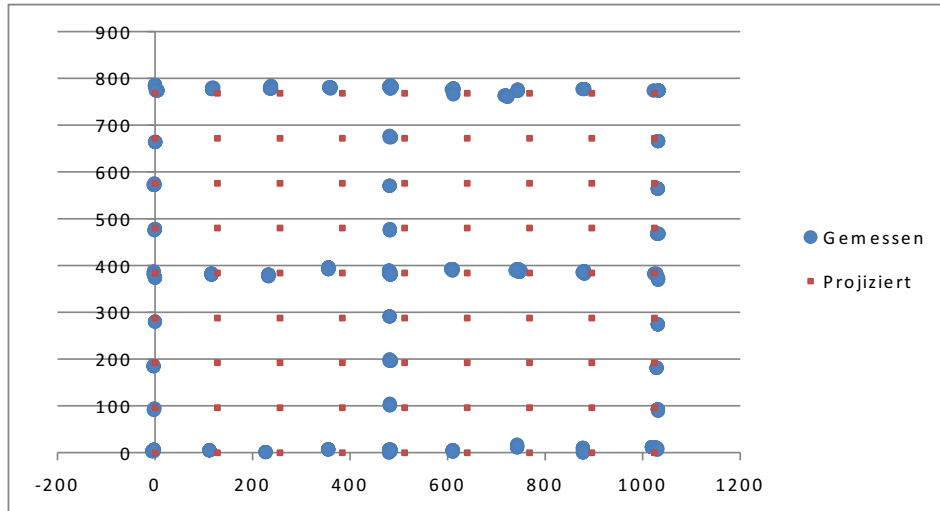


Abb. 46: Baryzentrisch geglätteter Integralansatz: Ausrichtung 0°

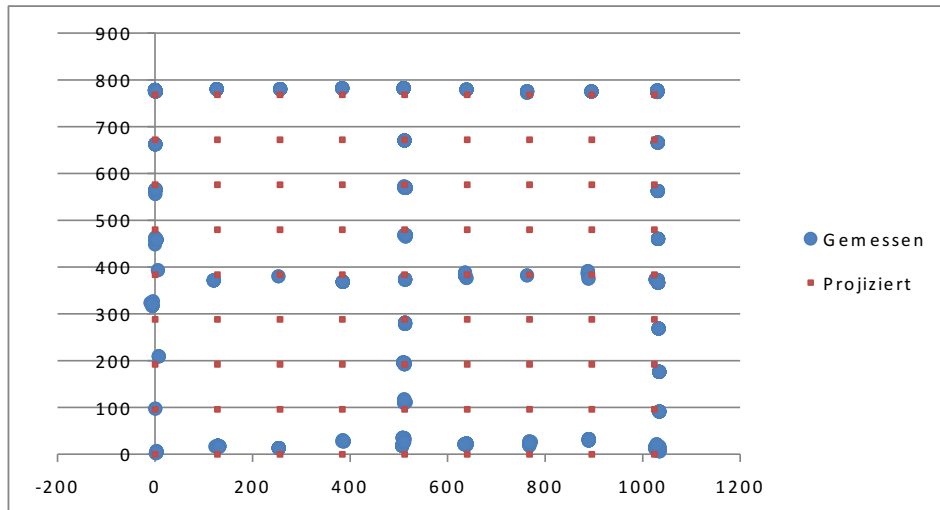


Abb. 47: Baryzentrisch geglätteter Integralansatz: Ausrichtung 30°

Generell nicht untersucht wurde, wie sehr sich der Benutzer bewusst ist, dass das Lämpchen zur Kamera zeigen muss. Die Schräge der Lampe muss der Schräge des Laptops angepasst werden. Nur so zeigt die Verlängerung der Lampe in Richtung Leinwand auch auf den Punkt, der getroffen werden soll. Ab einem Winkel von 45° wird dieser Effekt so stark, dass kaum mehr an die gewünschte Position gezeichnet werden kann.

Mausemulation

Der in dieser Arbeit entwickelte Prototyp emuliert eine Maus vollständig. Beim Klicken wurde auf eine Interpolation verzichtet. Falls geklickt und gezogen wird, greift eine Interpolation mit einer Mittelwertbildung über die letzten 6 Werte. Bei Klicken und Halten wird ein Rechtsklick ausgeführt. So kann das ganze Betriebssystem bedient und pixelgenau gezeichnet werden. In Abb. 48 wird Microsoft PowerPoint 2010 mit einer LED-Lampe gesteuert.

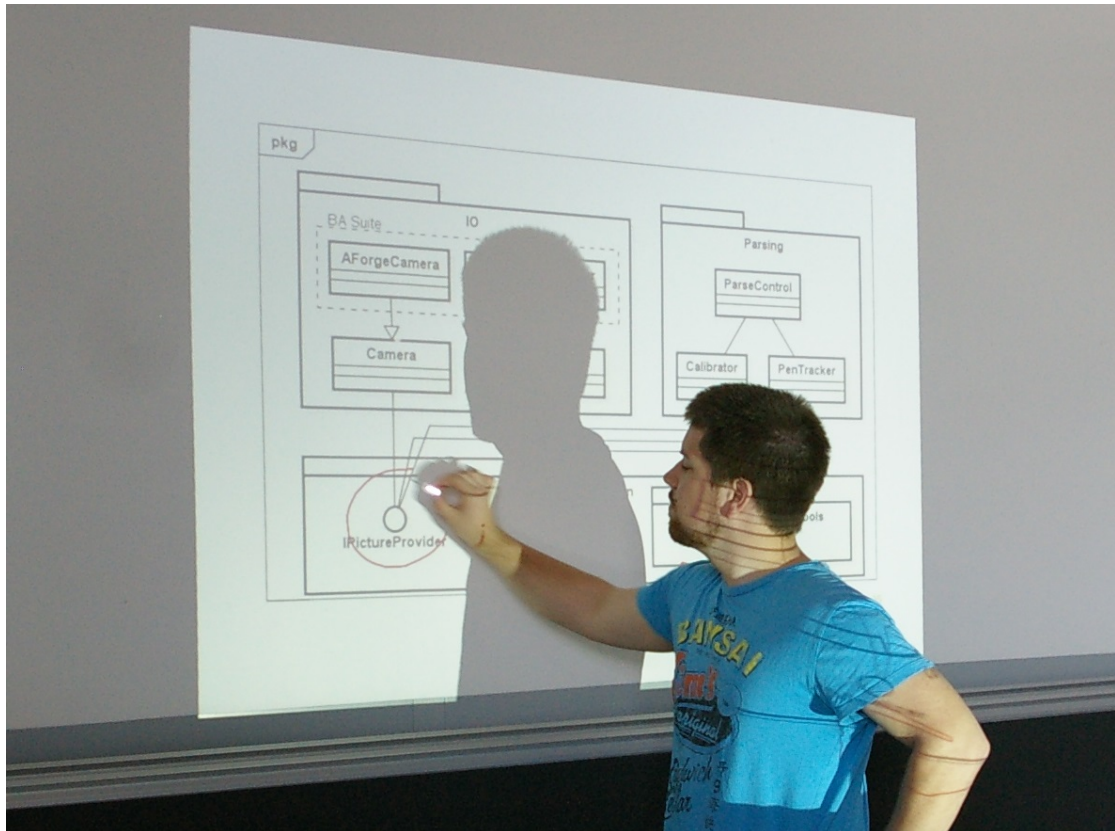


Abb. 48: Das Whiteboard in Aktion

4 Ausblick

4.1 Kamera & Stift

Die in dieser Arbeit verwendete Kamera hatte eine kleine Auflösung und eine störende automatische Belichtungskorrektur. Durch Nutzung einer besseren Kamera könnte sowohl die Genauigkeit des Resultates als auch die Stabilität des Inputs verbessert werden. Es wäre damit möglich, die schon etwas ältere Wacom-Technologie¹⁷ sowohl in Genauigkeit als auch in Reaktionszeit zu übertreffen.

Die Diversität der in handelsüblich verfügbaren Laptops integrierten Kameras könnte umgangen werden, wenn eine eigene Kamera mitgeliefert würde. Eine Infrarotkamera (mit Infrarot-Stift) wäre dafür von Vorteil. Die vorliegenden Algorithmen funktionieren ohne Weiteres auch auf einem Infrarotbild - wahrscheinlich zuverlässiger. Zudem ist eine Infrarotkamera weniger anfällig auf Störeinflüsse. Zusätzlich wäre nützlich, wenn die Aufnahmeeigenschaften einer eigenen Kamera per Software automatisch gesteuert werden könnten. Dann könnte sich das System bei Bedarf selbst neu kalibrieren.

Weiter optimieren liesse sich auch die Lichtverteilung des Stiftes. Sie ist momentan ziemlich ungleichmässig. Für bessere Resultate würde ein Stift entwickelt, der in alle Richtungen die identische Lichtmenge abgibt. So ist es nicht mehr von Belang, wie der Benutzer den Stift hält und neuen Benutzern wird die Bedienung erleichtert.

4.2 Kalibrierung

Für die Kamerakalibrierung gibt es zwei Ansätze, die weiterverfolgt werden können um mehr Referenzpunkte zu erhalten. Beide können benutzt werden um allfällige Korrekturfaktoren einzuführen und die sehr gute Genauigkeit noch weiter zu verbessern. Mit einer Kamera, welche automatisch kalibriert werden kann oder einer Infrarotkamera mit Farbkanälen für sichtbares Licht könnte die Kalibrierung während dem Betrieb laufend überwacht und so ein verrücken der Kamera detektiert werden. Eine komplette oder teilweise¹⁸ Nachkalibrierung könnte dann in-time gestartet werden. Zur Optimierung des Kalibrierungsvorganges gehört auch der Umgang mit Fehlerkennungen des Pen Tracking. Momentan gibt es viele falsch gefundene Stiftfunde direkt nach der Kalibrierung. Wenige Anpassungen im Code würden so die Nutzerfreundlichkeit verbessern und die Kalibrierungszeit verkürzen.

Ein Ansatz, der nicht weiter untersucht wurde, ist, dass der Benutzer die Kalibrierung komplett selbst vornimmt. Dies ist am Vorbild von herkömmlichen Touchpads orientiert. Der Benutzer tippt vorgegebene Positionen am Bildschirm (hier Leinwand) an und gibt so dem Kalibrierungssystem den benötigten Referenzinput. Der Hauptvorteil wäre die dadurch entfallende Kamerakalibrierung die in dieser Arbeit ziemlich aufwändig gestaltet ist.

¹⁷<http://www.wacom.com/en>

¹⁸Lediglich Anpassung von betroffenen Korrekturfaktoren

4.3 Input Emulation

Wenn weiterhin ein Betriebssystem mit dem hier entwickelten Präsentationssystem bedient werden soll, ist die Injektion von Punktpositionen in den Maustreiber die einzige gangbare Möglichkeit. Dabei sollte aber darauf geachtet werden, dass relative Mausbewegungen gesendet werden können, da es sonst mit gewissen Programmen, die Mausinputs auf einem sehr tiefen Level verarbeiten, zu Komplikationen kommen kann. Dies erfordert jedoch die Möglichkeit, die am PC eingestellte Mausbeschleunigung auslesen und Positionen verifizieren zu können, was nur mit grösserem Aufwand erreicht werden kann.

Touch Mit der Touch-Emulation unter Windows 8 unterstützt die in dieser Arbeit entwickelte Software alle von Microsoft implementierten Touch-Gesten. In Kombination mit einem Pen Tracker, der mehrere Pens detektieren kann, wären auch Multitouch-Gesten möglich. Somit könnten theoretisch mehrere Personen gleichzeitig einen PC bedienen. Das Nachbilden dieser Funktionalität unter anderen Betriebssystemen wäre zwar nicht trivial, aber aus Kompatibilitätsgründen wünschenswert. Was ebenfalls in Richtung Touch noch interessant zu untersuchen wäre, ist die Unterstützung für Multitouch. Es wäre vermutlich mit vertretbarem Aufwand zu realisieren, dass mehrere Stifte auf dem Kamerabild simultan erkannt werden können. Die native Unterstützung für Multitouch fehlt jedoch den meisten Betriebssystemen noch. Dies würde die Umsetzung erschweren.

Handdetektion In Richtung Touch geht auch die Vision, dass heutzutage moderne Systeme per Handgesten zu bedienen seien. Im Anhang in Kapitel 5.4 wird diskutiert, wie dies angegangen werden könnte. Dabei wird vor allem auf die noch offenen Probleme eingegangen.

5 Anhang

5.1 Kalibrierung: Bilderkennung

In diesem Kapitel werden die Ansätze zur Bilderkennung, die im Kapitel 2.1 evaluiert wurden, genauer beschrieben. Die Beschreibungen entsprechen jeweils den Testimplementierungen davon. Diese sind im Sandbox-Projekt zu finden. Es sind alle Ansätze, die im finalen Prototypen nicht mehr vorhanden und deshalb nicht fertig ausgearbeitet und nicht gut dokumentiert sind. Im folgenden Text wird davon ausgegangen, dass es eine Möglichkeit gibt, Rechtecke pixelgenau auf den Bildschirm zu zeichnen. Dies wurde genauer im Kapitel 3.3.1 behandelt.

Allgemeiner Ablauf Die Grundüberlegung jedes Ansatzes der Kalibrierung ist derselbe. Im ersten Schritt werden die Ecken des Bildschirms gefunden. Dann werden in weiteren Schritten zusätzliche Daten gesammelt damit man eine Grundlage für eine immer bessere Interpolation hat. Dazu können die im Folgenden beschriebenen Ansätze beinahe beliebig kombiniert werden.

5.1.1 Eckdetektion mit Differenzbildern und zufällig verteilten Rechtecken.

In diesem Unterkapitel wird der Ansatz beschrieben, der im Kapitel 2.1.1 evaluiert wurde. Deshalb wird hier nicht mehr auf die Analyse eingegangen. Bei diesem Ansatz werden keine zusätzlichen Libraries verwendet. Da die zu verwendenden Bitmapoperationen nativ (C#) sehr inperformant sind wurde eine eigene Bitmapklasse verwendet, welche die Bitmaps zu Bytearrays umwandelt. Damit können nur Matrizenrechnung und -analysen durchgeführt werden.

Ablauf Der genaue Ablauf der Testimplementation wird im Folgenden Schritt für Schritt beschrieben.

1. Der Bildschirm wird Schwarz gemacht. Das entsprechende Bild wird gespeichert.
2. Der Bildschirm wird weiss gemacht und das Differenzbild¹⁹ gemacht. Mit 4 Scanlinien wird von jeder Ecke ausgehend unter 45° ein Punkt gesucht, dessen Nachbarn in Richtung der gegenüberliegenden Ecke liegen und eine bestimmte Deckung besitzen. Es wird ein Quadrat mit Seitenlänge über 3-5 Pixel berücksichtigt.
3. Es werden 3 Rechtecke mit zufälligen Eckpunkten gezeichnet und die Eckpunkte gespeichert, je einer pro Farbkanal. Auf dem Differenzbild zum Ersten werden mit dem gleichen Verfahren zu jedem Farbkanal die Ecken gesucht.

¹⁹Beschreibung eines Differenzbildes: siehe Abschnitt 2.1.1

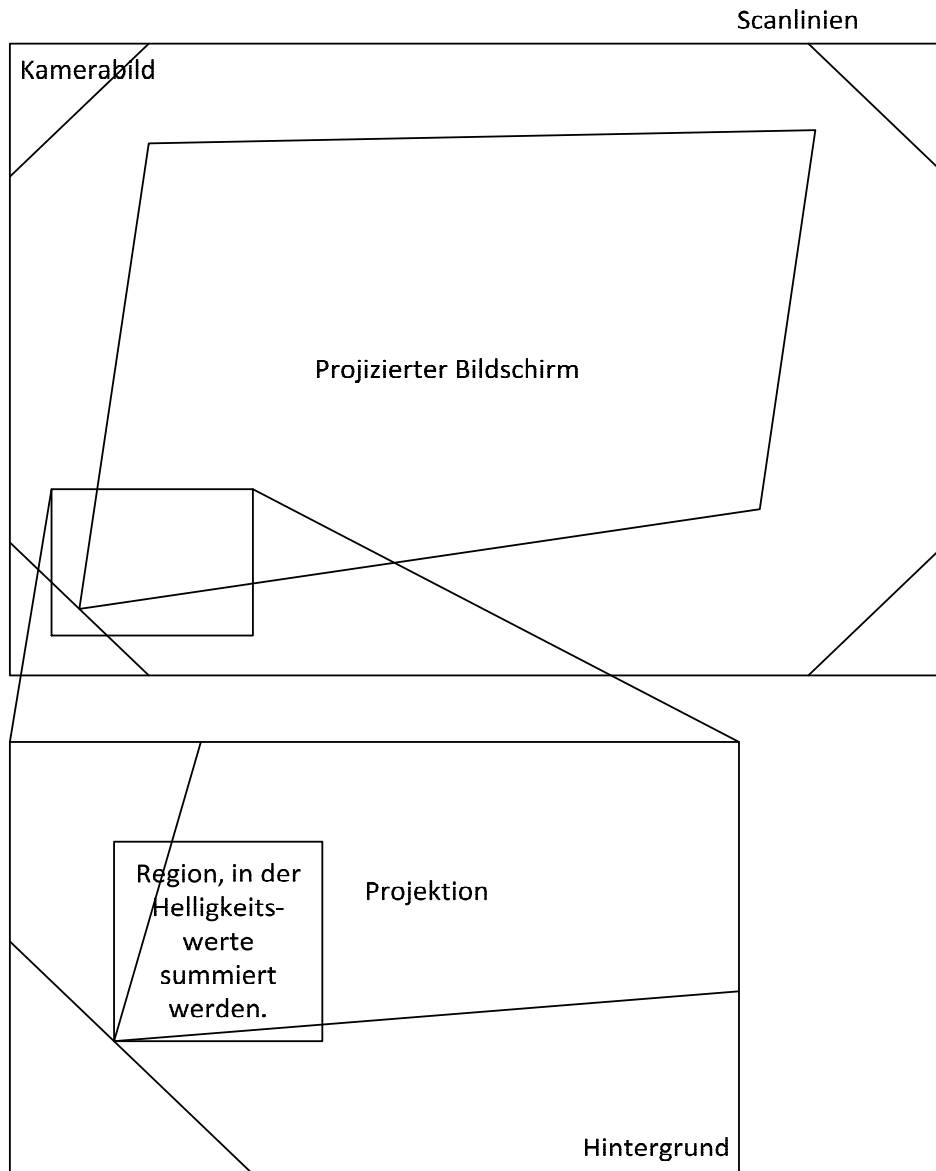


Abb. 49: Diagonale Scanlinien finden Eckpunkte.

5.1.2 Verwendung des AForge-Blob-Detectors

Beim verwendeten Framework handelt es sich um ein OpenSource Projekt, das nativ in C# geschrieben ist. Es bietet viele Möglichkeiten zur Bildebearbeitung. Eine genaue Analyse des Frameworks ist im Kapitel 3.1.2 beschrieben. Das nun beschriebene Verfahren wird in den folgend beschriebenen Lösungen immer eingesetzt. Dabei werden bei jeder Lösung nur geänderte und wichtige Aspekte erwähnt. Funktionierende Teile wurden jeweils in die Hauptlösung übernommen. Die verschiedenen Kalibrierungsmethoden

im Code sind keine 1:1 Umsetzungen der hier beschriebenen Ansätze, sondern Kombinationen davon. Der Code von verworfenen Umsetzungen wurde in anderen Umsetzungen teilweise übernommen und aktualisiert. Diese Aktualisierungen wurden nicht nachgepflegt aus dem Grund, dass diese Lösungen nicht als sinnvoll für ein Weiterverfolgen erachtet wurden.

Gesuchtes Muster Die folgenden Ansätze bilden ein mehrfarbiges Schachbrettmuster ab, das nach Farbkanälen aufgeschlüsselt werden kann und somit anhand der Ecken der Rechtecke viele Referenzpunkte liefert.

5.1.3 Einfache Differenzbild-Kalibrierung

Folgend wird der Ablauf des im Kapitel 2.1.1 analysierten Ansatzes erläutert. In der Implementation wird AForge verwendet, jedoch kann dieser Ansatz mit jedem Framework implementiert werden.

Ablauf

1. Als erstes Bild wird der Bildschirm schwarz gefärbt. Dieses Bild dient als Grundbild für alle folgenden Differenzbilder.
2. Der Bildschirm wird komplett weiss gefärbt. Auf dem Differenzbild²⁰ wird der grösste Blob gesucht. Dieser umfasst dann genau die Ecken des Bildschirms.
3. Es werden grüne und blaue Quadrate jeweils versetzt auf den Bildschirm projiziert, die bei jedem Frame leicht verschoben werden um mehr Referenzpunkte zu bekommen. Die Erkennung des roten Farbkanals ist am schlechtesten²¹, deshalb werden blau und grün verwendet. Zwei Farben sind ausreichend, wenn sie mit einem Feld Abstand projiziert werden. So gibt es keine Nachbarschaftskonflikte bei ihrer Erkennung.

²⁰Beschreibung eines Differenzbildes: siehe Abschnitt 2.1.1

²¹Wurde empirisch gemessen. Wissenschaftliche Messungen über das Erkennen von Farben mit Webcams fehlen noch.

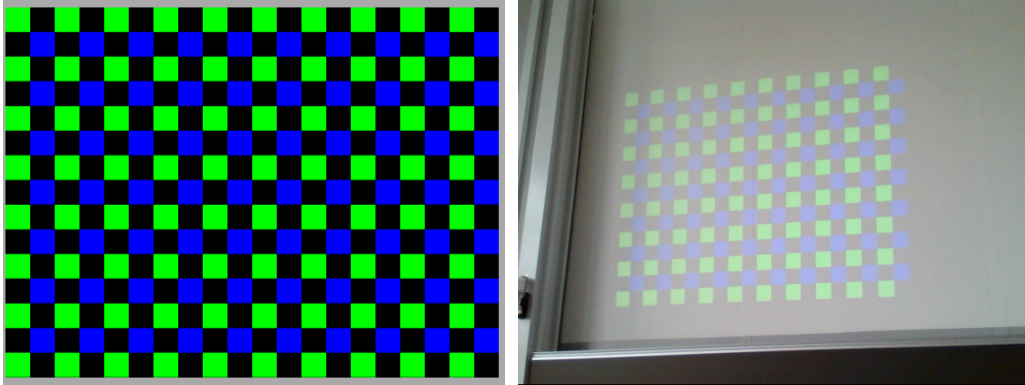


Abb. 50: Schachbrettmuster über zwei Farbkanäle

4. Diese Rechtecke werden anschliessend auf dem Kamerabild gesucht. Deren Zuordnung geschieht anschliessend mit einem in Kapitel 2.2 evaluierten Verfahren um die Position zu erhalten.

5.1.4 Histogramm-Analyse

Im Folgend beschriebenen Verfahren werden keine Differenzbilder generiert, sondern Helligkeitswerte des Bildes direkt analysiert. So können überdurchschnittlich helle Flächen (Projektionen) gefunden werden. Die restlichen Bildinformationen werden entfernt um eine Blob-Erkennung zu ermöglichen. Dieser Algorithmus enthält bereits die im Analyseteil erwähnte Optimierung 2.1.2, da er andernfalls ziemlich unbrauchbar ist.

Ablauf

1. Es wird eine weisse Fläche projiziert. Das Kamerabild wird in 2×2 gleich grosse Teile aufgeteilt. Auf jedem Teil werden alle Pixel, welche die Durchschnittshelligkeit unterschreiten auf Schwarz gesetzt. Danach werden die Teile wieder zusammengesetzt. Auf dem zusammengesetzten Bild wird nun der grösste Blob gesucht. Von diesem können nun die Eckpunkte bestimmt werden. Jene entsprechen den Bildschirmecken.
2. Es werden die grünen und blauen Quadrate aus dem vorherigen Verfahren projiziert. Diese Referenzbilder werden in 4×4 oder mehr Teile aufgeteilt und der blaue oder grüne Farbkanal separat analysiert. Dabei wird alles, was nicht ausserordentlich hell ist, entfernt. Danach werden die Bildteile wieder zusammengesetzt. In den zwei zusammengesetzten Bildern werden Blobs erkannt und analysiert.

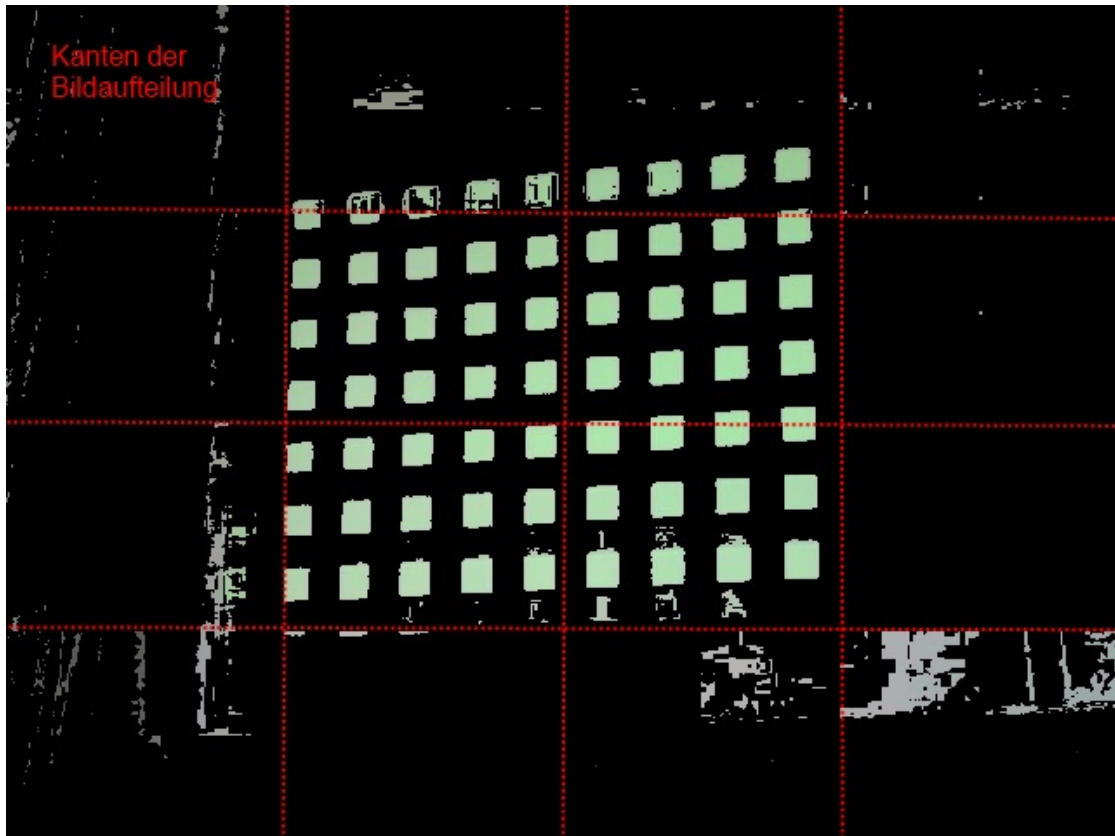
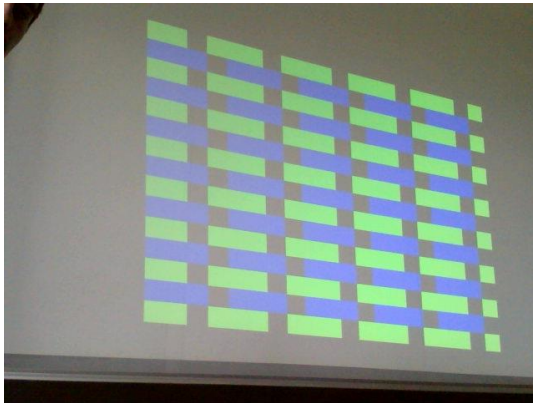


Abb. 51: Zweiter Schritt: Bei der Aufteilung 4×4 erkannte grüne Quadrate

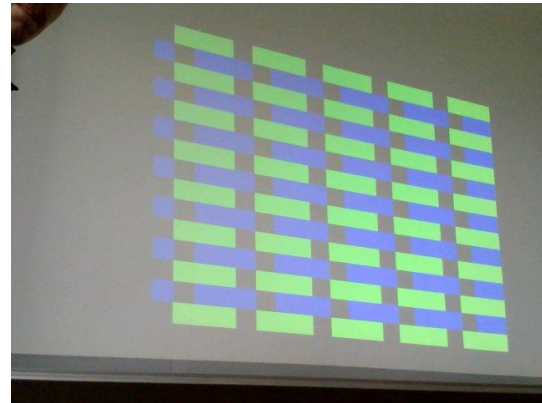
5.1.5 Erweitertes Differenzbild

Das nun beschriebene Verfahren setzt das gesuchte Referenzbild aus zwei projizierten Bildern mit ähnlicher Helligkeitsverteilung zusammen. Seine Analyse wurde im Kapitel 2.1.1 beschrieben. Der im Kapitel 3.4.1 beschriebene Ansatz ist ebenso eine Anwendung dieses Verfahrens.

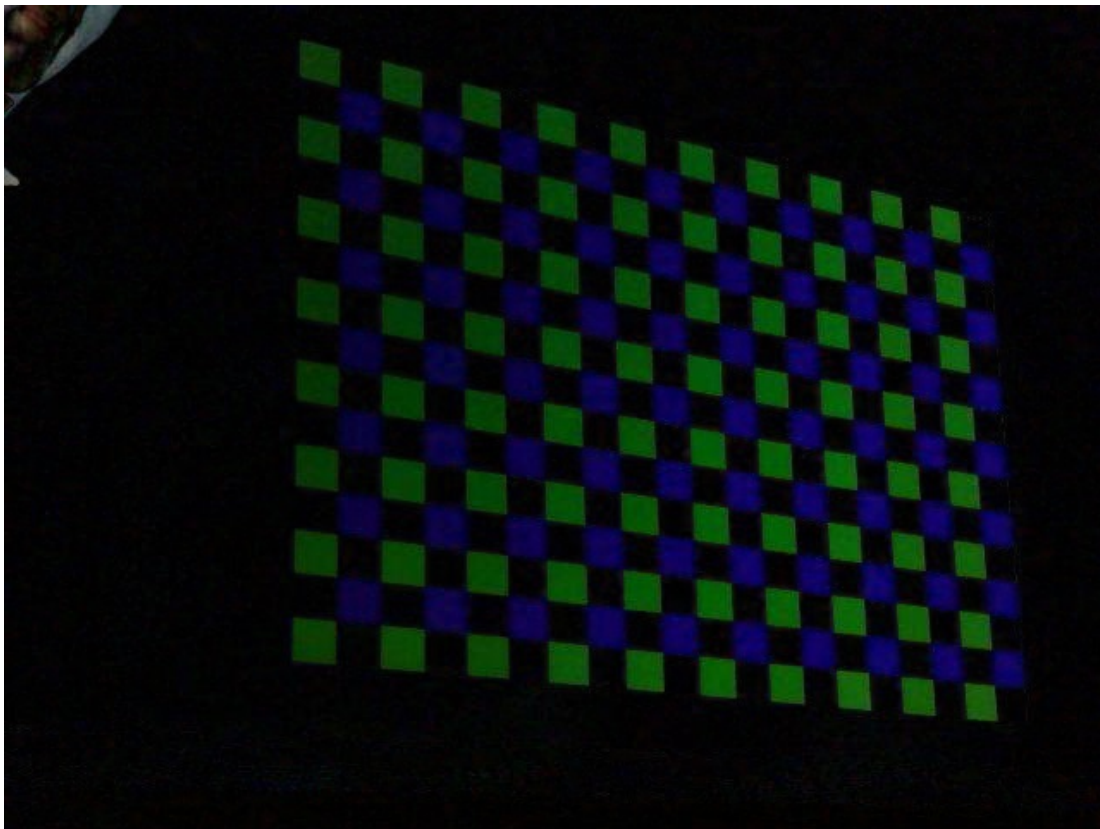
Ablauf Es werden jeweils zwei Bilder projiziert, deren Differenz genau das bekannte blau-grüne Muster ergibt. So können die Rechtecke auf dem Bild ganz scharf und klar erkannt und identifiziert werden. Durch das Bilden eines weiteren Differenzbildes mit den beiden erwünschten Farbkanälen können das meiste Bildrauschen und allfällige Unsicherheiten entfernt werden.



1) Frame x



2) Frame $x + t$



3) Differenzbild aus 1) und 2)

Abb. 52: Es wird gezielt ein Muster projiziert, welches im Differenzbild ein Schachbrett ergibt.

Störungen wie sie hier in der oberen linken Ecke vorhanden sind (Kopf einer Person) können dadurch entfernt werden, dass sie nicht innerhalb der erkannten Bildschirmecken liegen.

5.2 Kalibrierung: Zuordnung von Punkten

Der AForge BlobCounter gibt alle auf einem Kamerabild gefundenen Blobs zurück. Diese Daten beinhalten Schwerpunkt, Umfassungsrechteck, Fläche, Breite und Höhe. Diese Detektion könnte aber auch mit einem beliebigen anderen Framework durchgeführt werden. Anhand dieser gewonnen Positionsdaten wird die Zuordnung gemacht, welcher Blob zu welchem projizierten Quadrat gehört. Durch eine Filterung mit einer vordefinierten minimalen und maximalen Höhe/Breite können die Blobs gefiltert und Rauschen damit entfernt werden. Da die Ecken des Bildschirms bekannt sind, kann bestimmt werden, ob ein Blob im gültigen Bereich innerhalb liegt. Falls er das nicht tut, wird er entfernt. So können 99% der Fehlerkennungen eliminiert werden²². Die Schwerpunkte der Blobs sind mit Koordinaten des Kamerabildes beschrieben, was es schwierig macht, deren Position auf dem ursprünglichen Beamerbild zu bestimmen. Das liegt an der Verzerrung. So könnte z.B. die kleinste X-Koordinate zuunterst liegen und die grösste Y-Koordinate rechts oben. Je nach Verzerrung ist es so schon sehr schwierig die obere linke Ecke zu finden. Im Folgenden werden verschiedene Ansätze beschrieben, wie dies trotzdem möglich wäre.

5.2.1 Rekursiv

Beim rekursiven Ansatz wird angenommen, dass die Nachbarn eines Blobs eindeutig bestimmt werden können. Dies ist möglich anhand ihrer Distanzen, bereitet aber Probleme, wenn gewisse Blobs fehlen, da sie nicht erkannt wurden. Folglich wird dann ein anderer Blob als Nachbar detektiert, der eigentlich keiner ist.

Ablauf Nun wird der Grundalgorithmus beschrieben. Im Analyseteil wurde bereits auf Verbesserungen eingegangen, die auch weiter unten erwähnt sind.

1. Der Algorithmus bestimmt auf jedem Farbkanal denjenigen Blob, der am nächsten bei der oberen linken Ecke liegt. Durch die Positionsangabe (0,0) für grün oder (1,1) für blau als Startwert²³ ist bekannt, dass es sich um einen Eckpunkt handelt. Der Punkt wird markiert, dass er nicht nochmals bearbeitet werden muss.
2. Es werden also die zwei nächsten Nachbarn gesucht. Anhand der Differenz der Koordinaten kann bestimmt werden, ob der Punkt oberhalb (Y-Differenz ist grösser als die Differenz der X-Achse und negativ), rechts (X-Differenz ist grösser und positiv), unten oder links liegt.
3. Eine Funktion berechnet die Position, die der gefundene Blob haben muss (zwei in die entsprechende Achse verschoben). Diese Position kann anhand der Nachbarn bestimmt werden. Danach wird ein rekursiver Aufruf auf jedem unmarkierten Nachbar gestartet, nachdem der Ausgangspunkt als erledigt markiert wurde.

²²Empirischer Wert - Eine wissenschaftliche Auswertung liegt nicht vor.

²³Der grüne Farbkanal beginnt in der oberen linken Ecke, der blaue ist eine Rechteckgrösse versetzt dazu.

4. Der rekursive Aufruf prüft zudem, ob die übergeben Position plausibel erscheint (innerhalb der Projektion liegen kann) oder schon bestimmt wurde. Er bestimmt anhand dieser Position, wie viele Nachbarn zu erwarten sind.

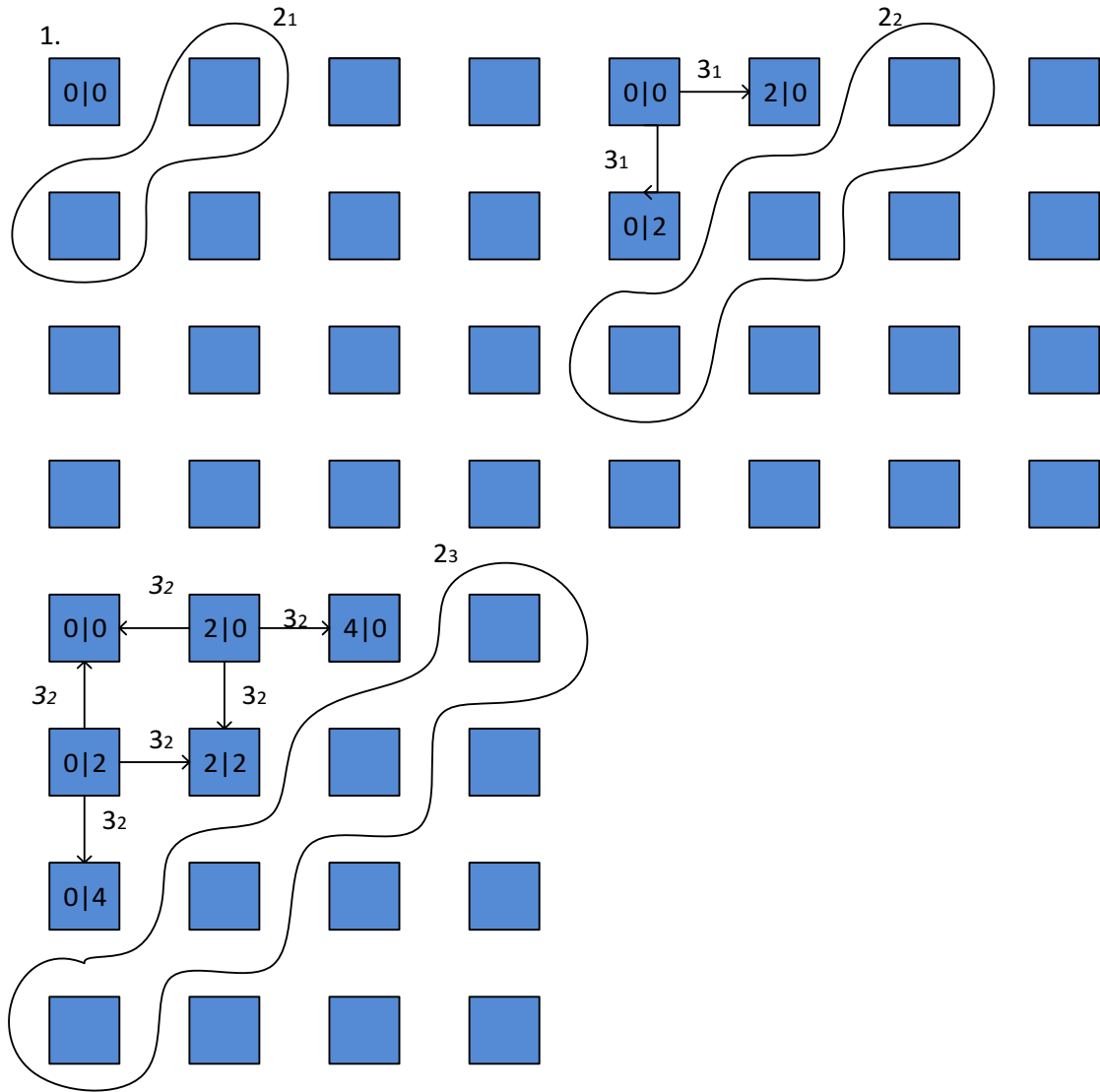


Abb. 53: Vorgehen beim rekursiven Ansatz

Verbesserungen Im Folgenden werden einige Verbesserungen aufgelistet, die unabhängig voneinander vorgenommen werden könnten.

- Es könnte nicht nur eine erwartete Blobposition gespeichert werden, sondern mehrere.

- Die Blobposition muss nicht bei der Iteration gespeichert werden, sondern jedes Mal, wenn der Blob als Nachbar gefunden wurde.
- Es könnte bei allen vier Ecken gestartet werden anstatt nur in der linken oberen Ecke.
- Die finale Blobposition wird festgelegt durch Auswahl der Position, die am häufigsten bestimmt wurde.
- Es gibt die Möglichkeit, die Position eines Quadrates anhand eines folgenden ähnlichen Bildes zu verifizieren. Dabei werden zur Verifikation ausserdem die Daten desjenigen Blobs verwendet, der den ähnlichsten Schwerpunkt hat und somit am ehesten demjenigen des vorherigen Bildes entspricht.
- Es werden nicht mehr die nächsten Nachbarn gesucht, sondern es wird analysiert, auf welcher Achse die Blobs liegen und dort gezielt der nächste Blob gesucht. So können diagonal liegende Punkte die Erkennung nicht mehr stören.

5.2.2 Nutzung der Interpolation

Mit einer Interpolation anhand der bereits gefundenen Eckpunkte des verzerrten Beamerbildes können die Positionen der Quadrate besser geschätzt werden. Es gibt dazu mehrere Möglichkeiten, die im Abschnitt 2.3 evaluiert wurden.

Ablauf

1. Man iteriert durch die Blobs und bestimmt zu jedem die Interpolation des Schwerpunkts.
2. Es wird überprüft, in welchem Quadrat der Schwerpunkt zu liegen kommt. Falls das Quadrat die gleiche Farbe hat, wird die Position als gültig angenommen.

5.2.3 K-Means Cluster-Zuordnung mit steigender Genauigkeit

Bei dem nun folgenden Verfahren wird kein mehrfarbiges Schachbrettmuster projiziert, sondern nur diejenigen Rechtecke, welche im aktuellen Schritt für die Analyse benötigt werden.

Ablauf Dieses Verfahren ist ein mehrstufiges Verfahren, das vom jeweils letzten Erkennungsschritt abhängt. Dies führt zu einer geringeren Fehlertoleranz.

1. Es werden die Eckpunkte des Bildschirms erkannt.
2. Alle Kanten werden in der Mitte geteilt und deren Mittelpunkte als Quadrat visualisiert.
3. Die erkannten Punkte werden mit K-Means den erwarteten Werten zugeordnet.

4. Aus dem Ergebnis resultieren weitere Rechtecke, auf denen das Verfahren wieder angewandt werden kann.

In der folgenden Abbildung sind die zwei ersten Schritte abgebildet, danach wird das Verfahren auf die erkannten Teilausschnitte wieder angewandt.

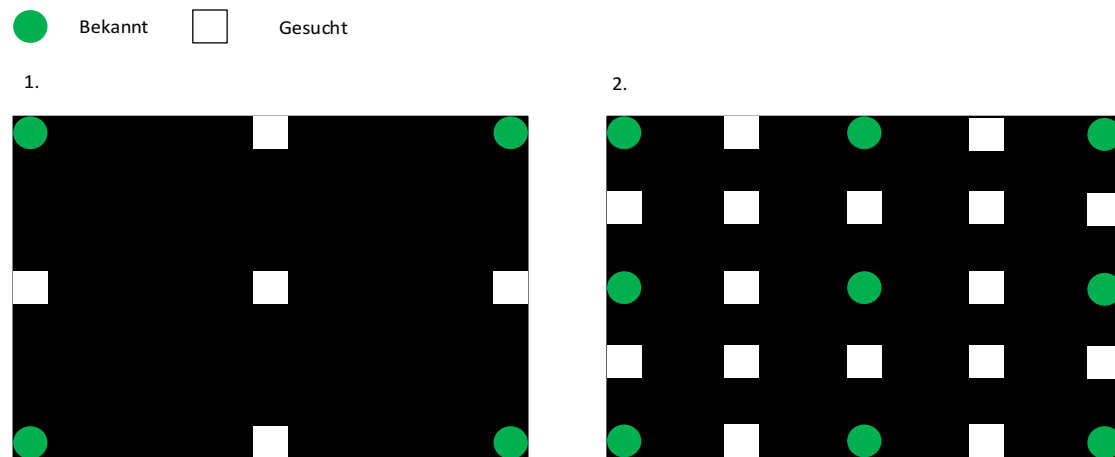


Abb. 54: Ablauf K-Means

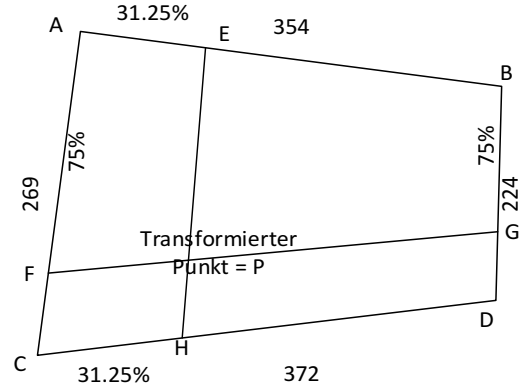
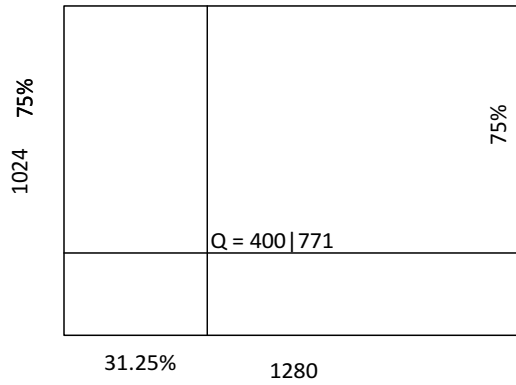
5.3 Interpolation

Interpolationsansätze beschreiben, wie anhand der gefundenen Referenzpunkte die genaueste Interpolation für ein Mapping eines Kamerapixels auf eine Bildschirmkoordinate resultiert. Diese Verfahren wurden testweise auch auf die Ecken des erkannten Bildschirms angewandt, um die Genauigkeit zu überprüfen.

5.3.1 Lineare Interpolation

Bei der Umsetzung mittels linearer Interpolation wird ausgenutzt, dass die Verhältnisse der Strecken in der Abbildung identisch bleiben. Dies ist hier nicht ganz korrekt, da die perspektivische Verzerrung nicht berücksichtigt wird, was sich nicht schwerwiegend auswirkt, jedoch zu einer gewissen Ungenauigkeit führt. Nachteile dieses Verfahrens sind zudem, dass in jedem Quadranten des gesuchten Punktes ein Punkt gefunden werden muss, der mit anderen zusammen auf dem Bildschirm ein Rechteck bildet. Diese Voraussetzung ist für die Eckpunkte gegeben, jedoch ein erschwerender Faktor wenn eine unregelmässige Verteilung von Referenzpunkten gegeben ist. Ausserdem kann dieses Verfahren nur in eine Richtung angewandt werden, nämlich von Bildschirmkoordinaten zu Kamerapixeln. Dies ist die falsche Richtung. Bei diesem Vorgehen müssten alle Abbildungswerte vorberechnet und bei Bedarf nachgeschaut werden. In Algorithmus (2) wird das mathematische Vorgehen beschrieben.

Algorithmus 2 Berechnung der linearen Interpolation



$$Ratio_X = \frac{Q_X}{Screen_X}$$

$$Ratio_Y = \frac{Q_Y}{Screen_Y}$$

$$E = A + \overrightarrow{AB} * Ratio_X$$

$$F = A + \overrightarrow{AC} * Ratio_Y$$

$$G = B + \overrightarrow{BD} * Ratio_Y$$

$$H = C + \overrightarrow{CD} * Ratio_X$$

$$P = \text{Schnittpunkt}(\overrightarrow{EH}, \overrightarrow{FG}) \Rightarrow$$

$$A_{EH}x + B_{EH} = C_{EH} \Rightarrow A_{EH} = Y_H - Y_E$$

$$B = X_E - X_H$$

$$C_{EH} = A_{EH} * X_E + B_{EH} * Y_E$$

Die Berechnung der anderen Koeffizienten läuft identisch.

$$Det = A_{EH} * B_{FG} - A_{FG} * B_{EH}$$

Wenn die Determinante gleich null ist, sind die Linien parallel. Dieser Fall kann ausgeschlossen werden.

$$X_P = \frac{(B_{FG} * C_{EH} - B_{EH} * C_{FG})}{Det}$$

$$Y_P = \frac{(A_{EH} * C_{FG} - A_{FG} * C_{EH})}{Det}$$

$$P = (X_P | Y_P), \text{ wenn } Det \neq 0$$

Genauigkeit Der lineare Ansatz wurde getestet und seine Genauigkeit gemessen. Dazu wurde ein Grid mit $\frac{1}{8}$ Unterteilungen auf die Leinwand projiziert. Danach wurden die Positionen am Rand und in der Mitte der X-Achse und der Y-Achse entlang mit einer LED-Lampe beleuchtet. In Abb. 55 ist sichtbar, dass der lineare Ansatz ziemlich gut funktioniert, wenn Beamer und Laptop mit 0° zueinander ausgerichtet sind. In Abb. 56 wurde der Winkel zwischen Beamer und Laptop auf 30° erhöht. Hier macht sich die perspektivische Verzerrung in X-Richtung bemerkbar. Strecken werden zu lang abgebildet²⁴.

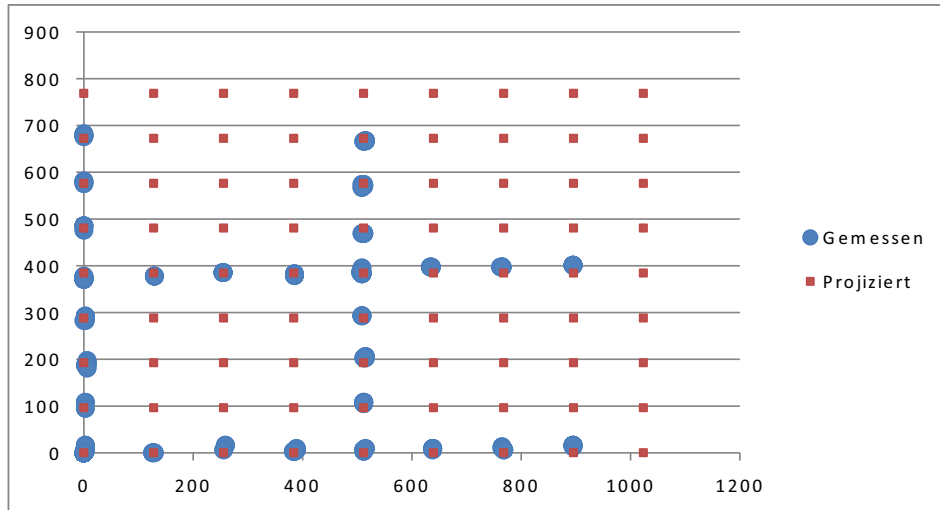


Abb. 55: Ausrichtung Beamer-Laptop: 0°

²⁴Interessant ist, dass die oben in Abb. 17 festgestellte Abweichung vom linearen Ansatz in dieser Messung sichtbar gut ist.

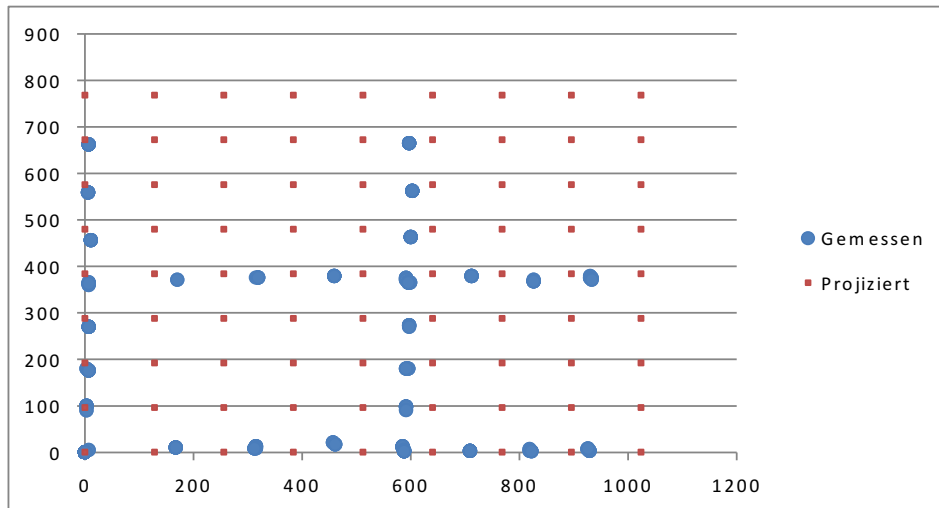


Abb. 56: Ausrichtung Beamer-Laptop: 30°

Bemerkungen zu den Messabbildungen Die zwei oben gezeigten Abbildungen bilden die Y-Achse nach oben ab. In Beamer- und Kamerabild wurde Y nach unten gemessen. Zudem bildet die Umsetzung des Ansatzes alle Punkte ausserhalb der Projektion auf den Punkt (0,0) ab. Da alle am Rand unten gemessenen Punkte ein bisschen ausserhalb liegen werden sie als leerer Punkt gewertet und sind darum in den Messabbildungen nur als Häufung um (0,0) sichtbar.

5.3.2 Baryzentrische Koordinaten²⁵

Funktionsweise Baryzentrische Koordinaten beschreiben in jedem n -Eck n Faktoren, die das Verhältnis zwischen den Flächen zwischen den Kanten und einem Punkt beschreiben. Deren Summe ist immer 1. Anhand dieser Faktoren lässt sich sehr einfach ein Rebasings durchführen. Das heisst, wenn die den Koordinatenursprüngen entsprechenden Eckpunkte in einem anderen System bekannt sind, kann die Position des Punktes ins Zielsystem übertragen werden. Dies ist in dieser Arbeit jeweils der Fall. Die gängigen Referenzpunkte sind jeweils die Beamerbild-Eckpunkte. Die Definition von baryzentrischen Koordinaten wird dem Leser hier nicht weiter eingeprägt. Diese kann er bei Interesse in einem der Paper [6, 8, 4, 7, 3] nachlesen.

Anwendung Baryzentrische Koordinaten berücksichtigen keine perspektivischen Verzerrungen, wie sie in dieser Arbeit vorkommen. Dies schränkt ihren Nutzen etwas ein. Sie können nur direkt verwendet werden, wenn ein Punkt innerhalb oder zumindest in der Nähe eines schon perspektivisch korrigierten Punktes liegt. Andernfalls ergeben sich Ungenauigkeiten, die sich ähnlich zu denen des linearen Ansatzes verhalten.

²⁵Für eine detailliertere Erklärung siehe http://en.wikipedia.org/wiki/Barycentric_coordinate_system

Nähe wird dadurch definiert, dass alle baryzentrischen Faktoren kleiner als 1.5 sein müssen. Wenn sie kleiner als 1 sind ist der Punkt innerhalb. Mit 1.5 darf er zu einer Kante eine Fläche gegen aussen aufspannen, die maximal der Hälfte der ganzen Fläche entspricht. Das entspricht einem maximalen Abstand der halben Dreieckshöhe zur Kante. Da die Nutzung von n -Ecken ohne zusätzlich Gleichungen weder zusätzliche Genauigkeit noch sonstige Vorteile bringt, werden baryzentrische Koordinaten bei diesem Schritt nur im Dreieck benutzt.

Durch eine Mittlung der resultierenden Koordinaten anhand mehrerer Dreiecke können die Störungen der Verzerrung etwas minimiert werden. Wenn genügend Referenzpunkte vorhanden sind, fällt das nicht mehr ins Gewicht.

Ergebnisse Wenn der Beamer und das Notebook nebeneinander stehen, funktioniert die Interpolation anhand der Eckpunkte ziemlich gut. Die Implementation wurde jedoch meist unter erschwerten Umständen (flacheren Winkeln) getestet. Hier gab es bei der linearen Interpolation gewisse Verschiebungen in horizontaler Richtung. Dies entspricht nicht der erwünschten Genauigkeit. Mit den baryzentrischen Koordinaten anhand der Eckpunkte gab es anfangs sehr viele Ungenauigkeiten. Zudem waren Knicke in den Linien über die Diagonale sichtbar.

Eine Mittlung der Ergebnisse aller vier Dreiecke hat jedoch geholfen, die Knicke zu eliminieren und die vertikalen Fehlerfaktoren zu eliminieren. Dafür gab es damit Probleme an den Rändern, da dort unrealistische Werte entstehen. In Tabelle 1 ist sichtbar, wie stark sich die perspektivische Verzerrung auswirkt.

Y Werte:			
Soll	links	mitte	rechts
0	-62	12	50
100	40	86	125
200	150	193	212
300	264	289	297
400	368	390	391
500	505	495	479
600	632	594	574
700	761	689	662
768	838	756	700

X Werte:											
Soll	0	100	200	300	400	500	600	700	800	900	1000
oben	10	152	263	382	486	593	686	772	857	939	1005
mitte	15	148	266	384	494	600	689	780	865	950	1023
unten	2	130	265	381	491	600	695	779	870	950	1034

Tab. 1: Genauigkeit der baryzentrischen Interpolation

Kompensation der Verzerrung Wenn nur drei oder vier Referenzpunkte für eine Interpolation verwendet werden, ist das Resultat korrekt, wenn das Kamerabild winkeltreu ist. Doch auch in diesem Fall gibt es Probleme, wenn mit diesem Verfahren eine Linie über mehrere Dreiecke gezogen wird. Diese hat dann an jedem Übergang eine neue Richtung. Das wird von einem Nutzer dadurch wahrgenommen, dass eine über die Mitte gezogene Linie nicht gerade ist, sondern Knicke enthält.

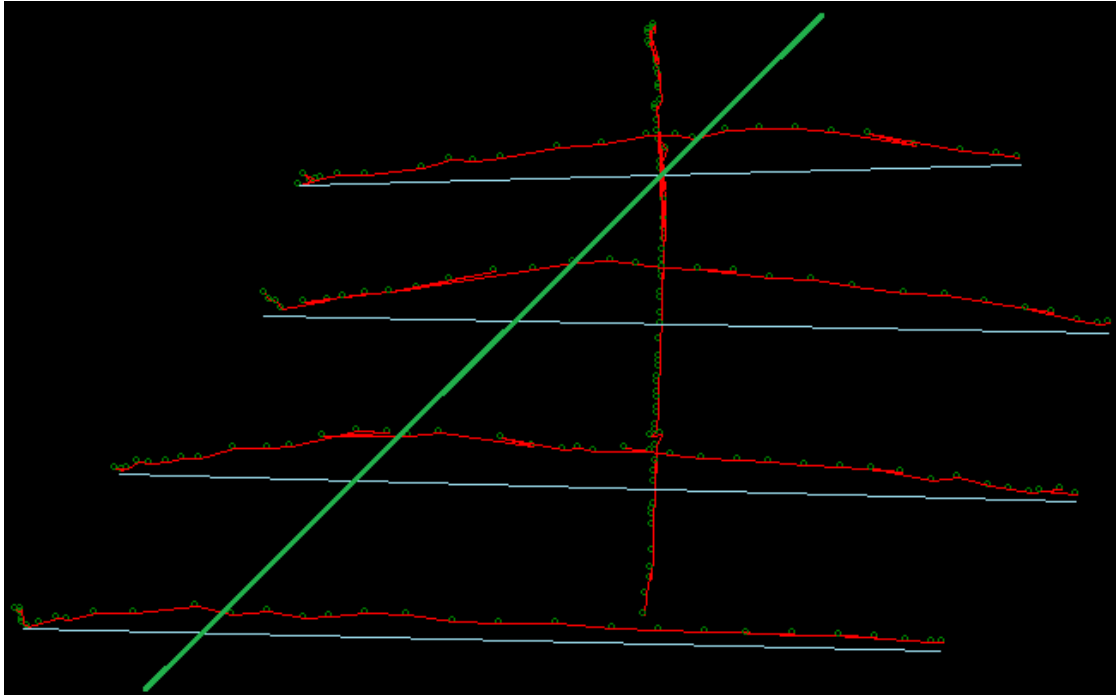
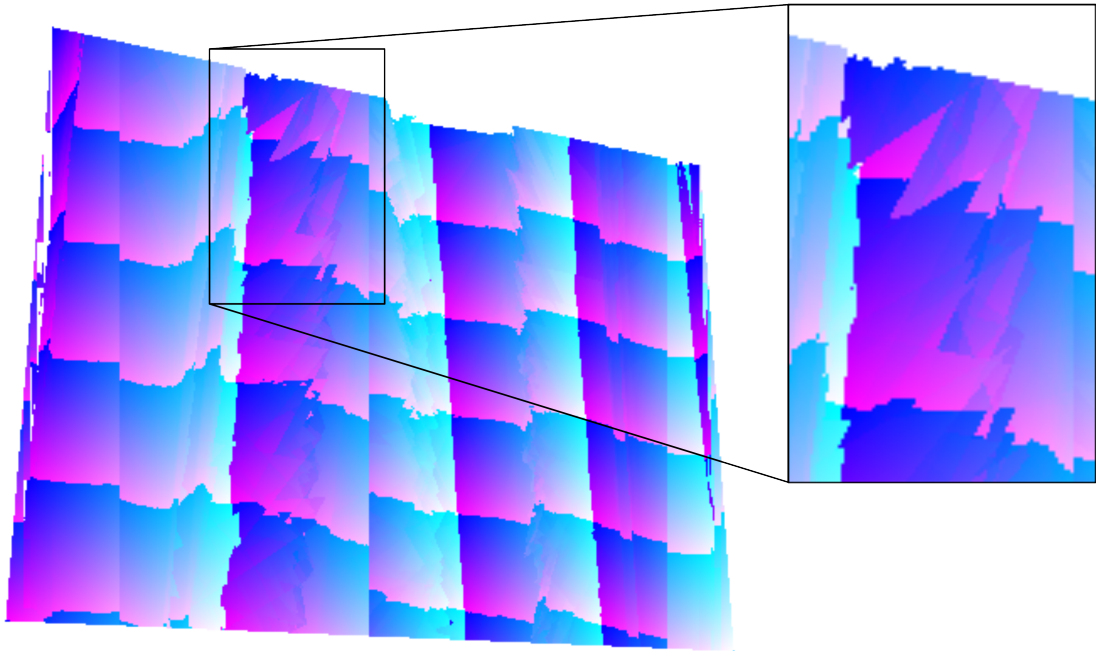
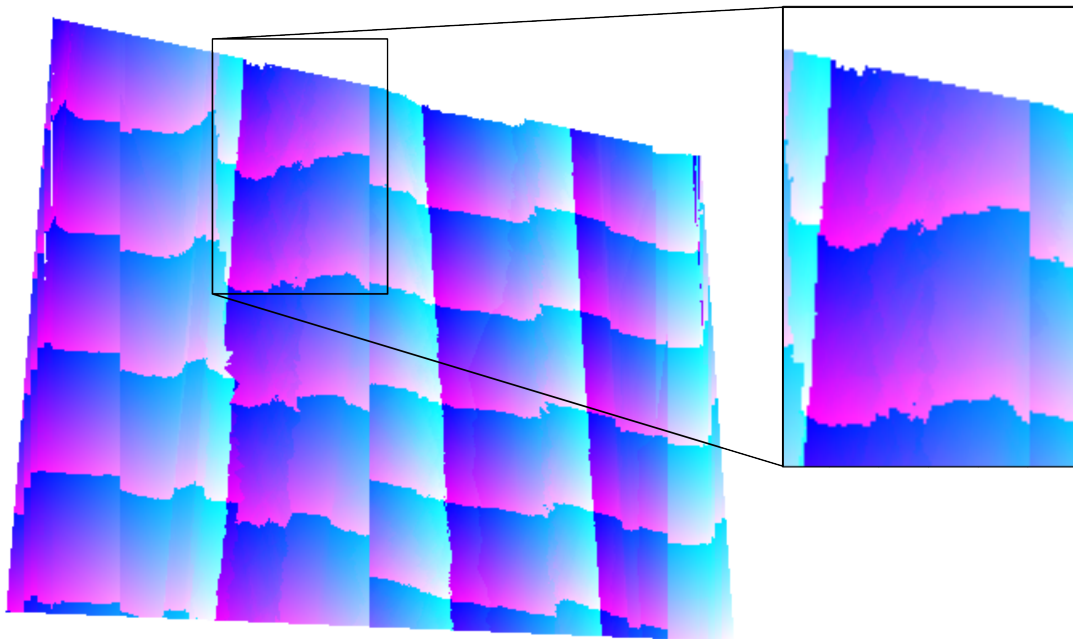


Abb. 57: Gezeichnete Linien (blau) resultieren in geknickten Linien (rot) über die Diagonale (grün)

Mittelung von 3 Resultaten:



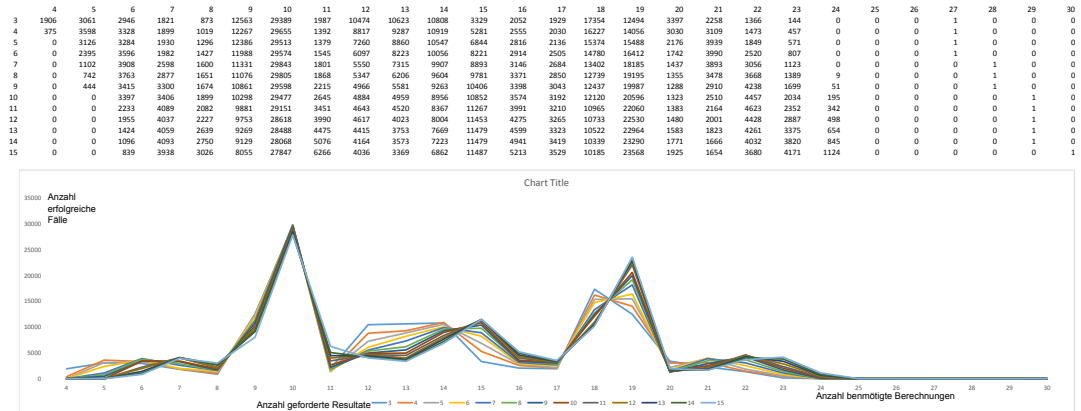
Mittelung von 15 Resultaten:



Die Rotwerte representieren die interpolierten X-Werte, die Grünwerte diejenigen der Y-Werte. Der Blauwert ist auf der ganzen Leinwand identisch.

Abb. 58: Interpolation anhand eines feinen Gitters mit baryzentrischen Dreiecken

Interpolation anhand eines feineren Gitters Die Anzahl berücksichtigter baryzentrischer Dreiecke ist hier sehr wichtig. In Abb. 58 ist sichtbar, dass mit der Mittlung von mehreren Resultaten lokale Unstimmigkeiten behoben werden. Das benötigt jedoch auch einen höheren Rechenaufwand, wie man in Tabelle 2 sieht. In dieser Tabelle sind die nötigen Berechnungen für eine gewisse Anzahl Dreiecke aufgeführt. Es gibt jedoch viele globale Unstimmigkeiten, die nicht korrigiert werden können. Diese sieht man daran, dass alle Linien eigentlich gerade sein müssten. Dies ist jedoch an mehreren Stellen (überall wo es Wellen und Knicke hat) nicht gegeben.



Tab. 2: Benötigter Aufwand für baryzentrische Berechnung

Gewichtung Da baryzentrische Koordinaten leider keine Abstandsfunktion definieren, können keine Nachbarn erkannt und keine Gewichtung durchgeführt werden um das Problem der Knicke zu lösen. Zudem gibt es das Problem, dass gewisse Dreiecke falsche Daten liefern, wie in Abb. 15 dargestellt. Daher werden regelmässige Rechtecke verwendet. Dabei wird nicht nur das Resultat eines Rechtecks analysiert, sondern auch das der direkt benachbarten Rechtecke. Durch eine entsprechende Gewichtung der Nachbar-Vierecke sollte das Resultat nicht verfälscht werden, aber die Knicke soweit geglättet, dass sie dem Nutzer nicht mehr auffallen.

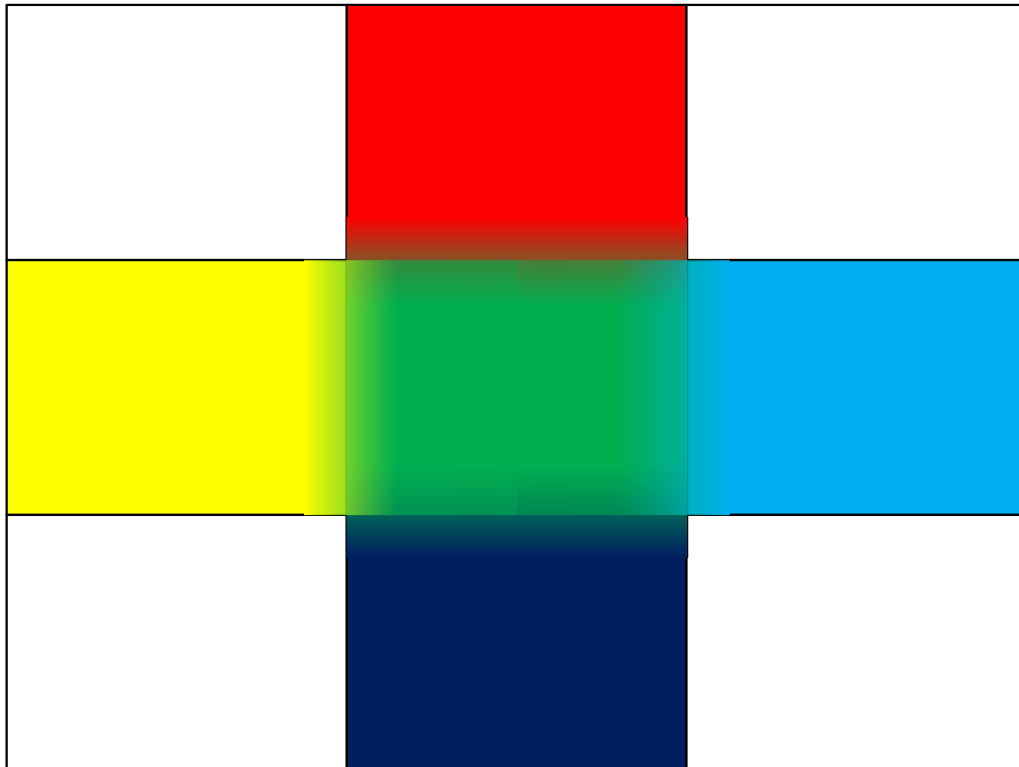


Abb. 59: Gewichtung der Vierecke

Hierbei entsprechen die Farben an den Übergängen der Gewichtung der Resultate.

5.4 Versuche zur Umsetzung von Touch

Dieser Exkurs befasst sich mit der Vision, eine projizierte Beamerwand mittels Touch zu bedienen. Der Benutzer soll die projizierte Fläche mit der Hand antippen und so das System hinter der Projektion steuern können. Diese Aufgabenstellung lässt sich über verschiedene Fragestellungen analysieren.

Relevante Informationen finden Zuerst muss klar werden, woraus die benötigten Informationen gewonnen werden. Dafür sind ein oder mehrere Kamerabilder denkbar²⁶. Ebenfalls fragt sich, wie die Informationen aus diesem Kontext gewonnen werden können. Es ist möglich, die Person vor der Leinwand gegen ein Basisbild zu vergleichen oder bewegte Teile mittels Differenzbildanalyse über zwei oder mehr Bilder zu erkennen. Es könnte der Person ein Handschuh angezogen werden, der eine eindeutige Farbe hat und so immer sogar in nur einem Kamerabild zu erkennen ist. Es wäre denkbar, eine Person mit komplizierteren Algorithmen über ein vorher konfiguriertes Pattern zu tracken.

²⁶3D-Analysesysteme wurden vollständig ausser Acht gelassen, da sie teurer sind als die in dieser Arbeit gesetzte Kostengrenze von Zusatzhardwarekosten (ca. 20.-).

Die Mathematik birgt mit Fouriertransformation und Direct Linear Transformation viele Möglichkeiten Patternmatching zu betreiben. Die in dieser Arbeit untersuchten Ansätze sind einfacherer Art und beschränken sich darauf zu zeigen welche allgemeinen Aspekte betrachtet werden müssen.

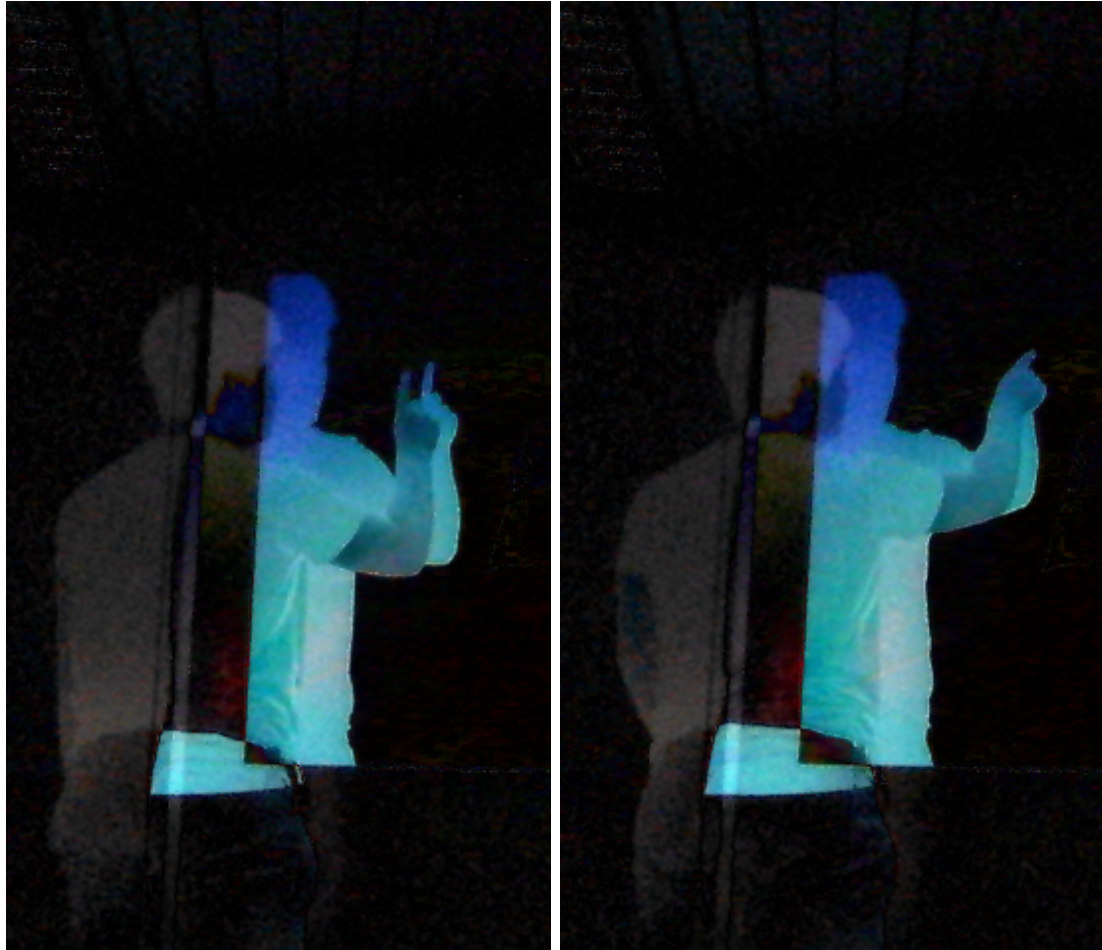
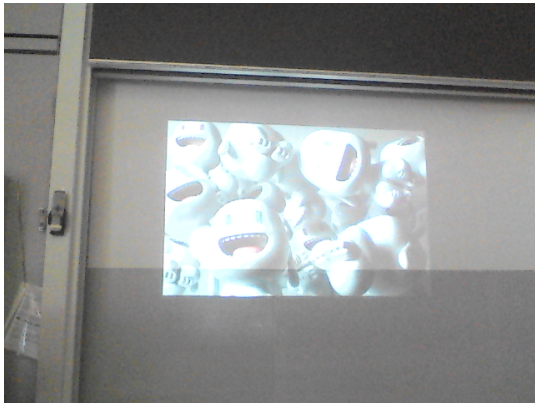
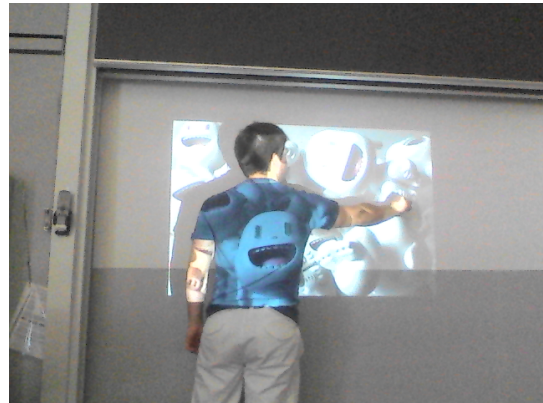


Abb. 60: Finger in Ruhestellung (links) - Finger tippt auf Leinwand (rechts)

Hand und Finger finden Die Hand einer Person zu finden ist mit wenig Aufwand gut zu erreichen. Über den unteren Rand des Kamerabildes wird die im Bild stehende Person mittels Differenzbilder gefunden. Die Position und das Ausmass des Körpers der Person ist mit wenig Spiel gut abschätzbar entlang zweier senkrechter Linien am Rand der Beine (siehe Abb. 61). So ist auch bekannt, in welchem Bereich sich Arm und Hand der Person befinden. Ein konkreter Ansatz für eine Vorgehensweise wird weiter unten beschrieben, nachdem kurz auf Gesten eingegangen wurde.



1) Basisbild



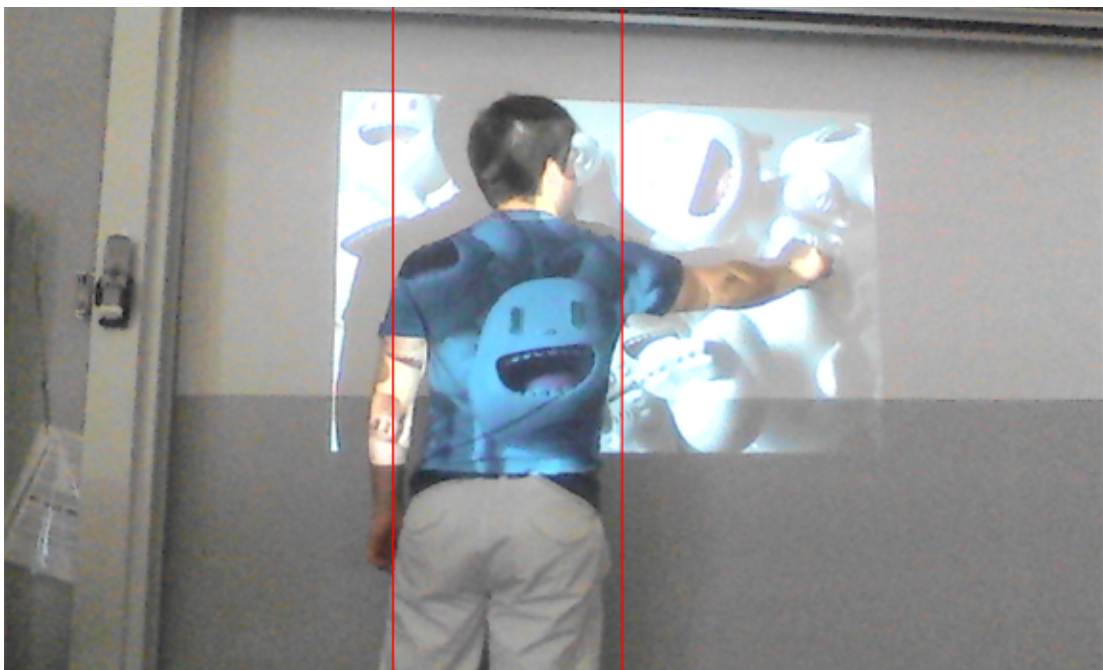
2) Frame



3) Differenz



4) entauschtes Differenzbild



5) Personenposition

Abb. 61: Vorgehen zum Finden einer Hand

Handgesten erkennen Handgesten wie Touch oder Greifen zu erkennen, ist nicht ganz trivial. Der hier vorgeschlagene Ansatz versucht den Körper der Person vom Hintergrund durch den Schatten zu trennen. Die Unterscheidung zwischen Schatten und Person geschieht auf der Differenz gegen das Basisbild. So bleiben alle nicht benötigten Informationen unbeachtet. In Abb. 62 ist der Unterschied zwischen Schatten und Arm auf dem Differenzbild sichtbar. Im Diagramm wurde analysiert, welche durchschnittlichen Farbwerte das Differenzbild in einem vertikalen Streifen besitzt. Der Streifen wurde von oben nach unten über 480 Pixel durchlaufen. Das Maximum bezeichnet den Schatten, das lokale Minimum den Arm. Sobald klar ist, an welcher Position sich der Übergang von Schatten zu Arm befindet, kann die Schattenfarbe im Originalbild nachgeschaut werden.

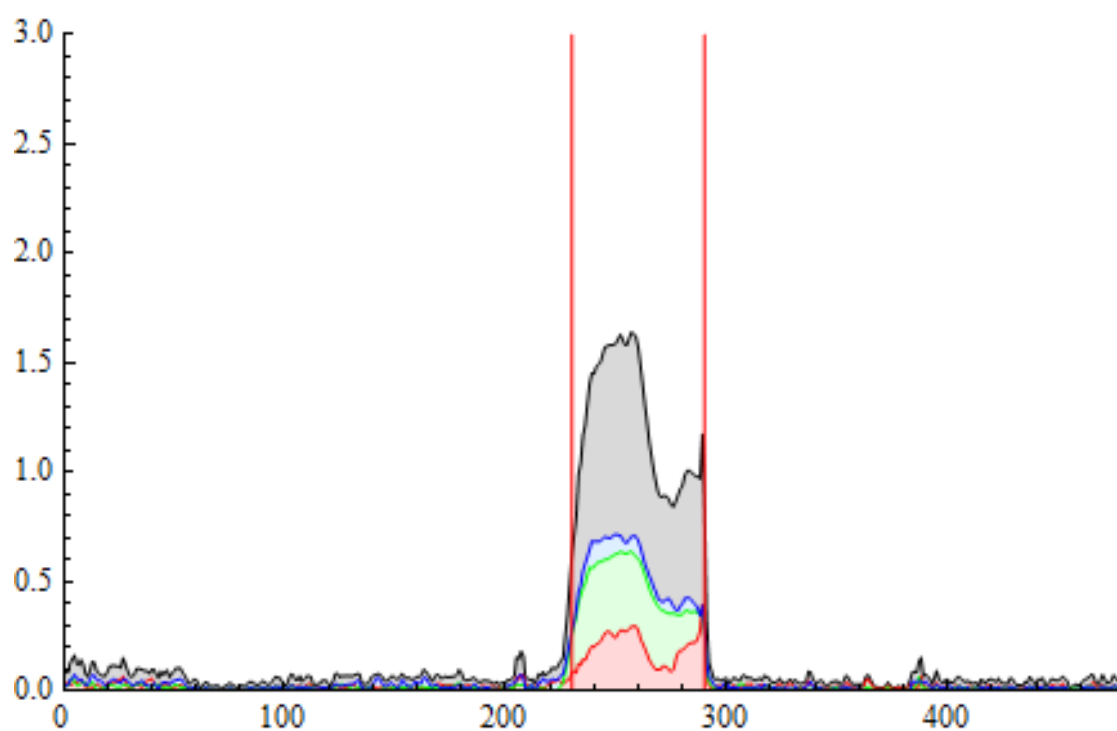
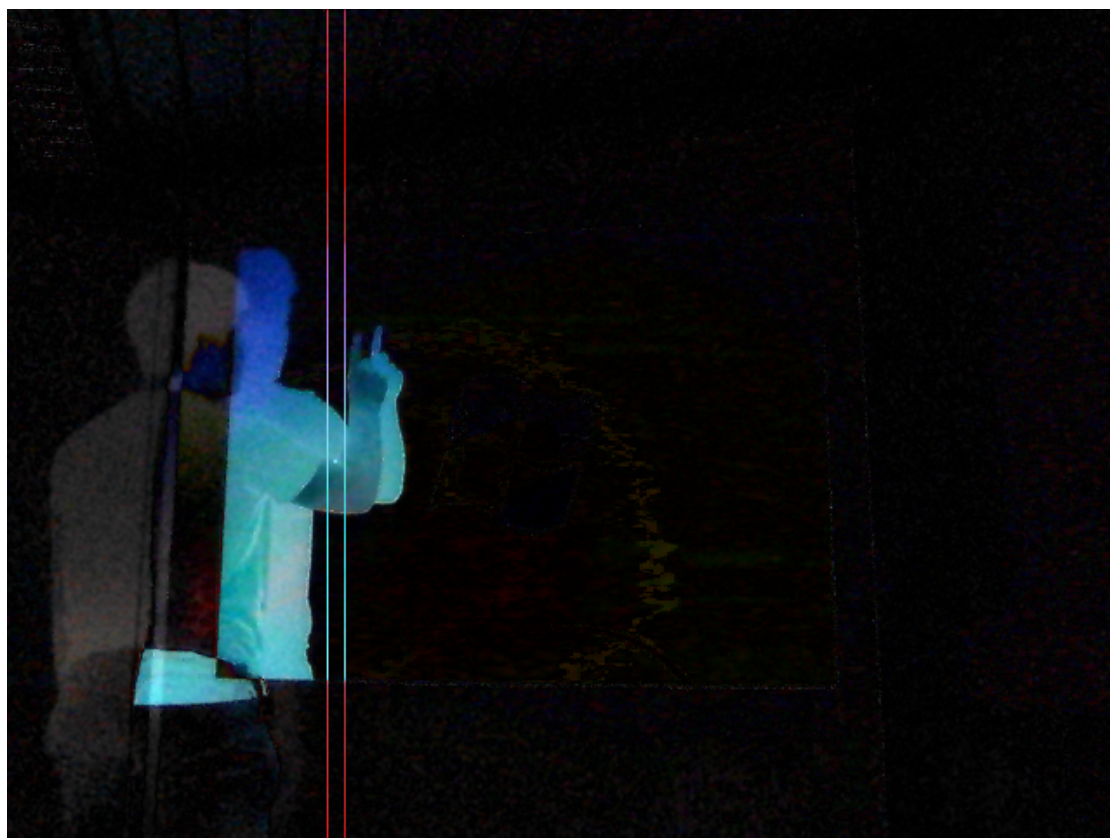


Abb. 62: Die Helligkeitsanalyse innerhalb des gekennzeichneten vertikalen Streifen des Differenzbildes (oben) ergibt im Schnitt diese Farbaufteilung (unten).

Die Schattenfarbe wird aus dem Originalbild als weiss markiert und vom eingeschränkten Differenzbild abgezogen. Dann bleibt der Arm übrig ohne Schatteninformation. Der Gesamtanteil an Schatteninformation sind die vom eingeschränkten Differenzbild abgezogenen Pixel. Unter Umständen könnte die Menge an Schattenpixel und deren Position für die Detektion verwendet werden ob die Person die Leinwand antippt oder nicht. Dies ist eine sehr vage Theorie und bedarf weiterer Untersuchungen.

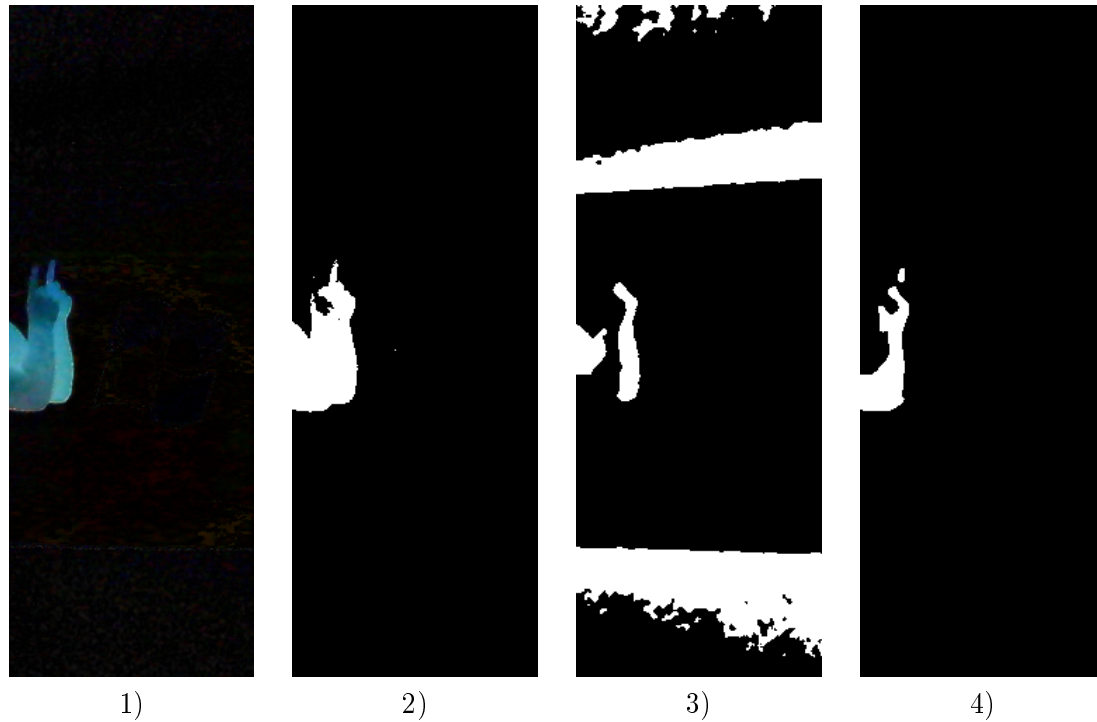


Abb. 63: 1) Differenzbild gegen Basis, 2) per Helldunkel-Schranke eingeschränktes Differenzbild, 3) Schattenfarbe weiss markiert aus dem Originalbild, 4) zweites minus drittes Bild

Da nun der Arm gefunden ist, könnte in diesem eingeschränkten Bereich ein Patternmatching durchgeführt werden. Ein Sample der Hand oder des Fingers könnte an Schlüsselpositionen gefunden werden. Die Information ob eine Geste wie Touch durchgeführt wurde, kann aus der streifenweisen Schattenanalyse geholt werden. Ist der Schatten in der richtigen Intensität am richtigen Ort, so fand ein Touch statt, sonst nicht. Einfach dürfte dieses Vorgehen jedoch nicht sein. Es gibt zu viele Arten, wie die Schattenverteilung sein könnte. Hier wird vorgeschlagen, sich zuerst auf das genaue Finden der Hand festzulegen. Sobald die Hand getrackt werden kann, ist es möglich, weitere Schlüsse aus den sich daraus ergebenden Analyse- und Messmöglichkeiten zu ziehen.

Eine Idee dazu ist, Bewegungen der Gelenke Schulter, Ellbogen und Hand aufzuzeichnen. Dies könnte über einen Ansatz mit Ausgleichsgeraden geschehen. Dazu wird von der Schulter mit einer radialen Scanline von unten über die Punktmenge des Armes gefahren.

Sobald ein Punkt auftaucht, ist entweder die Hand oder der Ellbogen gefunden worden. Je nach Entfernung des Punktes von der Schulter lässt sich das einfach entscheiden. Am Ellbogen wird das Verfahren wiederholt bis der erste Punkt der Hand gefunden wurde. Das Fahren der Scanline kann mathematisch als Koordinatentransformation von euklidischen zu Polarkoordinaten beschrieben werden. Dies vereinfacht das Modellieren der Scanline.

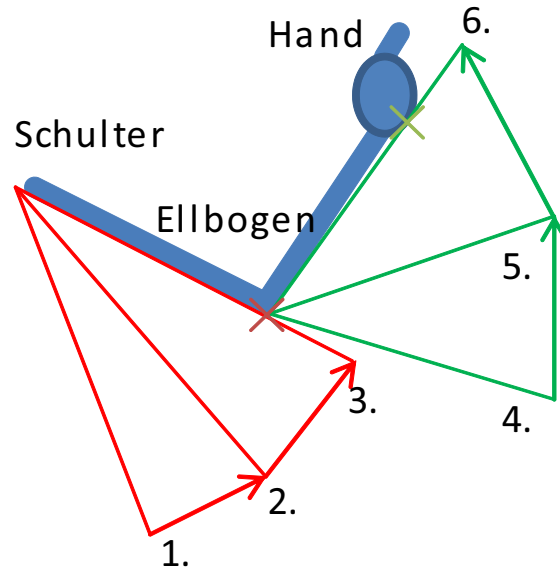


Abb. 64: Scanline-Ansatz um Gelenke zu finden

Ergebnisse Es ist schwierig zu sagen, ob es sinnvoll ist, Touch auf der Leinwand weiter zu verfolgen. Einerseits wäre diese Funktionalität innovativ und wünschenswert. Sie ist an keine weitere Hardware gebunden und die zugehörige Software könnte mit dem Betriebssystem zusammen ausgeliefert werden. Andererseits ist es sehr schwierig zu erkennen, ob eine Person die Leinwand angetippt hat. Bei schlechten Lichtbedingungen und komplizierteren projizierten Bildern konnte die Hand bei schneller Bewegung nicht einmal mehr mit bloßem Auge erkannt werden. Ob eine Webcam mit schlechter Auflösung, niedriger Framerate und schlechter Belichtung überhaupt den Input für eine zuverlässige Datenverarbeitung liefern kann ist fraglich. Es sollte möglich sein, die Hand zu tracken und bei einem geübten Benutzer die Finger. Dies allerdings nur bei optimalen Lichtbedingungen. Es ist hingegen nicht klar, wie dass mit hoher Trefferrate festgestellt werden kann, ob die Person die Leinwand angetippt hat. Das oben beschriebene Vorgehen für die Schattenanalyse befasst sich lediglich mit den Daten, welche in dieser Arbeit aufgenommen wurde. Es müsste systematischer untersucht werden, ob dies für jeden Fall zuverlässig gelöst werden könnte.

5.5 Quellenverzeichnis

Literatur

- [1] Janne Heikkila. Geometric camera calibration using circular control points. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 22(10), 2000.
- [2] John Tsotsos James MacLean. Fast pattern recognition using gradient-descent search in an image pyramid. *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, 2:873 – 877, 2000.
- [3] Anil N. Hirani Mathieu Desbrun Joe Warren, Scott Schaefer. Barycentric coordinates for convex sets. *Advances in Computational Mathematics*, (27):319 – 338, 2007.
- [4] Michael Floater Tao Ju Solveig Bruvoll Sukumar Kai Hormann, Scott Schaefer. Barycentric coordinates and transfinite interpolation. *SIAM Conference on Geometric Design and Computing*, (10), 2007.
- [5] Seth Rogers Stefan Schroedl Kiri Wagstaff, Claire Cardie. Constrained k-means clustering with background knowledge. *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577 – 584, 2001.
- [6] Petter Lidberg. Barycentric and wachspress coordinates in two dimensions: theory and implementation for shape transformations. *U.U.D.M. Project Report*, (3), 2011.
- [7] Bert Riffelmacher. Baryzentrische koordinaten & interpolation. *Seminar Graphische Datenverarbeitung*, 2010.
- [8] Gabriel Zachmann. Texturen 2. *Computergraphik II*, 2010.