UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Evaluation und Optimierung der Merkmalsetraktion in Gaze-Tracking-Systemen

Bachelorarbeit
zur Erlangung des Grades
Bachelor of Science
im Studiengang Computervisualistik

vorgelegt von

Baharak Rezvan

Betreuer: Dipl.-Inform. D. Droege, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Erstgutachter: Dipl.-Inform. D. Droege, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Zweitgutachter: Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im September 2014

Kurzfassung

Ziel der Bachelorarbeit war die Evaluation und Optimierung von verschiedenen Eye-Tracking Algorithmen für die subpixelgenaue Bestimmung der relevanten Merkmale bezüglich Genauigkeit. Die extrahierten Merkmale sind das Pupillen- und Glintzentrum. Die Algorithmen sind mit preiswertigen Kameras einsetzbar. Es wurde ein synthetisches Modell des Auges modifiziert und verwendet, mit dem die Methoden mit bekannter Ground Truth evaluiert wurden.

Abstract

The goal of this work is evaluation and optimization of several eye-tracking algorithms for estimation of relevant features regarding accuracy. The extracted features are pupil- and glintcenters. The algorithms are applicable to off the shelf cameras. A synthetic model of the eye was modified and utilized. The model was used to supply ground truth for the evaluation of the methods.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Vereinbarung der Arbeitsgruppe für Studien- und Abschlussarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	ja ☐	nein ☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	ja ☐	nein ☐

Koblenz, den 15. September 2014

Danksagung

Ich möchte mich bei meinen Eltern sowie Peter Decker, die mich immer unterstützen bedanken. Mein besonderer Dank gilt meinem Betreuer Detlev Droege für die konstante Unterstützung. Bei Prof. Paulus bedanke ich mich für die Gelegenheit, in seiner Gruppe meine Arbeit zu schreiben. Bei Sebastian Koslink bedanke ich mich für die große Hilfe.

When I know your soul, I will paint your eyes

Amedeo Modigliani

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Augen dienen nicht nur zum Sammeln von Information, sondern auch zum Kommunizieren mit der Welt. Diese Kommunikationsmöglichkeit durch die Augen wird in vielen Bereichen benutzt: In der Marktforschung, der Psychologie, den Neurowissenschaften, der Mensch-Maschine-Interaktion und so weiter. Gaze-Tracking ist das Verfahren, durch das der Blickpunkt des Auges berechnet wird. Ein Bestandteil davon ist das Eye-Tracking, welches versucht, das Auge zu lokalisieren. Außer den teuren Gaze-Tracking Geräten, die im Markt vorhanden sind, wird versucht, mit preiswerten Kameras ähnliche Ergebnisse zu erzielen. Aufgrund der günstigen Hardware müssen die folgenden Herausforderungen überwunden werden: Gering aufgelöste und verrauschte Bilder zu verarbeiten, Kopf- und Augenbewegungen, sowie die Reflexion des Monitors im Auge zu kompensieren.

Gaze-Tracking, wie wir es vorstellen, besteht aus drei Haupteilen: Augendetektion, Extraktion der Merkmalen und Gazeschätzung. Wir werden uns in dieser Arbeit mit der Augen- und Merkmalsdetektion beschäftigen. Dafür werden die Pupillen- und Glintzentren als zu bestimmende Merkmale extrahiert. Diese Merkmale sind Vorraussetzung für die Bestimmung des Blickpunkts.

In dieser Arbeit werden zuerst (in Kapitel 2) Grundbegriffe definiert, state-of-the-art Hardware und Algorithmen kategorisiert und vorgestellt. In Kapitel 3 werden die implementierte Pipeline, das Testframework, Hardware und die implementierten Algorithmen für die Pupillen- und Glintzentrumsberechnung präsentiert und die potentiellen Schwächen und mögliche Verbesserungen dazu beschrieben. Kapitel 4 befasst sich mit der Bewertung von den implementierten Methoden und deren Genauigkeit. Außerdem wird in diesem Kapitel das synthetische Modell, das für die Evaluation benutzt wurde, vorgestellt. Die Zusammenfassung und der Ausblick in Kapitel 5 beschließen die Arbeit.

Kapitel 2

Stand der Wissenschaft

In den letzten 30 Jahren wurde versucht Viedeo-based-Gaze-Tracking Systeme zu entwickeln. Diese Systeme kann man nach Hardware- und Software-Aufbau unterscheiden. Die Auswahl von Hardware beeinflusst die Auswahl von Algorithmen. Zum Beispiel: wenn nur gering aufgelöste Bilder zur Verfügung stehen, werden Algorithmen, die viele Pixel in einem Bereich vom Auge benötigen, ausgeschlossen. In diesem Kapitel werden die State-of-the-art Techniken und Algorithmen für Gaze-Tracking vorgestellt.
Vorab werden anwendungsspezifische Begriffe definiert, um Undeutlichkeiten zu vermeiden.

- Glint: Das Licht reflektiert mehmal zwischen Linse und Hornhaut. Dadurch entsehen sogenannten Purkinje Reflexionen. Die erste Purkinje Reflexion wird Glint genannt (siehe Abbildung 2.1).
- Gaze oder Gazepunkt: der Punkt, auf den das Auge anblickt.
- Eye-Tracking: Das Verfahren, durch das das Auge lokalisiert wird.
- Gaze-Tracking: Das Verfahren, durch das der Gazepunkt berechnet wird.
- Gaze-Schätzung: Das Verfahren, das die detektierten Merkmale in Video-Frames auf Gaze-Koordinaten abbildet.

2.1 Hardware-Kategorisierung

Die Gaze-Tracking Geräte sind entweder auf dem Gesicht gelagert (*headmounted*) oder von dem Gesicht entfernt (*remote*). Headmounted Geräte benötigen eine

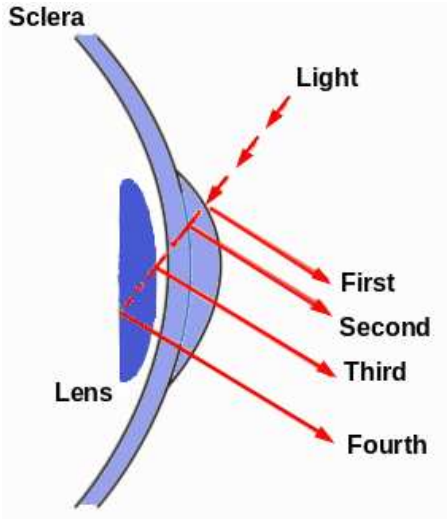


Abbildung 2.1: Die Reflexion vom Licht resultiert Purkinje Reflexionen[HJ10]

Schätzung der Kopfpose, kompensieren aber diesen Aufwand, indem sie mehr Mobilität für den Benutzer ermöglichen. Für Remote Geräte muss man nicht unbedingt die Kopfpose schätzen.
Hinsichtlich der Anzahl von benutzten Kameras und Lichtquellen teilen sich die Geräte in folgende Kategorien auf:[HJ10]

- Einzel-Kamera und Einzel-Licht (*Einzel-Glint*): Abbildung 2.2 zeigt ein Beispiel für einen solchen Gaze-Tracker. Shih et al.[SWL00] beweisen, dass die Verwendung von einem einzelnen Glint und Pupillenzentrum zu keiner kopfposeinvarianten Gazeschätzung führen kann. Diese Technik ist erfolgreich bei der Verwendung von festgelagerten Kameras und für minimale Kopfbewegungen.[HJ10]

- Einzel-Kamera und Multipel-Licht (*Multipel-Glint*): Diese Technik erlaubt mehr Kopfpose Invarianz. Es besteht die Möglichkeit, dass eine Glint wegfällt. Einschränkungen für Kopfbewegungen sind deswegen möglich.[HJ10]

- Multipel-Kamera und Multipel-Licht: Solche Geräte liefern robuste Ergebnisse, benötigen aber eine Stereokalibrierung.[HJ10]

Die Lichtquelle ist entweder *sichtbares Licht* oder *Infrarot*. So entsteht durch die Lichtquelle eine Unterteilung der Verfahren. Die Verfahren, die kontrolliertes IR Licht verwenden, sind *aktiv* im Gegensatz zu *passiv*en, die das sichtbare Licht einsetzen. Durch Anwendung von IR Licht wird die spekulare Reflexion beseitigt.

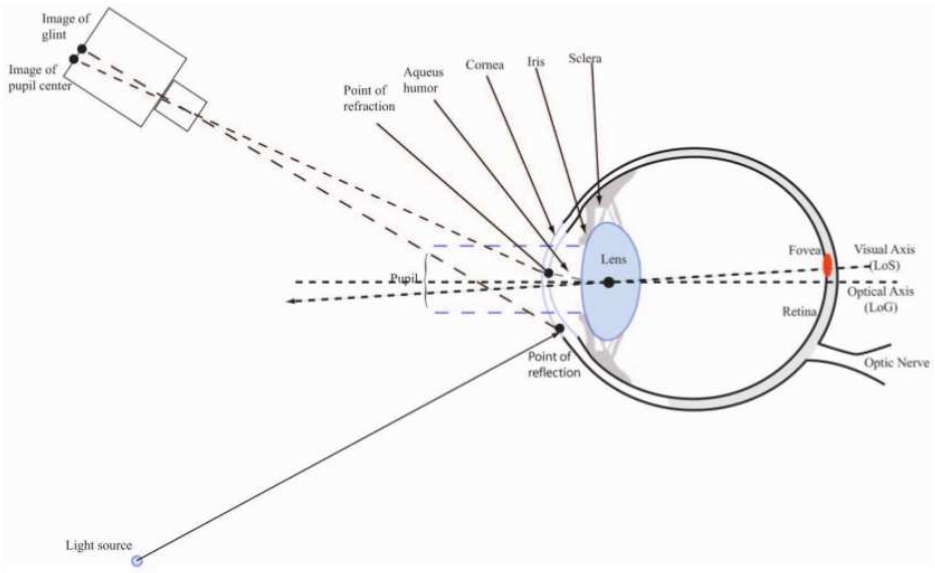


Abbildung 2.2: Modell der Struktur des Auges, Lichtquelle und Projektionen[HJ10]

Die Sklera und Iris reflektieren das IR Licht stark, während nur die Sklera sichtbares Licht stark reflektiert. Mit IR wird mehr Kontrast gewonnen und die Pupile erscheint im Gegensatz zu den passiven Verfahren mit deutlichem Rand. Allerdings sind Geräte mit IR Licht nicht für Außeneinsatz geeignet. Die Lichtquelle kann nah an der optischen Achse der Kamera angebracht werden (*on-axis* Licht), oder fern davon (*off-axis* Licht). In den Bildern, die bei on-axis Lichtquellen entstehen, sind die Pupillen hell, weil das meiste Licht reflektiert wird (ähnlich wie der rote Augen Effekt bei der Nutzung vom Blitz). Die durch off-axis aufgenommenen Bilder zeigen eine dunkle Pupille.[HJ10]

2.2 Software-Kategorisierung

In Bild 2.3 sind die Komponenten eines Gaze- und Eye-Trackers zu sehen. Bild Daten werden von einer oder mehreren Kameras geliefert. Die Augen werden detektiert und entweder direkt von der Applikation verwendet oder in mehreren Frames verfolgt. Anhand der Augen Information und möglicherweise der Kopfpose, wird der Gazepunkt geschätzt. Der berechnete Gazepunkt wird der Applikation weitergegeben.[HJ10]
Die Augendetektion ist eine herausfordernde Aufgabe. Es kann sein, dass die Augen nicht völlig offen sind, dass die Reflexionen vom Bilschirm stören oder Kopfbewegungen das Verfahren stören. Gering aufgelöste Bilder sorgen auch für Probleme.

Verschiedene Methoden sind für Detektion der Augen vorgeschlagen worden. Diese Methoden werden wie folgt unterteilt:

- Modelbasiert
- Merkmalbasiert

Merkmalbasierte Verfahren versuchen die Merkmle, die für die Augendetektion relevant sind, zu detektieren und lokalisieren. Ein Besipiel für solche Merkmale sind unter anderem Pupille und Glint. Daunys und Ramanauskas [DR04] präsentieren ein Mermalbasiertes Verfahren. Sie veruschen, den Pupillenrand mit Hilfe von Polynomialfittung zu berechnen. Anschließend werden zwei Linien horizontal bzw. vertikal durch die Mitte der Randpunkte gefittet. Der Schnittpunkt von diesen Linien ist das Pupillenzentrum. Diese und eine weitere Methode von Daunys werden in Kapitel 3 Abschnitt 3.4 in Detail vorgestellt.

Die Iris und Pupille haben die Form eine Ellipse. Dies wird in manchen *modelbasierten* Verfahren ausgenutzt. Daugman[Dau93] benutzt einen integrodifferentialen Operator um die Iriskontur und Pupillenkontur zu finden. Dieses Verfahren ist aktiv. Wie oben erwähnt, ist die Iriskontur in diesem Fall weniger deutlich. Seine Methode wird in kommerziellen Iriserkennungsgeräten benutzt.

Li et al.[LWP05] schlagen den *Starburst* Algorithmus vor. Der Algorithmus nutzt ein Merkmalbasiertes Verfahren, um die Randpunkte von der Pupille zu finden. Die Gradienten Werte entlang eines Strahles vom Startpunkt in der Mitte des Auges zu den Rändern hin werden mit einem Schwellwert verglichen. Dadurch entstehen Merkmalspunkte. Das Ausstrahlen wird rekursiv von gefundenen Merkmalspunkten wiederholt. Nach der Berechnung von den Randpunkten wird ein modelbasiertes Verfahren angewendet. Die Pupille wird berechnet, indem versucht wird, mit RANSAC eine Ellipse durch die Randpunkte zu fitten. Der Nachteil bei diesem Verfahren ist, dass der Schwellwert für die Gradientenstärke manuell gesetzt wird. Außerdem geht Straburst davon aus, dass der Glint zuvor eliminiert wird, was aufwändig ist. In Kapitel 3 Abschnitt 3.4 werden die Nachteilen vom Glinteliminieren bzw. -ignorieren im Detail disskutiert.

Außer Methoden, die von einer elliptsichen Form für Pupille oder Iris ausgehen, versuchen manche, ein Templatemodell für das Auge zu nutzen. Yuille et al. [YHC92] schlagen ein verformbares Templatemodell des Auges vor. Das Modell besteht aus zwei Parabeln, die die Augenlider repräsentieren (mit 11 Parametern) und einem Kreis, der die Iris repräsentiert. Das Modell wird auf das Bild gefittet. Wenn die Augen nicht weit offen sind, wird diese Methode problematisch. Außerdem ist die genaue Definition von dem Template komplex.

Nachdem die Augen detektiert und ihre Merkmale extrahiert wurden, kann Gazeschätzung mit verschiedenen Verfahren realisiert werden. Am häufigsten sind:

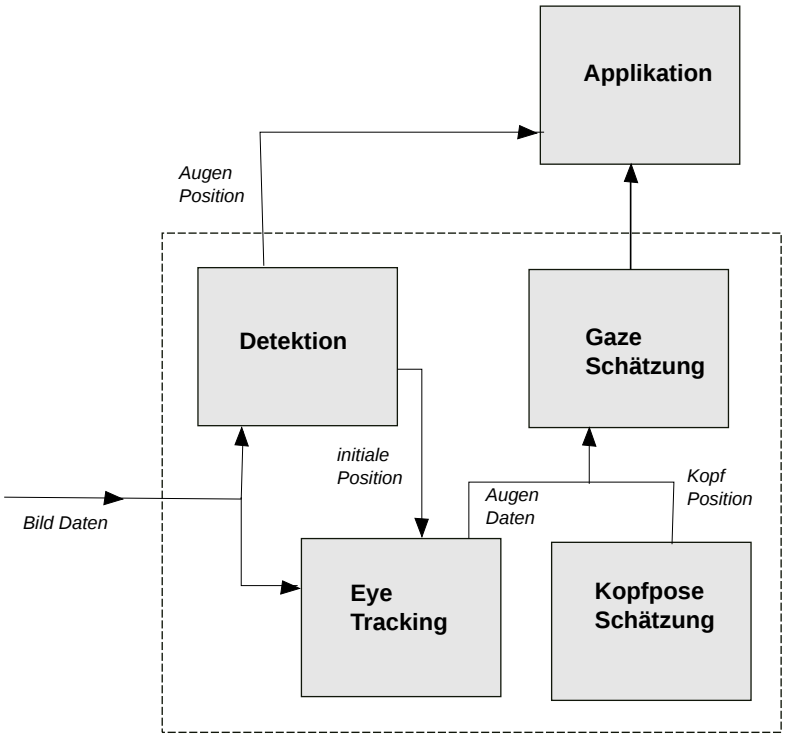


Abbildung 2.3: Komponenten des Video-basierten Gaze- und Eye-Trackings[HJ10]

- Modelbasiert: Solche Verfahren gehen davon aus, dass das *Point of Regard (PoR)* der Schnitt der Visuellen Achse mit einem Objekt in dem Sichtfeld ist. Für diese Methoden sind die Position und Ausrichtung der Kamera(s), Lichtquelle(n) und des Bilschirms oder Objektes mit hoher Genauigkeit nötig.

- Regressionsbasiert: Diese Methode benutzt Polynome, um das *PoR* als eine Funktion des Pupillen-Glint Vektors zu bestimmen.

Kapitel 3

Realisierung

In diesem Kapitel wird die realisierte Pipeline vorgestellt. Außerdem wird die Hardware vorgestellt. Weiter werden die Methoden, die realisiert wurden präsentiert und disskutiert. Für die Bewetung der ausgewählten Methoden wurde ein Framework aufgebaut. Der Aufbau des Frameworks wird auch in diesem Kapitel detailliert erklärt.

3.1 Pipeline

Auf dem Bild 3.1 ist eine Gaze-Tracking Pipeline dargestellt. Das Bild wird aufgenommen. Anhand der Bild Daten werden das Gesicht und die Augen detektiert und lokalisiert. Die Merkmale (Glintzentrum und Pupillenzentrum) werden aus der Augen Region extrahiert. Der Gaze wird Anhand dieser extrahierten Merkmale berechnet.
In dieser Arbeit wurde ein Framework entworfen, das die beschriebene Pipeline realisieren kann. Für den Realisierungsschritt wurden die Bilder, die von Wladimir Krebs zur Verfügung stehen, verwendet.
Das Gesicht und die Augen werden zuerst lokalisiert. Damit wird eine *Region Of Interest* definiert. So können die Augenmerkmale in kleineren ROIs detektiert werden. Die Detektion der Merkmale wird nach verschiedenen Methoden durchgeführt und verglichen. Die berechneten Merkmale sind das Pupillen- und Glintzentrum. Um verschiedene Merkmalsextraktionsmethoden evaluieren zu können, werden die Ergebnisse im XML Format gespeichert. So wird eine Doppelberechnung vermieden. Zum Beispiel stellen wir uns das folgende Scenario vor: X und Y seien zwei Methoden für die Berechnung der Pupillenmitte (Y kann auch eine geänderte Version von X sein). Die Ergebnisse von Methode X sollen mit den Ergebnissen von Methode Y für die Bilderserie 1 verglichen werden. Dafür sollte man die Ergebnisse für X nur einmal berechnen, analog für Y. Nach der Serialisierung sind die

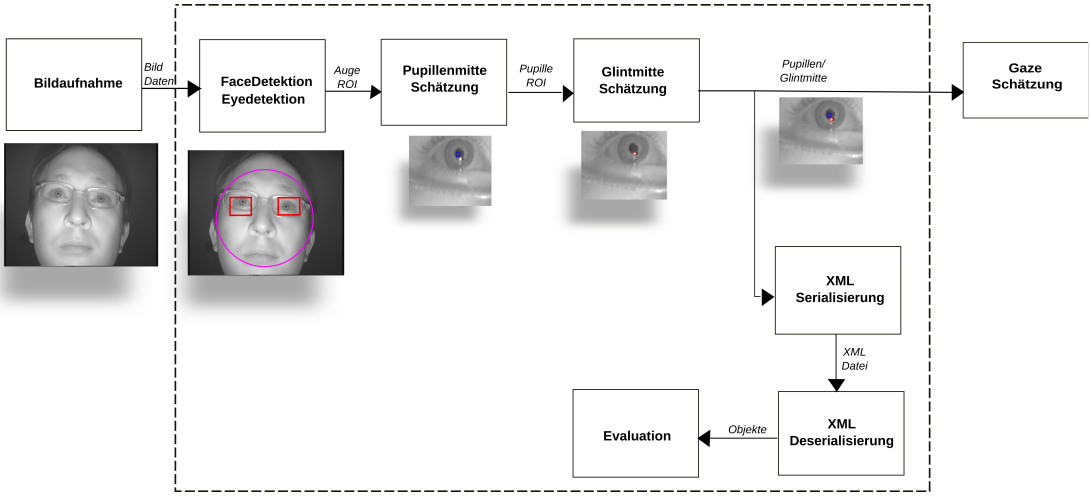


Abbildung 3.1: Die in dieser Arbeit genutzte Gaze-Tracking Pipeline. Die Bilder sind Beispiele für Ergebnisse jedes Schritts. Diese Arbeit befasst sich mit den Teilen, die in dem Rechteck abgebildet sind.

Ergebnisse jederzeit zugreifbar. Die Gazeschätzung wird nicht implementiert, aber es besteht die Möglichkeit, mit minimalen Ergänzungen die gefundenen Merkmale auf den Bildschirm abzubilden und damit einen vollständigen Gaze-Tracker umzusetzen.

3.2 Hardware

Die benutzten Bilder für die Realisierung sind mit einer Kamera mit analogem Ausgang, die zur Bar-Überwachung benutzt wurde, aufgenommen worden. Die Kamera ist extrem Licht empfindlich und lässt nur das Infrarote Licht durch. Es handelt sich so nach Abschnitt 2.1 um eine *aktive* Methode. Die Bilder sind 720 x 576 Pixel groß. Die Kamera ist unter dem Bilschirm angebracht. Die IR Lichtquelle (mehrere Leuchtdioden) ist unter der Kamera installiert. Nach Abschnitt 2.1 haben wir ein *off-axis* Licht. Wenn das beobachtete Objekt der Bildschirm ist, kommt es häufig vor, dass die Augen durch die oberen Lider verdeckt werden. Der Vorteil einer von unten (relativ zu dem Bildschirm) installierten Kamera im Vergleich zu einer oben angebrachten Kamera ist weniger Verdeckung der Augen durch die oberen Augenlider. Durch die IR Lichtquelle gibt es einen höreren Kontrast zwischen der Pupille und Iris. Damit ist der Pupillenrand einfacher zu detektieren.

3.3 Aufbau des Frameworks

Das Framework implementiert die in Abschnitt 3.1 beschriebene Pipeline. Es realisiert folgende Schritte:

- Das Bild lesen:
  Dafür werden *openCV* Funktionen benutzt. Während des Lesens werden alle Frames in einer Serie gelesen und für weitere Verarbeitungen bereit gestellt.

- Das Gesicht und die Augen detektieren:
  Für die Gesichtsdetektion wird der Haar Merkmalbasierte Cascade Classifier von *openCV* benutzt. Für die Augendetektion ist ein Versuch mit *openCV* Methoden gescheitert. Wir haben eine einfache Methode entwickelt, die versucht, die Augenregion relativ zu dem gefundenen Gesichtszentrum zu finden. In Abschnitt 3.4 wird mehr über Gesichts- und Augendetektion erklärt. Der Schwerpunkt dieser Arbeit liegt nicht auf den Methoden für Gesichts- und Augendetektion. Deswegen genügen uns diese Methoden, die befriedigende Ergebnisse liefern.

- Das Pupillenzentrum und Glintzentrum werden mit den ausgewählten Methoden berechnet:
  Die Pupille wird in der AugeROI (geliefert vom letzten Schritt) detektiert und das Zentrum davon berechnet. Danach wird der Glint detektiert und das Zentrum davon berechnet. Um diesen Schtritt zu realisieren, sind verschiedene Methoden implementiert. Für die Berechnung des Pupillenzentrums sind *center of gravity* , *after Daunys* und *ellipse fitting* und für die Schätzung des

Glintzentrums sind *center of gravity* und *polynomial fit* implementiert. Diese Methoden werden in Abschnitt 3.4 vorgestellt.

- Die Ergebnisse im XML Format serialisieren und in eine Datei speichern:
  Jede Klasse besitzt eine Funktion, die für Serialisierung zuständig ist. Das Gesicht kann z.B. seine Attribute im XML Format serialisieren und übergeben. Die XML Struktur besteht aus eine Szene, die aus eine Serie von Frames zusammengesetzt ist. Jeder Frame wiederum beinhaltet Gesichter, jedes Gesicht Augen, Augen wiederum Pupillen und Glints. Außer der Szene und den Frames haben alle Einträge unter anderem eine Nummer und einen Algorithmusname als Attribut. Die Nummer macht es einfacher bestimmen zu können, ob z.B. mehr als ein Gesicht gefunden wurde. Der Name des Algorithmus besteht aus einem String. Der String bezeichnet den Name von dem Algorithmus, der für die Berechnung von dem Eintrag benutzt wurde und die Version von dem Algorithmus. Zum Beispiel "DA10 " steht für Daunys Algorithmus, Version 10.

- Die XML Datei deserialisieren und die Attribute der Objekte für die Evaluation der Methoden benutzen:
  Jede Klasse, die serialisiert wurde (Gesicht, Augen, Pupillen, Glints), ist zuständig für die Deserialisierung der Attributen aus XML Einträgen in die Objekt Attribute. Zum Beispiel kann die Klasse Face die Augen, die ihr gehören, als Objekt wiederherstellen. Nach der Deserialisierung werden die erwünschten Ergebnisse (z.B. Glintzentrums Koordinaten) in eine Text Datei gespeichert.

- Evaluation:
  Die Ergebnisse von dem letzten Schritt sind in Text Dateien gespeichert. Sie werden zum Testen, für statistische Verfahren, für Plotts usw. benutzt. In Kapitel 4 werden die Evaluation und Test Methoden beschrieben.

3.4 Algorithmen

In Abschnitt 3.3 wurde der *openCV* Cascade Classifier erwähnt. Für die Detektion der Gesichter wird ein *machine learning based* Verfahren benutzt. [1] Dafür muss der Klassifikator trainiert werden. Wir benutzen den trainierten Gesichtsklassifikator von *openCV*, der in den Tutorialdaten vorhanden ist. Es wird davon ausgegangen, dass genau ein Gesicht auf dem Bild zu finden ist. Trotzdem findet die benutzte Methode auf manchen Bildern mehr als ein Gesicht. Diese false positives sind

[1] `http://docs.opencv.org/modules/objdetect/doc/cascade\_classification.html?highlight=cascadeclassifier#cascadeclassifier`

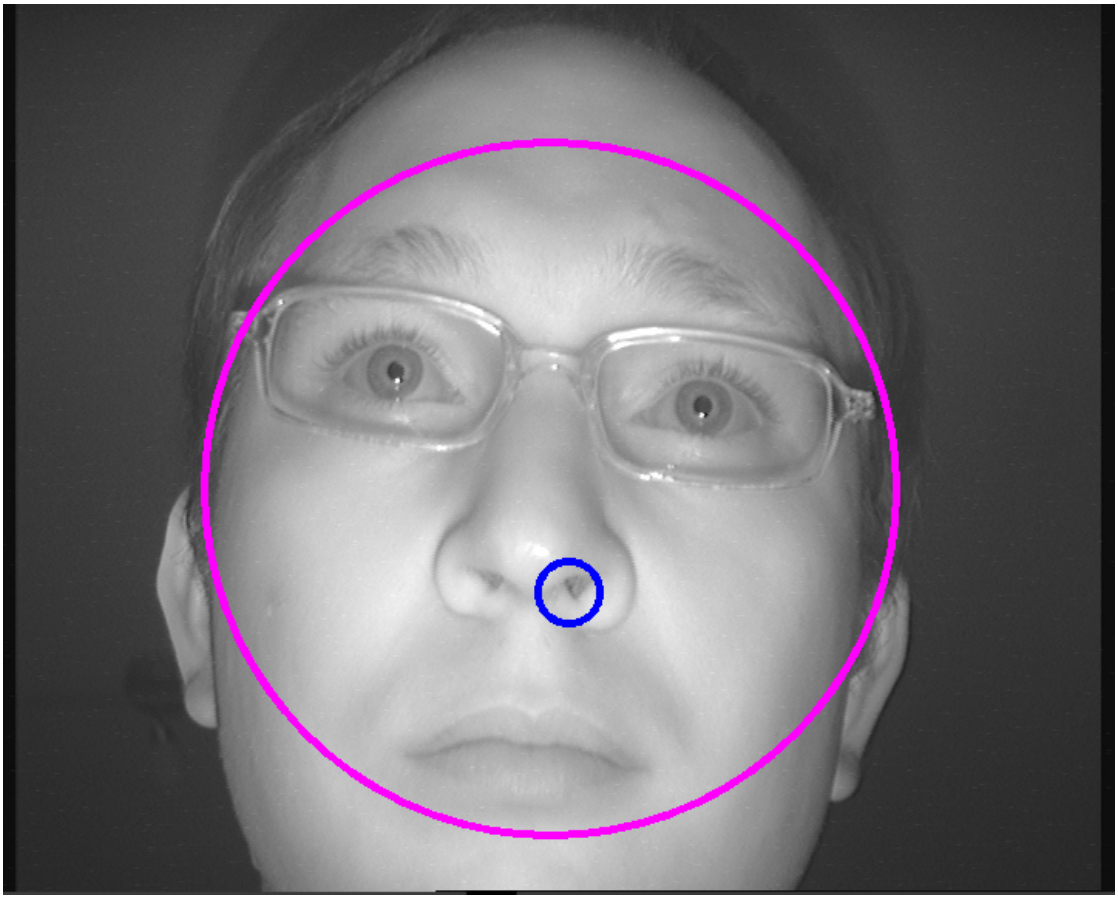


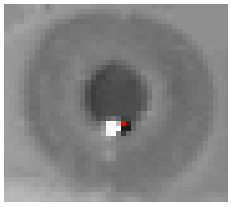
Abbildung 3.2: Die Augendetektion von *openCV* liefert oft solche Ergebnisse

bei unseren Bildserien klein und im Hintergrund. Um diese falschen Gesichter zu vermeiden, eliminieren wir diejenigen Gesichter, deren Größe eine geforderte Gesichtsgröße unterschreitet. Die geforderte Größe ist empirisch ermittelt.

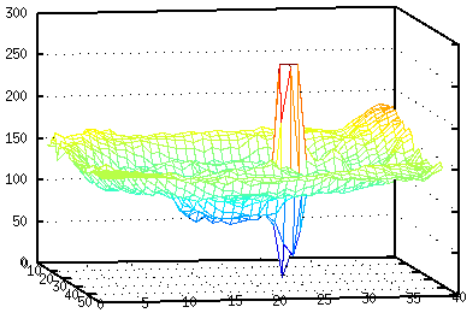
Wir haben versucht den *openCV* Klassifikator für die Augendetektion zu benutzen. Die Ergebnisse waren nicht zufriedenstellend. Ein Auge oder beide Augen wurden in vielen Frames nicht detektiert. Häufig wurde das Nasenloch stattdessen in diesen Frames als Auge detektiert (siehe Bild 3.2).

Deswegen haben wir eine einfache Methode entwickelt, um die Augenregionen zu finden. Es wird davon ausgegangen, dass die Augen sich auf einer bestimmten Position relativ zur Gesischtsmitte befinden. Um dieser Position wird eine Auge-ROI definiert und in den nachfolgend Schritten benutzt. Über die Größe der Region wird später diskutiert.

Für die Berechnung des Pupillenzentrums werden im Folgenden beschriebene Algorithmen benutzt:



(a) Eine Ausschnitt von AugeROI. Das mit rot übermalte Pixel bezeichnet den niedrigste Wert auf der Pupille



(b) Das Histrogramm zur AugeROI

Abbildung 3.3: Die Verteilung der Werte in der AugeROI

Zuerst stellen wir den *center of gravity* vor. Der ist ein Merkmalsbasiertes Verfahren und kann das Zentrum der Pupille subpixelgenau bestimmen. Die AugeROI wird zuerst binarisiert, so dass die Pixel in zwei Kategorien fallen: Pupille und nicht-Pupille. Der passende Schwellwert für die Binarisierung sollte wohlüberlegt werden. Das erste was einem einfällt ist, den niedrigsten Wert plus ein Offset als Schwellwert zu verwenden. Wie in Bild 3.3 zu sehen ist, befindet sich der niedrigsten Wert auf der Pupille nicht in dem Zentrum, sondern direkt neben dem Glint. Auf dem Bild wurde dieses Pixel mit Rot übermalt. Das Pixel entspricht dem niedrigsten Wert auf dem Histogramm. Mit der oben genannten Auswahl, basierend auf dem niedrigsten Wert, ist es möglich, dass das Pupillenzentrum nicht als Pupille kategorisiert wird.

Wenn das Offset klein ist, werden Teile von der Pupille nicht erwischt. Wenn das Offset groß ist, werden Teile von der Augenregion, die nicht zur Pupille gehören (z.B. Iris), als Pupille kategorisiert. Versuche zeigten, dass die empirische Auswahl den Offset in unterschiedlichen Frames zu unterschiedlich akzeptabelen Ergebnissen führt. Anders ausgedrückt, der oben beschriebene Schwellwert ist nicht allgemein gültig. Alternativ betrachten wir das kommulative Histogramm und wählen den zehnten Wert (in Abhängigkeit der Größe der Pupille in Pixeln) als Basis für den Schwellwert. Damit stellen wir sicher, dass mindestens die 10 dunkelsten Pi-

xel als Pupille kategorisiert werden. Zu dem genannten Wert wird ein empirischer Offset von 20 Prozent (diese Zahl kann sich für verschiedene Bildserien ändern) addiert.

Unabhängig davon, wie der Schwellwert gewählt wird, werden Teile von der Region falsch als Pupille binarisiert. Der Grund dafür ist, dass manche Teile wie z.B. Wimpern den gleichen Wert wie manchen Pupillenpixel haben. Manche der so entstehenden Segmente besitzen einen kleinen Umfang. In diesem Fall reicht ein einfaches *opening* um diese zu eliminieren. Aber wenn die falsch segmentierte Region groß ist, wird das *opening* nicht helfen. Deswegen passen wir die Augenregion, die im letzten Schritt gefunden wurde an, indem wir eine kleinere AugenROI definieren. So wird verhindert, dass störende Effekte wie die Wimpern mit in der ROI liegen. Die Größe der AugenROI wird empirisch für verschiedene Bilderserien gewählt. Größere AugeROIs werden besonders bei den synthetischen Bildern, die in Kapitel 4 vorgestellt werden, Probleme verursachen. Es ist klar, dass kleinere AugeROIs das Risiko erhöhen, dass die Pupille nicht ganz in der Region liegt. Besonders in Bilderserien, in denen die Bewegungen des Kopfes beträchtlich sind. In Abbildung 3.4 ist ein Beispiel für die genannten Probleme und deren Lösung zu sehen. So werden diese kleine Segmente, die nicht zur Pupille gehören, eliminiert und die Pupille kreisförmiger dargestellt.

Weiter wird der Mittelwert im Pupillenbereich jeweils in x und y Richtung berechnet (Siehe Gleichung 3.1). So wird das Pupillenzentrum bestimmt.

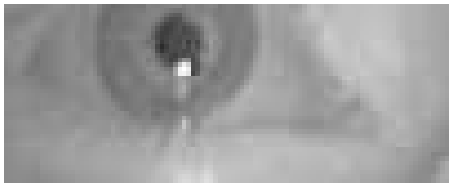
$$\begin{aligned} \mathbf{x_c} &= \frac{\mathbf{1}}{\mathbf{n}} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{x_i} \\ \mathbf{y_c} &= \frac{\mathbf{1}}{\mathbf{n}} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{y_i} \\ (x_i, y_i) &\in P \end{aligned} \tag{3.1}$$

In Kapitel 2 wurde die Methode nach Daunys und Ramanauskas[DR04] erwähnt. Wir benutzen ihre Methode um das Zentrum der Pupille zu berechnen und nennen den Algorithmus *after Daunys* . Zuerst muss der Pupillenrand gefunden werden. In dem Paper werden *Scan Linien* erwähnt. Eine *Scan Linie* ist eine gedachte Linie, die vertikal oder horizontal auf der Pupille liegt und den Rand der Pupille in 2 Punkten schneidet.

Unabhängig von dem angewendeten Algorithmus, sollen Randpunkte links, rechts, oben und unten auf der Pupille gefunden werden. Es ist wichtig, dass die Randpunkte symmetrisch sind. D.h. es muss für jedes Randpixel links ein assoziiertes Randpixel rechts geben. Analog gilt die Bedingung für Randpixel oben und unten. Anders ausgedrückt: jede *Scan Linie* muss die Pupille in beiden Richtungen entlang der Linie schneiden (siehe Abbildungen 3.5). Nachdem der Rand

(a) Augenregion definiert im Gesichts- und Augen-detektionsschritt

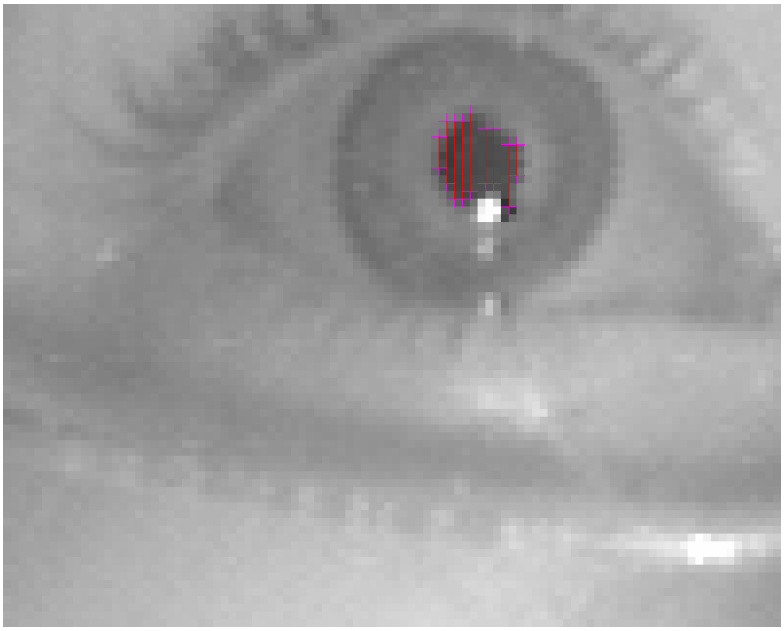
(b) Die Segmente in der Region, die gleiche Werte wie der Pupillenpixel besitzen, werden als Pupille segmentiert. Wenn diese groß sind, werden sie nicht durch *opening* eliminiert.



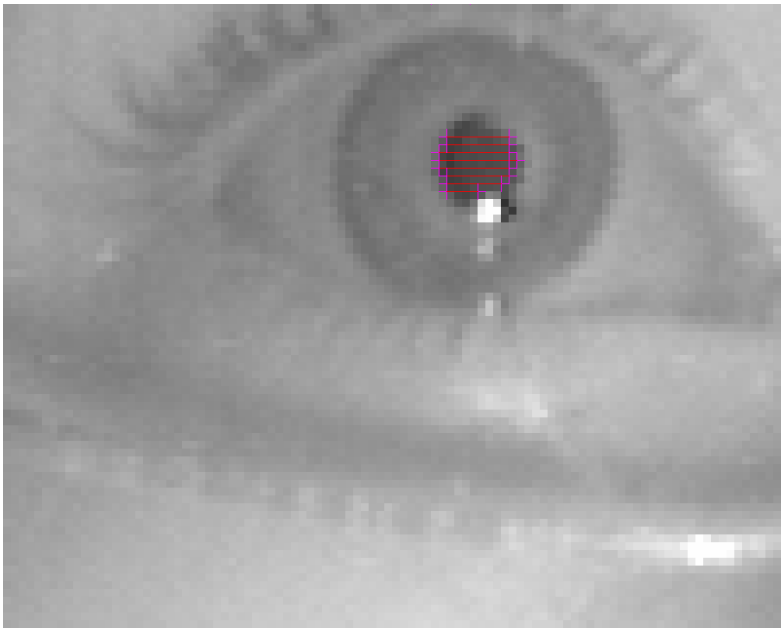
(c) Die AugeROI wird verkleintert, um unerwünschte Regionen auszuschließen.

(d) Unerwünschte Regionen wurden eliminiert.

Abbildung 3.4: Eliminieren der Region, die nicht zur Pupille gehört durch die Definition einer kleineren Region.

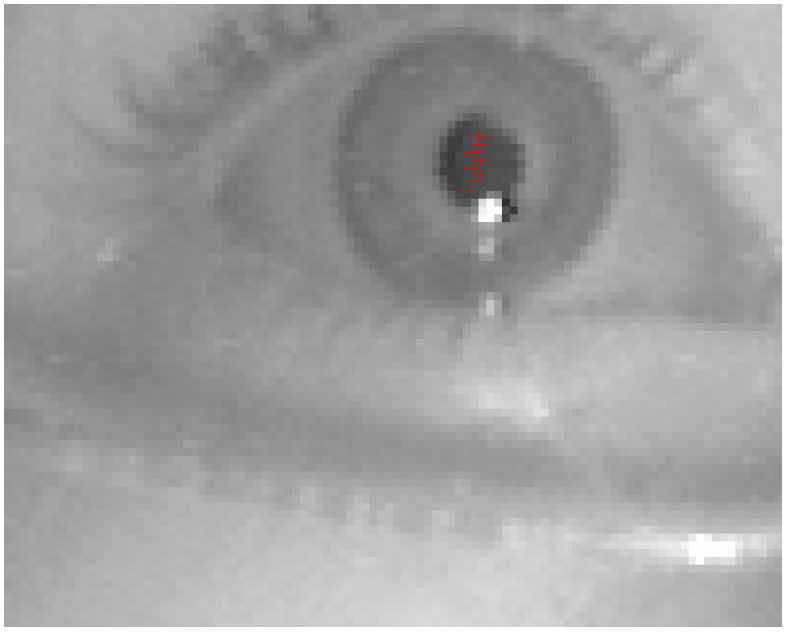


(a) Vertikale Scan Linien

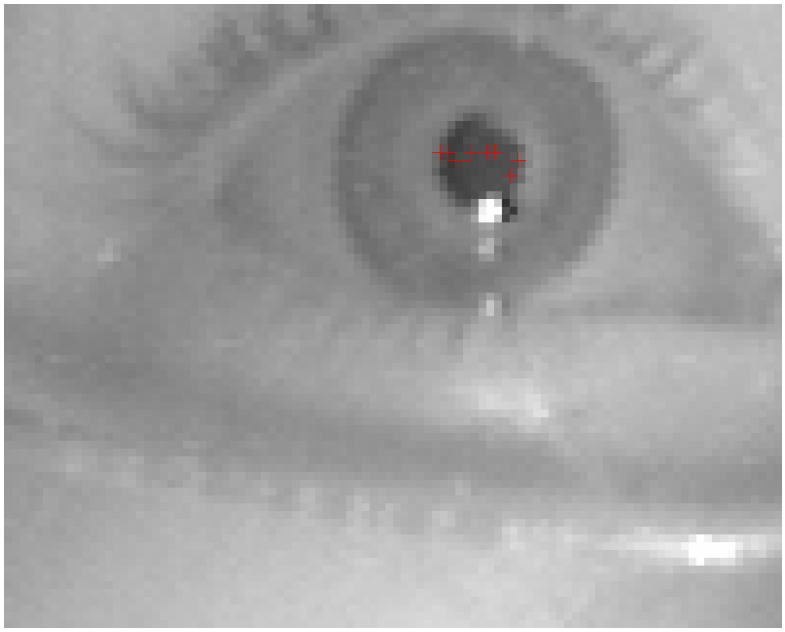


(b) Horizontale Scan Linien

Abbildung 3.5: Die gefundenen Randpunkte sind Paare



(a) Vertikale Mittelpunkte



(b) Horizontale Mittelpunkte

Abbildung 3.6: Die Mittelwerte von Koordinaten in horizontale und vertikale Richtungen

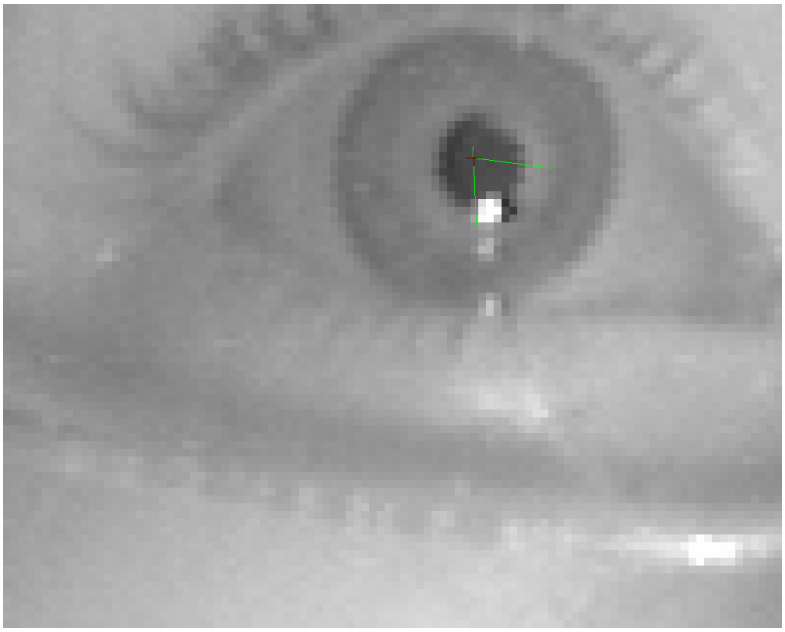


Abbildung 3.7: Die entstehende horizontale und vertikale Linie und ihr Schnittpunkt

gefunden wurde, werden die Mittelwerte der Koordinaten der zwei Randpixel auf der horizontalen *Scan Linie* berechnet. Es wird eine Linie durch die Mittelwerte gefittet und so entsteht eine Vertikale Linie in der Mitte von der Pupille. Für horizontale *Scan Linien* wird das gleiche Verfahren analog durchgeführt. Jetzt genügt es, die beiden Mittellinien schneiden zu lassen und damit das Pupillenzentrum zu bestimmen.

Für das Fitten der Linien wird die *openCV* Funktion *fitLine* benutzt. Die Funktion versucht mit dem M-estimator eine Linie zu fitten. Es muss durch einen Parameter bestimmt werden, welches Abstandsmaß benutzt werden soll. Wir haben uns aufgrund guter Ergebnisse für den Abstand $\rho(r) = 2(\sqrt{1 + \frac{r^2}{2}} - 1)$ entschieden.

Abbildung 3.8 zeigt, dass der Glint ein Teil von der Pupille verdeckt. Dadurch entstehen falsche Randpixel. Dies verursacht eine Verschiebung der Linien nach oben und links (siehe Bild 3.7). Daunys et al.[DR04] haben in ihrem Paper mit Bildern ohne Glint gearbeitet. Dieses Problem existiert für alle Methoden, die die Randpixel für weitere Berechnungen benutzen. Folgend werden wir die *ellipse fitting* Methode vorstellen und das Problem nochmal angehen.

3.4.1 Ellipsenfitting

Eine weitere Methode für die Berechnung des Pupillenzentrums ist *ellipse fitting* . Nachdem wir den Rand berechnet haben, versuchen wir eine Ellipse durch die

Randpixel zu fitten. Es muss berücksichtigt werden, dass die Randpixel nicht wie bei *after Daunys* symmetrisch sein müssen. Es kann ein Pixel links existieren, ohne dass das assoziierte Pixel rechts zur Verfügung steht. Die Methode für Ellipsenfitting ist schnell und genau genug, und damit für das echtzeite Gaze-Tracking geeignet. Im Folgenden werden wir zwei Ellipsenfitting Methoden vorstellen: Die verbesserte Fitzgibbon[FF95] Methode und RANSAC.

openCV stellt eine Funktion für Ellipsenfitting zur Verfügung. Sie ist eine Implementation von *Direct Least Squares Fitting of Ellipse* nach Fitzgibbon et al.[FF95]. Es gibt inzwischen Änderungen in der Implementation, die die Nachteile der ersten Version auszugleichen versuchen. Wir werden diese in einem späteren Absatz diskutieren. Gleichung 3.3 zeigt einen Kegel. $F(x,y) = ax^2 + bxy + cy^2 + dx + ey + f = 0$
oder in Vektorenform:

$$\mathbf{F_a}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{a} = \mathbf{0} \tag{3.2}$$

$$\mathbf{a} = \begin{pmatrix} a & b & c & d & e & f \end{pmatrix}^\top, \quad \mathbf{x} = \begin{pmatrix} x^2 & xy & y^2 & x & y & 1 \end{pmatrix}^\top \tag{3.3}$$

Die algebraische Distanz von einem Punkt $\mathbf{p_i} = (\mathbf{x_i}, \mathbf{y_i})$ von dem Kegel $\mathbf{F_a}(\mathbf{x}) = \mathbf{0}$ ist $\mathbf{F_a}(\mathbf{p_i})$. Um eine Ellipse zu fitten muss man die Gleichtung3.4 lösen, mit der Bedingung $(b^2 - 4ac < 0)$.

$$\min_{\mathbf{a}}(\sum_{i=1}^{N}(\mathbf{F_a}(\mathbf{p_i})) = \min_{\mathbf{a}}(\sum_{i=1}^{N}(\mathbf{p_i}\cdot\mathbf{a})^2) \tag{3.4}$$

Das Minimierungsproblem 3.4 mit der genannten Bedingung ist schwer zu lösen. Es sollte in Betracht gezogen werden, dass für jede $\alpha \neq 0$, $\alpha \cdot \mathbf{F_a}(\mathbf{x}) = \alpha \cdot \mathbf{x} \cdot \mathbf{a} = 0$ der gleiche Kegel wie 3.3 ist. D.h die Koeffezienten von dem Kegel dürfen skaliert werden. Damit ändert sich die Bedingung zu $b^2 - 4ac = 1$ oder in Vektorenform:

$$\mathbf{a}^\top \cdot \mathbf{C} \cdot \mathbf{a} = 1$$

$$\mathbf{C} = \begin{pmatrix} 0 & 0 & +2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ +2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.5}$$

C ist die *constraint* Matrix. Nach Bookstein [Boo79] und Gleichung 3.3 wird das Problem zu dem folgenden Minimierungsproblem reduziert:

$$\begin{aligned} &\min \|\mathbf{D} \cdot \mathbf{a}\|^2 \\ &\mathbf{a}^\top \cdot \mathbf{C} \cdot \mathbf{a} = 1 \end{aligned} \tag{3.6}$$

D ist die *design* Matrix.

$$\mathbf{D} = \begin{pmatrix} x_1^2 & x_1 y_1 & y_1^2 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N^2 & x_N y_N & y_N^2 & x_N & y_N & 1 \end{pmatrix} \tag{3.7}$$

Die Gleichungen werden umgeschrieben[2]:

$$\begin{aligned} &\mathbf{S}\mathbf{a} = \lambda \mathbf{C}\mathbf{a} \\ &\mathbf{a}^\top \cdot \mathbf{C} \cdot \mathbf{a} = \mathbf{1} \\ &\mathbf{S} = \mathbf{D}^\top \mathbf{D} \end{aligned} \tag{3.8}$$

$\mathbf{S} = \mathbf{D}^\top \mathbf{D}$ ist die *scatter* Matrix. Die 3.8 Gleichung ist mit *generalized Eigenwert* der ersten Gleichung und anwendung der zweiten Gleichung zu lösen. Nach Fitzgibbon et al.[FF95] existiert nur eine ellipsenförmige Lösung zu der Gleichung 3.4 mit der Bedingung 3.5. Maini [Mai05] argumentiert in seinem Paper, dass die beschriebene Methode für Ellipsenfitting zwei wesentliche Nachteile hat:

Erstens sind die Ergebnisse numerisch instabil. Dies zeigt sich besonders wenn den Code nicht in MATLAB geschrieben wird. Da z.B. ANSI C nur kleinere floating-point Zahlen als MATLAB erlaubt.

Zweitens: In manchen Fällen ist die Lokalisierung der optimalen Lösung mehrdeutig oder unmöglich. Wie gesagt behaupten Fitzgibbon et al. in ihrem Paper, dass nur eine ellipsenförmige Lösung zu dem Problem existiert. Die Behauptung ist nicht immer wahr. Vorallem wenn die Punkte exakt auf eine Ellipse liegen, liefert die Methode keine Lösung. Wegen der nummerischen Instabilität kommt diese Situation auch vor, wenn die Punkte fast auf einer Ellipse liegen.

Nachdem in dem Paper die Nachteile genannt werden, stellt Maini [Mai05] eine vebesserte Methode, nämlich *enhanced direct least-square fitting of ellipse (EDFE)*. Die zu fittende Punkte werden nochmal zentriert und skaliert. So liegen die Punkte in einem Quadrat (Länge 2, Zentrum=(0,0)). Die Gleichung 3.3 sieht jetzt so aus:

$$\mathbf{F}_{\hat{\mathbf{a}}} = \hat{\mathbf{x}} \cdot \hat{\mathbf{a}} = 0 \tag{3.9}$$

[2]Für detaillierte umrechnung von den Gleichungen und Matlab Code siehe `http://research.microsoft.com/en-us/um/people/awf/ellipse/`

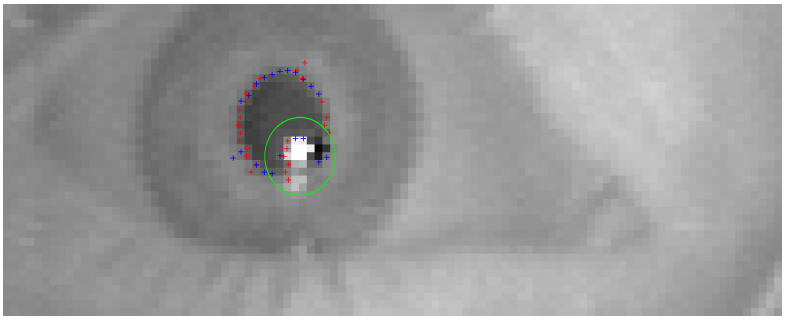


Abbildung 3.8: Durch die Verdeckung des Glints entstehen falsche Randpunkte (in dem grünen Kreis).

Nachdem die Lösung gefunden wird, müssen die Ergebnisparameter denormalisiert werden. Für das zweite Problem mit den Punkten, die exakt auf einer Ellipse liegen, hat Maini [Mai05] die folgende Lösung präsentriert: Nach dem die Punkte normalisiert werden sollte es überprüft werden, ob der Eigenvektor lokalisiert werden kann. Wenn nicht, wird bekanntes gaussches Rauschen auf die Punkte angewendet. Das Fitting wird danach durchgeführt. Die Verwendung vom Rauschen und Fitting danach wird M mal wiederholt. So entstehen M Ellipsen als Ergebnis. Das finale Ergebnis ist der Mittelwert der Parameter der entstandenen Ellipsen.

Das zuletzt erwähnte Verfahren wurde aber wegen dem Rechenaufwand nur wenn nötig empfohlen. Also nur, wenn es unmöglich ist eine Lösung zu finden, oder die Lösung numerisch instabil ist. In dieser Arbeit sind wir davon ausgegangen, dass die Randpixel der Pupille ungenau genug sind. Deswegen halten wir es für unnötig, die Bilder zu verrauschen.

Die *ellipse fitting* Funktion, die *openCV* zur Verfügung stellt basiert auf der ersten Version von Fitzgibbon et al.[FF95]. Die Nachteile wurden nicht ausgeglichen. Aber Fitzgibbon et al.[FF99] haben in neuere Implementationen versucht, die Probleme der älteren Versionen zu beheben.[3] Die erwähnte *ellipse fitting* Methode ist nicht iterativ und daher schnell. Die *Least Squares Fitting* Methode versucht im Allgemeinen die beste Form durch alle Punkte zu fitten. D.h. die Punkte, die durch schlechte Randbestimmung als Ausreißer bezeichnet werden, sind immernoch an dem Fitting Verfahren beteiligt. Dies ist in unserem Fall besonders nachteilig, weil das Glint auf unseren Bilder ein Teil des Pupillenrands verdeckt. Abbilgun 3.8zeigt das Problem, das durch dem Existenz des Glints entsteht.

Das Eliminieren und Erstezen des Glints mit einem ähnlichen Farbpixel wie von der Pupille ist keine Lösung. Dadurch bekommen wir nur einen neuen falschen Rand. Die Randpixel, die sich im Nachbarschaft des Glints befinden zu ignorieren,

[3]Die Änderungen von dem Algorithmus sind nicht in dem Paper sondern in dem neuen Matlab Code zu sehen. Siehe `http://research.microsoft.com/en-us/um/people/awf/ellipse/`

ist auch keine gute Idee. Damit werden die Randpixel, die die *ellipse fitting* Methode übergeben werden schlecht verteilt sein und zu ungenaue Ergebnisse führen.

Wir haben in dieser Arbeit noch eine Methode für die Berechnung des Pupillenzentums im Betracht gezogen. Die Methode basiert auf dem *random sample consensus (RANSAC)*. Diese wieder versucht das Zenrum der Pupille durch fitten einer Ellipse durch die Randpixel zu finden. Nach Hartley und Zisserman [HZ03] definieren 5 Punkte eine Ellipse. Die Gleichung für die Ellipse ist $F(x, y) = ax^2 + bxy + cy^2 + dx + ey + f = 0$. Das bedeuted, dass die Größe des *Subsets* 5 sein muss. Es wird für 5 zufällige Punkte aus der Menge der Randpixel die passende Ellipse gefunden. Die Abstände der anderen Punkte zu der gefundenen Ellipse werden berechnet. Dafür wird die algebraische Distanz als Abstandsmaß benutzt. Ein empirisch ermittelten Schwellwert entscheidet, ob ein Punkt zu der Ellipse gehört oder nicht. Dadurch werden *inlier* und *outlier* bestimmt. Das Verfahren wird nach eine bestimmten Anzahl an Iterationen beendet. Die Ellipse mit dem besten *inlier/outlier* Verhältnis ist das finale Ergebnis. Die Anzahl der nötige Wiederholungen in unserem Fall kann mit der Gleichung 3.10 berechnet werden. Dabei ist r die Wahrscheinlichkeit des korrekten Ergebnisses und w die Wahrscheinlichkeit, dass ein gewählter Punkt *inlier* ist.

$$I = \frac{\log(1 - r)}{\log(1 - w^5)} \tag{3.10}$$

So müssen für r=0,99 und w=0,6 zum Beispiel I=56,889 Wiederholungen durchgeführt werden.

3.4.2 Randberechnung

Um den Rand der Pupille zu berechnen, haben wir drei verschiedene Methoden angewendet.

- *contrast difference*: Die Pupille ist dunkler als die Iris. Wir suchen nach dem Übergang von der Pupille zur Iris. Es wird von einem Startpunkt angefangen und der Wert von dem aktuellen Pixel mit den Werten des ersten und zweiten Nachbars verglichen. Wenn beide Differenzen größer als ein Schwellwert sind, wird das aktuelle Pixel als Randpixel bezeichnet. Es muss darauf geachtet weren, dass der Startpunkt innerhalb der Pupille liegen muss. Als Startpunkt kann zum Beispiel das Ergebnis der *center of gravity* Methode gewählt werden. Die Suche in eine Richtung ist beendet, sobald ein Randpixel gefunden wird oder der Suchebereich überschrittet wird. Der Suchebereich sollte groß genug gewählt werden, damit der Pupillenrand abdegeckt wird aber nicht zu groß, so dass die Suche in irrelevantem Bereich weiterläuft. Die Auswahl des

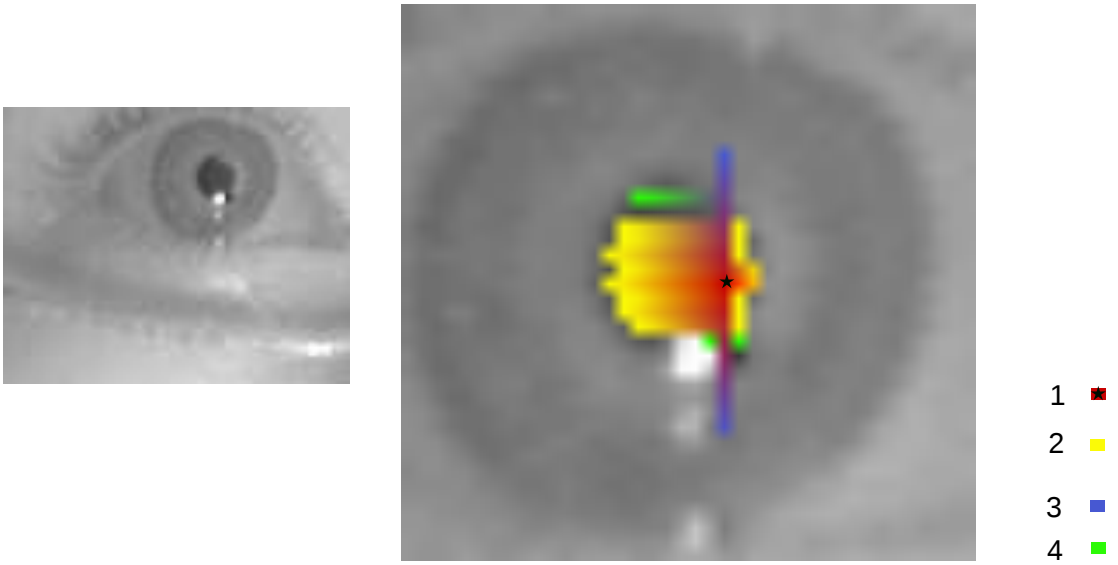


Abbildung 3.9: Links Beipiel für eine AugeROI und rechts eine Ausschnitt davon. 1 ist das erste Startpixel, 2 das gefundene Randpixel, 3 bezeichnet das letzte Startpixel in jeder Richtung und 4 das nicht gültige Randpixel. Von 1 fängt die Suche an. Es wird von dem aktuellen Startpunkt nach dem Randpixel gesucht, bis ein Randpixel gefunden wird oder die gesetzte Grenze (hier 1/8 der Länge bzw. Weite des AugeROIs) erreicht wird. Der Farbverlauf von 1 bis 2 zeigt die Suche von dem aktuellen Startpixel bis zum gefundenen Randpixel. Der benutzte Schwellwert ist hier 20. D.h. der Differenz zwischen jedem Pixel und seinem ersten und zweiten Nachbarpixel wird mit 20 verglichen. Das Startpixel bewegt sich nach oben und unten. So werden die linke und rechte Ränder gefunden. Ungültige Randpixel sind diejenigen, die keine assoziierten Randpixel in der anderen Richtung haben oder diejenigen, die zu nah aneinander sind. Analog ist die Suche in der vertikalen Richtung. So werden die unteren und oberen Ränder auch bestimmt.

Schwellwerts ist auch ausschlaggebend. Abbildung 3.9 zeigt das Verfahren. Mit dieser Methode können wir pixelgenaue Randpixel berechnen.

- *cubic fitting*: Wie in der Abbildung 3.11 zu sehen ist, hat der Helligkeitsverlauf in einer Bildzeile an dem Pupillenrand die Form eines Polynoms des dritten Grades. Dies ist in den Spalten analog zu sehen. Es wird ähnlich wie bei *contrast difference* von einem Startpunkt angefangen und in dem Suchbereich nach einem Sprung des Farbwertes gesucht. Damit wird es klar, ob wir uns in der Nähe von dem Pupillenrand befinden. Sobald einen Sprung gefunden wird, wird eine Gruppe von 5 benachbarten Pixel mit dem gefundenen Pixel als Zentrum gebildet. Danach wird versucht, mit der Gruppe ein Polynom mit der Gleichung $ax^3 + bx^2 + cx + d = 0$ aufzustellen. Um die

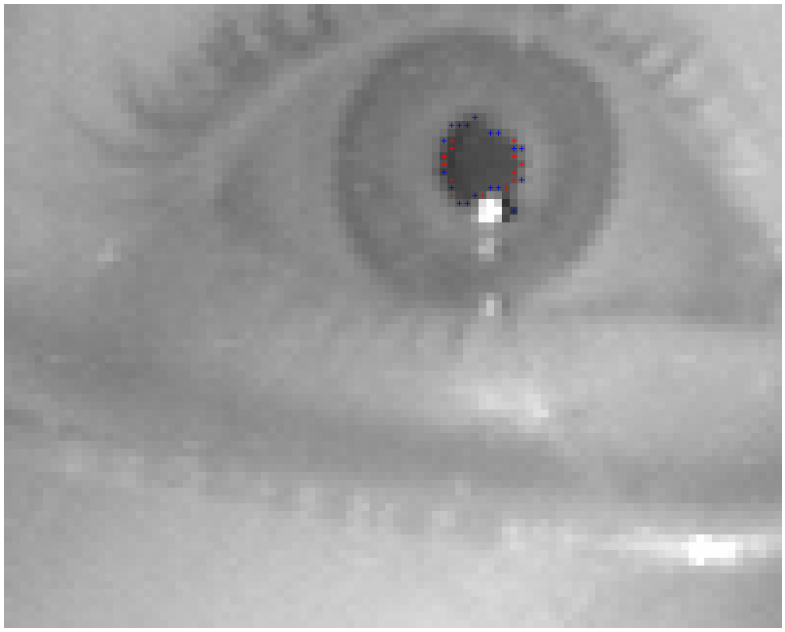


Abbildung 3.10: Ein Besipiel für den gefundenen Pupillenrand mit *contrast difference*. Die blaue Kreuze zeigen auf horizontalen und rote auf vertikalen Rand

Koeffezienten des Polynoms zu bestimmen stellen wir eine überbestimmte Gleichung mit 5 benachbarten Pixel auf und lösen die mit QR-Zerlegung. Der Wendepunkt des Polynomes ist das Randpixel (subpixelgenau). Er entspricht dem Nulldurchgang der zweite Ableitung oder $x_0 = -b/3a$. Um die degenerierten Konfigurationen auszuschließen überprüfen wir, ob das so gefundene x_0 sich zwischen dem 2-ten und 4-ten Pixel der 5 benachbarten Gruppenpixeln befindet. Wenn das nicht der Fall ist wird die 5-er Gruppe ignoriert und weiter gesucht. Der Startpunkt wird für die nächste Suche aktualisiert. Abbildung 3.13 zeigt zwei Gruppen am linken und rechten Rand der Pixel und die dazu passende kubische Funktion. So wird die Suche in einem bestimmten Radius nach oben, unten, links und rechts ausgeführt. Als erste Startpunkt wird das Ergebnis von *center of gravity* benutzt. Der Suchbereich wird empirisch gewählt.

3.4.3 Glintberechnung

Im Weiteren werden Algorithmen für die Berechnung des Glintzentrums vorgestellt. Der Erste Algorithmus ist ähnlich wie *center of gravity* für die Berechnung des Pupillenzentrums. Zuerst wird das Maximum in der ROI gesucht. Als Schwellwert der Binasierung wird 0.8 des Maximums angewendet. Die Zahl 0.8 ist empirisch ermittelt. Anders als die Pupille, befindet sich das Maximum des Glints in dem

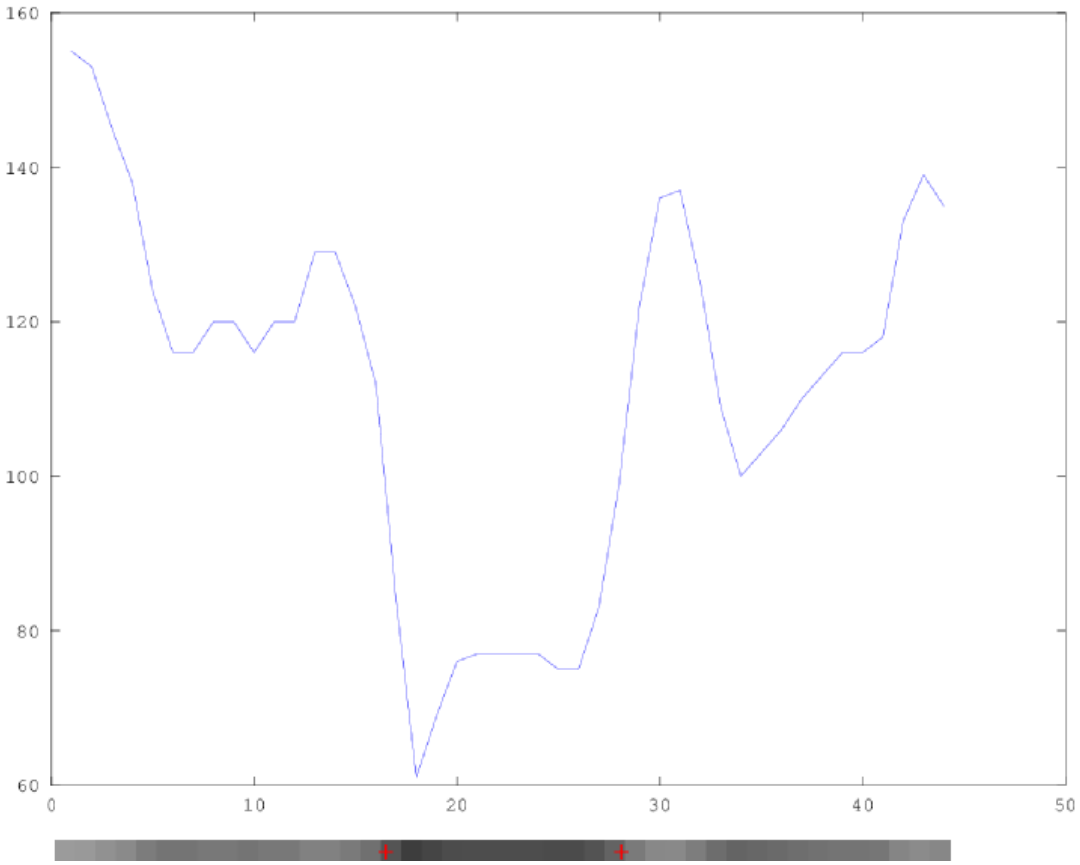
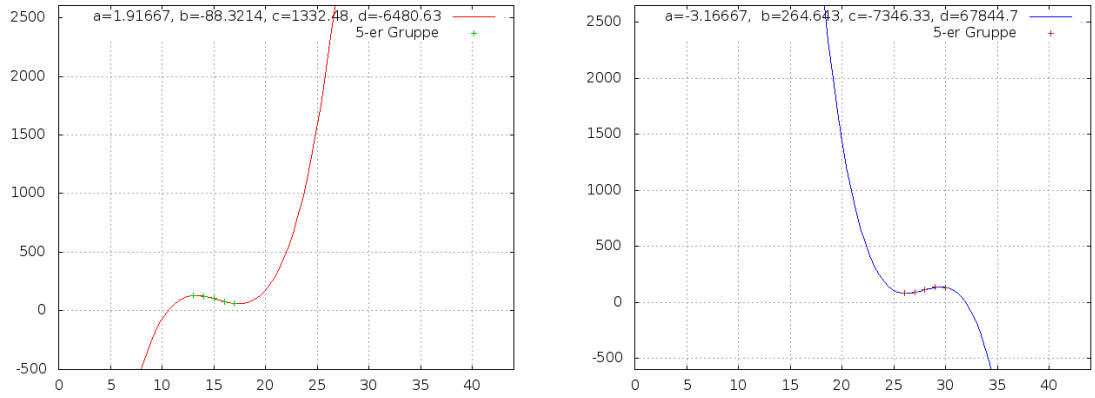


Abbildung 3.11: Eine Zeile aus dem Auge und das dazu entsprechende Histogramm. Die roten Kreuze zeigen die Randpixel



(a) Das gefundene Randpixel links **(b)** Das gefundene Randpixel rechts

Abbildung 3.12: Die 5-er Gruppen der benachbarten Pixel und dadurch gefittete Polynome

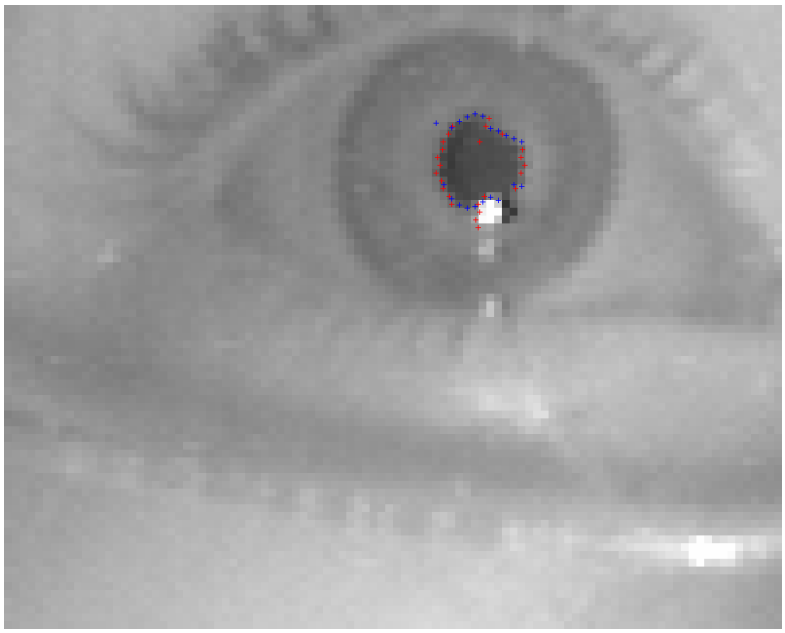


Abbildung 3.13: Ein Besipiel für den gefundenen Pupillenrand mit *cubic fitting*. Die blauen Kreuze zeigen auf den horizontalen und rote auf den vertikalen Rand

Zentrum des Glints. So ist eine Prozentzahl des Maximums der Region als Schwellwert zweckmäßig. Nach der Binarisierung ist die Region in Glint und nicht-Glint aufgeteilt. Dann wird der Mittelwert von X- und Y-Koordinaten der Glintpixel berechnet. Die berechnete Koordinaten sind die Koordinaten des Glintzentrums. Gleichung 3.13 fasst die Berechnung vom Schwerpunkt der Glintpixel zusammen. G ist die Menge der Glintpixel. Der beschriebene Algorithmus liefert subpixelgenaue Koordinaten und ist ein Merkmal-Basiertes Verfahren.

Wie auf dem Bild 3.15 zu sehen ist, haben die Werte der Glintnachbarpixel eine inverse Beziehung mit dem Abstand zu dem Glintzentrum. Deswegen kann die einfache *center of gravity* mit Gewichten geändert werden. So wird in Gleichung 3.13 die Koordinate des Glintpixels mit dem entsprechenden Wert in dem nicht-binarisiertem Bild gewichtet. Siehe Gleichung 3.16. Diesen neuen Algorithmus nennen wir *weighted center of gravity* .

$$\mathbf{x_c} = \frac{\mathbf{1}}{\mathbf{n}} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{x_i} \tag{3.11}$$

$$\mathbf{y_c} = \frac{\mathbf{1}}{\mathbf{n}} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{y_i} \tag{3.12}$$

$$(x_i, y_i) \in G \tag{3.13}$$

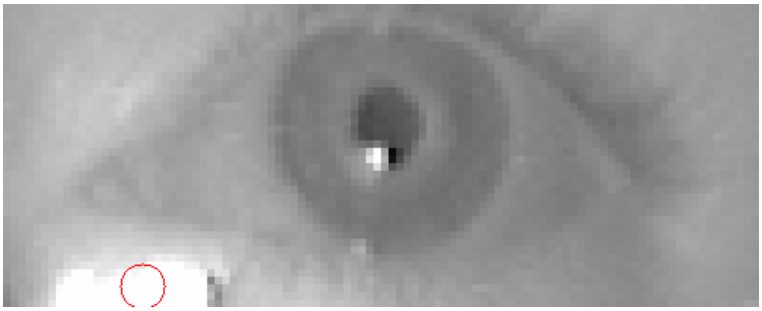
$$\mathbf{x_c} = \frac{\mathbf{1}}{\sum\limits_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{I(x_i, y_i)}} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{I(x_i, y_i) x_i} \tag{3.14}$$

$$\mathbf{y_c} = \frac{\mathbf{1}}{\sum\limits_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{I(x_i, y_i)}} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{I(x_i, y_i) y_i} \tag{3.15}$$

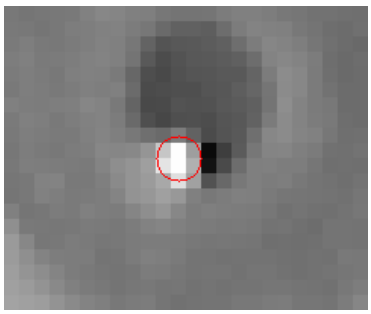
$$(x_i, y_i) \in G \tag{3.16}$$

Auf manchen Bildern sind Reflexionen vom Gesicht oder der Brille zu sehen. Diese Reflexionen können, wenn sie in der AugeROI sind, die Glintdetektion beeinträchtigen. Um dies möglichst zu vermeiden, definieren wir eine kleinere Region für die Detektion des Glintes. Bild 3.14 zeigt der Effekt der Reflexion und wie die kleinere Region dieses Problem behoben hat.

Weiter kann das Glint mit dem folgenden Algorithmus berechnet werden. Den Algorithmus nennen wir *polynomial fit* . Wie in der Abbildung 3.15.b zu sehen ist, können auf dem Histogramm der Glintregion Polynome zweiten Grades gefittet werden. So wird ein neuer Algorithmus basierend auf *polynomial fitting* entwickelt.

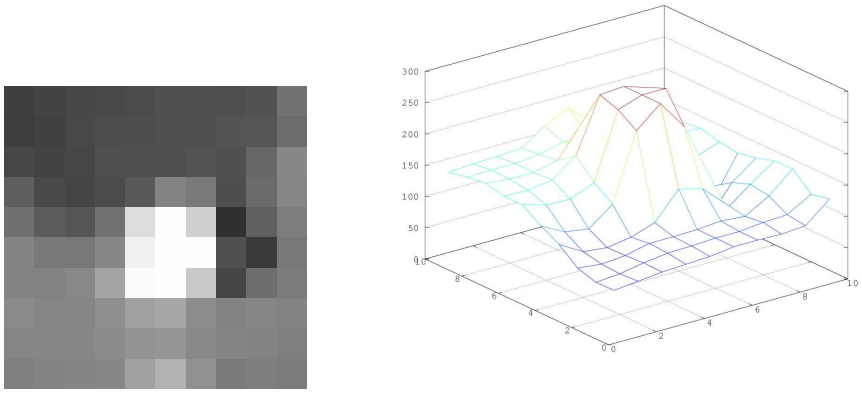


(a) Die Reflexionen in der Region ziehen den Schwerpunkt zu sich



(b) Die Wahl einer kleineren Region behebt dieses Problem

Abbildung 3.14: Reflexionen in AugeROI beeinträchtigen die Berechnung des Glintzentrums



(a) Ein Ausschnitt vom Bild, der das Glint und die Nachbarpixel zeigt

(b) Das entsprechende Histogramm zu dem Ausschnitt

Abbildung 3.15: Der Maximalwert vom Glint befindet sich in dessen Zentrum. Die Werte nehmen mit zunehmendem Abstand zum Glintzentrum ab.

Zuerst wird der Maximalwert in der Region bestimmt. Der Mittelwert der Koordinaten der Pixel mit Maximumwert ist das Zentrumpixel für den nächsten Schritt. Die 9-er Nachbarscheft von dem Zentrumpixel wird betrachtet. Es werden in x und y Richtung jeweils 3 Polynome des zweiten Grades gefittet. Dafür muss die Gleichung 3.17 gelöst werden. Die Maxima von den 3 Polynomen werden berechnet. Daraus werden die gewichteten Mittelwerte der Maxima in jede Richtung bestimmt.

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{\mathbf{A}} \times \underbrace{\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}}_{\mathbf{x}} \tag{3.17}$$

$$\mathbf{A} = \mathbf{x}^{-1} \times \mathbf{y}$$

Die Maxima werden so gewichtet, dass nur das Maximum von dem Mittleren Polynom mit 3 gewichtet wird und die anderen Maxima mit 1. Mit dieser Methode wird das Glintzentrum subpixelgenau bestimmt. Anstatt der 9-er Nachbarschaft kann eine größere Nachbarschaft betrachtet werden. In diesem Fall ist Gleichung 3.17 überbestimmt. Daher muss für das Lösen der Gleichung die Pseudoinverse der **x** Matrix berechnet werden. In dieser Arbeit wird diesen Algorithmus mehrfach benutzt, dabei besteht jedes Polynom aus mindestens 7 Pixel (siehe Abbildung 3.16).

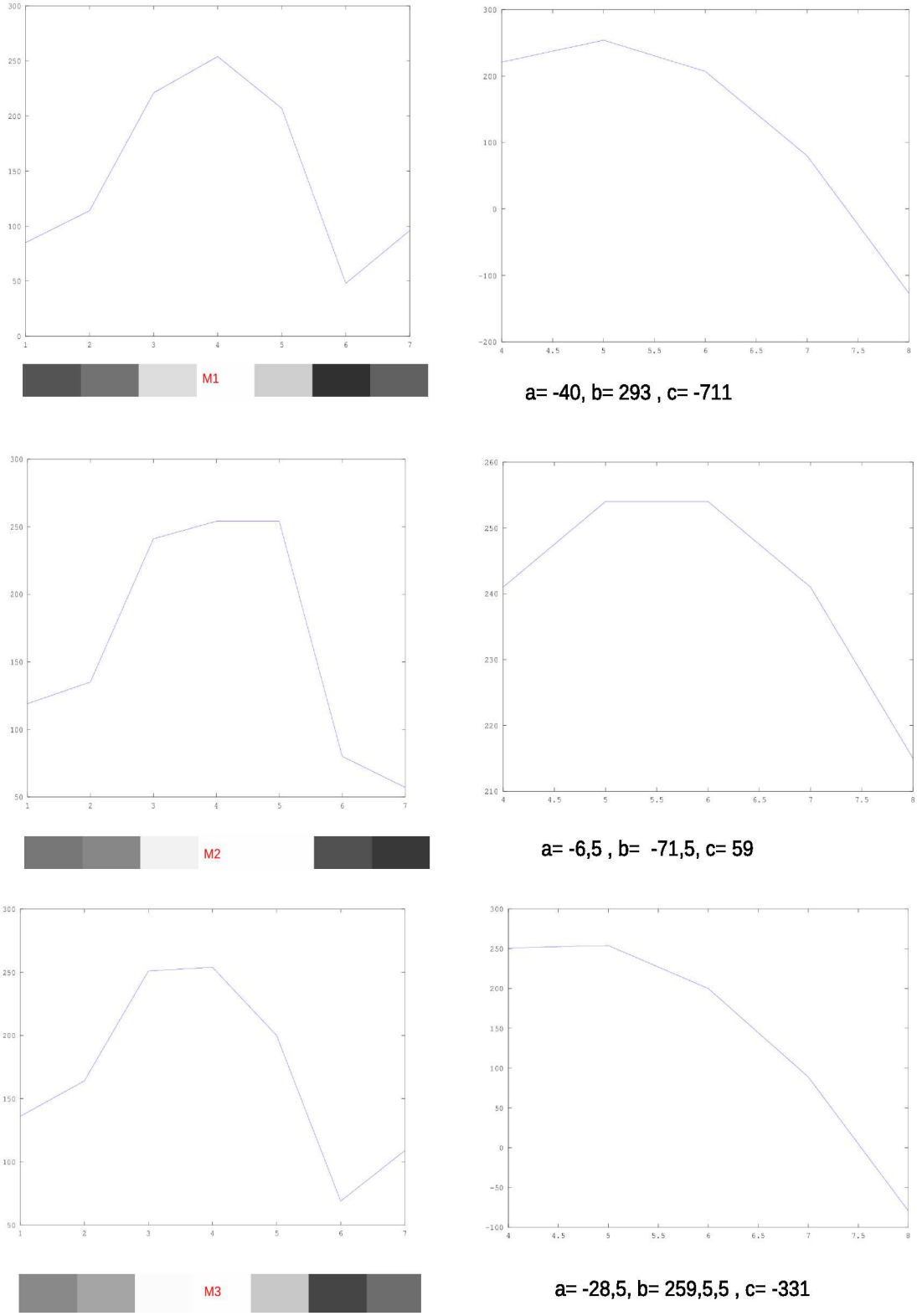


Abbildung 3.16: M2 ist das Zentrumpixel des Glintes. Auf dem Histogramm ist zu sehen, dass M2 mit seinen Nachbarn ein Polynom des 2-ten Grades bildet. Dieses hat M2 als Maximum. Die Reihen über und unter der Glintreihe bilden ähnlich 2 Polynome. Die entsprechenden Polynome sind rechts abgebildet.

Kapitel 4

Evaluation

In Kapitel 3 wurden die implementierten Algorithmen präsentiert. Während der Implementierung der Algorithmen wurden die Bilder von Wladimir Krebs als Eingabe für die Algorithmen benutzt. So konnte grob eingeschätzt werden, ob die gesuchten Merkmale auf den Bildern von einem Mensch und von den Algorithmen an einer ähnlichen Stellen gefunden werden. Um die Genauigkeit der Algorithmen subpixelgenau bestimmen zu können, brauchen wir eine Subpixelgenaue *Ground Truth*. In diesem Kapitel wird zuerst das Modell, das für die Evaluation der Algorithmen benutzt wird vorgestellt. Danach wird beschrieben wie die Ground Truth Daten bestimmt werden. Am Ende des Kapitels werden die Ergebnisse von der Evaluation bekannt gegeben.

4.1 Bilder für die Evaluation

Das Bestimmen der Ground Truth in realen Bilder ist im Gegensatz zu synthetischen Bilder eine große Herausforderung. Die Augen Bewegen sich ständig. Diese Bewegungen sind für die Wahrnehmung notwendig. Die Größe von Pupille und Iris, der Öffnungsgrad von den Liedern usw. ändern sich zufällig und sind nicht kontrollierbar. Dazu kommt die Tatsache, dass das Bestimmen der genauen Position von den Augenmerkmalen auf realen Bilder nicht trivial ist. Swirski et al.[SD13] präsentieren in ihrem Paper ein Blender Modell des Kopfes, das für die Evaluation des von ihnen vorgestellten Algortihmus benutzt wird.[1] Das Modell kann modifiziert werden damit z.B. beliebige Hautfarben, Pupillengrößen,... entstehen. Allerdings hat das Modell im Gegensatz zu den Testbildern, die im Implementierungsschritt benutzt wurden, keinen Glint. Außerdem weicht die Beleuchtung bei den entstehenden gerenderten Bildern von den realen Testbildern ab.

[1] Das Modell ist unter `http://www.cl.cam.ac.uk/research/rainbow/projects/eyemodelfit/` zu finden.

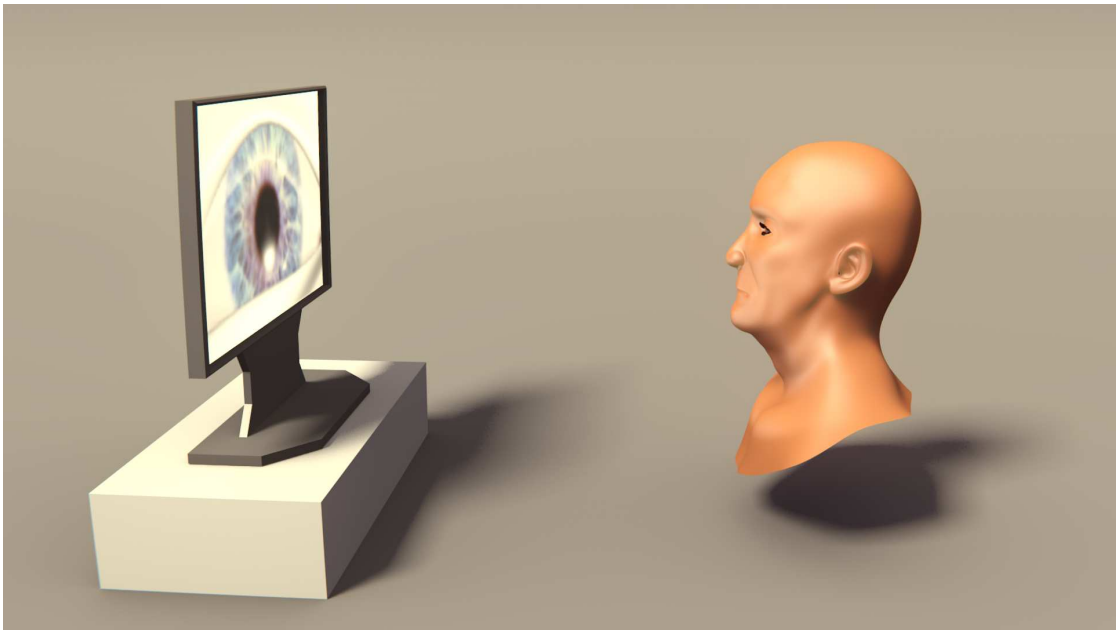


Abbildung 4.1: Schema der Hardware in dem geänderten Modell. Bild von Sebastian Koslink.

Wir haben in dieser Arbeit das genannte Modell modifiziert[2] und ein einfaches Verfahren benutzt um die subpixelgenaue Position von dem Pupillenzentrum finden zu können. Die Modifikationen sind wie folgt: Ein Bildschirm wurde vor den Kopf gesetzt sowie eine Kamera und eine Lichtquelle unter der Kamera. Die Bilder werden von der Kamera aufgenommen. Die Lichtquelle unter der Kamera ist die einzige Lichtquelle in dem Modell. So ähnelt das Modell der realen Hardware, mit dem Unterschied, dass die Lichtquelle keine Infrarot Lichtquelle ist. Um den Glint zu simulieren wird eine weiße Kugel an die Position der Lichtquelle gesetzt. Die Reflexion dieser Kugel sieht man als Glint auf dem Auge. Die Hautfarbe und der Reflexionsgrad von der Haut wurden geändert, damit die Haut und der Glint auseinanderzuhalten sind. Abbildung 4.1 zeigt das Schema von dem geänderten Modell.

Es wird eine Animation erzeugt, in der die Augen sich nach einem Muster bewegen und in bestimmte Richtungen schauen. So werden die Augenmerkmale in verschiedene Positionen extrahiert und für die Evaluation benutzt. Außerdem kann die genaue Blickposition auf dem Bildschirm im Modell bestimmt werden und für das Testen der Gaze Schäztungsalgorithmen benutzt werden. Abbildung 4.2 zeigt ein Beispiel für die gerenderten Bilder. Wie auf dem Bild zu sehen ist, ist nur ein Auge mit Wimpern gestaltet. Das haben wir von dem original Modell übernommen.

[2]Dank an Sebastian Koslink für die Modifikation von dem Blender Modell

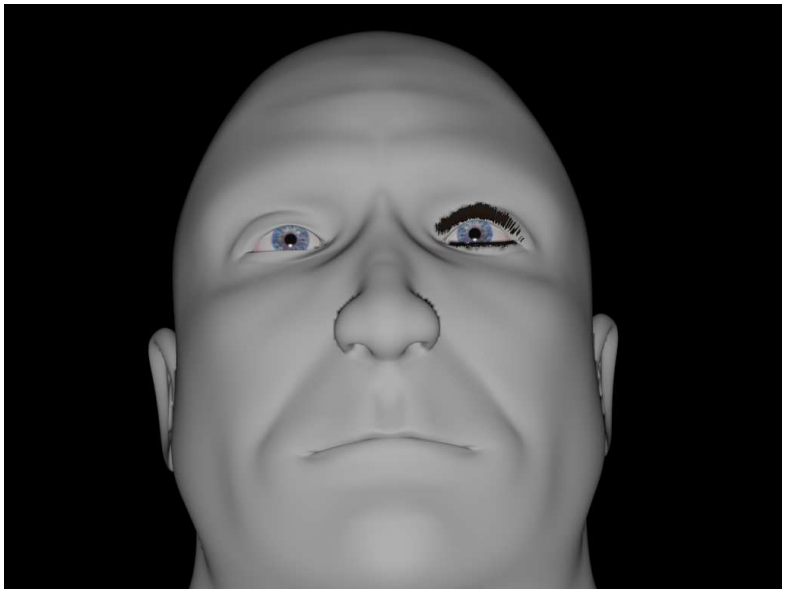


Abbildung 4.2: Ein gerendertes Bild von dem Blender Modell. Auflösung ist 1024x768Pixel

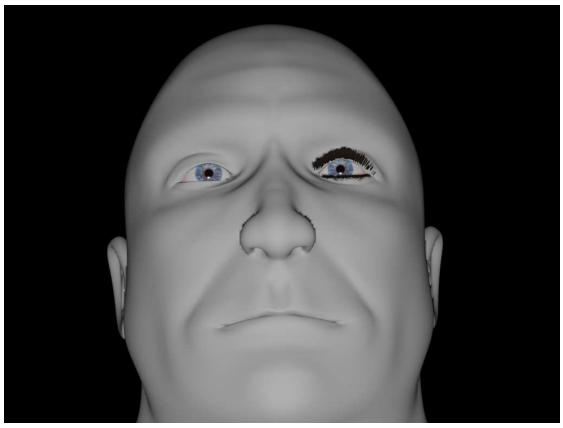
In dem Abschnitt 4.3 wird gezeigt, dass diese Asymmetrie zu Gunsten des Testen ist.

4.2 Ground Truth

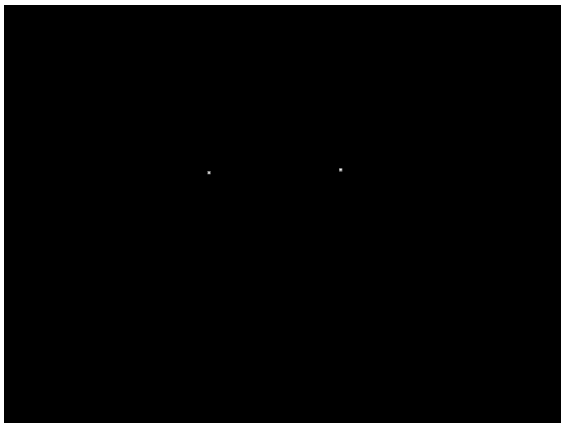
Die Merkmale, die von den implementierten Algorithmen in dieser Arbeit berechnet werden, sind das Pupillen- und Glintzentrum. Daher müssen Ground Truth für die Position dieser Zentren bestimmt werden. Die Ergebnisse der Algorithmen werden mit der Ground Truth verglichen.

Auf dem Blender Modell ist die Pupille als eine Scheibe gestaltet, die die Augenkugel berührt.[SD13] Um das Zentrum der Pupille auf den gerenderten Bilder zu bestimmen, haben wir einen weißen Punkt auf diese Scheibe gesetzt. Der Punkt wird auf eine separate Schicht gesetzt und in seperate Bilder gerendert. Als Ergenis haben wir haben wir zu jedem gerenderten Bild ein korrespondiertes Bild, das nur zwei weiße Punkte, nämlich die Pupillenzentren enthält (siehe Abbildung 4.3).

In Bild 4.4 ist der auf das Pupillenzentrum gesetzte weiße Punkt zu sehen. Die entstehende Struktur ist ähnlich wie die Struktur vom Glint in den realen Bildern (vergleiche mit dem Bild im Kapitel 3 Bild 3.16). Wir ziehen Nutzen aus dieser Tatsache um die Ground Truth des Pupillenzentrums mit dem *polynomial Fit* Algorithmus, der ursprünglich für die Glintzetrumdetektion implementiert wurde, zu bestimmen.



(a) Ein gerendertes Bild



(b) Die weißen Punkte auf dem Bild sind die Pupillenzentren auf dem linken Bild. (Die weißen Punkte auf dem Bild sind ungenau und übermalt, damit sie zu sehen sind).

Abbildung 4.3: Zu jedem gerenderten Frame wird ein korrespondierendes Bild erzeugt, das die Pupillenzentren enthält.

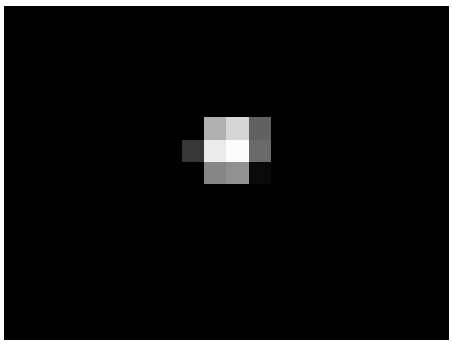


Abbildung 4.4: Der weiße Punkt, der das Pupillenzentrum bezeichnet, ähnelt dem Glint in realen Bilder.

Der durch die Reflexion entstehende Glint auf dem Blender Modell weicht von der Glintstruktur in realen Bilder ab (siehe Abbildung 4.5). Es ist erwähnenswert, dass die Auflösung der realen Testbilder 720x576 Pixel ist. Im Gegensatz zu den synthetischen Bilder, die für die Evaluation benutzt worden sind (1024x768 Pixel). Wenn die Auflösung von den synthetischen Bildern auf niedriger gestellt wird, besitzen die Ergebnisglints weniger Pixel. Dadurch werden sie ähnlicher zu den Glints der realen Bilder. Allerdings werden bestimmte Frames und Situationen wie in Bild 4.5 (c) nicht verbessert. Außerdem enthalten durch die Verminderung der Auflösung die weißen Punkte, welche auf die Pupillenzentren gesetzt worden sind, weniger Pixel. Weniger Pixel für die weißen Punkte beeinträchtigen den benutzten *polynomial fit* Algorithmus. Dadurch wird die Ground Truth für die Pupillenzentren ungenau oder fehlt.

Aufgrund der genannten Probleme wurden die zwei Algorithmen für die Berechnung des Glintzentrums auf den synthetischen Bilder ausgeführt und mit einander verglichen. Der Bedarf für einen Entwurf eines realistischeren Glints, und das Bestimmen der Ground Truth für denselben, bleibt.

4.3 Ergebnisse

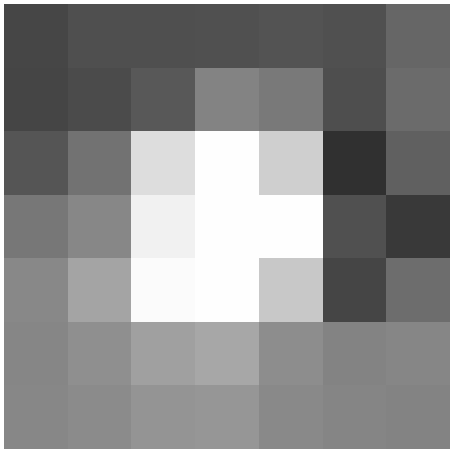
Die Algorithmen, die im Kapitel 3 Abschnitt 3.4 implementiert wurden, wurden wie folgt bewertet:

Die Algorithmen für die Berechnung der Pupillenzentren wurden auf den synthetischen Bilder ausgeführt und die Ergebnisse neben der Ground Truth geplottet. Anschließend wurden die Ergebniskoordinaten jedes Algorithmus und der Ground Truth Koordinaten in Matrizen gespeichert. So entstehen die Nx2 Matrizen wie in 4.1

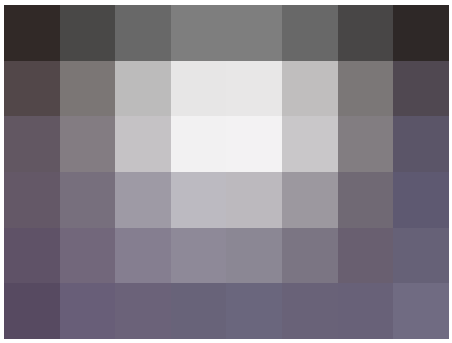
$$\mathbf{M} = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_N & y_N \end{pmatrix} \tag{4.1}$$

Jede Matrix, die Ergebnisse eines Algorithmus enthält, wird von der Ground Truth Matrix subtrahiert. Die Länge jedes Zeilenvektors der Ergebnisse wird in Nx1 Matrizen gespeichert. Diese entsprechen den Fehlern. Der Mittelwert und die Varianz der Elemente dieser Matrix wurden bestimmt. Die geplotteten Ergebnisse und statistische Berechnungen sind auf den Bilder 4.7 und 4.6 und der Tabelle 4.1 zu sehen.

Außer bei *RANSAC* zeigt das linke Auge stets bessere Ergebnisse. Die Vermutung ist, dass die Wimpern des rechten Auges mehr Fehler verursachen. In



(a) Beispiel für den Glint auf den realen Bilder. Diese Struktur bleibt einheitlich mit Bewegung der Augen.



(b) Beispiel für den Glint auf den synthetischen Bildern, erzeugt mit dem Blender Modell. Der Glint enthält mehrere Pixel. Der Farbverlauf auf der Region unterscheidet sich von dem auf den realen Bildern.

(c) Weiteres Beispiel für den Glint auf den synthetischen Bildern. Durch die Bewegung von Augen unterscheidet sich die Struktur vom Glint stark von der bei den realen Bildern.

Abbildung 4.5: Vergleich der Glintstruktur auf synthetischen und realen Bildern zeigt eine Abweichung

Abbildung 4.9 ist zu sehen, dass die Wimpern das Ergebnis der Binarisierung beeinträchtigen und in diesem Fall das *opening* auch nicht weiter hilft. Da wir das Ergebniss des *center of gravity* als Startpunkt für die Algorithmen, die einen Startpunkt benötigen, benutzen, verursacht der Fehler einen Domino-Effekt.

Wie zu erwarten, liefern die Algorithmen, die Randpixel benötigen (*after Daunys* ,*Fitzgibbon*,*RANSAC*), bessere Ergebnisse mit den Randpixeln, die von *cubic fitting* berechnet wurden. Dies liegt daran, dass *cubic fitting* im Gegensatz zu *contrast difference* subpixelgenau ist und eine robustere Methode fürs Randbestimmen anwendet. Die von *contrast difference* auf dem oberen und unteren Rand gelieferten Randpixel sind in meisten Bilder zu wenig. Diese Methode hatte jedoch auf den realen Bilder befriedigende Ergebnisse geliefert (siehe Abbildung 4.10 und vergleiche mit 3.10).

RANSAC liefert mit *contrast difference* unbefriediginde Ergebnisse. Die Vermutung ist, dass das gewählte Abstandsmaß (algebraische Distanz) nicht zweckmäßig ist. In Abbildung 4.11 sehen wir eine Beispielbild für das Ergebnis, berechnet von *RANSAC* und *contrast difference*. Die mit *contrast difference* berechneten Randpixel sind nur pixelgenau. Außerdem liefert die Methode Pixel, die in Wahrheit keine Randpixel sind. Diese sind eindeutig zwei Gründe dafür, dass *RANSAC* in Kombination mit *contrast difference* keine akzeptabelen Ergebnisse liefert.

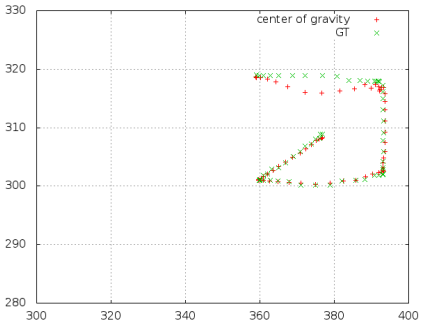
Wie in Abschnitt 4.2 erwähnt wurde, sieht der gestaltete Glint auf den synthetischen Bildern nicht realistisch aus. Außerdem fehlt eine Methode, um die Ground Truth für den Glint zu bestimmen. Deshalb haben wir die beiden Algorithmen für die Berechnung des Glintzentrums direkt miteinander verglichen. Die Ergebnisse dieses Vergleichs sind auf dem Bild 4.8 und der Tabelle 4.2 zu sehen.

Tabelle 4.1: Vergleich der Ergebnisse der Berechnung der Pupillenzentren mit der Ground Truth

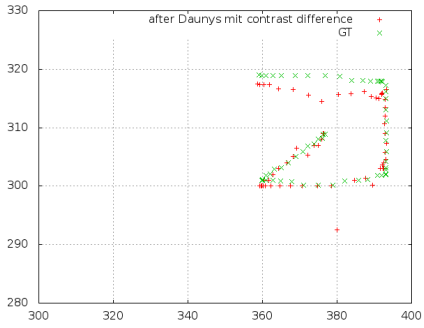
Auge	Algorithmus	Edgealgorithmus	Mittelwert	Standardabweichung
linkes	*center of gravity*	-	0.69577	0.68386
rechtes	*center of gravity*	-	2.0993	1.3478
linkes	*after Daunys*	*contrast difference*	1.4438	1.2882
rechtes	*after Daunys*	*contrast difference*	5.9341	14.059
linkes	*after Daunys*	*cubic fitting*	0.74185	0.52294
rechtes	*after Daunys*	*cubic fitting*	3.5661	10.911
linkes	*Fitzgibbon*	*contrast difference*	4.1924	1.4042
rechtes	*Fitzgibbon*	*contrast difference*	14.928	2.8502
linkes	*Fitzgibbon*	*cubic fitting*	0.79966	0.42278
rechtes	*Fitzgibbon*	*cubic fitting*	1.5066	1.0802
linkes	*RANSAC*	*contrast difference*	82.486	228.68
rechtes	*RANSAC*	*contrast difference*	44.966	106.78
linkes	*RANSAC*	*cubic fitting*	7.0344	24.152
rechtes	*RANSAC*	*cubic fitting*	4.5757	20.837

Tabelle 4.2: Vergleich der berechneten Glintzentren von *polynomial fit* und von *weighted center of gravity*

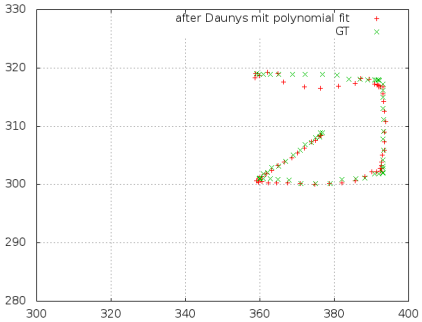
Auge	Mittelwert	Standardabweichung
linkes	3.3040	6.1794
rechtes	2.9334	5.4545



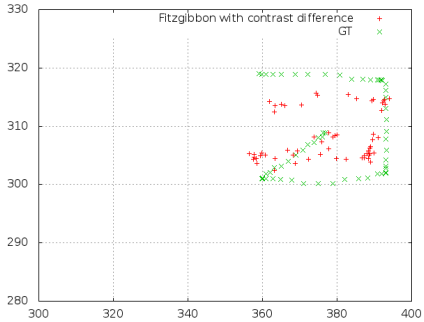
(a) Vergleich von *center of gravity* mit GT.



(b) Vergleich von *after Daunys* mit GT, dabei wurde *contrast difference* für das Randbestimmen benutzt.

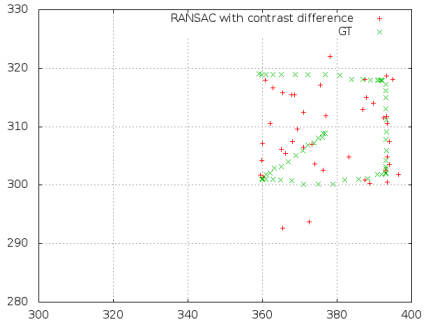


(c) Vergleich von *after Daunys* mit GT, dabei wurde *cubic fitting* für das Randbestimmen benutzt.

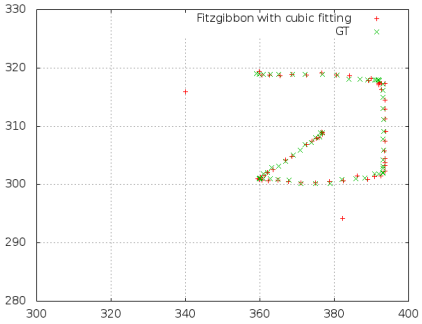


(d) Vergleich von *Fitzgibbon* mit GT, dabei wurde *contrast difference* für das Randbestimmen benutzt.

(e) Vergleich von *Fitzgibbon* mit GT, dabei wurde *cubic fitting* für das Randbestimmen benutzt.

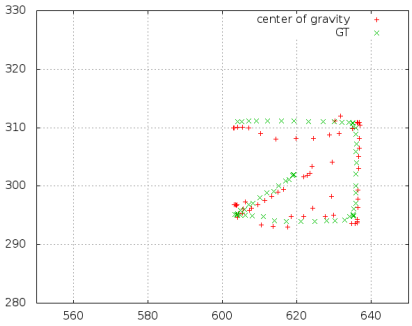


(f) Vergleich von *RANSAC* mit GT, dabei wurde *contrast difference* für das Randbestimmen benutzt.

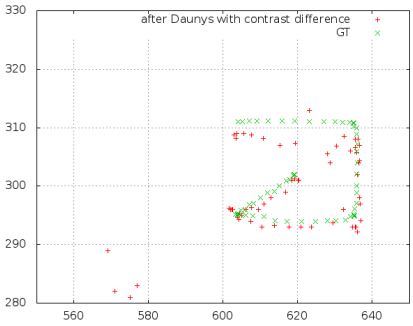


(g) Vergleich von *RANSAC* mit GT, dabei wurde *cubic fitting* für das Randbestimmen benutzt.

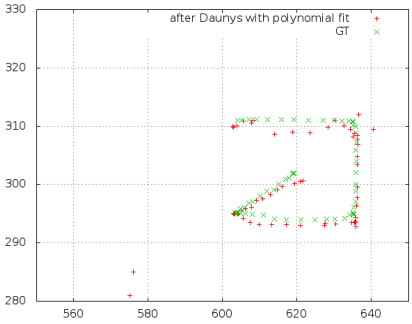
Abbildung 4.6: linkes Auge. Vergleich von den Algorithmen mit GT



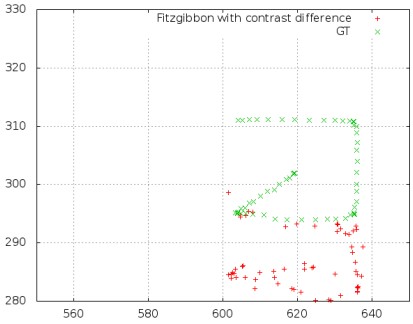
(a) Vergleich von *center of gravity* mit GT.



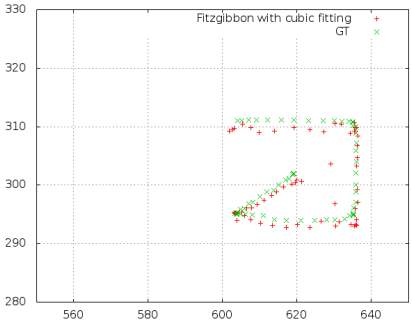
(b) Vergleich von *after Daunys* mit GT, dabei wurde *contrast difference* für das Randbestimmen benutzt.



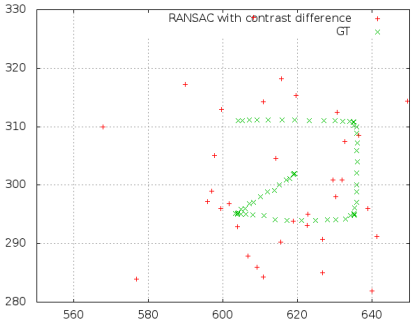
(c) Vergleich von *after Daunys* mit GT, dabei wurde *cubic fitting* für das Randbestimmen benutzt.



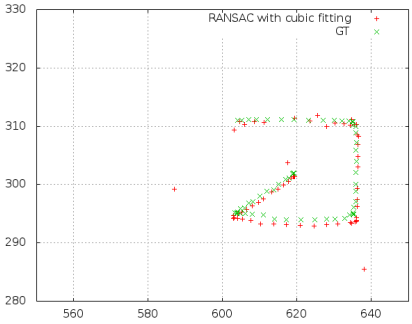
(d) Vergleich von *Fitzgibbon* mit GT, dabei wurde *contrast difference* für das Randbestimmen benutzt.



(e) Vergleich von *Fitzgibbon* mit GT, dabei wurde *cubic fitting* für das Randbestimmen benutzt.

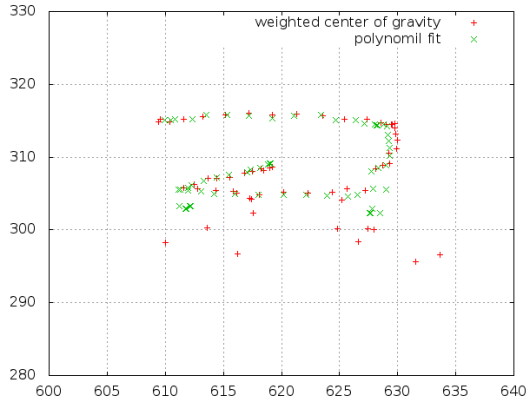


(f) Vergleich von *RANSAC* mit GT, dabei wurde *contrast difference* für das Randbestimmen benutzt.

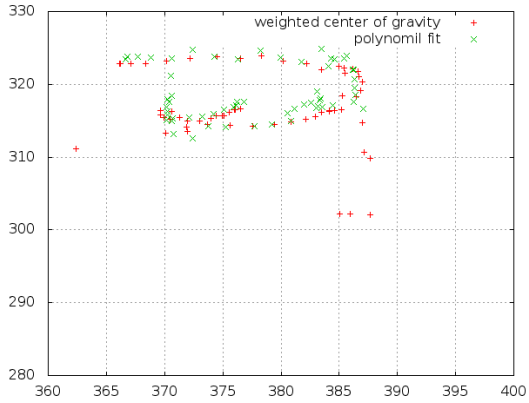


(g) Vergleich von *RANSAC* mit GT, dabei wurde *cubic fitting* für das Randbestimmen benutzt.

Abbildung 4.7: Rechtes Auge. Vergleich von den Algorithmen mit GT



(a) rechtes Auge. Vergleich *weighted center of gravity* mit *polynomial fit*.



(b) linkes Auge. Vergleich *weighted center of gravity* mit *polynomial fit*

Abbildung 4.8: Für den Glint vergleichen wir die beiden Algorithmen mit einander

(a) Die Wimpern sind auf dem binarisierten Bild zu sehen.

(b) Nachdem *opening* bleiben unerwünschte Pixel

Abbildung 4.9: Die Wimpern des rechten Auges verursachen einen Domino-Effekt.

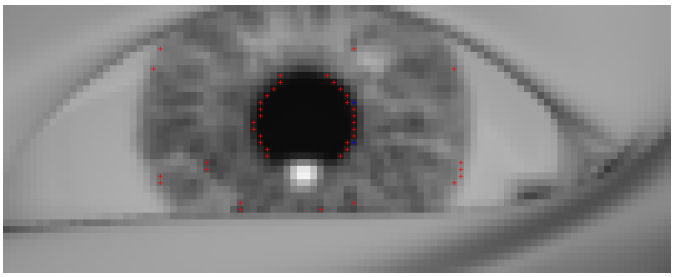


Abbildung 4.10: Am oberen und unteren Pupillenrand wurden nur wenige Randpixel gefunden (die blauen Kreuze).

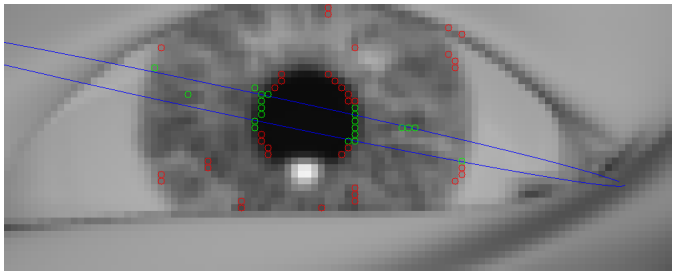


Abbildung 4.11: Ungeeignetes Abstandmaß für *RANSAC*. Die Randpixel sind mit *contrast difference* berechnet.

Kapitel 5

Zusammenfassung und Ausblick

Während dieser Arbeit wurde ein Framework für das Testen und Evaluieren der verschiedenen Algorithmen für Glint- und Pupillendetektion entwickelt. Das Framework wurde so aufgebaut, dass die Ergebnisse von den Algorithmen jederzeit vorhanden sind und eine Doppelberechnung nicht nötig ist. Mehrere merkmals- und modelbasierte Algorithmen für die Glintzentrum- bzw. Pupillenzentrumberechnung wurden implementiert und zuerst auf den zur Verfügung stehenden Testbilder getestet. Die Testbilder stammen von einer Kamera mit geriniger Auflösung. Zurzeit haben kostengünstigere Webcams eine höhere Auflösung. Für die zukünftige Arbeit könnten Bilder mit besserer Qualität benutzt werden. So wird das Berechnen der Merkmalen einfacher und trotzdem bleibt das Verfahren kostengünstig. Die Beobachtung der Ergebnisse auf den Testbildern zeigten befriedigende Ergebnisse.

Für die genaue Evaluation wurde ein vorhandenes synthetischen Modell modifiziert und als Ground Truth benutzt. Das Modell ähnelt der realen Situation bis auf manche Unterschiede wie z.B. einen unrealistischen Glint und nicht-Infrarote Lichtquelle. Außerdem fehlt die Möglichkeit für das Bestimmen der Ground Truth für den Glint. Die Vergleiche der Algorithmen für die Pupillendetektion zeigen, dass die Methoden außer *RANSAC* mit wenigen Modifikationen das Pupillenzentrum subpixelgenau berechnen können. Ein passendes Abstandsmaß könnte die Ergebnisse von *RANSAC* verbessern. Die Methoden, die für weitere Berechnungen die Randpixel benötigen, können mit dem pixelgenauen *cubic fitting* akzeptabelere Ergebnisse liefern. Allerdings beeinträchtigen die falschen Randpixel durch die Verdeckung der Pupille von dem Glint die Genauigkeit dieser Methoden. Eine Lösung für die Verdeckung der Pupille könnte in der Zukunft ein guter Ausgangspunkt für die Verbesserung der Methoden sein. Außerdem wurde in diese Arbeit die Lokalisierung der Augen nur am Rand behandelt. Da diese Lokalisierung für die nächsten Schritte ausschlaggebend ist, könnte mehr daran gearbeitet werden. Eine genauere Augendetektion kann einen genaueren Radius für die Augen liefern

und den Suchbereich für die Methoden verkleinern. Dadurch werden die Merkmale an den richtigen Positionen gesucht und false positive Fehler vermieden.

Literaturverzeichnis

[Boo79] Bookstein, F. L.: Fitting conic sections to scattered data. In: *Computer Vision, Graphics, and Image Processing* 9 (1979), S. 56–71

[Dau93] Daugman, John: High Confidence Visual Recognition of Persons by a Test of Statistical Independence. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 15 (1993), Nr. 11, 1148–1161. http://dx.doi.org/10.1109/34.244676. – DOI 10.1109/34.244676

[DR04] Daunys, Gintautas ; Ramanauskas, Nerijus: The Accuracy of Eye Tracking Using Image Processing. In: *Proceedings of the Third Nordic Conference on Human-computer Interaction*. New York, NY, USA : ACM, 2004 (NordiCHI '04). – ISBN 1–58113–857–1, 377–380

[FF95] Fitzgibbon, Andrew W. ; Fisher, Robert B.: R.B.: A Buyer's Guide to Conic Fitting. In: *British Machine Vision Conference*, 1995, S. 513–522

[FF99] Fitzgibbon, M. A. W. A. W.and Pilu P. A. W.and Pilu ; Fisher, R. B.: Direct least-squares fitting of ellipses. 21 (1999), Mai, Nr. 5, S. 476–480

[HJ10] Hansen, Dan W. ; Ji, Qiang: In the Eye of the Beholder: A Survey of Models for Eyes and Gaze. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32 (2010), Nr. 3, S. 478–500. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/TPAMI.2009.30. – DOI http://doi.ieeecomputersociety.org/10.1109/TPAMI.2009.30. – ISSN 0162–8828

[HZ03] Hartley, Richard ; Zisserman, Andrew: *Multiple View Geometry in Computer Vision*. 2. New York, NY, USA : Cambridge University Press, 2003. – ISBN 0521540518

[LWP05] Li, Dongheng ; Winfield, David ; Parkhurst, Derrick J.: Starburst: A Hybrid Algorithm for Video-based Eye Tracking Combining Feature-based and Model-based Approaches. In: *Proceedings of the 2005 IEEE*

Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops - Volume 03. Washington, DC, USA : IEEE Computer Society, 2005 (CVPR '05). – ISBN 0–7695–2372–2–3, 79–

[Mai05] Maini, Eliseo S.: Robust Ellipse-specific Fitting for Real-time Machine Vision. In: *Proceedings of the First International Conference on Brain, Vision, and Artificial Intelligence*. Berlin, Heidelberg : Springer-Verlag, 2005 (BVAI'05). – ISBN 3–540–29282–9, 978–3–540–29282–1, 318–327

[SD13] Świrski, Lech ; Dodgson, Neil A.: A fully-automatic, temporal approach to single camera, glint-free 3D eye model fitting [Abstract]. In: *Proceedings of ECEM 2013*, 2013

[SWL00] Shih, Sheng-Wen ; Wu, Yu-Te ; Liu, Jin: A Calibration-Free Gaze Tracking Technique. In: *ICPR*, 2000, S. 4201–4204

[YHC92] Yuille, Alan L. ; Hallinan, Peter W. ; Cohen, David S.: Feature Extraction from Faces Using Deformable Templates. In: *Int. J. Comput. Vision* 8 (1992), August, Nr. 2, 99–111. `http://dx.doi.org/10.1007/BF00127169`. – DOI 10.1007/BF00127169. – ISSN 0920–5691