# Diplomarbeit

TECHNISCHE UNIVERSITÄT CHEMNITZ

TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Elektrotechnik und Informationstechnik

Professur Robotersysteme

# Diplomarbeit

Entwicklung einer offenen Softwareplattform für Visual Servoing
Praktische Realisierung am Beispiel von Gesichtserkennung und -verfolgung

Sören Sprößig

Chemnitz, den 28. Juni 2010

**Prüfer:** Prof. Dr.-Ing. Jozef Suchý
**Betreuer:** Dipl.-Ing. Gunnar Zschocke

**Sprößig, Sören**
Entwicklung einer offenen Softwareplattform für Visual Servoing
Diplomarbeit, Fakultät für Elektrotechnik und Informationstechnik
Technische Universität Chemnitz, Juni 2010

# Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Chemnitz, den 28. Juni 2010

Sören Sprößig

Technische Universität Chemnitz
Fakultät für Elektrotechnik und Informationstechnik
Professur Robotersysteme

28.10.2009

# Aufgabenstellung

für

## Diplomarbeit

Name, Vorname: Sprößig, Sören geb. am: 14.05.1984

Studiengang: Elektrotechnik

Studienrichtung: Automatisierungstechnik

Thema: Entwicklung einer offenen Softwareplattform für Visual Servoing

(Ausführliche Aufgabenstellung siehe Rückseite)

Die wissenschaftliche Arbeit ist als Einzelarbeit/Gruppenarbeit anzufertigen *)

Betreuer: Prof. Dr.-Ing. Jozef Suchý

Tag der Ausgabe: 28.10.2009

Abgabetermin: 27.04.2010

Tag der Abgabe:

Prof. Dr.-Ing. habil. A. Farschtschi
Vorsitzender des Prüfungsausschusses

Prof. Dr.-Ing. J. Suchý
Verantwortlicher Hochschullehrer

Ausführliche Aufgabenstellung:

Entwicklung einer offenen Softwareplattform für Visual Servoing. Praktische Realisierung am Beispiel von Gesichtserkennung und –verfolgung.

Ziel dieser Arbeit ist es, eine flexibel zu verwendende Plattform für Visual Servoing Aufgaben zu erstellen, mit der eine Vielzahl von verschiedenen Anwendungsfällen abgedeckt werden kann. Schwerpunkte sind die Untersuchung, die detaillierte Beschreibung sowie die Implementierung verschiedener Verfahren der Gesichtserkennung und –wiedererkennung (engl. „face detection/recognition").

Aus den gewonnenen Erkenntnissen und dem sich ergebenden Anforderungsprofil an die zu entwickelnde Plattform, leitet sich die anschließende Realisierung einer eigenständigen Anwendung ab. Hierbei ist weiterhin zu untersuchen, wie die neu zu entwickelnde Software zukunftssicher und in Hinblick auf einen möglichen Einsatz in Praktika umgesetzt werden kann.

# Acknowledgements

„Portions of the research in this paper use the FERET database of facial images collected under the FERET program, sponsored by the DOD Counterdrug Technology Development Program Office.“

Für weitere Informationen siehe [46] und [47].

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Liste der Algorithmen

# Abkürzungsverzeichnis

| | |
|---|---|
| .NET | Software-Plattform von Microsoft |
| API | *„Application Programmiung Interface“*, Programmierschnittstelle |
| CART | *„Classification and Regression Tree“*, eine Entscheidungsbaumstyp |
| CCD | *„Charge-coupled device“*, integriertes Bauteil, z.B. verwendet als Lichtsensor |
| CLR | *„Common Language Runtime“*, Laufzeitumgebung des .NET-Frameworks |
| CMOS | *„Complementary Metal Oxide Semiconductor“*, Halbleiterbauelement, bei dem p- und n-Kanal-MOSFET auf einem gemeinesamen Substrat liegen |
| D-LDA | *„Direct Linear Discrimant Analysis“*, siehe Abschnitt 2.4.2 |
| DCT | Diskrete Kosinus-Transformation |
| DF-LDA | *„Direct Fractional Linear Discrimant Analysis“*, siehe Abschnitt 2.4.2 |
| DWT | Diskrete Wavelet-Transformation |
| EMF | *„Enhanced Fisher Model“*, Erweitertes LDA-Verfahren |
| F-LDA | *„Fractional Linear Discrimant Analysis“*, siehe Abschnitt 2.4.2 |
| GA | Genetischer Algorithmus |

| | |
|---|---|
| GFC | *„Gabor Fisher Classificator“*, Bildklassifikator, der Gabor-Wavelets verwendet |
| HSV | Hue-Saturation-Value, Farbmodell |
| ICA | *„Independent Component Analysis“*, höherdimensionale Verallgemeinerung der PCA |
| KOS | Koordinatensysteme |
| LBP | *„Local Binary Patterns“*, Ansatz in der Texturanalyse, der auf Analyse der lokalen Pixel-Nachbarschaft beruht. |
| LDA | *„Linear Discrimant Analysis“*, siehe Abschnitt 2.4.2 |
| MDA | *„Model-driven Architecture“*, modellgetriebene Softwarearchitektur |
| MDF | *„Most Discriminant Feature“*, siehe Abschnitt 2.4.2 |
| MEF | *„Most Expressive Feature“*, siehe Abschnitt 2.4.2 |
| OPVS | *„Open Platform Visual Servoing“*, Name des im Rahmen dieser Diplomarbeit entstandenen Programms |
| PAC | *„Probaly Approximately Correct“*, Entscheidungsklassifikator, der zu >50% richtig liegt |
| PC | Personal Computer |
| PCA | *„Principal Component Analysis“*, siehe Abschnitt A.2.5 |
| PDM | *„Point Distribution Model“*, Punktverteilungsmodell, Bilderkennungsverfahren |
| RGB | Rot-Grün-Blau, Farbmodell |
| RSAT | *„Rotated Summed Area Table“*, siehe SAT |
| SAT | *„Summed Area Table“*, siehe Integralbild |

| | |
|---|---|
| SURF | *„Speeded Up Robust Features“*, Featuredetektor für Bilder |
| SVD | *„Singular Value Decomposition“*, Singulärwertzerlegung einer Matrix |
| SVM | *„Support Vector Machine“*, Klassifikatorverfahren |
| SVN | Subversion, ein Versionsverwaltungssystem |
| TCP | *„Tool Center Point“*, Werkzeugposition eines Industrieroboters |
| TCP/IP | *„Transmission Control Protocol/Internet Protocol“*, Netzwerkprotokoll |
| UML | *„Unified Modelling Language“*, Grafische Sprache zur Modellierung von Software und Systemen |
| USB | *„Universal Serial Bus“*, universelle Peripherieschnittstelle |
| XML | *„Extensible Markup Language“*, Auszeichnungsprache zur Darstellung hierarchischer Datensätze |
| XSD | *„XML Schema“*, Strukturdefinition für XML-Dateien |

# Formelzeichenverzeichnis

| | |
|---|---|
| $I_I(x_p, y_p)$ | Integralbild; Summe über die Bildpixel bis zur Bildkoordinate $(x_p, y_p)$ |
| $I_{I,45°}(x, y)$ | um 45° rotiertes Integralbild |
| $S_B$ | Streuung der Bilder zwischen allen Klassen in der Trainingsmenge |
| $S_T$ | totale Streuung über alle Bilder der Trainingsmenge |
| $S_W$ | Streuung über alle Bilder einer Klasse in der Trainingsmenge |
| $W$ | Optimierungskriterium bei PCA/LDA-Verfahren |
| $\Gamma_i$ | Trainingsbild $i$ im Eigengesichter-/Fishergesichterverfahren in vektorieller Schreibung. |
| $\mathbf{C}$ | Kovarianzmatrix |
| $\mathbf{R}$ | Rotationsmatrix |
| $\Omega_i$ | in Eigengesichtsraum projiziertes Trainingsbild $i$ |
| $\Phi_i$ | zentriertes Trainingsbild $i$ im Eigengesichter-/Fishergesichterverfahren |
| $\Psi$ | Durchschnittsgesicht im Eigengesichter-/Fishergesichterverfahren zur Zentrierung der Traininsbilder. |
| $\epsilon$ | allgemeine Bezeichnung für den in einer Berechnung gemachten Fehler; weiterhin bei Eigengesichter-/Fishergesichterverfahren der Abstand zwischen in Eigengesichtsraum projizierten unbekannten und trainierten Gesichtsaufnahmen |
| $\phi$ | Drehwinkel um die $x$-Achse $^\circ$ |
| $\psi$ | Drehwinkel um die $z$-Achse $^\circ$ |
| $\theta$ | Drehwinkel um die $y$-Achse $^\circ$ |
| $\vec{T}$ | Translationsvektor |
| $w$ | Gewichtung |

# 1 Einleitung

## 1.1 Motivation

Ziel dieser Diplomarbeit ist es, eine flexibel zu verwendende Plattform für Visual Servoing-Aufgaben zu Erstellen, mit der eine Vielzahl von verschiedenen Anwendungsfällen abgedeckt werden kann. Kernaufgabe der Arbeit ist es dabei, verschiedene Verfahren der Gesichtserkennung (engl. *„face detection“*) und -wiedererkennung (engl. *„face recognition“*) zu betrachten und an ausführlichen Beispielen vorzustellen. Dabei sollen allgemeine Grundbegriffe der Bildverarbeitung und bereits bekannte Verfahren vorgestellt und ihre Implementierung im Detail dargestellt werden. Aus den dadurch gewonnen Erkenntnissen und dem sich ergebenden Anforderungsprofil an die zu entwickelnde Plattform leitet sich anschließend die Realisierung als eigenständige Anwendung ab. Hierbei ist weiterhin zu untersuchen, wie die neu zu entwickelnde Software zukunftssicher und in Hinblick auf einen möglichen Einsatz in Praktika einfach zu verwenden realisiert werden kann. Sämtliche während der Arbeit entstandenen Programme und Quellcodes werden auf einem separaten Datenträger zur Verfügung gestellt. Eine komplett funktionsfähige Entwicklungsumgebung wird als virtuelle Maschine beigelegt.

## 1.2 Verständnis des Begriffs Visual Servoing

In [56] beschreiben die Autoren Visual Servoing als *„...control based on feedback of visual measurements...“*, also jedwede Form von Regelung eines Roboters auf Basis visueller Datenerfassung.
Die Autoren von [4] verweisen in ihrer Begriffsklärung auf eine *„...von Sanderson und Weiss eingeführte Taxonomie visueller Hilfssysteme...“* [51]. Auch in dieser Betrachtung werden die visuell erfassten Informationen direkt in der Roboterregelung integriert (*„dynamic look-and-move vs. direct visual servo“*). Weiterhin unterscheiden beide zwischen der Art wie die visuell erfassten Information verarbeitet werden. So spielt es eine signifikante Rolle, ob die Informationen zur Lokalisation des Roboters im Raum verwendet werden (*„position-based visual servoing“*) oder ob aus einem erfassten Bild Merkmale direkt Aufschluss über Steuerparameter geben sollen

(„*image-based visual servoing*“).
Mit dieser Arbeit soll nun ein allgemeinerer Ansatz zum Thema versucht werden - wir verstehen den Begriff *visual servoing* hier als die Gesamtheit aller durch digitale Bildverarbeitung ausgelösten Roboteraktionen.

# 2 Methoden der Gesichtserkennung

## 2.1 Einleitung

Das Gebiet der Gesichtserkennung ist seit Mitte der 60er Jahre des 20. Jahrhunderts ein weltweit intensiv bearbeitetes Forschungsgebiet. Gute Ergebnisse werden hier seit circa zehn Jahren mit dem Erscheinen günstiger und vor allem leistungsfähiger Hardware und neuer Algorithmen erzielt. Eine Suche im Online-Archiv des IEEE Explorers nach *face recognition* bringt allein für das Jahr 2009 mehr als 500 Veröffentlichungen hervor, was zeigt, dass dieses Thema sich nach wie vor großen Interesses erfreut. Vor allem die Anwendung im Sicherheitsbereich sensibler Einrichtungen (Forschungslabore, Flughäfen u.ä.) erfordert eine stetige Verbesserung bekannter Algorithmen.

Bei näherer Betrachtung der Thematik muss zunächst die Begrifflichkeit der *Gesichtserkennung* (engl. *„face detection and feature extraction“*) - das Auffinden von Gesichter über ihre Merkmale in Bildern - von der der *Gesichtswiedererkennung* (engl. *„face recognition“*) - das Wiedererkennen bereits bekannter Gesichter - unterschieden werden.

Eine gute historische Übersicht der Forschung der letzten 60 Jahre und einen weiten Überblick über verschiedene Ansätze zur Gesichtserkennung ist in [72] zu finden. Aktuellere Arbeiten werden in [69] betrachtet und grob verglichen. Ein mannigfaltiger Überblick über verschiedene Ansätze und Papers zu deren Implementierung ist schließlich unter [22] zu finden.

Ziel dieser Arbeit soll es sein, eine praktikable, einsatzfähige Lösung für ein einfaches Gesichts-Tracking zu finden und die Realisierung im Detail zu beschreiben. Für spätere Verwendung in der Ausbildung wird die Methode auch für Anfänger auf dem Gebiet der Bildverarbeitung anwendbar ausformuliert und in einer einfach zu benutzenden Software hinterlegt.

## 2.2 Vorbetrachtungen

### 2.2.1 Abgrenzung der Aufgabenstellung

Auf dem Gebiet der Bildverarbeitung sind unter dem Aspekt der Gesichtserkennung sehr viele Methoden publiziert worden, die teilweise extrem unterschiedliche Ansätze verfolgen. Ein Großteil der Arbeiten beschränkt sich dabei auf Extraktion aller Informationen aus einer Aufnahme (engl. *„acquisition from still images“*), was für diese Arbeit auch festgelegt wurde. Für die Gesichtserkennung wird das Verfahren des *Haar-Klassifikators* im Detail vorgestellt. Mögliche Erweiterungen und Verbesserungen werden zusätzlich eingeführt. Abschließend werden verschiedene Implementierungen hinsichtlich ihrer Ausführungsgeschwindigkeit untersucht. Für die Gesichtswiedererkennung können die existierenden Algorithmen grob in *Modell-basierte* (engl. *„model-based“*) und *Aussehens-basierte* (engl. *„appearance-based“*) Verfahren unterteilt werden, wobei diese Diplomarbeit nur ausgewählte Vertreter letzterer Kategorie behandeln wird. Detailliert vorgestellt werden die Verfahren *Eigengesichter* und *Fischergesichter*, die gegeneinander evaluiert werden. Im Anschluss werden zusätzlich noch aktuell veröffentlichte Verbesserungen dieser beiden Verfahren vorgestellt.

Den Abschluss bildet die Einführung eines einfachen Verfahrens zum Verfolgen einer bekannten Person, das die eingeführten Verfahren fusioniert und so die Grundlage der in den folgenden Kapiteln vorgestellten Softwarelösung darstellt.

Einen Überblick über weitere Methoden zur Gesichtserkennung und -wiederkennung ist schließlich in Abschnitt 2.6 zu finden.

### 2.2.2 Vergleichbarkeit der Ergebnisse

Viele Arbeiten (u.a. [69]) weisen auf die schlechte Vergleichbarkeit der bekannten Algorithmen hin. Zum jetzigen Zeitpunkt existiert noch kein abschließend als Standardverfahren bezeichenbares Verifikations- und Validierungsverfahren für die verschiedenen Realisierungen. Dies liegt unter anderem in der unterschiedlichen Auffassung begründet, was eine „korrekte Funktion“ des Algorithmus bedeutet. So kann die Güte der eingesetzten Klassifikatoren über die *ROC-Kurve* (engl. *„receiver operating characteristic“*) bewertet werden, bei der die *Sensitivität* (auch *Richtigpositiv-Rate*) über die *Spezifität* (auch *Richtignegativ-Rate*) der Erkennung in einem kartesischen Koordinatensystem abgetragen wird. Für andere Anwendungen kann die Ausführungsgeschwindigkeit des Algorithmus wichtiger sein als die Erkennungsrate (z.B.

Einsatz in Echtzeitsystemen), was wiederum direkt mit den Kosten[1] des Algorithmus korreliert.
Hauptproblem der Vergleichbarkeit zwischen den Algorithmen aber ist das verwendete Bildmaterial. Sehr viele vorgestellte Algorithmen werden lediglich auf spezielle Testaufnahmen angewendet, was die Ergebnisse schlecht deutbar macht. Dabei existiert eine Vielzahl nutzbarer Gesichtsdatenbanken, die zur vergleichenden Bewertung der Ergebnisse verwendet werden können ([79], Tabelle 2.2.2).
In dieser Arbeit selbst wurden Validierungen mit der AT&T-Datenbank vorgenommen. Weiterhin wurde die Robustheit der Algorithmen mit einer selbst zusammengestellten kleinen Gesichtsdatenbank mit je neun Frontalaufnahmen bekannter Personen geprüft, die zum Test auch mit künstlichen Bildfehlern und Bildrauschen versehen wurden. Abschließend wurden tatsächliche Aufnahmen mit der final einzusetzenden Firewire-Kamera im Labor aufgenommen.

## 2.3 Gesichtserkennung (face detection)

### 2.3.1 Einleitung

Die erste Anwendung des *Haar-Klassifikators* wurde von Viola und Jones in [63] beschrieben.[2] Eine Erweiterung um weitere diagonale Merkmale und einen nachgeschalteten Optimierungsalgorithmus wurde von Lienhart und Maydt in [31] ergänzt und stellt das gegenwärtige Standardverfahren von OpenCV zum Auffinden von Gesichtern in Bildern dar.
Im Gegensatz zum allgemeinen Ansatz wird im von Viola und Jones beschriebenen Verfahren nicht das zu verarbeitende Bild skaliert und an einen festen Detektor übergeben, sondern der Detektor selbst skaliert. Anschließend wird er erneut über das Bild geschoben und die Erkennung mit dem neuen Detektor wiederholt. Dies reduziert die benötigte Zeit und den benutzten Speicher erheblich, da keine Zwischenbilder mehr berechnet werden müssen.

### 2.3.2 Haar-ähnliche Merkmale

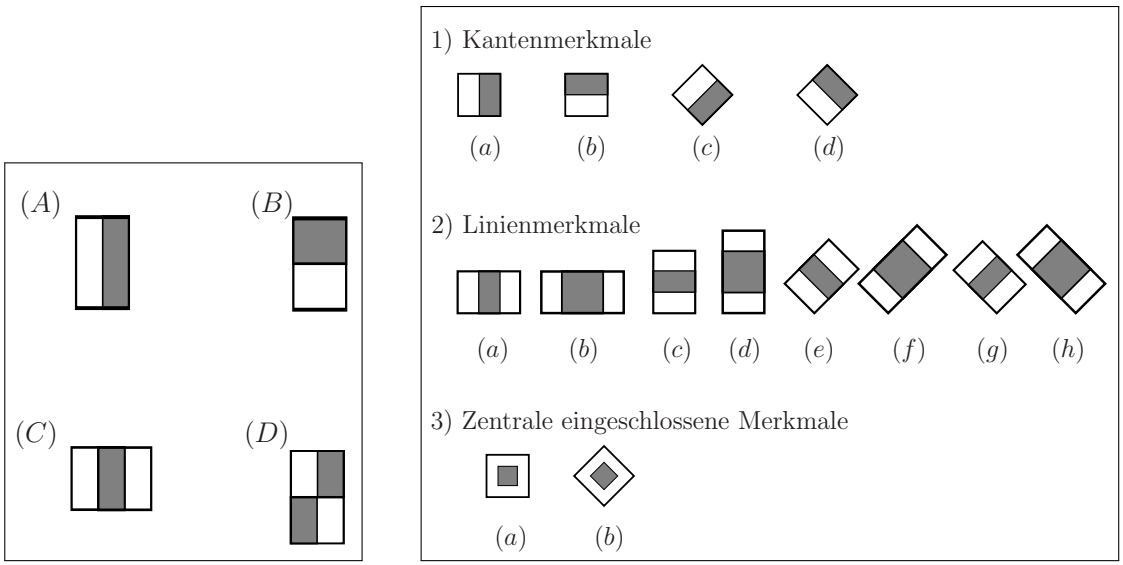
Der Name Haar-Klassifikator ist der englischen Bezeichnung des Verfahrens „*a cascade of boosted classifiers working with haar-like features*“ entlehnt, wobei eben

[1] Die Aufwendigkeit eines Algorithmus und die damit verbundene benötigte Rechenkapazität wird auch als Kosten bezeichnet.
[2] Der Ansatz an sich wurde allerdings bereits 1995 von Papageorgiou et al. in [43] beschrieben. Die effektive Implementierung allerdings erfolgte erst durch Viola/Jones.

Tabelle 2.1: Einige Gesichtsdatenbanken im Überblick

| Datenbank | Beschreibung |
|---|---|
| AT&T „Database of faces“ [74] | *umfangreiche Datenbank für verschiedene Posen*; Graustufenbilder; je zehn Aufnahmen von insgesamt 40 Personen; Männer und Frauen; verschiedene Hautfarben; Frontalaufnahmen mit leichter Drehung um die Longitudinalachse; leichte Änderung der Beleuchtung und des Gesichtsausdrucks; teilweise mit Brille und Bart |
| Color FERET Database [75] | *zur Zeit größte verfügbare Farbbilddatenbank, teilweise Aufnahme einzelner Personen über Jahre verteilt*; Farbbilder; insgesamt 14126 Aufnahmen in 1564 Aufnahmesessions mit insgesamt 1199 verschiedenen Personen; Männer und Frauen; für 365 Personen sind mehrere Aufnahmen mit Zeitversatz von bis zu 2 Jahren erstellt worden |
| Sheffield Face Database (früher UMIST) [76] | *große Datenbank, häufig für Vergleiche in Arbeiten verwendet*; Graustufenbilder; insgesamt 564 Bilder; 20 Personen, verschiedene Geschlechter, Hautfarben; verschiedene Aufnahmen von Profil- bis Frontalaufnahmen |
| Testdatenbank Sören Sprößig | *kleine Testdatenbank mit Aufnahmen aus dem Internet, manuell beschnitten und zentriert*; Farb- und Graustufenbilder; insgesamt 36 Bilder; 3 Personen, je 10 Bilder + 6 Testbilder; eine Frau, zwei Männer; Frontalaufnahmen |
| Yale Face Database [77] | *kleine Datenbank mit Änderung des Gesichtsausdrucks*; Graustufenbilder; je elf Aufnahmen von insgesamt 15 Personen; Männer und Frauen; verschiedene Hautfarben; Frontalaufnahmen; starke Änderung der Beleuchtung und des Gesichtsausdrucks; teilweise mit Brille und Bart |
| Yale Face Database B [78] | *sehr umfangreiche Bibliothek zur Untersuchung von veränderter Pose und Beleuchtung*; Graustufenbilder; je 576 Aufnahmen (neun Posen zu je 64 verschiedenen Beleuchtungen) von zehn Personen; zusätzlich für alle Posen eine Aufnahme mit Hintergrundbeleuchtung; Männer und Frauen; verschiedenen Hautfarben; teilweise mit Brille und Bart |

jene Haar-ähnlichen Merkmale wiederum ihren Namen vom *Haar-Wavelet* beziehen (siehe dazu A.2.6).




(a) nach Viola/Jones (b) nach Lienhart/Maydt

Abbildung 2.1: Haar-ähnliche Merkmale

Viola und Jones haben eine Größe von 24×24 Pixel für Trainingsbilder $x$ ermittelt, was für die von ihnen definierten Merkmale circa 160.000 mögliche skalierte Merkmale bedeutet. Dabei ist zu beachten, dass mit dieser Anzahl eine deutliche Überdefinition der benötigten Merkmale stattfindet. Lienhart und Maydt haben durch Einführung neuer Merkmale und der daraus resultierenden Reduzierung der Redundanzen deren Anzahl auf 117.941 reduzieren können, was nach wie vor einen erheblichen Rechenaufwand in der Trainingsphase bedeutet.
Die Aufgabe des Lernverfahrens (Algorithmus 2.1) ist es daher, die $T$ stärksten Merkmale anhand vorhandener Trainingsbilder zu adaptieren. Als Lernalgorithmus wird ein *Boostingverfahren*[3] namens *AdaBoost* in einer auf die Problemstellung angepassten Form verwendet. Dabei wird der *schwache Klassifikator* $h$ trainiert, der die wenigsten Trainingsbilder falsch klassifiziert. Dieser verwendet das Haar-ähnliche Merkmal $f$, den Schwellwert $\theta$ und die Polarität $p$, die die Richtung der Ungleichung bestimmt:

$$h(x, f, p, \theta) = \begin{cases} 1 & p\, f(x) < p\, \theta \\ 0 & sonst \end{cases}$$

Im Bild werden Haar-ähnliche Merkmale mit Hilfe eines sogenannten Integralbilds (engl. *„integral image“*) oder auch SAT (engl. *„SAT = summed area table“*) als

[3]Eine kurze Einführung in Boosting und weitere maschinelle Lernalgorithmen liefert A.2.7.

**Algorithmus 2.1:** Boostingverfahren mit $T$ Hypothesen für alle möglichen Merkmale $f$

**Gegeben** *n Bilder* $(x_1, y_1), ..., (x_n, y_n)$ *mit* $y_i = 1$ *für positive (enthalten Gesicht) und* $y_i = 0$ *negative (enthalten kein Gesicht) Trainingsbilder*;
**Initialisiere** *Gewichte* $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ *für alle m positiven und alle l negativen Trainingsbilder*

**für** $t = 1, ..., T$ **tue**

**Normalisiere** die Gewichte $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$ **Wähle** den besten schwachen Klassifikator anhand des gewichteten Fehlers $\epsilon_t$ mit

$$\epsilon_t = \min_{f,p,\theta} \sum_i w_{t,i} \left| h(x_i, f_t, p_t, \theta_t) - y_i \right|$$

**Setze** $h_t(x) = h(x, f_t, p_t, \theta_t)$ mit $f_t, p_t, \theta_t$ von $\epsilon_t$;
**Aktualisiere** die Gewichte

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

mit $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ und $e_i = \begin{cases} 0 & x_i \text{ korrekt klassifiziert} \\ 1 & \text{sonst} \end{cases}$

**Ende**

**Ergebnis** *der entstandene starke Klassifikator* $C(x)$ *lautet*

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \textit{sonst} \end{cases}$$

*mit* $\alpha_t = \log \frac{1}{\beta_t}$

Lookup-Tabelle effektiv überprüft. Ein *Integralbild* zu einem Punkt $P(x_p, y_p)$ ist dabei als die Summe aller oberhalb und links von $P$ liegenden Pixel im Bild $I$ definiert[4]:

$$I_I(x_p, y_p) = \sum_{x_i<x_p, y_i<y_p} I(x_i, y_i)$$

Verwendet man nun die kumulative Zeilensumme $s$, erhält man mit

$$s(x, y) = s(x, y-1) + I(x, y) \tag{2.1}$$
$$I_I(x, y) = I_I(x-1, y) + s(x, y) \tag{2.2}$$

unter den Vorbedingungen $s(x, -1) = 0$ und $I_I(-1, y) = 0$ einen effektiven Algorithmus, der die Berechnung aller Werte im Integralbild in einem Durchlauf ermöglicht. Aus dem Beispiel (siehe Abbildung 2.2) ist schnell ersichtlich, dass zur Berechnung der Summe aller Pixel eines beliebigen Rechtecks $R(x, y, h + x, w + y)$ im Bild $I$ genau vier Lookups

$$A_R = I_I(x-1, y-1) + I_I(x+w-1, y+h-1) - I_I(x-1, y+h-1) - I_I(x+w-1, y-1)$$

benötigt werden und die benötigte Zeit damit konstant ist. Mit Hilfe dieser Integralbilder werden nun die Haar-ähnlichen Merkmale effektiv berechnet - für die Merkmale vom Typ $(A)$ und $(B)$ zum Beispiel als Differenz zweier rechteckiger Bereiche mit nur sechs Lookups.

Lienhart und Maydt führen in [31] zusätzlich noch das Integralbild für gedrehte Rechtecke (engl. *„RSAT = rotated summed area table“*) ein, um damit auch die zusätzlichen um 45° rotierten Merkmale zu berechnen:

$$I_{I,45^\circ}(x, y) = \sum_{x' \le x, x' \le x-|y-y'|} I(x', y')$$

Die Berechnung dieser Lookup-Tabelle erfolgt dann auf Basis des Integralbilds in zwei Durchläufen unter der Vorbedingung

$$I_{I,45^\circ}(-1, y) = I_{I,45^\circ}(-2, y) = I_{I,45^\circ}(x, -1) = 0$$

[4] Zur Darstellung muss dieses Integralbild noch auf das verwendete Farbformat von $n$ Bit normiert werden: $I_I^{(n)}(x_p, y_p) = \frac{I_I(x_p, y_p)}{\max I_I(x_p, y_p)} \cdot (2^n - 1)$

mit

$$I_{I,45^\circ}(x,y) = I_{I,45^\circ}(x-1,y-1) + I_{I,45^\circ}(x-1,y) + \\ +I(x,y) - I_{I,45^\circ}(x-2,y-1) \quad (2.3)$$

$$I_{I,45^\circ}(x,y) = I_{I,45^\circ}(x,y) + I_{I,45^\circ}(x-1,y+1) - I_{I,45^\circ}(x-2,y) \quad (2.4)$$

wobei 2.3 dabei einem Durchlaufen des Integralbilds von links nach rechts und oben nach unten entspricht und 2.4 dem entgegengesetzten Verarbeiten von rechts nach links und unten nach oben. Anschließend können auch um 45°gedrehte Rechtecke durch Ablesen von vier Werten

$$A_{R,45^\circ} = I_{I,45^\circ}(x+w,y+w)+I_{I,45^\circ}(x-h,y+h)-I_{I,45^\circ}(x,y)-I_{I,45^\circ}(x+w-h,y+w+h)$$

berechnet werden.

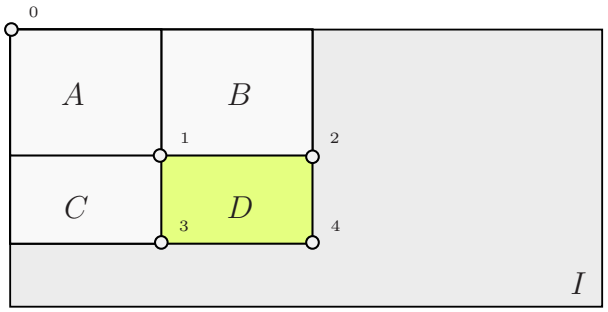



Abbildung 2.2: Integralbild und seine Anwendung

**Beispiel** Das Integralbild $I_{I,4}$ für Punkt 4 ist die Summe der Pixel über alle vier Rechtecke $A, B, C, D$. Somit ergibt sich die Summe der Pixel in Rechteck $D$ zu $I_{I,4} - I_{I,2} - I_{I,3} + I_{I,1}$. [a]

[a] $I_{I,1}$ muss hier erneut addiert werden, da es sowohl in $I_{I,3}$ als auch in $I_{I,2}$ enthalten ist.

Viola und Jones konnten zeigen, dass für $T = 200$ Merkmale bereits brauchbare Ergebnisse erreicht werden konnten. Bei einer Erkennungsrate von $D = 95\%$ wurde gerade eins von 14.084 Bilder fälschlicherweise als Gesicht erkannt ($p_f \approx 7,1 \cdot 10^5$). Dies ist unter realen Bedingungen aber leider noch nicht ausreichend. Der auf der Hand liegende Ansatz einer weiteren Verbesserung des Klassifikators durch Hinzufügen zusätzlicher schwacher Lerner ist relativ einfach möglich, wodurch allerdings die Laufzeit des Verfahrens weiter ansteigen würde.[5]

### 2.3.3 Ein neuer Ansatz - die Haar-Kaskade

Das prinzipielle Funktionieren der Verwendung eines geboosteten Klassifikators auf Merkmalsbasis war damit nachgewiesen, nur wurde eine praktischere Implementie-

[5] David Wolpert nannte dieses Problem der Verbesserung eines Merkmals durch Verschlechterung eines anderen in [67] „no-free-lunch-theorem“, nach der amerikanischen Redensart *„there ain't no such thing as a free lunch“* - es gibt kein geschenktes Mittagessen.

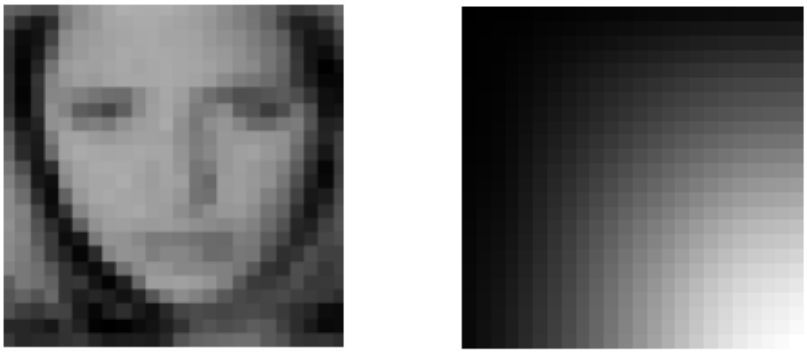


Abbildung 2.3: Beispielgesicht (links) und zugehöriges Integralbild (rechts) bei 24x24 Pixeln

rung mit viel höherer Genauigkeit benötigt. Viola und Jones änderten daher das Verfahren nochmals ab: Statt aufwendig nach Bereichen zu suchen, die ein Gesicht enthalten, wurde der Algorithmus schlicht umgedreht. Es wird daher in der praktischen Implementierung des Haar-Klassifikators (Abbildung 2.4) nach Bereichen gesucht, die mit großer Wahrscheinlichkeit kein Gesicht enthalten. Viele solcher starken Klassifikatoren wurden dann in einer Kaskade nacheinander abgelegt (engl. *„rejection cascade“*)[6]. In jeder Stufe $F_n$ dieser Kaskade führt das Resultat „Kein Gesicht“ direkt zum Abbruch der gesamten Kaskade und wertet die gesamte untersuchte Region als „Kein Gesicht“. Dieses Abweisen (engl. *„rejection“*) und die Anordnung der einzelnen Klassifikatoren - einfach-strukturierte Klassifikatoren zu Beginn der Kaskade, komplexe gegen Ende - führt zu einer deutlichen Beschleunigung der Codeausführung.

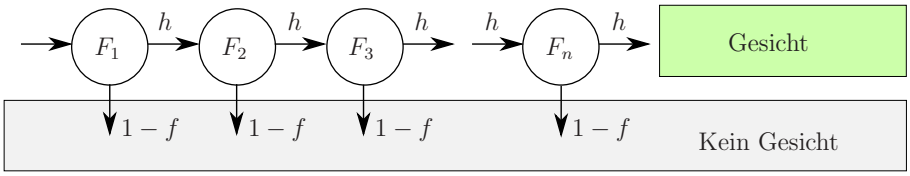



Abbildung 2.4: Haar-Klassifikator „abweisende Kaskade“

[6]Diese Kaskade kann man sich auch als einen stark unbalancierten (degenerierten) Baum vorstellen.

Für eine Kaskade mit $n$ Stufen gilt

$$F = \prod_{i=1}^{n} f_i \tag{2.5}$$

$$D = \prod_{i=1}^{n} d_i \tag{2.6}$$

für die Falsch-Positiv-Rate $F$ und die Erkennungsrate $D$. Betrachten wir diese Erkennungsraten nun als Wahrscheinlichkeiten, wird der Vorteil der Kaskadenanordnung schnell deutlich. Bei einer Erkennungsrate von $d_i = 99\%$ wird in einer zehnstufigen Kaskade eine gesamte Erkennungsrate von $D = 0,99^{10} \approx 90\%$ erreicht. Die fälschliche Erkennung eines Gesichts aber verringert sich deutlich: bei einer Falsch-Positiv-Rate von $d_i = 30\%$ ist die gesamte Falsch-Positiv-Wahrscheinlichkeit nur noch $D = 0,3^{10} \approx 6 \cdot 10^{-6}$.
Das von Viola und Jones gewählte endgültige Lernverfahren (Algorithmus 2.2) ermöglicht es dem Nutzer, zu Beginn des Lernverfahrens die maximale Falsch-Positiv-Rate $f$ und minimale Erkennungsrate $d$ festzulegen. Der Algorithmus erhöht dann automatisch die Anzahl der schwachen Klassifikatoren, bis die gewünschten Raten $f_i$ und $d_i$ mit den Testdaten erreicht werden. Anschließend wird der starke Klassifikator mit einem Validierungsdatensatz[7] überprüft. Werden die gewünschten Erkennungsraten $f_i$ und $d_i$ dann noch nicht erreicht, wird eine weitere Stufe in die Kaskade eingefügt.

### 2.3.4 Vergleich zweier Beispielimplementierungen

Im Folgenden sollen zwei Realisierungen des Haar-Klassifikators miteinander verglichen werden. Als Referenz dient eine in C geschriebene native Win32-Anwendung unter Verwendung der OpenCV-Bibliothek (siehe C.1). Anschließend werden Güte und Geschwindigkeit mit einer *managed code*-Implementierung in IronPython unter Verwendung der Emgu-Bibliothek verglichen (siehe C.2), da diese die Zielimplementierung darstellen soll.

#### Parametrisierung

Die OpenCV-Implementierung des Haar-Klassifikators stellt eine Vielzahl verschiedener Parameter zur Verfügung, die Einfluss auf die Güte und Geschwindigkeit des Ergebnisses haben. So gibt der Parameter `scaleFactor` an, wie stark der Klassifikator in jeder Iteration skaliert wird (Standardwert ist 1.1, eine Erhöhung sollte

[7] Validierungsdatensatz, vgl. A.2.7

**Algorithmus 2.2:** Lernalgorithmus der Haar-Kaskade

**Vorgabe** *maximale Falsch-Positiv-Rate $f$, minimale Erkennungsrate $d$ für jede Stufe*;
**Vorgabe** *Gesamt-Falsch-Positiv-Rate $F$*;
**Vorgabe** *Menge der Positivbeispiele $P$, Menge der Negativbeispiele $N$*;

**Setze** $f_0 = 1.0$, $d_0 = 1.0$

**solange** $f_i > F$ **tue**
    $i \leftarrow i + 1$ ;
    $n_i = 0; f_i = f_{i-1}$ ;
    **solange** $f_i > f \cdot f_{i-1}$ **tue**
        $n_i \leftarrow n_i + 1$ ;
        **Verwende** *$P$ und $N$, um den Klassifikator mit $n_i$ schwachen Lernen für die aktuelle Stufe zu Trainieren*;
        **Validiere** *kompletten Kaskadenklassifikator mit dem Validierungsdatensatz um $f_i$ und $d_i$ zu bestimmen*;
        **Verringere** *den Schwellwert des $i$-ten Klassifikators bis der starke Klassifikator der aktuellen Stufe mindestens eine Erkennungsrate von $d \cdot d_i$ hat (dies ändert auch $f_i$)*
    **Ende**
    $N \leftarrow \emptyset$ ;
    **wenn** $f_i > F$ **dann**
        **Validiere** *die aktuelle Stufe mit den Kein-Gesicht-Datensätzen und füge alle fälschlich als Gesicht erkannten Bilder in $N$ ein*
    **Ende**
**Ende**

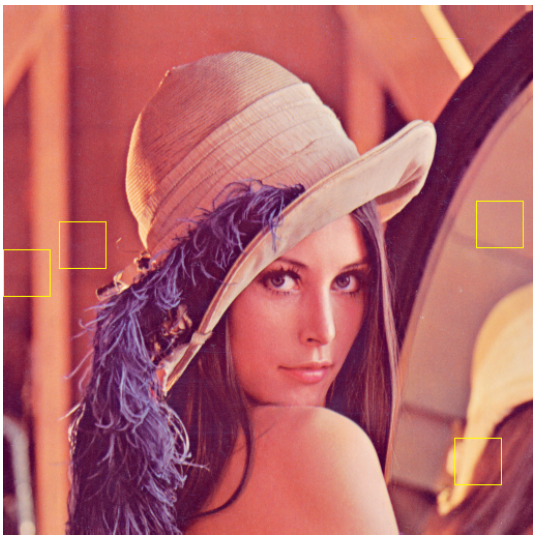
das Verfahren beschleunigen). Mit `minNeighbors` wird dem Klassifikator mitgeteilt, wieviele Objekte in direkter Pixelnachbarschaft eines gefundenen Objekts auftreten müssen, um den Bereich tatsächlich als ein solches Objekt zu werten. Dies führt zu einer Verbesserung der Güte des Verfahrens, da ein falsch-positiver Treffer nicht gehäuft auftreten sollte und so unterdrückt werden kann (Standardwert ist 3, eine Erhöhung sollte die Anzahl der gefunden Gesichter verringern). Mit `minSize` kann angegeben werden, welche Größe ein Objekt in Pixeln mindestens haben muss, um gefunden zu werden. Über die Bitmaske `flags` schließlich kann das Verfahren selbst gesteuert werden. Es sind die Schalter `DO_CANNY_PRUNING` (schließt vor Anwendung des Klassifikators irrelevante Bildbereiche mit zu vielen und zu wenig Kanten aus), `SCALE_IMAGE` (skaliert anstelle des Klassifikators das Bild), `FIND_BIGGEST_OBJECT` (gibt nur das größte gefundene Objekt zurück) und `DO_ROUGH_SEARCH` (führt eine grobe Suche aus, die abbricht, sobald in einem Bildbereich ein Objekt gefunden wurde) verfügbar - standardmäßig ist allerdings kein Schalter aktiv.
OpenCV liefert bereits eine Vielzahl verschiedener angelernter Klassifikatoren mit. Für die Untersuchung wurde der *„tree-based 20x20 gentle adaboost frontal face detector“* `haarcascade_frontalface_alt2.xml` von Rainer Lienhart verwendet. Da in der gewünschten Anwendung mit einem Livestream gearbeitet werden soll, wird die Auflösung der verwendeten Testbilder auf 640x480 px begrenzt. Es wird die Laufzeit und die Güte für die Parameter `minNeighbors` (untersuchte Werte: 0, 1, 2, 3) , `scaleFactor` (Werte: 1.1, 1.3, 1.5) und `minSize` (Werte: 10 bis 90 in Zehnerschritten) ausgewertet. Die Messungen werden für die Schalter `DO_ROUGH_SEARCH`, `DO_CANNY_PRUNING` und `SCALE_IMAGE` ausgeführt. Untersucht werden die Testbilder `lena` [B], `g20-09` [B] und `nmun-08` [B].

### Ergebnis

Zur Durchführung der Tests wurde der automatische Testskript `runner.py` erstellt, der einen Testlauf für alle Kombinationen aller definierten Parameter startete und die Messergebnisse in CSV-Dateien (Messzeit) und PNG-Dateien (erkannte Gesichter) ablegte. Anschließend wurde in allen so erzeugten Bildern manuell die Anzahl der falsch negativen und falsch positiven Ergebnisse bestimmt und die Messergebnisse in einer Exceldatei zusammengeführt. Alle Testergebnisse sind im Projektverzeichnis gepackt abgelegt. Durchgeführt wurde der Test auf einem Intel Core2Duo (2x2800MHz) mit 4 GB RAM unter Windows 7 (64-bit). Da es sich dabei um ein Multitasking-Betriebssystem handelt, kann eine Ausführung eines Testlaufs „am Stück“ nicht garantiert werden. Die durch die erzwungenen Unterbrechungen einhergehenden Messfehler der Laufzeit können nicht sicher bestimmt werden, weshalb die gemessenen Laufzeiten nicht direkt miteinander verglichen werden sollen, son-

dern lediglich die Tendenzen ausgewertet wurden. Die entsprechenden Diagramme sind in Abschnitt D abgelegt.

(a) Lena (1.5;3;50)



(b) Lena (1.5;1;20), ausschließlich false-positives durch ungünstige Parameter

(c) G20 (1.5;1;40), Duplikate und nicht erkannte Profilaufnahmen

(d) NMUN08 (1.1;1;40), false-positives und nicht erkannte verdrehte Gesichter

Abbildung 2.5: Darstellung verschiedener Resultate, Parameter in Klammern (scalingFactor;minNeighbors;minSize)

Als Ergebnis konnte festgestellt werden:

- Die Laufzeiten für das Laden des zu untersuchenden Bildes und der Haarkaskade unterscheiden sich nicht signifikant zwischen Emgu- und C-Implementierung. Die Laufzeit für die Erkennung mit der Haarkaskade ist in der Emgu-Variante im Mittel 20ms langsamer. Die Gesamtlaufzeit eines Tests allerdings dauert in der *managed-code*-Variante circa 1,5s länger, was auf die Natur des Verfahrens zurückzuführen ist (Laden des Programms in die .NET-Laufzeitumgebung).

- Wird der Parameter `minNeighbors` auf Null gesetzt, treten extrem viele Duplikate auf. In der weiteren Untersuchung wurde dieser Wert daher ausgelassen.

- Die eigentliche Wahl des Parameter `minNeighbors` (wenn >0) hat keine signifikante Auswirkung auf die Güte und Laufzeit des Verfahrens.

- Der Parameter `scaleFactor` hat deutliche Auswirkungen auf die Laufzeit und Güte des Algorithmus. Ein hoher Wert (im Test 1,5) führte zu deutlich weniger false-positives als ein kleiner Wert (im Test 1,1). Gleichzeitig erhöhte sich dadurch allerdings auch die Anzahl der nicht erkannten Gesichter (false-negatives). Die Laufzeit verringerte sich mit steigendem `scaleFactor`, was sich über die geringere Anzahl von nötigen Rechenschritten erklären lässt.

- Der Parameter `minSize` hat deutliche Auswirkung auf Laufzeit und Güte. Erwartungsgemäß ist die Laufzeit indirekt proportional, da sie die Anzahl der zu untersuchenden Bildausschnitte bestimmt. Je größer der kleinste zu untersuchende Bildausschnitt ist, desto weniger Bildausschnitte müssen untersucht werden und desto schneller ist der Algorithmus. Wird die `minSize` größer als die im Bild vorhandenen Gesichter gewählt, werden keine Gesichter erkannt. Wird `minSize` im Gegenzug allerdings deutlich kleiner als die vorhandenen Gesichter gewählt, treten viele false-positives auf.

- Die besten Ergebnisse (wenige oder keine false-positives, wenige nicht erkannte Gesichter) erzielt das Verfahren dann, wenn der Parameter `minSize` sich in der Nähe der Größe der im Bild vorhandenen Gesichter bewegt.

- Ungünstige Parameter führten zu Duplikaten (mehrfache Erkennung ein und des selben Gesichts)

- Das Flag `DO_ROUGH_SEARCH` führte zu keiner signifikanten Beschleunigung des Algorithmus, teilweise war die Abarbeitung sogar langsamer.

- Das Flag `SCALE_IMAGE` führt in der Emgu-Variante zu einer deutlichen Verlangsamung der Haar-Kaskade (circa um Faktor 2), vor allem für die beiden Bilder `g20-09` und `nmun-08`, deren Größe sich im Bereich der gewünschten Anwendung befindet.

- Das Verfahren erkennt Gesichter verschiedener Hautfarben zuverlässig. Weiterhin funktioniert es auch bei teilweiser Verdeckung des Gesichts (zum Beispiel durch Kopfbedeckungen oder Brillen).

- Das Verfahren hat Probleme, verdrehte Gesichter oder Profilaufnahmen sicher zu erkennen, was wahrscheinlich der zu Grunde liegenden Trainingsdatenbank mit Frontalaufnahmen geschuldet ist.

Daraus lassen sich folgende Schlussfolgerungen für die Implementierung ziehen:

- Das Verfahren ist für die gewünschte Anwendung praktikabel, da es für die untersuchten Bilder für kleine Parameter `scaleFactor` und `minFaceSize` in allen Tests weniger als 550ms benötigte, für auf die Szene optimierte Parameter sogar nur um die 100ms. Die Emgu-Variante kann eingesetzt werden, da die beobachteten Geschwindigkeitsdefizite in der vorgesehenen Implementierung durch einmaliges Vorladen und im Speicher Halten des Objekts vermieden werden können.

- Duplikate können in der gewünschten Anwendung ignoriert werden, da sie in der Weiterverarbeitung durch Beschneiden des Bildes und der nachfolgenden Gesichtswiedererkennung gleiche Ergebnisse liefern. Ebenso müssen falsepositives in der Nachverarbeitung erkannt werden, was beispielsweise durch die unter 2.5 beschriebene Vorverarbeitung passieren kann.

- Der Parameter `minSize` ist in der Anwendung parametrierbar zu gestalten, um eine Anpassung an die Aufnahmeszene vornehmen und die Güte zur Laufzeit optimieren zu können.

- Die Flags `SCALE_IMAGE` und `DO_ROUGH_SEARCH` werden nicht verwendet. Für die Bilderfassungsroutine während der Lernphase des Eigenface-Moduls wird das Flag `FIND_BIGGEST_OBJECT` verwendet, um hier mögliche kleine falsepositives auszuschließen.

### 2.3.5 Zusammenfassung

Die Haar-Klassifikator hat sich als sehr guter Algorithmus erwiesen, um ein Bild schnell nach Gesichtern zu durchsuchen. Mit einer einfachen Vorverarbeitung des Bildes (adaptiver Histogrammausgleich, siehe A.4.4) ist das Verfahren außerdem unempfindlicher für Helligkeits- und Kontrastschwankungen der verwendeten Bilder. Das Verfahren erzielt eine hohe Erkennungsrate bei gleichzeitig geringer Falsch-Positiv-Rate. Auch können teils verdeckte Gesichter noch zuverlässig erkannt werden. Durch seine Natur ist das Verfahren nicht nur auf Gesichter beschränkt, vielmehr können beliebige feste Objekte mit einer Vorzugsbetrachtungsrichtung angelernt und erkannt werden (siehe dazu [53]). OpenCV bringt bereits mehrere vollständig trainierte Haar-Kaskaden für die Erkennung von Gesichtern (frontal und im

Profil), Gesichtsteilen (Augen, Mund, Nase) und auch kompletten Menschen mit und kann somit direkt ohne Training eingesetzt werden.
Nachteilig am Verfahren ist die zeitaufwändige Trainingsphase. Je nach gewünschter Qualität des Klassifikators müssen mehr als 5.000 Positiv- und noch deutlich mehr Negativbeispiele angelernt werden. Für das Gebiet der Gesichtserkennung stehen dafür jedoch vielfältige freie Bibliotheken zur Verfügung (Tabelle 2.2.2). Weiterhin stellten Profilaufnahmen oder schräge Aufnahmen das ursprüngliche Verfahren nach Viola/Jones vor Probleme. Hierfür wurden allerdings mittlerweile weiterführende Arbeiten veröffentlicht (z.B. [71], [64]), die sich dieser Probleme annehmen.
Gute Ergebnisse, vor allem auf rechenschwacheren Systemen wie Handys, konnten mit dem stark verwandten LBP-Verfahren (engl. *„local binary patterns“*) erreicht werden. Entsprechende Arbeiten ([14], [24], [3]) sollten vor Einsatz des Verfahrens in diesem Hardwarebereich konsultiert werden.

## 2.4 Gesichtswiedererkennung (face recognition)

### 2.4.1 Eigengesichter als PCA-Verfahren

Für die Aufgabe der Gesichtswiedererkennung sind viele Verfahren beschrieben worden[8]. Ein sehr einfaches Verfahren stellt hier das der *Eigengesichter* (engl. *„eigenfaces“*) dar, das erstmalig von Turk und Pentland 1991 in [61] beschrieben wurde. In ihm wird die These betrachtet, dass nur bestimmte *„lokale und globale Merkmale“* eines Bildes wichtige Informationen zur Identifizierung eines Gesichts liefern. Das Ziel ist es also, eine reduzierte Darstellung eines Gesichts mit hohem charakteristischen Informationsgehalt zu finden.[9] Das Verfahren selbst stellt eine Anwendung der *Hauptkomponentenanalyse*[10] dar und wird in verschiedenen Zusammenfassungen zur Gesichtserkennung[11] den Aussehens-basierten (engl. *„appearance based“*) oder auch ganzheitlichen[12] Ansätzen (engl. *„holistic approaches“*) zugeordnet. Ursprünglich sollte das Verfahren zur Rekonstruktion von Gesichtern verwendet werden - mit vorhandener Gesichtsbasis (dem Eigengesicht) sollten Gesichter aus den Differenzbildern wieder hergestellt werden. Eine erste Beschreibung des Ansatzes der Dimensionsreduzierung über sogenannte *Eigenpictures* findet sich bereits in [57], al-

[8] siehe auch hierzu [72]
[9] Genau dieser Ansatz ist es auch, der in der Bildkompression (z.B. bei JPEG2000) verwendet wird. Eine Erklärung dazu liefert die Tatsache, dass natürliche Bilder signifikante statistische Redundanzen enthalten, was in [50] detailliert beschrieben wird.
[10] siehe A.2.5
[11] vgl. [69], [37], [72]
[12] im Sinne von „das ganze Bild verwendende“

lerdings wird hier nur die Rekonstruktion eines Gesichts anhand des Eigenpictures beschrieben, nicht jedoch die Möglichkeit der Gesichtswiedererkennung aufgegriffen.

### Vorbetrachtungen

Um Eigengesichter anwenden zu können, müssen wir uns von der gewohnten traditionellen Bildrepräsentation als Matrix[13] verabschieden. Ein einkanaliges Bild der Größe $m \times n$ Pixel wird nun nicht als eine Matrix $I_{(m,n)}$ aufgefasst, sondern als Spaltenvektor der Dimension $m \cdot n$. Das ursprüngliche Format muss daher zusätzlich gespeichert werden, um eine Rekonstruktion zu ermöglichen.

### Mathematische Beschreibung des Verfahrens

Es seien $\Gamma_1, \Gamma_2, ..., \Gamma_M$ Trainingsbilder eines Gesichts in vektorieller Schreibung. Aus diesen Bildern wird der Mittelwert $\Psi$ - das *Durchschnittsgesicht* - über

$$\Psi = \frac{1}{M} \sum_{i=1}^{M} \Gamma_i$$

errechnet. Unter Verwendung von $\Psi$ wird anschließend für jedes Trainingsbild ein zugehöriges Differenzbild $\Phi_i$ errechnet

$$\Phi_i = \Gamma_i - \Psi \qquad \forall i \in \{1, ..., M\}$$

Aus diesen Differenzbildern $\Phi_i$ wird anschließend mit $A = [\Phi_1 \Phi_2 ... \Phi_M]$ die Kovarianzmatrix $C$ erstellt

$$C = \frac{1}{M} \sum_{i=1}^{M} \Phi_i \Phi_i^T = AA^T$$

Die Matrix C hat dabei die Dimension $(m \cdot n) \times (m \cdot n)$, was ein Berechnen der $m \cdot n$ Eigenvektoren rechnerisch sehr aufwändig macht. Allerdings ist die Anzahl der signifikanten Bildinformationen (linear unabhängiger Eigenvektoren) durch die geringe Anzahl der Trainingsbilder - es gilt $M \ll (m \cdot n)$ - beschränkt, weshalb wir uns Dank der zum Einsatz kommenden Hauptkomponentenanalyse auf $M$ Eigenvektoren beschränken können. Zur Bestimmung dieser Eigenvektoren verwenden wir die Matrix $L = A^T A$ und setzen

$$\begin{aligned} A^T A v_i &= \mu_i v_i \\ A A^T A v_i &= \mu_i A v_i \end{aligned} \tag{2.7}$$

[13] siehe A.4.1

Dabei sind $v_i$ Eigenvektoren von $L$. Es folgt, dass $Av_i$ ein Eigenvektor von $C$ ist. Aus den $M$ Eigenvektoren von $L$ kann somit auf die wichtigen $M$ Eigenvektoren von $C$ geschlossen werden

$$u_l = \sum_{k=1}^{M} v_{lk}\Phi_k \qquad l = 1, ..., M$$

Diese Eigengesichter $u_l$ spannen dabei den sogenannten *Gesichtsraum* (engl. *„face space“*) auf. Die Relevanz eines Eigenvektors und des damit verbundenen Eigengesichts ist dabei von der Ordnung des Eigenvektors abhängig.

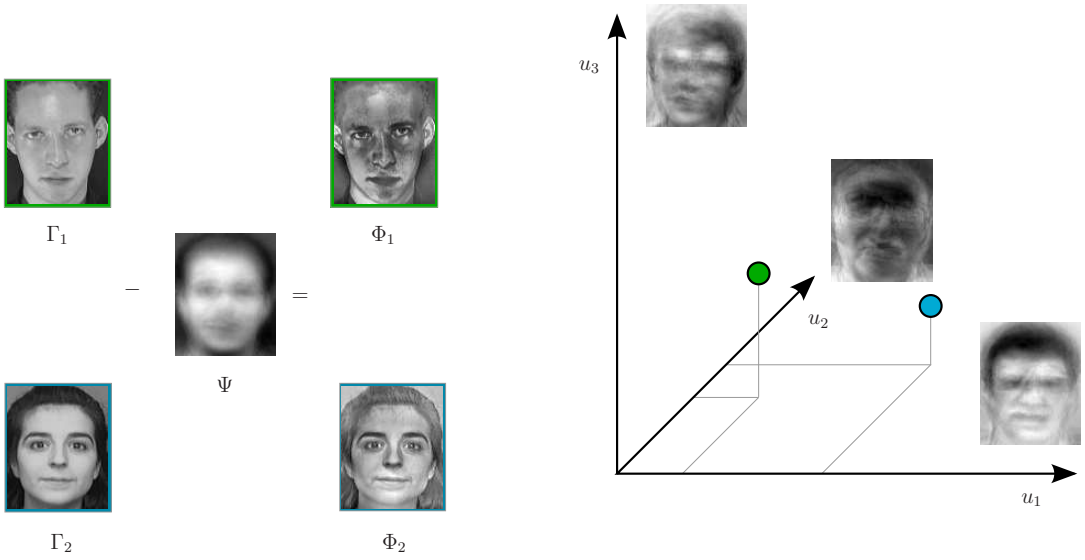



Abbildung 2.6: Projektion zweier Trainingsbilder $\Gamma_1, \Gamma_2$ in den Gesichtsraum aufgespannt aus drei Eigengesichtern $u_1, u_2, u_3$

### Vergleich Eigengesicht mit unbekanntem Bild

Um nun eine Aussage über ein dem Klassifikator unbekanntes Bild $\Gamma_u$ treffen können, überführen wir zunächst alle zentrierten Trainingsbilder $\Phi_i$ in ihre Eigengesichtskomponenten[14]:

$$\Omega_i = u^T\Phi_i \qquad \forall i \in \{1, ..., M\}$$

Wir erhalten für jedes Bild einen Spaltenvektor $\Omega_i$, den man sich als Koordinate in einem $M$-dimensionalen Raum vorstellen kann[15]. Man kann sich die einzelnen Komponenten des Vektors aber auch als Gewichte der jeweiligen Eigengesichter vorstellen ($\Omega_{i,1}$ als Anteil des ersten Eigengesichts usw.), in deren Summe sich die Rekonstruktion des Bildes ergibt.

[14] Man spricht auch von „in den Gesichtsraum projizieren“.
[15] vgl. Abbildung 2.6 für $M = 3$

Für eine Wiedererkennung wird daher das Bild $\Gamma_u$ ebenfalls in seine Eigengesichtskomponenten überführt

$$\Omega_u = u^T(\Gamma_u - \Psi)$$

und mit einem beliebigen Mustererkennungsalgorithmus die „Nähe“ von $\Omega_u$ zu allen projizierten Gesichtern $\Omega_i$ im Gesichtsraum untersucht. Als einfachste Implementierung kann der euklidische Abstand[16] verwendet werden:

$$\epsilon_i = \|(\Omega_i - \Omega_u)\|$$

Das Testbild $\Gamma_u$ gehört dann zu der Personen $P$, zu deren Eigengesichtsprojektion der kleinste Abstand besteht

$$P = \arg\min_k \epsilon_k$$

Über bestimmte Schwellwerte des Abstands können weiterhin die Ergebnisse „kein bekanntes Gesicht“ und „kein Gesicht“ ermittelt werden. Diese Werte sind allerdings systemimmanent und müssen speziell für den Anwendungsfall bestimmt werden.

Abbildung 2.7: Durchschnittsgesicht (oben links) und Eigengesichter (spsoDB Trainingsdatenbank mit je 5 Bildern pro Person)

[16] Weitere Distanzmaße sind in Anhang A.2.7 *Maschinelles Lernen* zu finden.

### Dimension des Gesichtsraums

Die Dimension des Gesichtsraums entspricht der Anzahl der verwendeten Eigengesichter. In der ersten Beschreibung des Verfahrens wurde diese mit $\dim u = M$ angegeben. Pentland und Turk konnten allerdings bereits in [61] weiterführend zeigen, dass sogar nur $M' = M - p$ Eigengesichter für eine vollständige Basis des *Gesichtsraums* benötigt werden, wobei $p$ die Anzahl der unterschiedlichen Personen in der zu Grunde liegenden Trainingsdatenbank war. In der weiterführenden Arbeit [45] wird die Frage detailliert behandelt und weitere Implementierungshinweise gegeben.
Interessant ist allerdings, dass die Ordnung der Eigengesichter recht gut mit den Details der Trainingsgesichter[17] korreliert (siehe Abbildung 2.8). Daraus kann der Schluss gezogen werden, dass für weniger unterschiedliche Gesichter weniger Eigengesichter zur Wiedererkennung benötigt werden als für recht ähnliche Gesichter, die sich nur in Details unterscheiden.

### Untersuchung des Algorithmus

Zur Untersuchung des Eigengesichtsverfahrens wurde eine Beispielimplementierung in MATLAB vorgenommen. Die Unterteilung erfolgte dabei in logische Codeeinheiten:

1. Laden der Trainingsdaten aus einer Beschreibungsdatei (Anhang C.3),
2. Berechnung von Durchschnittsgesicht, zentrierten Trainingsbildern, Eigengesichtern und projizierten Gesichtern (Anhang C.4),
3. Erkennung eines Testbilds (Anhang C.5) und
4. Vergleich von ermittelter und erwarteter Klasse.

Als Bildmaterial wurden von drei Politikern zehn frontale Gesichtsaufnahmen aus dem Internet geladen (siehe B) und auf eine einheitliche Größe beschnitten. Anschließend wurden die Eigengesichter und die projizierten Gesichter berechnet. Die Erkennung wurde mit je einem unbekannten Bild jeder Person durchgeführt. Zur Prüfung der erhaltenen Ergebnisse wurde eine Kontrolluntersuchung mit der AT&T-Gesichtsdatenbank[18] vorgenommen. Für alle verwendeten Gesichtsdatenbanken sind die Beschreibungsdateien (`XXX-train.txt` und `XXX-test.txt`) hinterlegt, ebenso liegen Skripte zur exemplarischen Ausführung und zum automatischen Test bei.

[17] Signaltheoretisch betrachtet mit der Frequenz des Signals
[18] siehe Tabelle 2.2.2

### Ergebnisse

Die Ergebnisse zu den vorgenommenen Testläufen sind in Abschnitt E dargestellt. Für die Gesichtsdatenbank `spsoDB` konnte festgestellt werden, dass ab drei Trainingsbildern pro Personen bei Ausnutzung aller Eigengesichter 100% Treffergenauigkeit mit den Testbildern erreicht wird. Werden alle zehn zur Verfügung stehenden Testbilder verwendet, reicht es, die ersten fünf Eigengesichter zu verwenden um eine 100%-ige Erkennungsrate zu erreichen.
Die Berechnung der Eigengesichter und Projektion der Trainingsbilder in den Gesichtsraum dauerte für die 30 Trainingsbilder 280ms, die Projektion eines Testbilds und der Vergleich mit bereits projizierten Trainingsbildern pro Testbild 2ms. Da die initiale Berechnung der Eigengesichter nur bei Anlernen neuer Personen durchgeführt werden muss, ist das Verfahren auch für den gewünschten Einsatz in einem Livestream verwendbar.
Für die `att` Datenbank wurde ab sechs Trainingsbildern pro Person bei Verwendung aller 279 Eigengesichter eine Trefferquote von 90% erzielt. Wurden alle zehn Trainingsbilder pro Person verwendet, genügte es, die ersten 13 Eigengesichter zu verwenden, um ebenfalls eine Genauigkeit von 90% zu erzielen.
Für die Zielanwendung wird daher die Anzahl der aufzunehmenden Bilder pro Person und die Zahl der zu verwendeten Eigengesichter parametrierbar angelegt, um die Güte des Verfahrens konfigurierbar zu gestalten.

### Abschließende Betrachtung

Eine positive Eigenschaft des Verfahrens ist es, dass auftretende teilweise Verdeckungen im Testbild[19] - sei es durch Rauschen und Aufnahmefehler oder aber durch getragene Mützen oder Brillen - durch die rekonstruktiven Eigenschaften des Verfahrens ausgeglichen werden und trotzdem zu guten Ergebnissen führen[20].
Demgegenüber sind in der Literatur eine Vielzahl von Nachteilen des Verfahrens beschrieben, die hier erwähnt werden sollen. So beschreiben Turk und Pentland [61] selbst, dass das Verfahren anfällig für wechselnde Hintergründe ist, da nicht speziell das Gesicht codiert wird, sondern das gesamte vorliegende Bild in die Berechnung eingeht. Weiterhin sei die Genauigkeit des Verfahrens sehr stark von Position und Größe des Gesichts abhängig. In der hier implementierten Zielanwendung wird jedoch das Gesicht bereits zuvor über ein anderes Verfahren extrahiert und normiert, so dass diese Einflüsse begrenzt werden können. Abschließend empfehlen die Autoren, dass nicht nur frontale Aufnahmen als Trainingsbilder verwendet werden sollten,

[19] *„reduced sensitivity to noise“*, [72]
[20] Dies ist beobachtbar an der korrekten Klassifikation der Bilder `test-4.png` bis `test-6.png` aus der `spsoDB`-Datenbank.

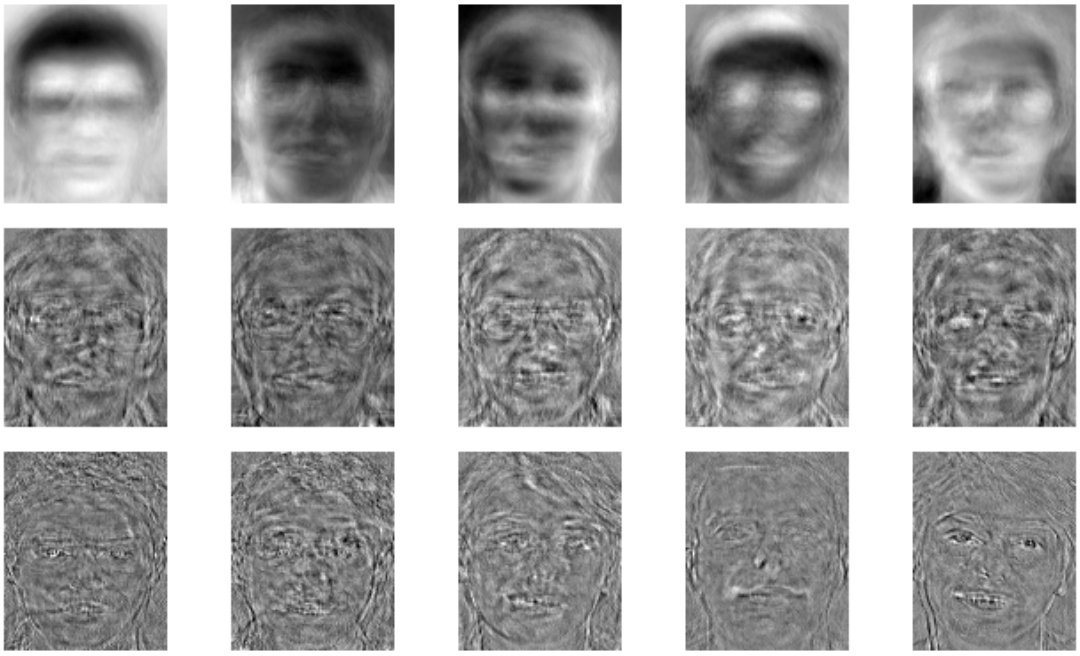


Abbildung 2.8: Zusammenhang Ordnung des Eigengesichts von der kodierten Frequenz (AT&T-Datenbank: Eigengesichter 1 - 5, 136 - 140, 355 - 359)

sondern auch leicht verdrehte Aufnahmen in die Trainingsphase eingehen sollen. Dies wurde übernommen und wird im Trainingsprogramm der Zielanwendung gefordert.

Belhumeur, Hespanha und Kriegman [7] weisen darauf hin, dass das Verfahren zu einer Projektion $W$ führt, die die gesamte Streuung (engl. *„total scatter“*) $S_T$ über die Trainingsbilder optimal abzubilden versucht:

$$\begin{aligned} S_T &= \sum_{k=1}^{M} (\Gamma_k - \Psi)(\Gamma_k - \Psi)^T \\ W_{opt} &= \arg\max_W \left| W^T S_T W \right| \end{aligned} \tag{2.8}$$

Sind die Gesichter unter stark verschiedenen Beleuchtungssituationen aufgenommen worden, geht vor allem diese Information in die Hauptkomponentenanalyse ein und die Gesichtsklassen verschmieren ineinander. Als mögliche Lösung für dieses Problem untersuchen die Autoren daher das Auslassen der drei ersten Eigengesichter, da diese hauptsächlich die Beleuchtungsunterschiede enkodierten. Es ist allerdings nicht wahrscheinlich, dass ausschließlich diese ersten Eigengesichter die Beleuchtungsunterschiede beinhalten und diese durch Auslassung entfernt werden können.

Shakhnarovich [54] schlägt zur Umsetzung der Hauptkomponentenanalyse die Lösung über eine *Singulärwertzerlegung* (engl. *„SVD = singular value decomposition“*)

vor:

$$\mathbf{X} = \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T$$

Dabei sei $\mathbf{X}$ von der Dimension $(M \times N)$ und es gelte $M \geq N$. Weiterhin seien $\mathbf{U}$ von der Dimension $(M \times N)$ und $\mathbf{V}$ von der Dimension $(N \times N)$ und bestehen beide aus orthonormalen Spaltenvektoren. Per Definition der SVD ist ein Singulärwert von $\mathbf{X}$ die Quadratwurzel eines Eigenwerts von $\mathbf{X}\mathbf{X}^T$. Weiterhin sei $\mathbf{U} = u$, womit die SVD eine Berechnung der Gesichtsraums ohne die aufwändige Berechnung der Kovarianzmatrix ermöglicht. Abschließend weist auch Shakhnarovich nochmals eindringlich auf das Problem der Klassenähnlichkeit durch Helligkeitsunterschiede bei Eigengesichtern hin.
Moghaddam und Pentland wählen in [40] einen wahrscheinlichkeitstheoretischen Ansatz, um den erhaltenen Gesichtsraum in den Hauptunterraum (engl. *„principal subspace“*) $F$ und sein Komplementät $\overline{F}$ aufzuteilen und so die Beleuchtungseinflüsse aus dem verwendeten Gesichtsraum zu entfernen.
Den am häufigsten verwendeten Ansatz stellt allerdings das *Fishergesichter*-Verfahren dar, das im folgenden Abschnitt genauer betrachtet werden soll.

### 2.4.2 Fisherfaces als Erweiterung unter Verwendung der LDA

#### Einleitung

Die Grundlage für das *Fishergesichter* (engl. *„fisherfaces“*) genannte Verfahren wurde durch Swets und Weng in [59] mit der These gelegt, dass das Resultat einer Hauptkomponentenanalyse die sogenannten „ausdrucksstärksten Merkmale“ (engl. *„most expressive features (MEF)“*) sind, welche allerdings hauptsächlich aus Beleuchtungsunterschieden resultierten. Besser wäre es daher, die „am stärksten trennenden Merkmale“ (engl. *„most discriminating features (MDF)“*) zu finden. Diese Idee wurde in [7] aufgegriffen, in dem den bereits beschriebenen Beleuchtungsproblemen über den Ansatz des *Lambertschen Kosinusgesetzes* [66] eine theoretische Betrachtung beigefügt wurde. Es ist damit möglich, die Abhängigkeit der Lichtstärke eines ideal diffus reflektierenden Flächenstücks (als *Lambert-Fläche* bezeichnet) vom Betrachtungswinkel $\theta$ zu

$$I(\theta) = A\cos(\theta)L$$

beschreiben. Übertragen auf die Gesichtserkennung folgerten Belhumeur et al., dass alle Bilder einer Lambert-Fläche in ein und dem selben dreidimensionalen, linearen Unterraum des Gesichtsraums liegen müssen, wenn sie von einem festen Betrachtungspunkt aus aufgenommen wurden. Gelingt es also, die durch die Beleuchtung

entstehende Streuung innerhalb einer Klasse einzufangen, kann eine bessere Trennung der Klassen erfolgen und ebenfalls eine bessere Abbildung in den Gesichtsraum erreicht werden.

Dafür macht sich das Fishergesichterverfahren den Umstand zu Nutze, dass die Trainingsdaten bereits Klassen zugeordnet sind.[21] Damit kann das von Fisher [19] beschriebene Verfahren der *linearen Diskriminanzanalyse* (LDA) angewendet werden, um die größtmögliche Diskriminanz innerhalb einer Klasse zu erreichen.

Im Vergleich zu den Eigengesichtern[22] ändert sich das Optimierungskriterium zu

$$W = \arg\max_{W} \left| \frac{W^T S_B W}{W^T S_W W} \right|, \tag{2.9}$$

wobei die Streuung innerhalb einer Klasse (engl. *„within scatter“*) $S_W$ und die Streuung zwischen allen Datenwerten (engl. *„between scatter“*) $S_B$ definiert sind zu

$$S_B = \sum_{i=1}^{C} N_i(\sigma_i - \sigma)(\sigma_i - \sigma)^T \tag{2.10}$$

$$S_W = \sum_{i=1}^{C} \sum_{x_k \in X_i} (x_k - \sigma_i)(x_k - \sigma_i)^T \tag{2.11}$$

Dabei ist $C$ die Anzahl der verwendeten unterschiedlichen Personen bzw. die Anzahl der Klassen, $X_i$ die Menge aller $x_k$ Trainingsbilder für Klasse $i$, $N_i$ die Anzahl der Trainingsbilder für Klasse $i$, $\sigma_i$ das Durchschnittsgesicht bzw. der Mittelwert der Trainingsbilder von Klasse $i$ und $\sigma$ das Durchschnittsgesicht über alle Klassen. Damit erhalten wir zur Lösung

$$S_B w_i = \lambda_i S_W w_i \qquad \forall i = 1, 2, ..., m$$

Es existieren maximal $m = c - 1$ Eigenwerte ungleich Null, was die obere Grenze $m$ für die zu verwendenden Eigenvektoren festlegt.

### Small-sample-size-Problem

Auch das Fisherfaces-Verfahren leidet unter dem sogenannten *small-sample-size-Problem*, das die schlechte Klassifikationsgüte (engl. *„classification performance“*) auf Grund der im Vergleich zur Bilddimension sehr viel kleinere Anzahl der zur

[21] Es existieren somit $N_i$ Trainingsbilder für jede der $i$ Klassen $C_i$.
[22] vgl. Gleichung 2.8

Verfügung stehenden klassenspezifischen Trainingsdaten beschreibt[23]. Somit hängt die Separierbarkeit nicht direkt mit der Klassifikationsgenauigkeit zusammen. Weiterhin neigt das Verfahren damit zur *Überanpassung*[24] (engl. *„over-fitting“*) an die Trainingsdaten. Zur Lösung dieses Problems wird in allgemeinen Beschreibungen [7] der Fisherfaces ein klassischer PCA-Schritt zur Dimensionsreduzierung eingeführt. Andere Erweiterungen, die ohne diese Hauptkomponentenanalyse auskommen, werden im Ausblick dieses Abschnitts kurz vorgestellt sowie auf die entsprechenden Papers verwiesen.

### Umsetzung des Algorithmus

Der untersuchte Algorithmus wurde auf Grund des small-sample-size-Problems analog zum Eigenfaceverfahren implementiert. Lediglich die Einbeziehung der Klasseninformation in die Zwischenklassen- und Klassen-immanente Streuung während der Fisherface-Berechnung wurde vorgenommen (siehe Quelle C.6).

### Vergleich

Die erzielten Resultate hinterlassen einen zwiespältigen Eindruck. Deutliche Vorteile konnte das Verfahren dann erzielen, wenn die Anzahl der Trainingsgesichter einer Person groß war und viele verschiedene Klassen vorhanden waren (siehe Vergleich Eigenfaces/Fisherfaces für `att`-Datenbank im Anhang). Waren nur wenige Klassen vorhanden (`spsoDB`-Datenbank), lag die Güte der Fisherfaces unter der der Eigenfaces. Dies wurde ebenso von Martínez und Kak in [38] beobachtet, in welchem die Autoren PCA und LDA für verschiedene Datenbanken vergleichen. Dabei konnten sie nachweisen, dass entgegen allgemeiner Annahmen LDA nicht grundsätzlich performanter arbeitet als PCA. Sie stellen fest, dass LDA genau dann eine geringere Trefferquote aufweist, wenn die Trainingsdatenbank sehr klein ist - was sich mit der gemachten Beobachtung für `spsoDB` deckt.

Angemerkt werden soll an dieser Stelle noch, dass LDA zwar Merkmale mit weniger Dimensionen erzeugt und damit eine schnellere Berechnung der Distanzen in der Testphase ermöglicht, allerdings durch die zusätzliche Berechnungen langsamer in der Trainingsphase ist. In der Referenzimplementierung dieser Arbeit wurde daher auf die Implementierung der Fisherfaces verzichtet.

[23] Es existieren zwar Gesichtsdatenbanken mit sehr vielen Beispielgesichtern verschiedener Personen, jedoch sind grundsätzlich sehr viel weniger Bilder einer Person als Gesamtbilder vorhanden.
[24] vgl. Abschnitt A.2.7

### Weiterführende Arbeiten

Zur Lösung des small-sample-size-Problems sind diverse auf LDA aufbauende Verfahren beschrieben worden. In [35] greifen Lotlikar et al. die Idee einer „gebrochenen Dimensionierung“ (engl. *„fractional dimensionality“*) auf und führen die *„fractional LDA“* (F-LDA) als inkrementelles Verfahren ein. Zu diesem Zweck wird die Wichtung (engl. *„weighting function“*) $w(d)$ in

$$S_B = \sum_{k=1}^{c} \sum_{l=1}^{c} w^{(k)}(d) \left(\Psi^{(k)} - \Psi^{(l)}\right) \left(\Psi^{(k)} - \Psi^{(l)}\right)^T$$

eingeführt, wodurch Klassen, die nahe zusammen liegen und daher wahrscheinlich zu Problemen führen können, stärker in die Rechnung eingehen. Die rechenintensive Methode wurde von den Autoren leider nicht direkt auf die Gesichtswiederkennung, sondern lediglich auf recht niederdimensionale - teils sogar künstlich erzeugte - Testdaten angewendet.
Chen et al. beschreiben in [15] eine LDA-Erweiterung namens „D-LDA“. Dieses direkte LDA-Verfahren verwendet einen anderen Ansatz um das small-sample-size-Problem zu umgehen. Die Autoren konnten nachweisen, dass auch der Nullraum zwischen den Klassen diskriminative Informationen enthält und eine PCA damit unweigerlich zu Informationsverlust führen wird. Die Autoren zeigten weiterhin, dass ihr Verfahren einem Vergleichsverfahren [34] im Sinne der Erkennungsgenauigkeit, Trainingsgeschwindigkeit und Stabilität überlegen ist. Leider fand kein direkter Vergleich mit etablierten Methoden oder Gesichtsdatenbanken statt.
Allerdings verwenden Lu et al. in [36] D-LDA und F-LDA als Ausgangspunkte für ihr Hybridverfahren DF-LDA und führen auch entsprechende Tests gegen Eigenfaces und Fisherfaces durch. DF-LDA verwendet zunächst die D-LDA zur Dimensionsreduzierung und erhält so einen small-sample-size-freien Unterraum. Dafür wird das Fisher-Kriterium auf

$$W = \arg\max_{W} \frac{\left|W^T \widetilde{S_B} W\right|}{\left|W^T (S_W + \widetilde{S_B}) W\right|}$$

mit

$$\begin{aligned} \widetilde{S_B} &= \sum_{l=1}^{c} \Phi \Phi^T \\ \Phi_i &= \left(\frac{M_i}{M}\right)^{\frac{1}{2}} \sum_{j=1}^{c} \left(w(d_{i,j}) \left(\Psi_i - \Gamma_j\right)\right) \end{aligned}$$

geändert, wobei $d_{i,j}$ als der euklidische Abstand der Durchschnittsgesichter der Klassen $i$ und $j$ definiert ist. Die Wichtung $w$ muss eine Funktion sein, die schneller abnimmt, als $d_{i,j}$; die Autoren empfehlen $w(d_{i,j}) = (d_{i,j})^{-2p}$ für $p \in \mathcal{N}, p \geq 2$.

Dadurch kann die Dimension mit einem F-LDA-Schritt weiter reduziert und die diskriminative Güte zusätzlich gesteigert werden. In den durchgeführten Tests auf die `att` und `UMIST` Datenbanken konnte gezeigt werden, dass D-LDA Eigenfaces erheblich und Fisherfaces immer noch deutlich im Sinne der Klassifikationsgüte übertrifft. DF-LDA allerdings führte für alle Tests zu noch besseren Ergebnissen und bewegte sich in der `att` Datenbank ab 15 Merkmalen bei einer Erkennungsrate von 95% - für `UMIST` reichten dafür sogar lediglich vier Merkmale.

In [32] schlagen Liu und Wechsler zwei „erweiterte Fisher-Modelle“ (engl. *„enhanced fisher linear discriminant model“*) vor, die zu besseren MDF führen sollen. Mit EMF-1 versuchen sie, die Eigenwerte von $C$ mit der meisten spektralen Dichte in den Trainingsbildern zu verwenden als auch die Eigenwerte der klassen-immanenten Streumatrix $S_w$ dabei nicht gegen Null konvergieren zu lassen. In *EMF-2* wird die Diskriminanzanalyse um einen zusätzlichen *„whitening“*-Schritt erweitert. Dabei konnten die Autoren der Arbeit zeigen, dass ihre Verfahren weniger zur Überanpassung neigen und bei Tests mit der FERET-Datenbank die Erkennungsrate um 10-15% verbessern.

Aufbauend auf diesen EMF führen Liu und Wechsler in [33] den sogenannten Gábor-Fisher-Klassifikator (GFC) von (engl. *„gabor-fisher-classificator“*) ein, der die Bilddaten zuerst in einen „angereicherten Gábor-Merkmals-Vektor“ (engl. *„augmented gabor feature vector“*) überführt. Dafür wird das Gábor-Wavelet

$$\phi_{\mu,\nu}(z) = \frac{\|k_{\mu,\nu}\|^2}{\sigma^2} e^{\frac{-\|k_{\mu,\nu}\|^2 z\|^2}{2\sigma^2}} \left( e^{\mathbf{i} k_{\mu,\nu} z} - e^{-2\frac{\sigma^2}{2}} \right)$$

verwendet, wobei hier $\mu, \nu$ für Orientierung und Skalierung des Gábor-Kerns stehen, $z = (x, y)$ ist und $k_{\mu,\nu}$ den sogenannten Wavevektor (engl. *„wave vector“*) bezeichnet. Diese Wavelettransformation wird durchgeführt, da sie die Bilderfassung und -verarbeitung in Säugetierhirnen gut nachbildet. Die Autoren erreichen bei Tests mit 600 Personen aus der `FERET` Datenbank sehr gute Ergebnisse, die Eigenfaces und Fisherfaces immer übertreffen. Für $m = 62$ verwendete Merkmale wird sogar eine Erkennungsrate von 100% erreicht.

Das Gebiet der Diskriminanzanalyse ist nach wie vor ein Schwerpunkt in der Gesichtswiedererkennung. So werden zum Beispiel in [70] weiterführende verallgemeinerte Diskriminanzanalyseverfahren (engl. *„generalized discriminant analysis“*) vorgestellt und verglichen, die deutlich bessere Ergebnisse liefern als die hier betrachteten LDA-Verfahren.

## 2.5 Realisierter Erkennungsalgorithmus

### 2.5.1 Allgemeiner Aufbau

Der in dieser Arbeit zum Einsatz kommende Algorithmus besteht aus zwei Phasen: der *Trainingsphase*, in der neue Gesichter über eine Software angelernt werden, und der *Erkennungsphase*, in der in einem Live-Bild der Kamera bekannte Gesichter gesucht werden.

### 2.5.2 Trainingsphase

In der Trainingsphase werden mehrere Frontalaufnahmen eines Gesichts über eine Kamera aufgenommen. In jedem dieser Bilder wird über eine *Haar-Kaskade* das größte Gesicht gesucht und extrahiert. Zur Bildverbesserung wird die Aufnahme des Gesichts über den in [48] beschriebenen Algorithmus verbessert: Über ein ovales Fenster wird die Intensität über das Gesicht ausgeglichen, um so den Hintergrund besser ignorieren zu können. Zusätzlich wird ein Histogrammausgleich durchgeführt, um Charakteristika des Gesichts deutlich hervorzuheben. Anschließend werden in diesem Gesicht erneut über eine Haar-Kaskade linkes und rechtes Auge sowie der Mund gesucht. Anhand dieser Daten wird das Bild ausgerichtet und auf ein vordefiniertes Format zugeschnitten, so dass die Augen aller angelernten Personen auf der gleichen Höhe liegen.

Die so erhaltenen neuen Gesichter werden der Gesamttrainingsmenge hinzugefügt und mit einem eindeutigen Label für die anzulernende Person versehen. Abgeschlossen wird die Trainingsphase durch Neuberechnung der *Eigengesichter*, die als Konfigurationsdatei in der Erkennungsphase zur Verfügung gestellt werden.

### 2.5.3 Erkennungsphase

In der Erkennungsphase werden erneut über eine Haar-Kaskade alle Gesichter in den aktuell zur Verfügung stehenden Bildern gesucht. Gefundene Gesichter werden extrahiert und, wie in der Trainingsphase beschrieben, normalisiert. Die verarbeiteten Gesichter werden nun in die vorhandenen Gesichtsräume projiziert und zugeordnet. Bei erfolgreicher Erkennung einer bekannten Person wird die Position im Bild und das Label der erkannten Person zur Weiterverarbeitung im Servoing-Teil der Anwendung zur Verfügung gestellt.

## 2.6 Vorstellung weiterer Verfahren

Die in dieser Arbeit vorgestellten Methoden stellen nur einen Bruchteil der untersuchten Verfahren zur Gesichtserkennung dar. In diesem Abschnitt soll auf weitere, in diesem Zusammenhang interessante Papers verwiesen werden.

### 2.6.1 Gesichtserkennung

Rowley, Baluja und Kanade untersuchen in [49] die Verwendung *neuronaler Netze*[25] für die Erkennung aufrechter Frontalaufnahmen in Bildern. Nach Extraktion eines 20x20 Pixel großen Teilbilds und dessen Vorverarbeitung durch Beleuchtungs- und Histogrammausgleich wird es in eine 20x20 Pixel große Eingangsschicht gegeben. Daran anschließend existieren drei verschiedene Arten versteckter Schichten (engl. *„hidden layers“*), die jeweils unterschiedliche Segmente der an der Retina anliegenden Information mit ihren „receptive fields“ analysieren. So untersuchen beispielsweise

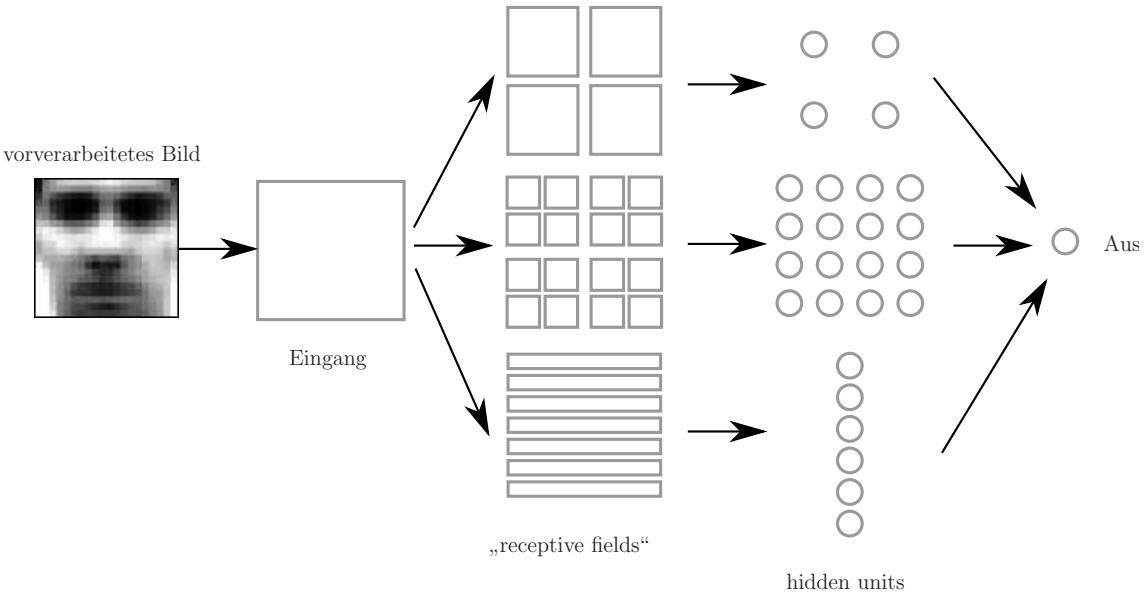


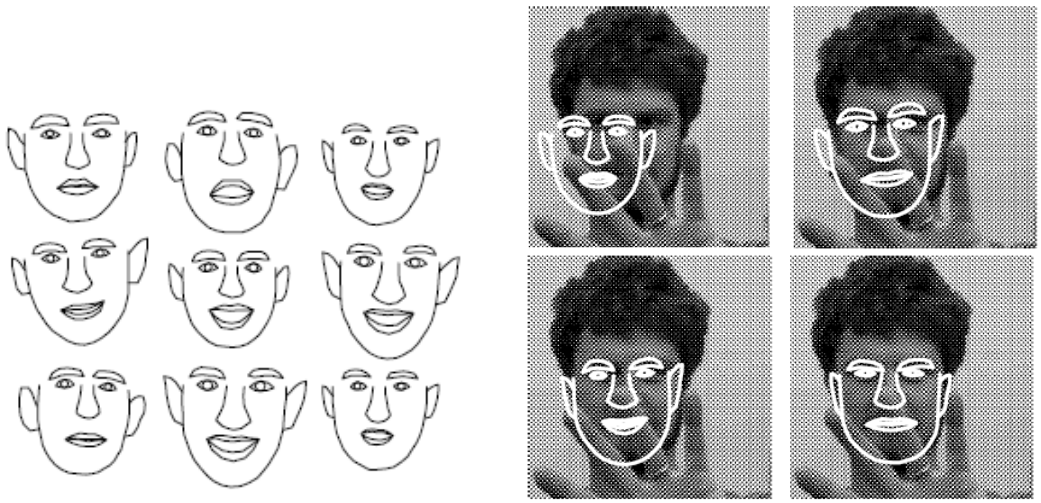

Abbildung 2.9: Gesichtserkennung über neuronales Feed-Forward-Netz

horizontale Streifen das Vorhandensein von Augen und Nase, wohingegen quadratische Ausschnitte die eigentliche Position der Augen prüfen. Es können mehrere der beschriebenen verdeckten Schichten aufeinander folgen. Die Ausgangschicht stellt eine einzelne Gleitkommazahl dar, die angibt, ob der Ausschnitt ein Gesicht enthält oder nicht. Zum Anlernen wurden Positiv-Trainingsdaten manuell vorbereitet und übereinander gelegt, Negativ-Trainingsdaten über einen Bootstrapping-Algorithmus

[25] genauer gesagt *feed-forward* Netze

aus Bildern ohne Gesichter generiert. Es erfolgten Untersuchungen zur Ausführungsgeschwindigkeit und deren Verbesserung (z.B. über Bildpyramiden, siehe Abschnitt A.4.3). Im Vergleich zu aktuellen Methoden fällt die recht kleine Trainingsdatenmenge (16000 Positivbilder, nur 9000 Negativbeispiele) auf. Die beobachtete Erkennungsrate liegt zwischen 77,9% und 90,3%. Die Untersuchungen wurden 1998 auf damals aktueller Hardware gemacht (zwei bis vier Sekunden für ein 320x240 Pixel großes Eingangsbild auf einem 200MHz SGI Indigo 2 Prozessor) und sollten zum aussagekräftigen Vergleich mit der Haar-Kaskade auf aktueller Hardware nachvollzogen werden.

Einen völlig anderen Ansatz verfolgen Lanitis et al. in [30]. Ausgehend von den durch Cootes et al. in [16] als aktive Formmodelle (engl. *„active shape models“*) bezeichneten Punktverteilungsmodellen (engl. *„PDM = point distribution models“*) beschreiben sie die Merkmale eines Gesichts durch seine groben Umrisse und verwenden *genetische Algorithmen* (engl. *„GA = genetic algorithms“*), um diese PDM auf ein im Bild vorhandenes Gesicht aufzufitten. Sie erreichten damit eine recht schnelle Erkennung von Gesichtern und konnten Personen über die Gesichtsform sogar zuverlässig wiedererkennen, sahen ihre Arbeit selbst aber nur als Grundlage für eine weitere Untersuchung der Methodik an. Interessant anzumerken ist das gute Fitting trotz teilweiser Okklusion, erkennbar in Abbildung 2.10(b).



(a) PDM von Gesichtern

(b) Fitting der PDM nach erster, zweiter, fünfter und zehnter Iteration

Abbildung 2.10: Gesichtserkennung mit Point Distribution Models und Genetic Algorithms

### 2.6.2 Gesichtswiedererkennung

Bartlett et al. stellen in [5] mit dem ICA-Verfahren (engl. *„independent component analysis“*) die Frage, ob es nicht auch Zusammenhänge höherer Ordnung zwischen den einzelnen Bildinformationen gibt, anstatt sich nur wie im PCA-Ansatz mit den kleinsten Quadraten zu beschäftigen. Es werden zwei Architekturen für diesen Algorithmus der *„unabhängigen Komponenten“* eingeführt, die auch als MATLAB -Code zur Verfügung gestellt werden. Die Autoren konnten zeigen, dass beide Architekturen in der FERET-Datenbank besser performen als die Eigenface-Methode. Eine vergleichende Studie von Sharkas und Elenien [55] aus dem Jahr 2008 untersucht Eigenfaces, Fisherfaces und ICA-Verfahren mit der AR- und AT&T-Datenbank, führt allerdings diese Analyse nicht mit Rohbilddaten, sondern mit zuvor mit diskreter Wavelettransformation (DWT) und diskreter Kosinustransformation (DCT) aufbereiten Daten aus. Ihre Ergebnisse sind nicht ganz so eindeutig: sie betrachten kein Verfahren als den anderen deutlich überlegen, stellen aber fest, dass ICA-II im Durchschnitt aller Tests leicht vorn liegt. Die Vorverarbeitung der Daten durch die DWT verbessert die Performanz des LDA-Verfahrens, die mit DCT vorverarbeiteten Daten verbessern das Ergebnis des PCA-Verfahrens. Als Schlußfolgerung empfehlen die Autoren weitere Experimente mit der Vorverarbeitung der Daten, da dies deutlicher zur Verbesserung der Performanz führe und fragen schließlich, ob Vorverarbeitung nicht der insgesamt bessere Weg sei, anstatt immer neue rechenintensivere Methoden vorzustellen.

### 2.6.3 Gesichtssegmentierung

Einen interessanten Ansatz für die Aufgabe der Gesichtserkennung und Segmentierung in einzelne Komponenten wie Augen, Nase und Mund liefert die recht neue Arbeit von Kim und Dahyot [28], da sie zur Erkennung hervorstechender Punkte eines Gesichts den in der Bildverarbeitung gut bekannten und weitverbreitet implementierten SURF-Detektor[26] verwendet um Merkmalsvektoren von Gesichtern zu extrahieren. Anschließend verwenden sie eine SVM[27] als Klassifikator. Eine Implementierung ist sehr einfach mit OpenCV möglich, da sowohl SURF-Detektor als auch SVM-Klassifikator hier bereits implementiert sind. Die Autoren erreichen

[26] Der SURF-Detektor (engl. *„Speeded Up Robust Features“*) stellt einen skalierungs- und rotationsinvarianten Feature-Detektor für Bilder dar. Eingeführt wurde er von Bay et al. in [6].

[27] (engl. *„support vector machine“*) zu deutsch etwa „Stützvektormethode“; ein Klassifikatorverfahren, dass versucht, vorhandene Objekte möglichst deutlich voneinander zutrennen. Es verwendet dafür hochdimensionale Stützvektoren und führt die Trennung in hochdimensionalen Räumen durch. Eine gute Einführung für die Verwendung von SVM in der Mustererkennung liefert Burges' Tutorial [12].

mit der Methode extreme Geschwindigkeitsvorteile gegenüber anderen Verfahren, da kein „Windowing“ über das gesamte Bild durchgeführt werden muss. Die Arbeit verbessert damit die bereits durch Osuna et al. beschriebene Idee [42] der Verwendung von SVMs in der Gesichtserkennung deutlich.

### 2.6.4 Nicht untersuchte Verfahren

Ausgeschlossen wurden in dieser Arbeit wissensbasierte Verfahren (z.B. die „multiresolution rule-based method“ von Yang und Huang [68]), die in einer Art Expertensystem Regeln über den Aufbau eines Gesichts formulieren und auf verschiedenen Auflösungsstufen[28] prüfen. Weitere Ansätze, die sich nur auf Merkmale wie Hautfarbe und Textur beziehen, wurden ebenfalls nicht näher betrachtet, da eine Farbaufnahme mit der angestrebten Zielplattform nicht möglich ist. Weiterhin sollten sämtliche wahrscheinlichkeitstheoretischen Methoden wie *Bayes-Klassifikatoren* oder *Hidden Markov Model* unbetrachtet bleiben, da die Einführung der zu Grunde liegenden stochastischen Methoden den Umfang dieser Arbeit übersteigen würde.

[28] vgl. Abschnitt A.4.3

# 3 Beschreibung der zu entwickelnden Softwareplattform

## 3.1 Einleitung

Einen weiterer Schwerpunkt in dieser Arbeit stellt die Erstellung einer neuen Softwareplattform für *Visual Servoing* dar. Ziel war es, eine modulare generische Software unter Verwendung moderner Techniken zu entwickeln, die auch weiterhin für PC-basierte Visual Servoing-Projekte oder auch Praktika an der Professur verwendet werden kann. Um eine Weiterverwendung über diese Arbeit hinaus zu ermöglichen, soll das Hauptaugenmerk auf Struktur und Dokumentation gelegt werden.

## 3.2 Anforderungsprofil

**Modell** Um die Anforderung an Struktur und Dokumentation gewährleisten zu können, wurde die Software über den Softwareentwicklungsansatz der *Modellgetriebenen Architektur* (engl. *„model driven architecture, MDA“*) entwickelt. Die Kernthese dieses Ansatzes ist die strikte Trennung der Funktionalität von der verwendeten Technik. Vor Beginn der Programmierung wurde daher die gesamte Funktionalität in der *UML* notiert und liegt der Arbeit in digitaler Form bei (siehe dazu Abschnitt F.11 und Abschnitt A.1).

**Dokumentation und Selbstbeschreibung** Zusätzlich zur Beschreibung der Funktionalität in UML wurde Wert auf die in-code Dokumentation gelegt, aus der automatisch eine API-Dokumentation erzeugt werden kann. Eine Erweiterung ist somit für spätere Nutzer und Entwickler sehr einfach möglich. Für alle von der Software verwendeten Konfigurations- und Datenformate wurde das für Menschen lesbare Textformat *XML* gewählt. Für jedes dieser Formate werden *XML Schemas*[1] zur

[1] Als eine Analogie kann man sich XML-Dateien als Formulare vorstellen, die der Nutzer ausfüllt. XML Schemas stellen dann eine Ausfüllhilfe bereit und prüfen die Korrektheit und Vollständigkeit der Daten.

Verfügung gestellt, so dass sich die Dateien selbst beschreiben können und ihr Inhalt durch externe Tools validierbar ist. Damit wird die Möglichkeit der fälschlichen Bedienung durch den Benutzer minimiert.

**Moderne Paradigmen und Software** Eine weitere wichtige Anforderung ist es gewesen, bei der Realisierung moderne Softwareentwicklungsparadigmen zu verfolgen und sich auf bekannte und etablierte *Entwurfsmuster*[2] (engl. *„pattern“*) zu beziehen. Die Anwendung selbst wurde komplett objektorientiert (siehe Abschnitt A.1.2 ) in C# realisiert und steht somit nativ unter Windows ab XP mit .NET-Framework zur Verfügung. Bei allen verwendeten Bibliotheken wurde jedoch darauf geachtet, dass eine Bindung an das Mono-Framework existiert, um eine Verwendung auch unter Unix und Linux zu ermöglichen.

**Freie Software und Open Source** Bei allen verwendeten Bibliotheken wurde wert darauf gelegt, dass es sich um freie Software im Sinne der freien Verwendung in der Forschung handelt. Die meisten Komponenten stehen sogar als Open Source im Quellcode zur Verfügung.

**Modularisierung** Das realisierte System gliedert sich in die *Kernanwendung* (engl. *„runtime core“*) und die von ihr geladenen Projektdateien (engl. *„projects“*). Projektdateien selbst stellen eine Beschreibung der durchzuführenden Visual Servoing Aufgabe dar und setzen sich wiederum aus einer Vielzahl einzelner Module zusammen. Jedes Modul implementiert dabei vom System vorgegebene Schnittstellen[3], die im nächsten Abschnitt näher beleuchtet werden. Dadurch wird die Aufgabe selbst in logische Teile zerlegt und ist für weitere Anwendungsfälle einfach adaptierbar. Auch können Teile der Software beliebig ausgetauscht werden, ohne die Gesamtsoftware neu erstellen zu müssen.

**Versionsverwaltung und Continuos Integration** Die Speicherung der gesamten Diplomarbeit und der entstandenen Software erfolgte in einem Versionsverwaltungssystem, um die Entstehung nachvollziehbar zu gestalten. Änderungen können so einfach integriert und vor allem nachvollzogen werden. Weiterhin wird für die Software eine virtuelle Maschine bereitgestellt, die die gesamte Entwicklungsumgebung

[2] Entwurfsmuster stellen eine Art Lösungsschablone für wiederkehrende Entwurfsprobleme in der Informatik dar. Die beste Einführung in die Arbeit mit Patterns findet sich im Standardwerk [23], dem sogenannten „Gang of Four“ Buch.

[3] Diese Schnittstelle kann man sich bildlich als eine Fähigkeit des Objekts vorstellen, ein Rolle, die das Modul im Projekt spielt.

kapselt und einen automatischen Build-Dienst integriert. Damit wird der Erstellvorgang der Software maximal automatisiert und der Einstieg für neue Entwickler vereinfacht.

## 3.3 Komponentenmodell

Der These der „Trennung der Anliegen“ (Abschnitt A.1.2) folgend, existieren die im anschließenden beschriebenen Rollen. Die Definition dieser Rollen als Schnittstelle (engl. *„interface“*) stellt sicher, dass sich jede Komponente, die eine Rolle implementiert, gegenüber dem System gleich verhält (die gleichen Methodenaufrufe unterstützt) und somit beliebig austauschbar ist.

**Modulinformation, Konfiguration, Ablaufkontrolle und Injektion von Referenzen** Um ein ladbares Modul selbst beschreibend zu implementieren, existiert die Schnittstelle `IModuleInformation`. Ein Modul, das diese Schnittstelle implementiert, liefert Informationen über seinen Namen, die aktuelle Version, den Autor und die unterstützten Konfigurationsparameter zurück. Um ein Modul über seine Para-

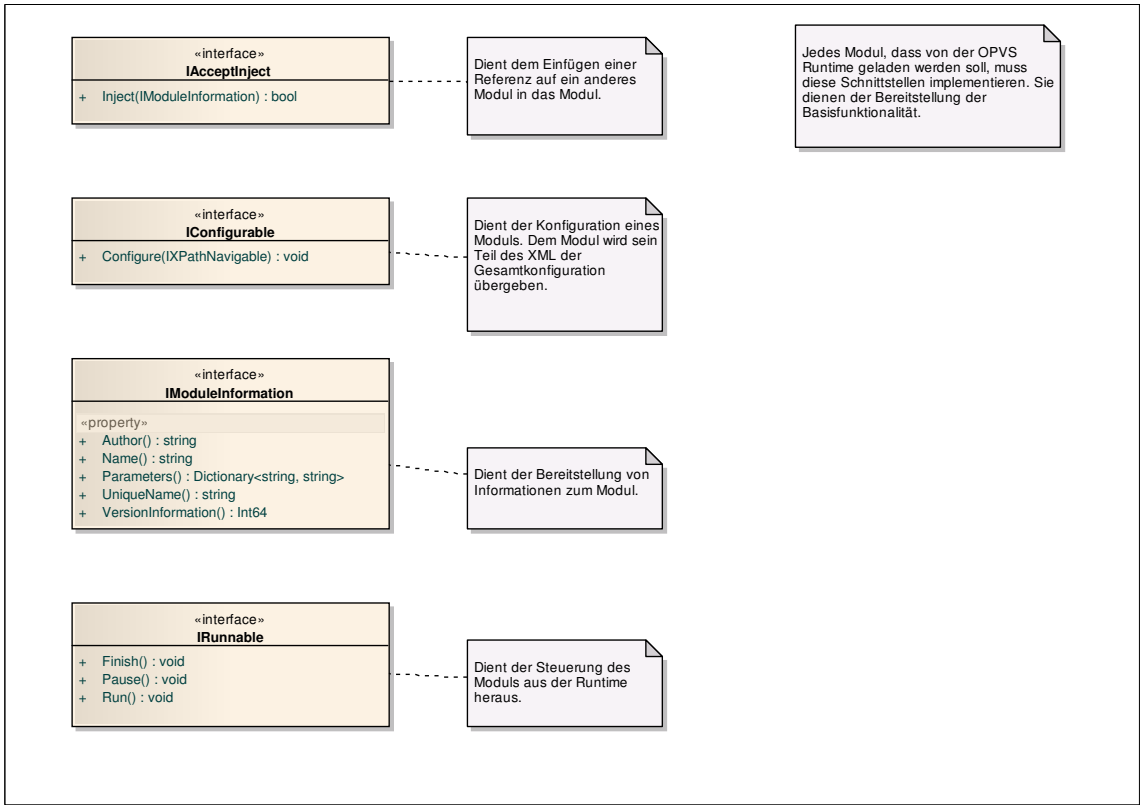



Abbildung 3.1: UML-Klassendiagramm für zu implementierende Schnittstellen zur Modulinformation, Konfiguration und Ablaufkontrolle

meter konfigurieren zu können, muss es die Schnittstelle `IConfigurable` implementieren. Diese Schnittstelle stellt die Methode `Configure` bereit, die als Parameter einen XML-Konfigurationsausschnitt übergeben bekommt. Dieser Konfigurationsausschnitt ist für alle Module gleich: er besteht aus dem Root-Element `configuration` und beliebig vielen Kindelementen `parameter`, die jeweils die Attribute `name` und `value` bereitstellen. Die Konfiguration des `GenericCamDrivers` hat beispielsweise diese Form:

```
1 <configuration>
2         <parameter name='camid' value='0'/>
3 </configuration>
```

Ein solches Modul kann in der OPVS-Runtime nur dann verwendet werden, wenn es die Schnittstelle `IRunnable` implementiert. Diese ermöglicht es der Laufzeitumgebung, das Modul zu starten, zu pausieren und die Ausführung zu stoppen. Schließlich ist es über die Schnittstelle `IAcceptInject` möglich, dem Modul über die Methode `Inject` die Referenz auf ein anderes Modul zu übergeben. Dies wird beispielsweise dazu genutzt, dem `FaceDetector`-Modul ein Bilderfassungsmodul in Form einer `GenericCamDriver`-Instanz zur Verfügung zu stellen.

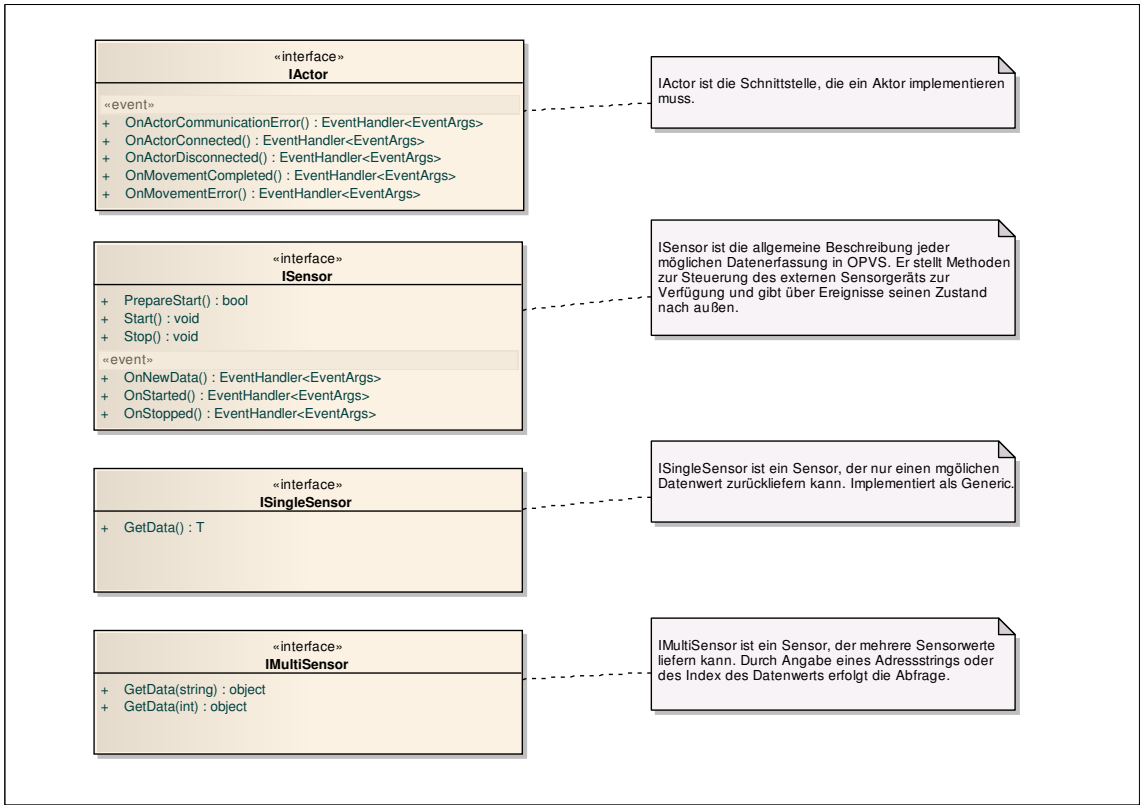



Abbildung 3.2: UML-Klassendiagramm für zu implementierende Schnittstellen für Sensoren und Aktoren

**Sensor** Sensoren dienen der Software als Informationsquellen und stellen die zu verarbeitenden Daten zur Verfügung. Sie sind somit die Schnittstelle zur realen Welt. OPVS unterscheidet zwischen Sensoren, die nur ein Datum zur Verfügung stellen (Schnittstelle `ISingleSensor`) und Sensoren, die verschiedene Typen zurückgeben können (Schnittstelle `IMultiSensor`). Jeder nutzbare Sensor implementiert zusätzlich die allgemeine Schnittstelle `ISensor`, die Methoden zur Vor- und Nachbereitung der Datenschnittstelle und EventHandler für auftretende Ereignisse zur Verfügung stellt.

**Aktor** Aktoren dienen der Interaktion mit der realen Welt und stellen die Senken des Systems dar. Sie werden als *State-Maschinen* realisiert und genügen dem in Abbildung 3.3 dargestellten Zustandsmodell.

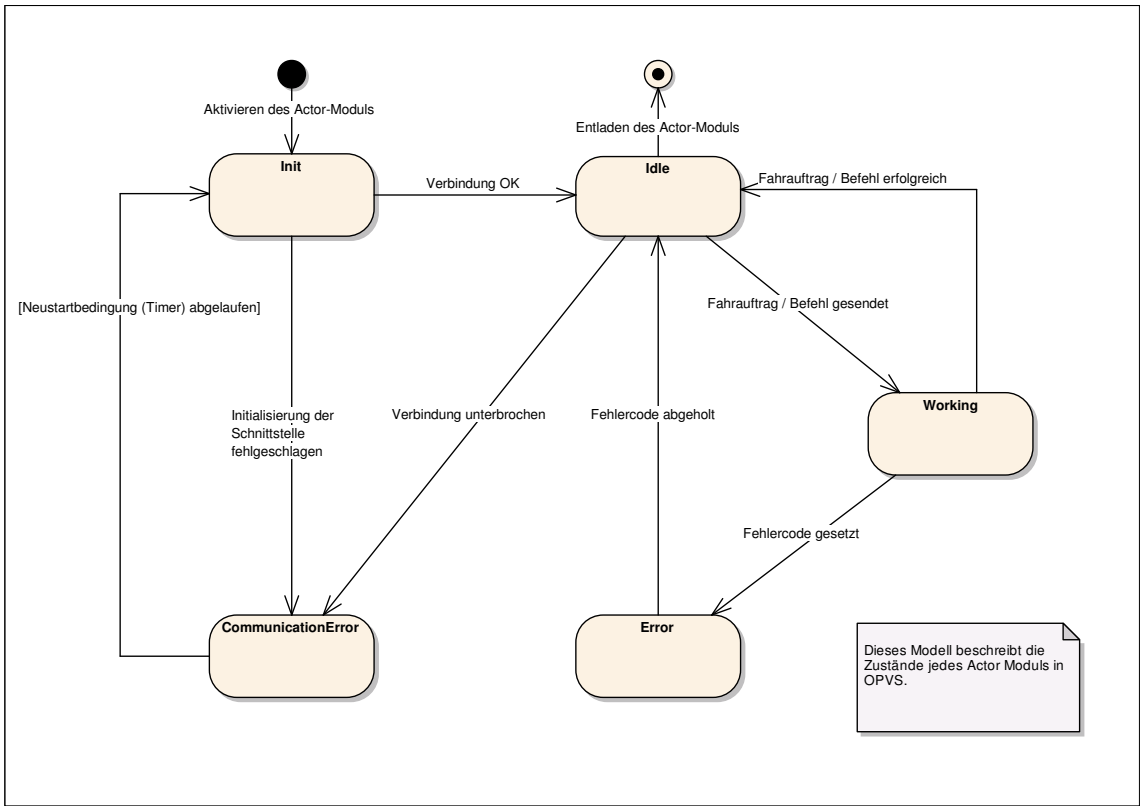



Abbildung 3.3: UML-Zustandsdarstellung eines Aktors

**Weltbeschreibung** Eine Weltbeschreibung dient OPVS dazu, einen kompletten Überblick über alle Sensoren, durch Sensoren erkannte Objekte und Aktoren der aktuellen Szenerie intern abzubilden. Dazu hält es Referenzen auf sogenannte Weltobjekte (Objekte die `IWorldObject` implementieren). Weltobjekte halten dazu ihre

karthesischen Koordinaten relativ zum im Weltobjekt gespeicherten Koordinatenursprung. Über einen Enumerator können diese Weltobjekte durch Visualisierungen ausgelesen und dargestellt werden. In dieser Arbeit findet keine Implementierung von Weltbeschreibungen oder -objekten statt. Diese Funktion wurde nur zur Vollständigkeit des Systems in Hinblick auf spätere Anwendung (z.B. Würfelerkennung im Raum mit Stereokamera) bereits vorgesehen.

**Visualisierung** Eine Visualisierung dient der Darstellung der inneren Zustände des Hauptprogramms und kann zur Darstellung der Weltbeschreibung oder Aufbereitung der Sensorsignale verwendet werden. Sensoren und Weltobjekte werden der Visualisierung über ihre Konfiguration zugeordnet. Bei der Initialisierung einer Visualisierung (Schnittstelle `IVisualization`) wird dem Visualisierungsmodul eine `Parent`-Komponente als Referenz übergeben, in die anschließend die eigentliche Ausgabe des Moduls erfolgt.

**Klassifikator** Ein Klassifikator dient der Verarbeitung und Aufbereitung verschiedenster Sensorinformationen. Die Schnittstelle `IClassificator` ist daher als *generischer Datentyp* (engl. *„generic“*) vom abstrakten Typ `T` definiert, was bedeutet, dass die Implementierung eines Klassifikators ihren Datentyp selbst vorgibt. So kann zum Beispiel ein Klassifikator, der ein neuronales Netz verwendet, den Datentyp `float` für seine Eingänge festlegen, eine Kamera jedoch `Image<TColor,TDepth>` verwenden. Die Schnittstelle selbst ist asynchron vorgesehen, da die Laufzeit einer Klassifikation nicht vorher bekannt ist und das Hauptprogramm sonst blockieren würde. Das bedeutet weiterhin, dass ein Klassifikator Erfolg oder Misserfolg über entsprechende Ereignisse (`OnClassified, OnNotClassified`) zurückmelden muss.
Klassifikatoren, die vom Benutzer neu angelernt werden können, implementieren die Schnittstelle `ITrainable`. Ist für dieses Anlernen zusätzlich eine grafische Benutzeroberfläche vorgesehen, wird diese durch `ITrainableInteractive` bereitgestellt.

**Programm** Die Schnittstelle `IProgram` stellt ein Hauptprogramm für OPVS zur Verfügung, welches die Funktion von `ISensor`, `IActor` und `IClassificator` steuert und so die eigentliche Funktion des Programms realisiert. Die Realisierung dieser Hauptprogramme erfolgt in IronPython. Sensoren, Aktoren und Klassifikatoren werden über Objektreferenzen bereitgestellt. Alle verarbeitenden CallBack-Events von Sensoren, Aktoren und Klassifikatoren werden in diesem Hauptprogramm hinterlegt.

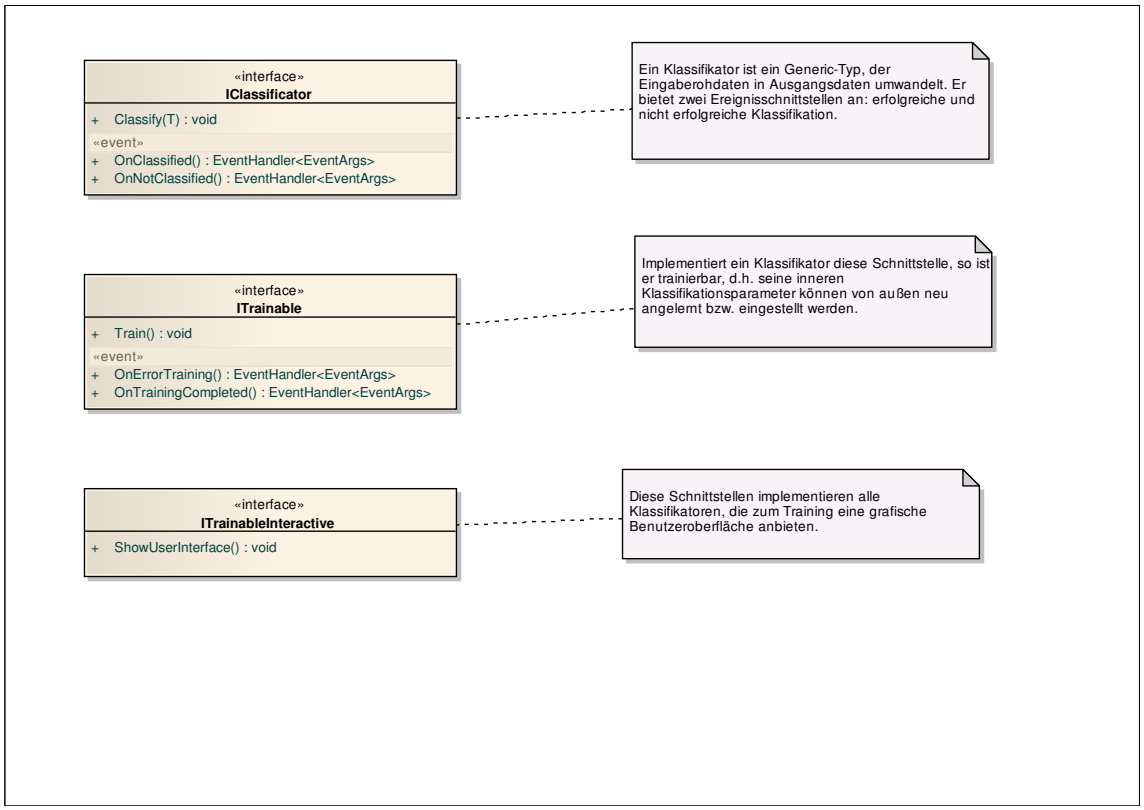



Abbildung 3.4: UML-Klassendiagramm über Schnittstellen für Klassifikatoren

# 4 Referenzimplementierung in C#

## 4.1 Hinleitung

Die Referenzimplementierung für OPVS wurde als Gesichtserkennung in Form eines Trackers erstellt. Dieser folgt einer anhand ihres Gesichts erkannten Person und verwendet einen Stäubli-Roboter, um auf diese Person zu zeigen. Dabei erfolgt die Zuordnung eines Bildpunkts zu einer Tool-Koordinate des Roboters über eine einfache antrainierte Lookup-Tabelle. Es empfiehlt sich daher, die Kamera möglichst hinter dem Roboter zu positionieren, um die Fehler durch perspektivische Verzerrung gering zu halten. Durch Verwendung dieser Lookup-Tabelle entfällt sowohl die Kamerakalibrierung als auch die umständliche Implementierung einer 3D-Pose-Schätzung beziehungsweise Verfahren der Auto-Stereographie über eine Bewegung der Kamera.
Es existieren drei Betriebsmodi, die im Folgenden als Use-Cases detailliert dargestellt werden:

1. Trainingsphase der Gesichtswiedererkennung,
2. Trainingsphase der Lookup-Tabelle Kamera-zu-Roboterarm und
3. Ausführungsphase des Gesichtstrackings

## 4.2 Modellbeschreibung der Anwendung

### 4.2.1 Komponentenmodell

Die Referenzimplementierung setzt das in Kapitel 2.6.4 beschriebene Komponentenmodell um. Die realisierten Komponenten sind in Tabelle 4.1 aufgeführt.

### 4.2.2 Use-Case-Diagramme

Die Trainingsphase der Gesichtswiedererkennung wird in Abbildung 4.2 beschrieben. Durchgeführt wird diese über eine Trainingsoberfläche, die `FaceRecognizer` zur Verfügung stellt. Eine Beschreibung der Oberfläche ist in Abschnitt 4.4.1 zu finden. Eine

Tabelle 4.1: Realisierte Komponenten der Referenzimplementierung

| Klassenname | Realisiert | Beschreibung |
|---|---|---|
| `GenericCamDriver` | `ISensor` | ein generischer Kameratreiber für USB-Kameras |
| `FaceDetector` | `IClassificator` | eine Gesichtserkennung über Haar-Kaskaden |
| `FaceRecognizer` | `IClassificator` | eine Gesichtswiedererkennung nach Eigengesichtmethode |
| `FaceTracker` | `IProgram` | Hauptprogramm der Anwendung,realisiert die Gesichtsverfolgung |
| `Lookup2DTool` | `IClassificator` | dient dem 2D-Lookup der Roboterarmstellung |
| `StaeubliDriver` | `IActor` | dient der Ansteuerung des Stäubliroboters |
| `DisplayFace` | `IVisualisation` | dient der Darstellung des Kamerabilds und des gefundenen Gesichts |
| `TextLogger` | `IVisualisation` | dient der Ausgabe von Debugginginformationen |

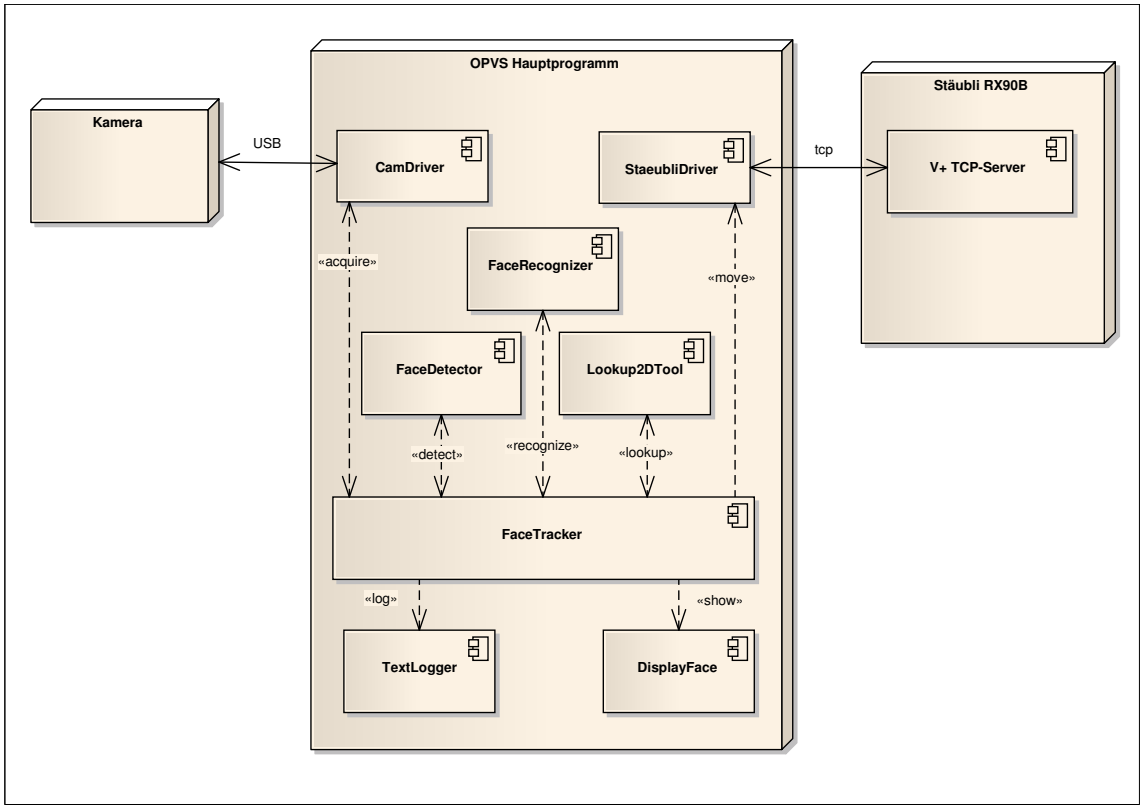



Abbildung 4.1: UML-Komponentendarstellung der Referenzimplementierung

Beschreibung aller Anwendungsfälle der Trainingsphase der Lookup-Tabelle für die Zuordnung von Bildpunkte zu Tool-Positionen des Roboters ist in Abbildung 4.3 gegeben. Die Trainingsoberfläche wird wiederum in Abschnitt 4.4.1 erklärt. Die Beschreibung der Abläufe während der Ausführungsphase des Gesichtstrackings stellt schließlich Abbildung 4.4 dar.

### 4.2.3 Klassendiagramme

#### GenericCamDriver

In Abbildung 4.5 ist das Klassenmodell des `GenericCamDrivers` dargestellt. Er ist als ein `ISingleSensor` implementiert, der Bilder als Emgu.CV-Datentyp `Image<Bgr, byte>` zurückgibt. Zur Konfiguration über `IConfigurable` steht lediglich der Parameter `camId` zur Verfügung, der den Index der zu verwendenten Kamera angibt.

#### FaceDetector

Das Klassenmodell des Klassifikators `FaceDetector` ist in Abbildung 4.6 dargestellt. Als Konfigurationsparameter stehen die folgenden Parameter zur Verfügung:

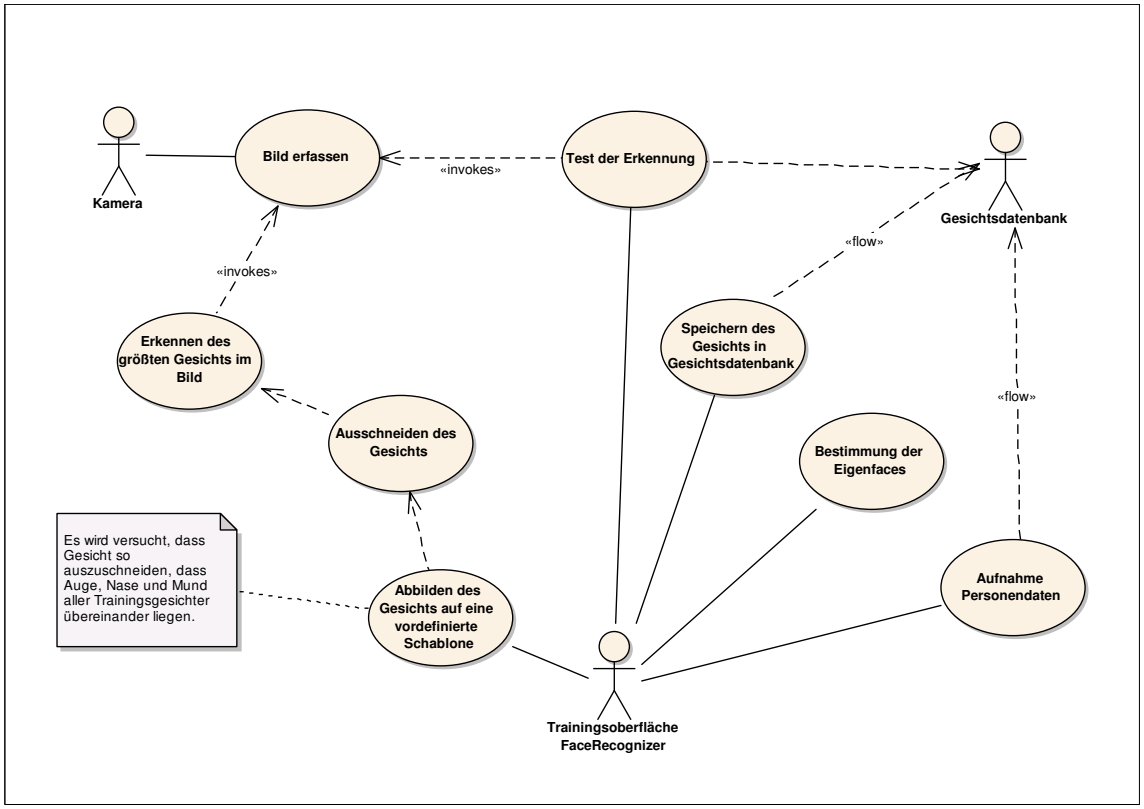



Abbildung 4.2: UML-Use-Case: Trainingsphase der Gesichtswiedererkennung

| Parameter | Default | Beschreibung |
|---|---|---|
| `cascade` | `haarcascade_frontal-face_alt2.xml` | Zu verwendende Haar-Kaskade (als XML-Datei) |
| `minNeighbors` | 1 | Anzahl weiterer Gesichter in direkter Nachbarschaft |
| `scaleFactor` | 1.1 | Skalierungsfaktor der Kaskade |
| `minSize` | 30 | Mindestgröße eines Gesichts |
| `flags` | 0 | Flags, Erklärung siehe Abschnitt 2.3.4 |

Tabelle 4.2: Parameter: `FaceDetector`

### FaceRecognizer

In Abbildung 4.7 ist das Klassenmodell des Klassifikators `FaceRecognizer` dargestellt. Dieser kann über `IConfigurable` die in Tabelle 4.3 aufgeführten Parameter erhalten. Im Unterschied zu `FaceDetector` kann `FaceRecognizer` durch den Benutzer um weitere zu erkennende Personen erweitert werden und stellt dazu eine Oberfläche über `ITrainableInteractive` zur Verfügung. Das Neutrainieren der Eigengesichter wird über `ITrainable` ausgeführt.

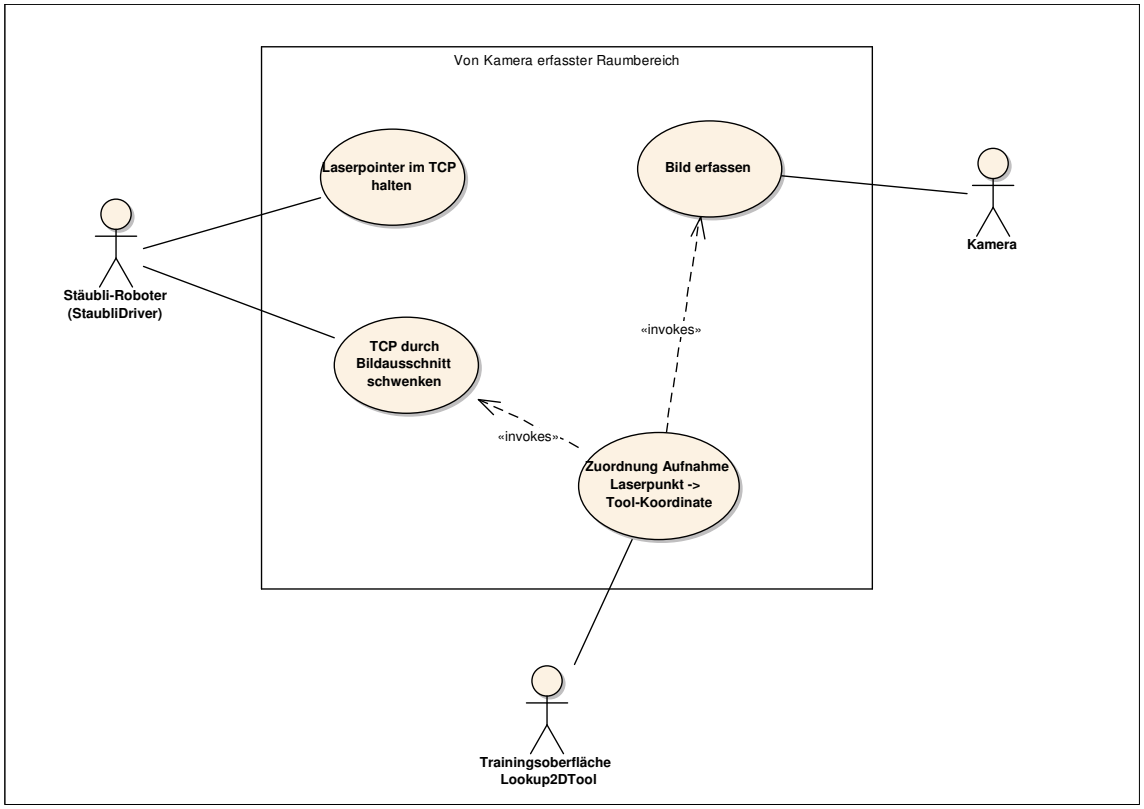



Abbildung 4.3: UML-Use-Case: Trainingsphase der Lookup-Tabelle Kamera zu Roboterarm

| Parameter | Default | Beschreibung |
|---|---|---|
| `trainDB` | - | Trainingsdatenbank |
| `noTrainingImages` | 5 | Anzahl der aufzunehmenden Trainingsbilder |
| `noEigenfaces` | 5 | Anzahl der zu verwendenden Eigengesichter |
| `matchDistance` | - | Größte Distanz, die noch als gültiger Match akzeptiert wird |

Tabelle 4.3: Parameter: `FaceRecognizer`

### Lookup2DTool

In Abbildung 4.8 ist das Klassenmodell des Klassifikators `Lookup2DTool` dargestellt. Dieser Klassifikator kann über `IConfigurable` die in Tabelle Tabelle 4.4 aufgeführten Parameter erhalten. Wie `FaceRecognizer` unterstützt `Lookup2DTool` das Anlernen der Tabelle über `ITrainableInteractive` und `ITrainable`.

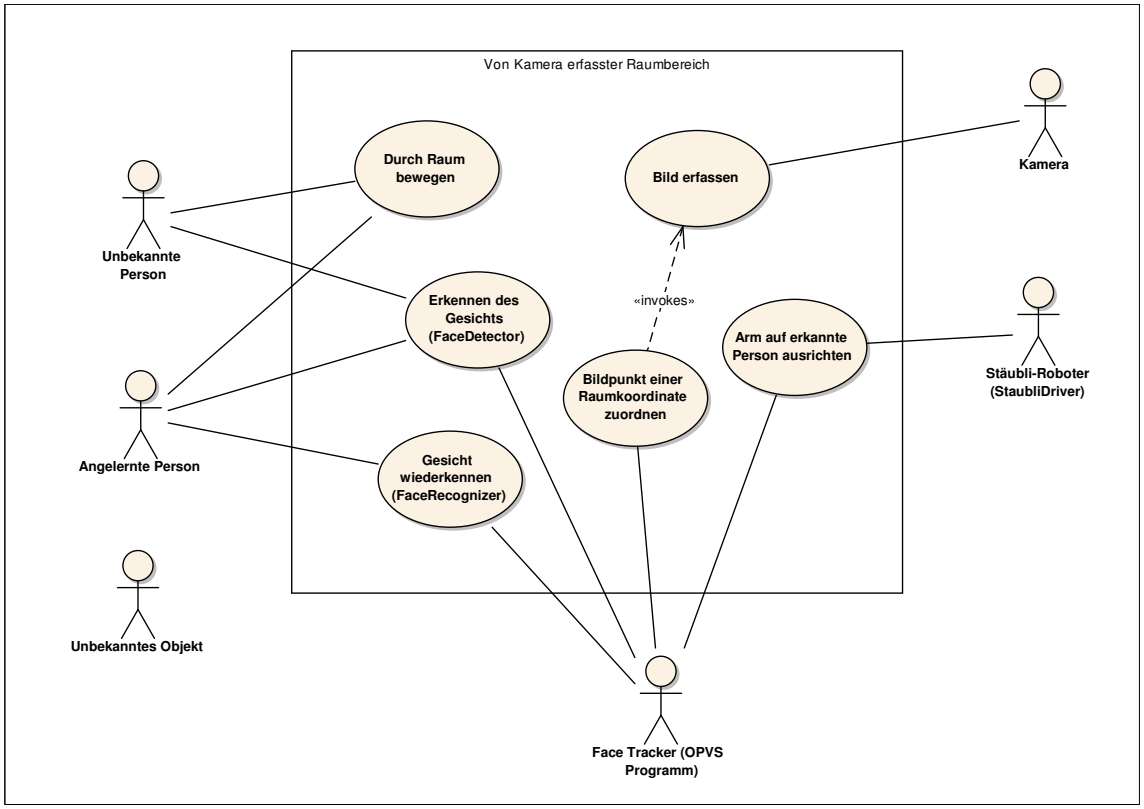



Abbildung 4.4: UML-Use-Case: Ausführungsphase des Gesichtstrackings

| Parameter | Default | Beschreibung |
|---|---|---|
| `useTable` | - | Dateiangabe einer bereits angelernten Lookup-Tabelle |

Tabelle 4.4: Parameter: `Lookup2DTool`

### StaubliDriver

In Abbildung 4.9 ist das Klassenmodell des Aktors `StaubliDriver` dargestellt. Bewegungen führt er als Tool-Bewegungen (sechs Koordinaten X,Y,Z und yaw,pitch,roll) über die Schnittstelle `IActorMoveToolXYZYPR` aus. Seine über `IConfigurable` definierbaren Parameter stellt Tabelle 4.9 dar.

| Parameter | Default | Beschreibung |
|---|---|---|
| `host` | `127.0.0.1` | Host bzw. IP-Addresse, auf der der V+-TCP-Server läuft |
| `port` | `50000` | Port, auf dem der V+-TCP-Server lauscht |

Tabelle 4.5: Parameter: `StaubliDriver`

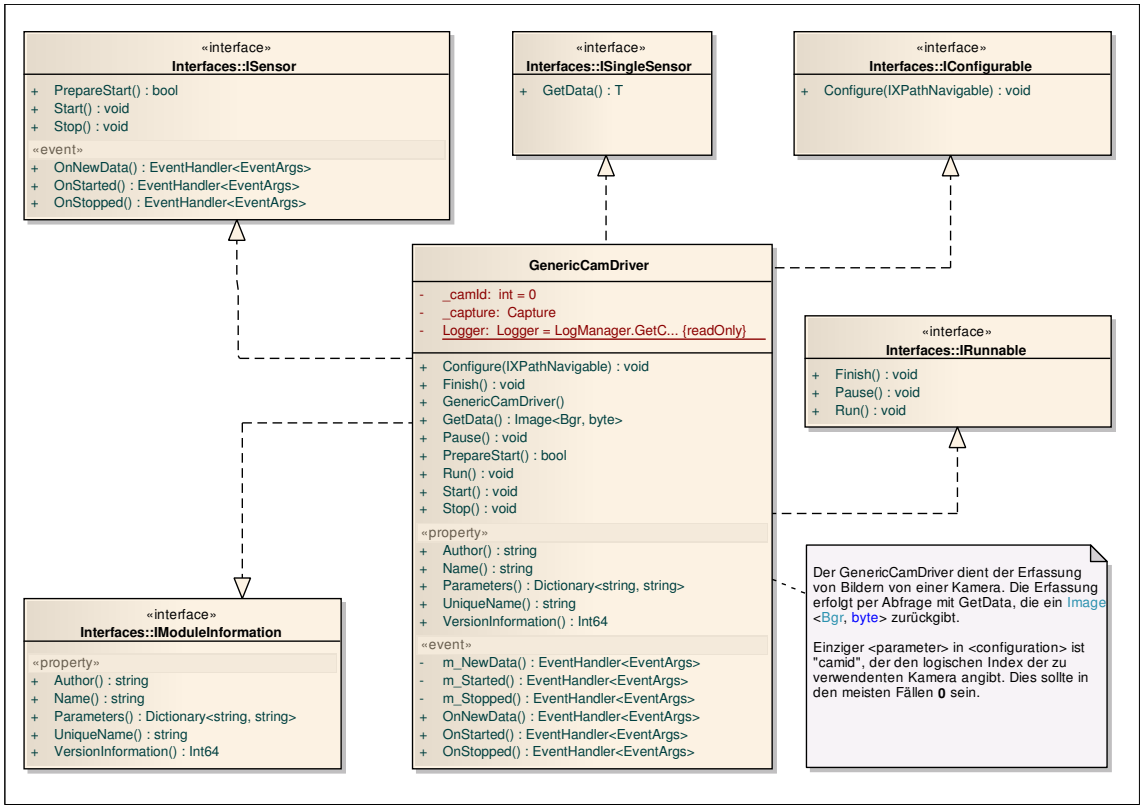



Abbildung 4.5: UML-Klassendiagramm: `GenericCamDriver`

### TextVisualization

`TextVisualiazion` ist eine sehr einfache Implementierung einer Visualisierung. Sie dient lediglich dazu, interne Zustände der beteiligten Klassen in Textform auszugeben und dem Benutzer somit eine Debugansicht in das System hinein zu ermöglichen. Zur internen Verwendung kommt dazu das NLog-Framework, das die interne Debugausgabe in eine Datei umlenkt. `TextVisualization` verhält sich analog zum Shell-Befehl `tail -f` und stellt die Textausgabe fortlaufend in einem Textfenster dar. Das Klassendiagramm ist in Abbildung 4.10 angegeben.

### DisplayFace

`DisplayFace` ist eine erweiterte Implementierung einer Visualisierung. Sie ermöglicht die Darstellung eines ausgewählten `GenericCamDrivers` und der in ihm mit `FaceDetector` und `FaceRecognizer` erkannten Gesichter. Das Klassendiagramm ist in Abbildung 4.10 dargestellt.

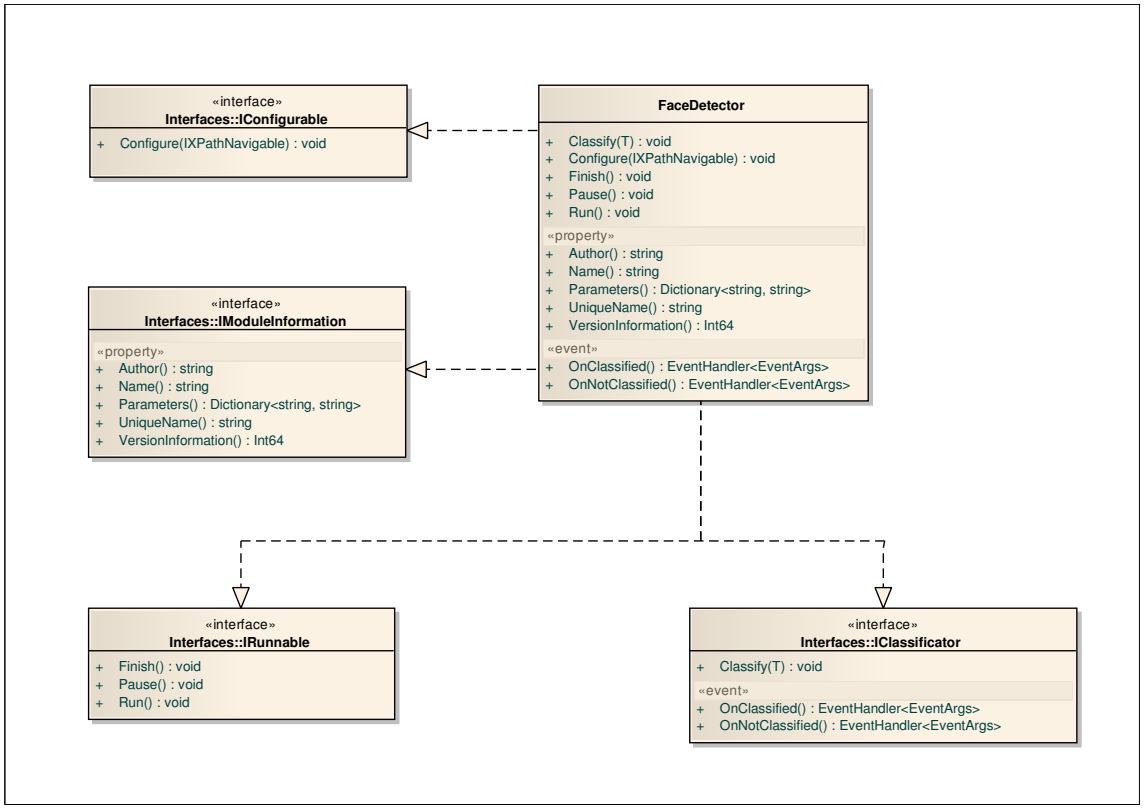



Abbildung 4.6: UML-Klassendiagramm: `FaceDetector`

## 4.3 Betrachtung der Software

### 4.3.1 Projektdateien

Zur korrekten Ausführung von OPVS sind das .NET-Framework, das EmguCV-Framework und das OPVS -Framework mit dem FaceTracking-Projekt auf dem Arbeitsplatz zu installieren. Weiterhin müssen folgende Hardwarevorkehrungen getroffen sein:

1. Roboter ist einsatzbereit,
2. Netzwerkverbindung zu Roboter besteht und
3. mindestens eine Kamera ist am Arbeitsplatz angeschlossen.

## 4.4 Inbetriebnahme der Software

Zur Inbetriebnahme der Software sollte zunächst der verwendete Stäubli RX90 im Labor in Betrieb genommen werden (Anschalten der Strom- und Druckluftversor-

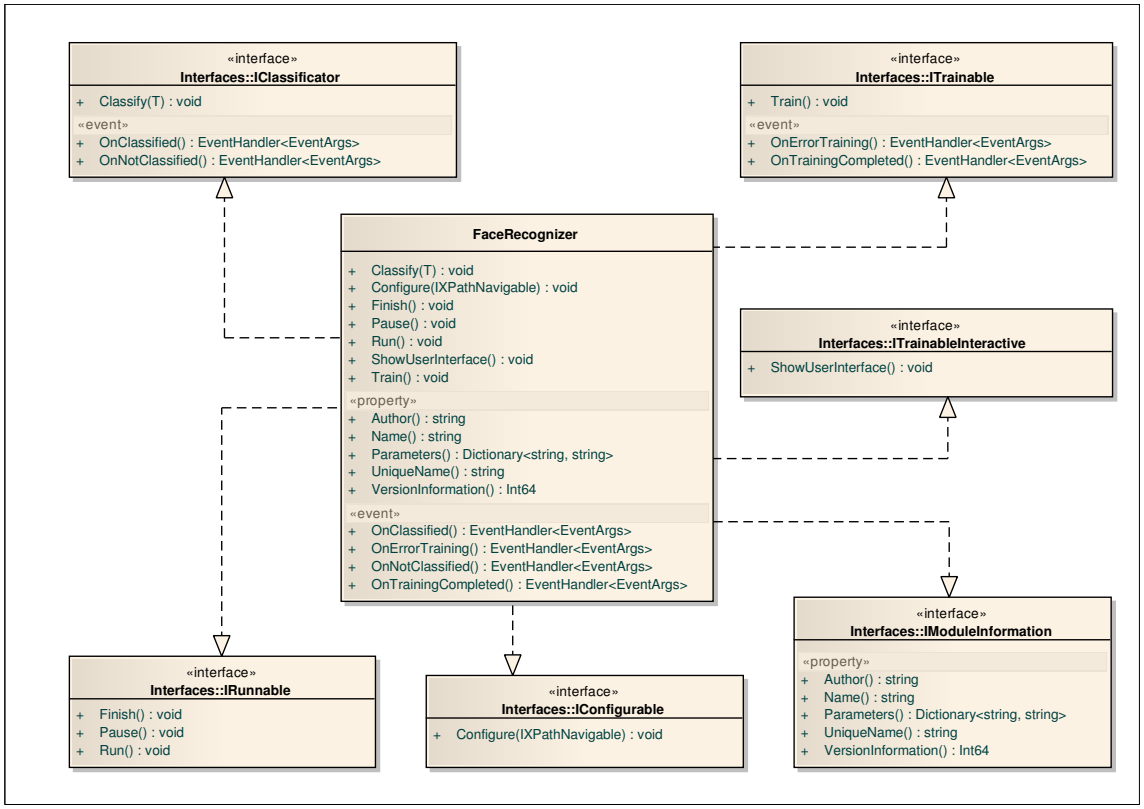



Abbildung 4.7: UML-Klassendiagramm: `FaceRecognizer`

gung). Anschließend muss die Verbindung vom Arbeitsplatz-PC aus mit Adept vorgenommen und der TCP-Server als Task in der Robotersteuerung gestartet werden. Unter der IP `172.16.194.192` auf Port `50000` ist der TCP-Server anschließend erreichbar. Vom Arbeitsplatz-PC aus kann nun ein Verbindungstest zu diesem Server mit einem beliebigen TCP-Client[1] vorgenommen werden. Ist dieser Test erfolgreich, wird `OPVS Workbench` gestartet und das Projekt `FaceTrackingSystem.xml` geladen. Mit `Runtime|Start Runtime...` wird das Projekt schließlich ausgeführt.
In der `OPVS Workbench` ist eine detaillierte Betrachtung des geladenen Projekts sowie seiner verwendeten Komponenten möglich. Außerdem können während der Ausführung im Projekt definierte Visualisierungen betrachtet werden. Ist die `OPVS Runtime` nicht in Betrieb, können die im Projekt vorhandenen Klassifikatoren trainiert werden.

### 4.4.1 Trainingsoberfläche

**FaceRecognizer** Für das Training der Gesichtswiedererkennung stellt `FaceRecognizer` eine Trainingsoberfläche zur Verfügung. Diese ist in drei Teile unterteilt: links

[1] z.B. `telnet, PuTTy, netcat`

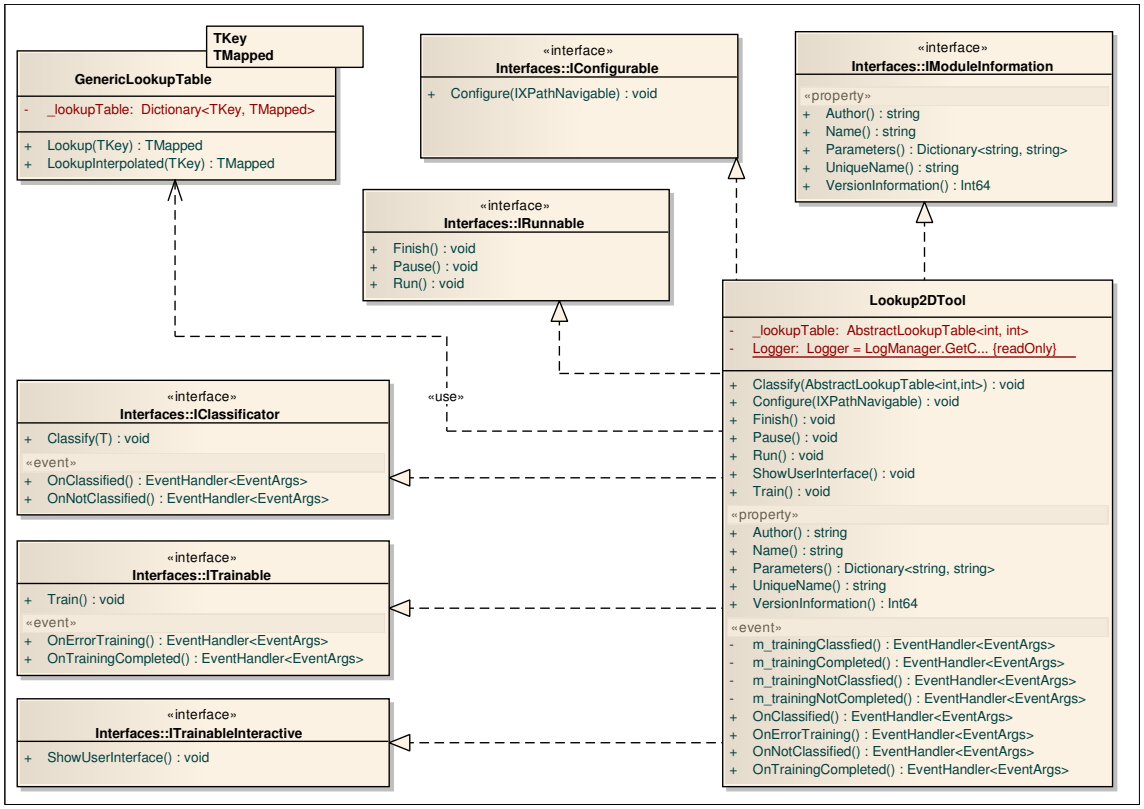



Abbildung 4.8: UML-Klassendiagramm: `Lookup2DTool`

Personenkonfiguration, mittig das aktuelle Kamerabild mit dem größten gefunden Gesicht, rechts die aktuelle Trainingsaufgabe. Zum Anlernen einer weiteren Person wird zunächst eine neue Person hinzugefügt. Anschließend werden fünf verschiedene Bilder der Person aufgenommen und in der Trainingsansicht in die Gesichtsschalone eingepasst. Alle Trainingsdaten werden automatisch im über die Konfiguration von `FaceRecognizer` definierte Trainingsverzeichnis abgelegt. Durch Drücken der Schaltfläche „Eigenfaces trainieren“ wird die Eigengesichts-Basis neu erstellt und steht der `OPVS Runtime` ab sofort zur Verfügung.

**Lookup2DTool** Das Antrainieren der Lookup-Tabelle erfolgt über die Zuordnung von Bildpunkten zu einer Roboterstellung (Abbildung 4.11). Dafür stehen die beiden Betriebsmodi manuelles oder automatisches Training zur Verfügung. In beiden Fällen werden Pixel in der Aufnahme eines Kamerabildes einer Roboterstellung zugeordnet. Zu diesem Zweck wird in den TCP des Roboters ein aktiver Laserpointer eingebracht, der im Kamerabild einen deutlichen Lichtpunkt erzeugt. Sofern möglich, wird dieser über einen Farberkennungsalgorithmus automatisch im Bild gesucht und hervorgehoben. Ist durch die Lichtverhältnisse oder das von der Kamera genutzte Farbformat keine automatische Erkennung dieses Lichtpunkts möglich, kann er vom

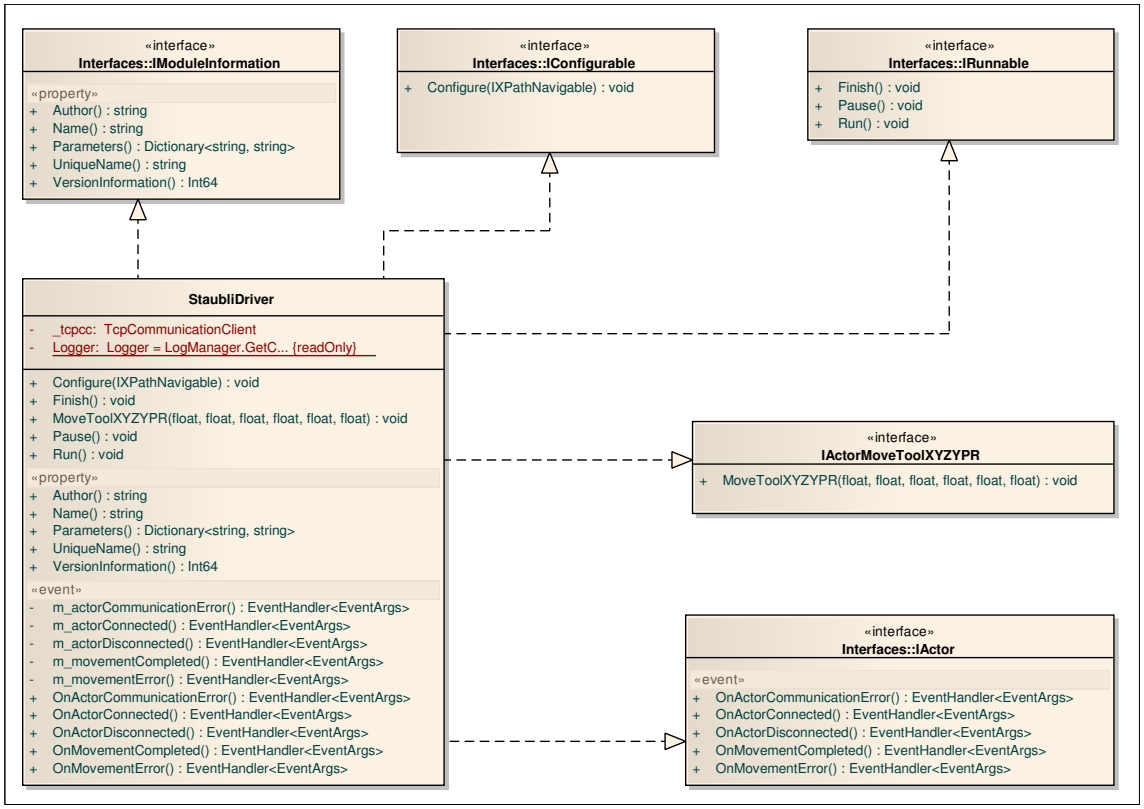



Abbildung 4.9: UML-Klassendiagramm: `StaubliDriver`

Benutzer manuell im Bild markiert werden. Die Bewegung des Roboters erfolgt in beiden Fällen manuell über das Bedienpad oder die im Trainingsprogramm vorgesehene Schaltflächen. Eine Mehrfachaufnahme ist im sogenannten Bulkmodus möglich, in dem sowohl Bilddaten als auch Roboterposition zyklisch abgefragt werden. Nach erfolgtem Training wird die Lookup-Tabelle in einer Datentabelle angezeigt und kann im Projektverzeichnis abgespeichert werden. Das Verfahren einer 2D-Lookup-Tabelle stellt natürlich keine komplette Rekonstruktion des 3D-Raums dar und ist daher anfällig für sphärische Verzerrungen und Bewegung der Kamera nach erfolgter Kalibrierung. Demgegenüber steht die Möglichkeit eines schnellen Wechsels der verwendeten Kamera mit einem Verzicht auf Kamerakalibrierung und die Methoden der Stereobilderfassung. Die besten Ergebnisse erzielt das Verfahren, wenn die Kamera „hinter“ dem Roboter aufgestellt wird und zentral auf die zu erfassende Szene blickt (siehe Abbildung 4.11).

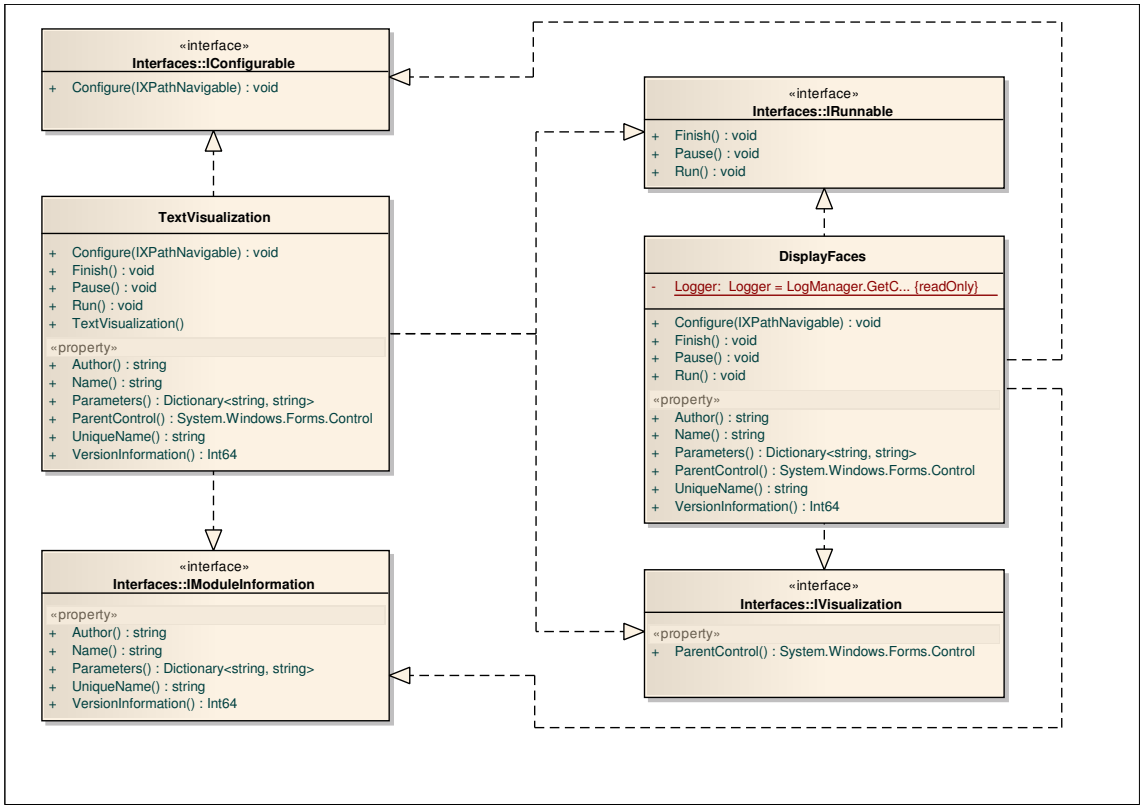



Abbildung 4.10: UML-Klassendiagramm: `TextVisualization` und `DisplayFace`

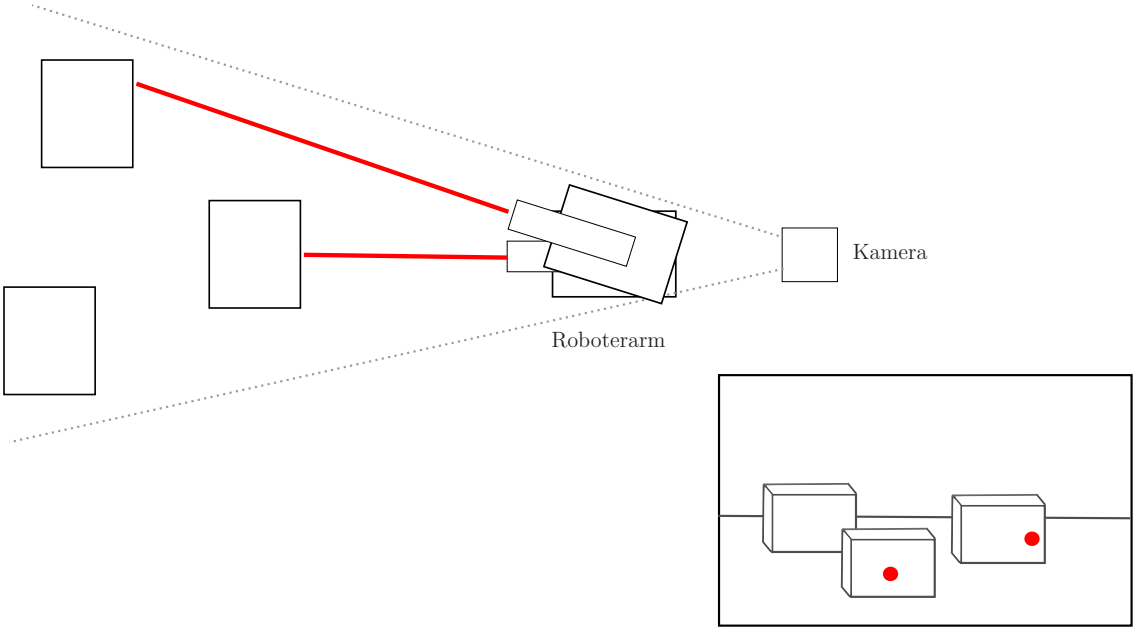



Abbildung 4.11: Zuordnung eines Bildpunkts zu einer Roboterarmposition mittels 2D-Lookup-Tabelle

### 4.4.2 $V^+$ TCP-Server

| Befehl | Parameter | Funktion |
|---|---|---|
| GETAPIVER | - | Ermittelt die auf dem Server verwendete Softwareversion. Dient der Versionierung der Schnittstelle, da eine Gegenstelle so den unterstützten Befehlssatz abfragen kann. |
| GETHELP | - | Liefert alle auf dem Server unterstützten Befehle zurück. |
| GETLASTERRROR | - | Liefert den zuletzt im Roboter gesetzten Fehlercode zurück. |
| SETPOS | X Y Z Y P R [appro] | Führt eine Bewegung des Roboters zu den angegebenen Koordinaten durch ($V^+$ :`MOVE TRANS`). Ist der optionale Parameter `appro` angegeben, wird eine Approach-Bewegung (langsame Annäherung an die endgültige Zielkoordinate mit ($V^+$ :`APPROS`) durchgeführt. |
| GETPOS | - | Liefert die gegenwärtige Position ($V^+$ :`HERE`) in X,Y,Z,yaw,pitch,roll zurück |
| SETSPEED | speed_factor | Setzt den Geschwindigkeitsmultiplikator im Roboter. Werte: 0-100, Auflösung $10e^{-6}$ |
| GETSPEED | - | Liefert den letzten gesetzten Geschwindigkeitsmultiplikator des Roboters zurück. |
| GETTOOL | - | Liefert die letzte `TOOL`-Konfiguration des TCP zurück. |
| SETTOOL | X Y Z Y P R | Setzt die TOOL-Konfiguration des TCP. |
| ISMOVING | - | Prüft, ob der Roboter noch in Bewegung ist. Rückgaben: `1` (in Bewegung), `0` (idle), `ERROR` (Fehler beim Auslesen) |
| SETDO | signal | Setzt im Roboter das übergebene digitale Signal. Da ein Auslesen nicht möglich ist, müssen alle digitalen Ausgänge Teil der Initialisierung des Programms sein. |

Tabelle 4.6: Unterstützte Befehle $V^+$-TCP/IP-Server

Für die Ansteuerung des Stäubli-Roboters wurde ein neues $V^+$-Programm erstellt. Dieses stellt über einen TCP/IP-Server auf Port 50000 ein eingeschränktes Befehlsset (siehe Tabelle 4.6) für die Roboterinteraktion zur Verfügung. Der TCP/IP-Server ist dabei als State-Maschine implementiert (vgl. Abbildung 4.12).

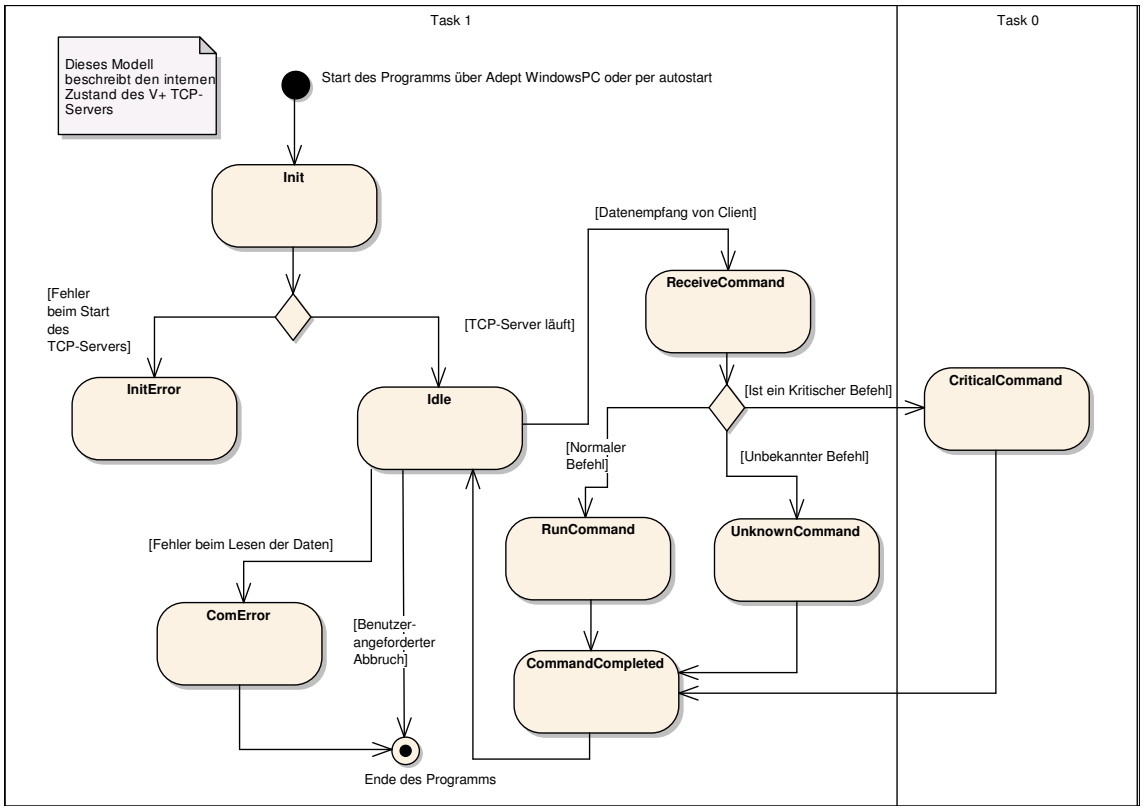



Abbildung 4.12: UML-Zustandsmodell des V+-TCP/IP-Servers

# 5 Zusammenfassung und Ausblick

In dieser Arbeit wurden Verfahren der Gesichtserkennung und -wiederkennung theoretisch eingeführt und nachvollziehbar in MATLAB implementiert. Vor- und Nachteile der behandelten Verfahren wurden diskutiert, auf bekannte Verbesserungen und Erweiterungen wurde verwiesen. Weiterhin wurden die besprochenen Verfahren Performanceuntersuchungen unterzogen und gegeneinander auf verschiedene Standarddatensätze angewendet. Aus diesen Messungen wurden Rückschlüsse auf die anschließende Parametrierung der Referenzimplementierung gezogen.

Zur Verwendung in Visual Servoing-Projekten wurde eine neue Softwareplattform „Open Platform for Visual Servoing “ theoretisch erarbeitet und nach modernen Paradigmen der Softwareentwicklung in einer *managed-code* Umgebung implementiert. Dabei konnte gezeigt werden, dass auch anspruchsvolle Visual Servoing-Aufgaben wie ein Gesichtstracking dank gestiegener Hardwareleistungsfähigkeit und vorhandener ausgereifter Bildverarbeitungsbibliotheken in einer objektorientierten interpretierten Umgebung implementierbar sind. Durch die von vornherein angestrebte Flexibilität von OPVS ist ein einfacher Baukasten zur Formulierung weiterer Visual Servoing-Probleme entstanden, der in der Zukunft mit weiteren Bauteilen gefüllt werden kann. Denkbar wäre beispielsweise die Implementierung weiterer Sensoren, wie die eines `1394CamDriver`s zur Anbindung der im Labor vorhandenen Firewirekameras, ein `StereoCamDriver`, der direkt Rauminformationen zur Verfügung stellt oder auf der Aktorenseite auch ein `KukaDriver`, um Bewegungen mit vorhandenen Kuka-Robotern ausführen zu können. Durch den modularen Aufbau der Software muss zu deren anschließender Verwendung lediglich eine Referenz im Projekt geändert werden.

Das in dieser Arbeit realisierte OPVS -Projekt „FaceTracker“ stellt eine einfache Implementierung der Möglichkeiten dar, das an vielen Stellen erweitert werden könnte. Zur Beschleunigung der Gesichtserkennung wäre es beispielsweise möglich, eine Art Bootstrapping der Bilderfassung durchzuführen, bei der die Umgebung zuerst ohne Personen aufgenommen wird. Von anschließend aufgenommenen Livebildern könnte dieses „Leerbild“ abgezogen werden, um nur die geänderten Bildbereiche zu ermitteln. Eine Anwendung der Haar-Kaskade auf diesen deutlich kleineren Bildausschnitt kann dann zu einer signifikanten Beschleunigung der Bilderkennung führen. Auch

der Prozess der eigentlichen Gesichtswiedererkennung kann durch Einführung einer Klassifikationsentscheidung auf Basis einer Mehrheitsmeinung verbessert werden, die unter der Annahme errechnet wird, dass an einer Stelle im Bild nicht zwei Personen erkannt werden können. Damit ist es möglich, die erkannte Person über konsekutive Bildaufnahmen sicherer zu erkennen („im Bildauschnitt ist seit 20 Frames Person A erkannt, dies ignoriert die potentiell fälschliche Erkennung als Person B in Frame 21“). In der Loslösung des Trackingproblems vom Auslesen aller Informationen aus einem einzigen Bild und dem damit verbundenen Übergang zur Betrachtung eines vorliegenden Videostreams liegen noch weitere Optimierungsmöglichkeiten. So kann die Qualität des Trackings durch Einbringen vorhandenen a-priori-Wissens verbessert werden. Da sich eine Person immer regelmäßig durch ein kontinuierlich erfasstes Bild bewegt, ist beispielsweise kein Springen von einer Bildecke in die entgegengesetzte möglich.

# A Grundlagen zum Verständnis dieser Arbeit

## A.1 Informatische Grundlagen

### A.1.1 Verwendete Techniken

Die Umsetzung der Referenzimplementierung erfolgte mit MATLAB Release 7.9.0.

Die Realisierung von OPVS erfolgte in C# für .NET 3.5SP1 unter Verwendung diverser, im Folgenden benannter Tools. Die Realisierung der Ablaufsteuerung verwendet IronPython, eine Pythonimplementierung für die Microsoft Common Language Runtime (CLR). Eine umfassende Einführung in das leider nicht mehr ganz aktuelle Python 2.5 liefert das exzellente Grundlagenwerk [17]. Sämtliche Konfigurations- und Projektdefinitionsdateien liegen als XML-Dateien vor. Aufbau und Format der Dateien sind über XML-Schemas selbsterklärend abgelegt. Zum Bearbeiten wird ein XML-Editor mit integrierter XSD-Validierung empfohlen.

Geschrieben schließlich wurde die Diplomarbeit mit LaTeX unter Windows mit MiKTeX 2.8.

### A.1.2 Objektorientierung

Die Grundlagen der Objektorientierung können in [29] nachgelesen werden. Grundlegende Begriffe und wichtige Prinzipien des objektorientierten Softwareentwurfs, die in der Realisierung dieser Arbeit eine entscheidende Rolle gespielt haben, sollen hier allerdings genannt und ihrer Idee nach kurz vorgestellt werden.

**Grundbegriffe** Die kleinste Einheit in der objektorientierten Softwareentwicklung stellt das *Objekt* dar. Ein Objekt unterliegt einem Lebenszyklus, der vom Programm gesteuert wird: es wird vor der Verwendung im Speicher erzeugt (über einen *Konstruktur instanziert*), vom Programm benutzt (über Attribute und Methoden sind die internen Zustände manipulierbar) und abschließend wieder aus dem Speicher entfernt. Ein Objekt ist dabei die Instanz einer *Klasse*. Die Klasse stellt hier ein

abstraktes Modell, eine Art Bauplan des Objekts dar.[1] Zur Verwendung untereinander können Objekte aktiv durch ihre Methoden angesprochen werden oder passiv über *Ereignisse* andere Komponenten selbst informieren. Weiterhin repräsentieren *Schnittstellen* (engl. *„interfaces“*) Klassen abstrakt zuordenbare Eigenschaften oder Funktionalitäten. Implementiert eine Klasse eine Schnittstelle, so muss sie für die in der Schnittstelle angegebenen Methoden eine eigene Implementierung bereitstellen.[2] Über *Vererbung* können Abstraktionen abgebildet werden. Eine abgeleitete Klasse erbt dabei alle Methoden und Eigenschaften ihrer Basisklasse.[3]

**Prinzip der einzigen Verantwortung (single responsibility principle)** Dieses Axiom besagt, dass einem Modul genau eine Aufgabe zugeordnet ist und umgekehrt für eine Aufgabe genau ein Modul verantwortlich ist. Dies reduziert den Änderungsaufwand in der Software auf genau einen Punkt, wenn sich eine Anforderung ändert. In OPVS existiert so z.B. nur ein `GenericCamDriver`, der von der Gesichtserkennung in Betriebs- und Trainingsphase verwendet wird.

**Prinzip der Trennung der Anliegen (separation of concers)** Dieses Prinzip besagt, dass für jede Aufgabe genau eine festgelegte Komponente zuständig ist, die Aufgabe also nicht über mehrere Module verteilt realisiert ist. Dies reduziert die Abhängigkeiten einzelner Komponenten untereinander und führt zu einer besseren Abstraktion der Gesamtlösung. In OPVS beispielsweise sind alle möglichen Operationen in strikt von einander getrennte Objektklassen wie Aktoren, Sensoren und Klassifikatoren getrennt, Funktionalitäten also nicht über eine solche Domänengrenze hinweg implementiert.

**Prinzip: Wiederholungen vermeiden (don't repeat yourself)** Die Grundaussage dieses Prinzips ist es, dass eine bestimmte Funktionalität nur einmalig in der Software umgesetzt wird. Dem liegt die Idee zu Grunde, dass im Falle einer notwendig werdenden Änderung diese nur an einer einzigen Stelle im Quellcode eingepflegt werden muss. Das Prinzip führt damit aber auch zu einer stärkeren Abstraktion im Quellcode, da kleinere Abweichungen in sonst gleichen Kontrollflüssen dem Modul von außen beigefügt können werden müssen.

[1] Ein beliebtes Beispiel in der Literatur ist die Klasse „Auto“, der das Objekt „VW Passat“ gegenübersteht.

[2] In unserem Beispiel müsste ein „Auto“ mit Automatikgetriebe die Schnittstelle „Automatikgetriebe“ implementieren.

[3] So könnte im Beispiel die Klasse „Auto“ eine Generalisierung der Basisklasse „Fahrzeug“ darstellen.

**Prinzip: Offen für Erweiterung, geschlossen für Änderung (open-closed-principle)** Dieses Prinzip beschreibt eine Kernanforderung an ein modulares Softwaresystem: Es muss für spätere Erweiterungen über neue Erweiterungsmodule offen sein, die dem System jeweils nur die neue Funktionalität beifügen. Dabei darf eine Erweiterung aber unter keinen Umständen eine Änderung am Kern oder bereits vorhandenen Modulen der Software nötig machen.
In OPVS stehen daher für die häufigsten Aufgaben einer Visual Servoing-Anwendung abstrakte Schnittstellen wie `ISensor`, `IActor` oder `IClassificator` bereit, die als Grundlagen für ein Erweiterungsmodul verwendet werden können. Ein neuer Robotertreiber wird keine Änderung an der OPVS Runtime benötigen.

**Prinzip der Trennung der Schnittstelle von der Implementierung (program to interfaces)** Die Grundidee dieses Prinzips ist es, dass ein modulares System beliebig erweiterbar sein soll. Da aber während der Entwicklung des Systems nie alle möglichen späteren Erweiterungsmodule bekannt sein können, darf die Implementierung nicht auf spezielle Klassen erfolgen, sondern nur gegen Schnittstellen.
In OPVS bedeutet dies, dass jedes Modul, egal ob Sensor, Aktor oder Klassifikator, von der Runtime über `IRunnable` steuerbar ist, ohne dass diese speziell mit Instanzen von `GenericCamDriver` oder `FaceDetector` arbeitet - sie startet beispielsweise alle Komponenten über ihre Schnittstelle `(IRunnable)module.Run()`.

**Prinzip: Umkehr der Abhängigkeiten (dependency inversion principle)** Auch dieses Prinzip soll zu einer Reduzierung der Abhängigkeiten zwischen einzelnen Softwaremodulen führen. Bezogen auf OPVS bedeutet dies, dass ein `GenericCamDriver` nicht wissen muss, dass seine Sensordaten in einem `FaceTracker` verarbeitet werden können. Ein `FaceTracker` wiederum muss Bildinformationen als Eingangsdaten erhalten, er muss jedoch nicht wissen, dass diese Daten aus einem speziellen `GenericCamDriver` stammen. Für `FaceTracker` ist es lediglich entscheidend, dass ihm ein Sensor Daten im Format `Image<TColor,TDepth>` zur Verfügung stellt.
Dieses Prinzip wird in OPVS außerdem über das *dependency injection*-Pattern angewendet. Es beschreibt eine Schnittstelle, um einer Komponente von außen eine weitere zur Laufzeit einzufügen (`FaceTracker` bekommt von `OPVS Runtime` während der Ausführung eine Referenz auf `GenericCamDriver` übergeben), ohne dass das spezielle Objekt bereits in der Entwicklung verfügbar oder vorhanden sein muss. Dies entkoppelt die einzelnen Module schlussendlich weiter voneinander und macht sie so beispielsweise separat testbar.

**Prinzip: Umkehrung des Kontrollflusses (inversion of control)** In der Literatur wird dieses Prinzip als das sogenannte *Hollywood-Prinzip* beschrieben, da seine Grundidee auf der Aussage „Don't call us, we'll call you." besteht. Die dahinterstehende Idee ist es, dass einzelne Module sich nicht untereinander selbst aufrufen sollten, sondern dies von einer Hauptkomponente, die als Dispatcher arbeitet, von außen erledigt wird. In OPVS übernimmt diese Rolle die OPVS Runtime.

## A.2 Mathematische Grundlagen

Zum Verständnis der in dieser Arbeit verwendeten Gleichungen wird die Kenntnis von Matrix- und Vektorrechnungsgrundlagen vorausgesetzt - eine gute Einführung dazu liefert [11]. Weiterhin sind statistische Grundlagen von Vorteil.

### A.2.1 Koordinaten

Koordinatensysteme (KOS) dienen der eindeutigen Beschreibung von Punkten im zu beschreibenden Raum. In einer *Koordinate* werden Werte aller Raumachsen durch Zahlenwerte beschrieben. In dieser Arbeit werden wir ausschließlich mit dem Sonderfall der affinen (oder auch geradlinigen) orthogonalen Koordinaten arbeiten. Diese Beschreibung unserer realen Welt wird auch als *kartesisches Koordinatensystem* bezeichnet. Es zeichnet sich dadurch aus, dass alle Raumachsen orthogonal aufeinander stehen. Zweidimensionale kartesische Koordinaten werden in Vektorschreibweise als $p^{(2)} = [x_p, y_p]^T$ angegeben, dreidimensionale Koordinaten als $p^{(3)} = [x_p, y_p, z_p]^T$.
Zur Beschreibung einer Szene verwenden wir das sogenannte *Weltkoordinatensystem*. Dieses definiert einen absoluten Bezugspunkt $O$ im Raum (meistens den Koordinatenursprung des KOS), zu dem alle anderen Objekte im Raum relativ positioniert werden (Translation $T$).
Da Punkte eine infinitesimal kleine Raumausdehnung (0-D) haben und keine Orientierung, benötigen wir zur genauen Beschreibung der Lage eines Objekts im Weltkoordinatensystem zusätzlich noch Informationen über ihre Ausrichtung, die sogenannte Rotation $R$.

### A.2.2 Translation und Rotationen

Die gewünschte Verwendung beider Koordinatensysteme macht eine einfache Möglichkeit, um *Koordinaten* in einen Punkt im Koordinatensystem $\vec{X}_A = [x_a, y_a, z_a]^T$ in einen Punkt im Koordinatensystem $\vec{X}_B = [x_b, y_b, z_b]^T$ zu überführen, nötig. Dies wird über eine *Translation* (Verschiebung) des Koordinatenursprungs von $\vec{X}_A$ in den Koordinatenursprung von $\vec{X}_B$ und anschließende *Rotation* (Verdrehung) erreicht. Wie

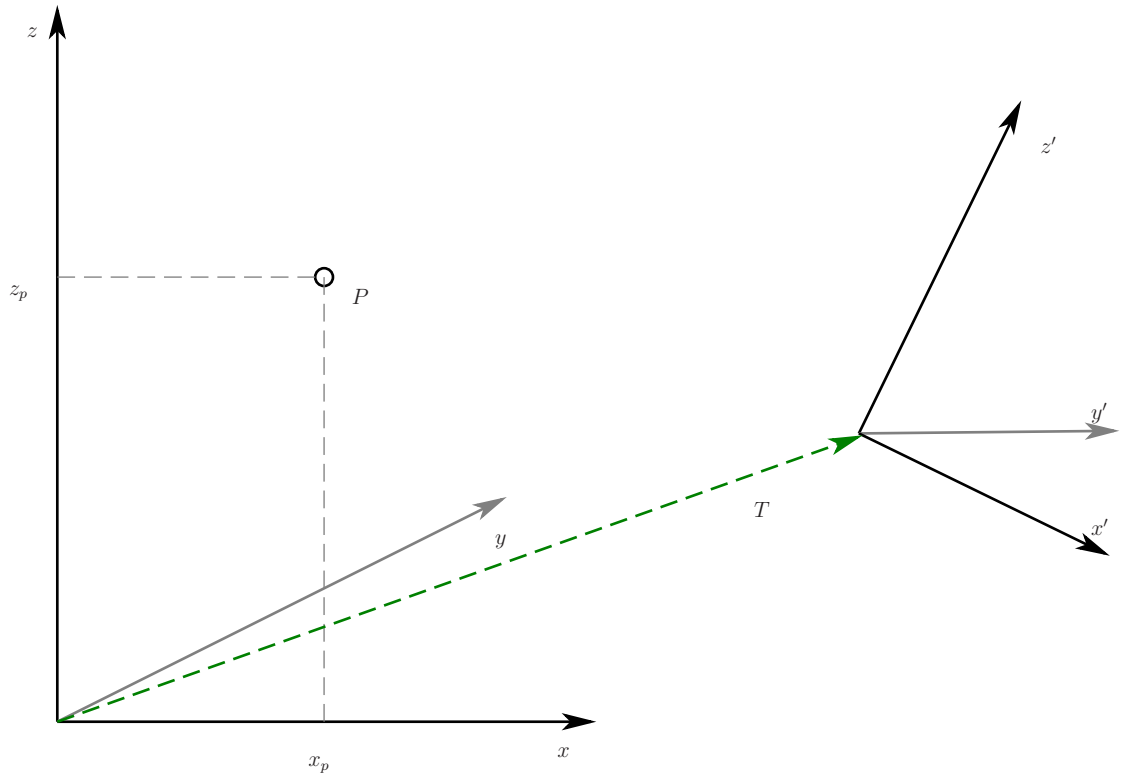



Abbildung A.1: Weltkoordinatensystem mit Punkt $P$, Translation $T$ in verdrehtes Objektkoordinatensystem

[26] (7.2.1) einführt, wird die Translation als Vektorsubtraktion mit dem Translationsvektor $\vec{T}$ und die Rotation als Matrixmultiplikation mit der Rotationsmatrix $\mathbf{R}$ durchgeführt:

$$\vec{X}_B = \mathbf{R}\left(\vec{X}_A - \vec{T}\right) \tag{A.1}$$

Diese Rotationsmatrix $\mathbf{R}$ ist eine Zusammenfassung dreier einzelner Rotationen:

1. Rotation um $x$-Achse um den Winkel $\phi$ mit $\vec{X}'_A = \mathbf{R}_x(\phi)\vec{X}_A$:

$$\mathbf{R}_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \tag{A.2}$$

2. Rotation um die entstandene $y$-Achse um den Winkel $\theta$ mit $\vec{Y}''_A = \mathbf{R}_y(\theta)\vec{Y}'_A$:

$$\mathbf{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \tag{A.3}$$

3. Rotation um die entstandene $z$-Achse um den Winkel $\psi$ mit $\vec{Z}'''_A = \mathbf{R}_z(\psi)\vec{Z}''_A$:

$$\mathbf{R}_z(\psi) = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.4}$$

Die inverse Transformation von $\vec{X}_B$ nach $\vec{X}_A$ muss unter Einhaltung dieser Reihenfolge passieren, da die verwendete Matrixmultiplikation und damit die gesamte Rotation nicht kommutativ ist.

**Rodrigues-Transformation** Die *Rodrigues-Transformation* beschreibt eine Rotation nicht über die bislang beschriebenen $3 \times 3$-Matrizen $\mathbf{R}$, sondern als Rotationsvektor $\vec{r} = [r_x r_y r_z]$, um den gedreht wird. Der Drehwinkel $\theta$ ergibt sich aus der Länge des Rotationsvektors zu $\theta = |\vec{r}|$. Damit erhält man die Rotationsmatrix

$$\mathbf{R} = \cos\theta I + (1 - cos\theta)\vec{r}\vec{r}^T + \sin\theta \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ r_y & r_x & 0 \end{bmatrix}$$

Die Überführung von aus der Matrixnotation in die Rodrigues-Notation erfolgt umgekehrt mit

$$\sin\theta \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ r_y & r_x & 0 \end{bmatrix} = \frac{(\mathbf{R} - \mathbf{R}^T)}{2}$$

Diese Darstellung einer Rotation mit drei Komponenten ist für numerische Berechnungen schneller zu optimieren als die geläufigen neun Komponenten der Rotationsmatrizen. Die in dieser Arbeit verwendete Bibliothek OpenCV verwendet daher intern die Rodrigues-Notation.

### A.2.3 Homogene Koordinaten und Transformationen

Die in A.2.2 eingeführten Transformationen Translation und Rotation werden durch unterschiedliche Matrizen beschrieben. Die Idee der homogenenen Koordinaten ist es, einen Formalismus zu finden, der alle diese Transformationen durch eine einheitliche Multiplikation ermöglicht.

Die Überführung von *kartesischen Koordinaten* in homogene Koordinaten findet durch Hinzufügen einer vierten Dimension $t$ statt:

$$\vec{X}_A \longrightarrow \vec{X}'_A = [tx_a, ty_a, tz_a, t]$$

Alle Transformationen werden nun verallgemeinert durch (4×4)-Matrizen dargestellt. Wir erhalten für die Translation $T$ um $\vec{T} = [x_T, y_T, z_T]^T$

$$T = \begin{pmatrix} 1 & 0 & 0 & x_T \\ 0 & 1 & 0 & y_T \\ 0 & 0 & 1 & z_T \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Für die Rotationen um die drei Koordinatenachsen gilt dann

$$\begin{aligned} \mathbf{R}_x(\phi) &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ \mathbf{R}_y(\theta) &= \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ \mathbf{R}_z(\psi) &= \begin{pmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

Weiterhin führen wir die Skalierung $\mathbf{S}$ als

$$\mathbf{S} = \begin{pmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

ein.

### A.2.4 Faltung

Eine *Faltung* oder auch *Konvolution* zweier Funktionen $f(t)$ und $g(t)$ auf den Intervall $D$ wird als $f * g$ notiert und ist definiert als

$$(f * g)(t) = \int_D f(\tau) g(t - \tau) d\tau$$

Verstehen kann man den Faltungsoperator als eine Gewichtung einer Funktion durch eine andere. Bei geeigneter Wahl der faltenden Funktion lassen sich somit verschiedene *Filterwirkungen* erzielen.

Betrachten wir ein Bild nun als diskrete Funktionen $f$ (eine genaue Einführung dazu folgt in [26]), können wir die *diskrete Faltung* mit der *Filtermaske* (auch *Kernel*) $g$ einführen als

$$(f * g)(n) = \sum_{k \in D} f(k)g(n-k).$$

Bildlich gesprochen wird jeder Bildpunkt des Originals auf einen Bildpunkt im Resultat unter Einbeziehung seiner Nachbarpixel mit der Maske $g$ abgebildet.

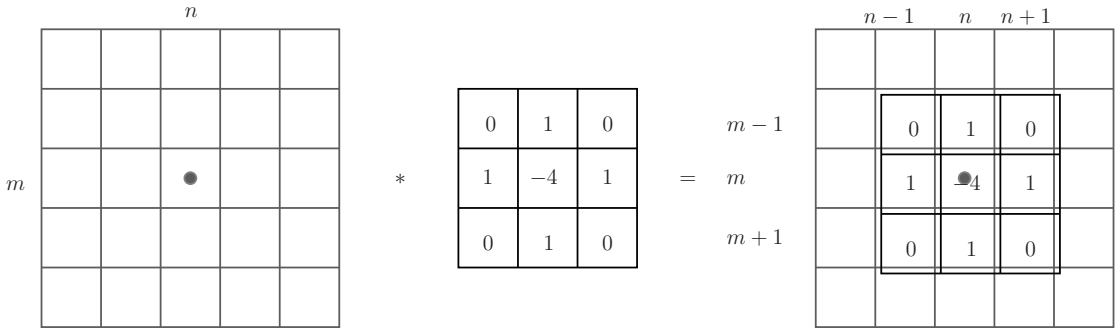



Abbildung A.2: Darstellung der diskreten Faltung mit einer (3×3)-Filtermaske (Laplace-Filter zur Kantenerkennung)

Diese Operation wird in der digitalen Bildverarbeitung auch als *Nachbarschaftsoperator* bezeichnet. Eine detaillierte Einführung dazu liefert erneut [26].

### A.2.5 Hauptkomponentenanalyse PCA

Das Verfahren der *Hauptkomponentenanalyse* (engl. *„PCA = principal component analysis“*) beschreibt eine Methode, um Daten aus einem hoch-dimensionalen Raum in einen niedriger-dimensionalen Unterraum bei möglichst geringem Informationsverlust abzubilden. Sie wurde erstmals 1901 von Karl Pearson in [44] beschrieben. Die Methode ist speziell in der Bildverarbeitung auch unter dem Namen *Karhunen-Loeve-Transformation* bekannt.

**Verfahren** Ausgehend von $n$ Datensätzen mit je $p$ Merkmalen wird eine reduzierte Darstellung gesucht, die mit $q$ Merkmalen ($q < p$) bei minimalem Informationsverlust auskommt. Im Kern wird mathematisch eine Hauptachsentransformation durchgeführt, da versucht wird, die Korrelation zwischen den Merkmalen zu minimieren und sie entlang der Achsen mit dem stärksten Einfluss neu auszurichten. Dies

wird durch Überführen der Kovarianzmatrix in eine Diagonalmatrix erreicht, da hier die nicht auf der Hauptdiagonale liegenden Elemente die Korrelation der einzelnen Werte angeben. In ihrer Realisierung liegt damit eine Kleinste-Quadrate-Methode für die statischen *Momente zweiter Ordnung* (*Varianz*) vor. Aus den Datenvektoren

$$X_j = (x_1, ..., x_p) \qquad \forall j = 1, ..., n$$

erhalten wir mit dem *Mittelwert* $\overline{X}$ (*Moment erster Ordnung*)

$$\overline{X} = \frac{1}{n} \sum_{j=1}^{n} X_j$$

die *Varianz* $\mathrm{var}(x)$ und *Kovarianz* $\mathrm{kov}(x, y)$ zu

$$\begin{aligned} \mathrm{var}(X_j) &= \frac{1}{n-1}(X_j - \overline{X})^T (X_j - \overline{X}) \\ \mathrm{kov}(X_i, X_j) &= \frac{1}{n-1}(X_i - \overline{X})^T (X_j - \overline{X}) \end{aligned}$$

Fassen wir nun alle möglichen Kovarianzen der Datenvektoren $X_j$ in der *Kovarianzmatrix* $\mathbf{C}$ zusammen

$$\mathbf{C} = \begin{pmatrix} \mathrm{kov}(X_1, X_1) & \cdots & \mathrm{kov}(X_1, X_n) \\ \vdots & \ddots & \vdots \\ \mathrm{kov}(X_n, X_1) & \cdots & \mathrm{kov}(X_n, X_n) \end{pmatrix}$$

Für diese Kovarianzmatrix bestimmen wir die Eigenwerte $\lambda_j$ und führen sie als Diagonalelemente in der Diagonalmatrix $\mathbf{\Lambda}$ auf. Die zugehörigen Eigenvektoren spannen dann den reduzierten Unterraum $\mathbf{\Gamma}$ auf. Es gilt hier $\mathbf{\Lambda} = \mathbf{\Gamma}^T \mathbf{C} \mathbf{\Gamma}$. Durch Projektion der Datenvektoren $X_j$ in den Unterraum $\mathbf{\Gamma}$

$$X_j \mapsto Y_j = \mathbf{\Gamma}^T X_j$$

erhält man die reduzierte Darstellung $Y_j$. Die Wahl der $q$ größten Eigenwerte zur Dimensionsreduzierung hat entscheidenden Einfluss auf die Größe des Abbildungsfehlers und spielt daher auch bei der Anwendung der Methode bei den Eigengesichtern eine entscheidende Rolle.

### A.2.6 Wavelets

*Wavelets* (vom französischen „onde“ = „Welle“, übertragen ins Deutsche etwa „Wellchen“) ist die Bezeichnung der Funktionen, die die Grundlage einer sogenannten Wavelet-Transformation bilden. Die Wavelet-Transformation stellt eine Sonderform einer Zeit-Frequenz-Transformation dar[4], die hier allerdings nicht eingeführt werden soll. Es sei hierzu auf [27] und [65] verwiesen, die beide eine gute Einführung in die Thematik geben.

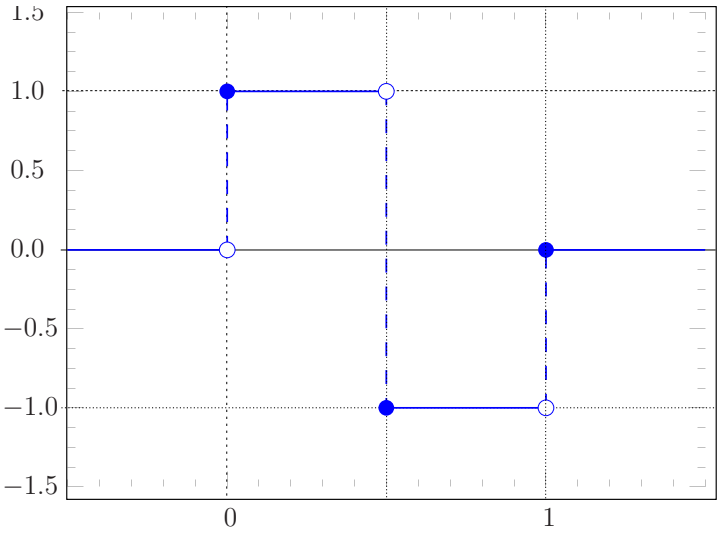



Abbildung A.3: Darstellung des Haar-Wavelets

Erwähnt soll an dieser Stelle nur werden, dass die in dieser Arbeit thematisierte Haar-Kaskade (siehe Abschnitt 2.3.2) ihren Namen durch ihre visuelle Ähnlichkeit zum *Haar-Wavelet* erhielt, welches der deutsche Mathematiker Alfred Haar 1901 in die Wavelet-Theorie einbrachte.

### A.2.7 Maschinelles Lernen

**Hinführung** *Maschinelles Lernen* ist ein Forschungsgebiet der Künstlichen Intelligenz. Es beschreibt eine Fülle verschiedener Ansätze mit deren Hilfe eine schlüssige Zuordnung von Datenvektoren zu Klassen vorgenommen werden kann. In dieser Arbeit wurden ausschließlich Verfahren der Statistik betrachtet und absichtlich stochastische Methoden (Bayes'sche-Netze, Markov-Netzwerke) unbetrachtet gelassen. Bei statistischen Methoden unterscheidet man zwischen autonomen Verfahren (auch *generative Algorithmen*), die eine *Klassifikation* ohne Eingreifen eines Menschen nur

[4] ähnlich der Fourier-Transformation

aus den Daten heraus vornehmen, und überwachten Verfahren, bei denen Expertenwissen mit in die Klassifikation einfließt. Im folgenden sollen in der Arbeit erwähnte Verfahren und Hilfsmittel kurz vorgestellt werden und weiterführende Literatur angegeben werden.

### Distanzmaße

Distanzmaße (auch *Norm*) dienen der Bestimmung der Unterschiedlichkeit (Distanz) $d$ zweier isodimensionaler Datenvektoren $x$ und $y$ und stellen eine Grundlage vieler multivariater Methoden der Datenanalyse dar. Es ist eine Vielzahl möglicher Distanzmaße bekannt, von der die in dieser Arbeit verwendeten nun eingeführt werden sollen.

**Minkowski-Distanz** Die nach dem deutschen Mathematiker und Physiker Hermann Minkowski benannte *Minkowski-Distanz*

$$d = \left[ \sum_{k=1}^{n} |x_k - y_k|^p \right]^{\frac{1}{p}}$$

stellt eine Verallgemeinerung einiger bekannter Abstandswahrnehmungen dar. So ist $p = 1$ die sogenannte *Manhattan-Distanz* (auch *City-Block-Distanz*), bei der die Distanz als die Summe der absoluten Differenzen der einzelnen $k$ definiert wird. Für $p = 2$ erhält man die im $\mathbb{R}^2$ aus dem Satz des Pythagoras bekannte *Euklidische-Distanz*. Für $p \to \infty$ schließlich erhält man die *Maximum-Distanz* (auch *Tschebyschow-Distanz* genannt), die den maximalen Abstand $\max_k |x_k - y_k|$ liefert.
Bei allen diesen *Metriken* müssen zwei wichtige Gesichtspunkte beachtet werden: Sie sind nicht skaleninvariant[5] und gehen nicht auf die Korrelation zwischen den Merkmalen ein.

**Mahalanobis-Distanz** Die *Mahalanobis-Distanz* behebt diese Probleme und ist unter Verwendung der *inversen Kovarianzmatrix* $\mathbf{C}^{-1}$ definiert als

$$d = \sqrt{(x-y)^T \mathbf{C}^{-1} (x-y)}$$

Im Gegensatz zu anderen Verfahren wird hier auch die Ausdehnung des Datenraums - eben über die Kovarianz - berücksichtigt. Der Vorteil dieses Verfahrens liegt damit

[5] Fassen wir den Datenvektor als $n$-dimensionalen Spaltenvektor aller $n$ bestimmen Zufallsvariablen $x_n$ auf, so überdecken sehr große Werte $x$ kleinere, womit Unterschiede zwischen den einzelnen Werten nicht gleich gewichtet betrachtet werden.

auf der Hand: Es findet eine Normierung der Datenwerte innerhalb der Datenvektoren über die Varianz statt, wodurch eine Entzerrung erreicht wird. Im Resultat ist die Distanz skalen- und translationsinvariant. Die verwendete Kovarianzmatrix enthält dabei auf der Hauptdiagonalen die Varianzen der einzelnen Datenwerte des Datenvektors. Ist diese Kovarianzmatrix eine Einheitsmatrix[6], so verhält sich $\mathbf{C}^{-1}$ als neutrales Element und es liegt wieder die bereits bekannte *euklidische Distanz* vor.

**Kosinus-Distanz** Einige Arbeiten verwenden die sogenannte *Kosinus-Distanz*, die sich aus dem *Skalarprodukt*

$$\vec{x} \cdot \vec{y} = |\vec{x}||\vec{y}| \cos \gamma$$

zu

$$d_{\cos} = 1 - \frac{\sum_{k=1}^{n} x_k y_k}{\sqrt{\sum_{k=1}^{n} x_k^2 \sum_{k=1}^{n} y_k^2}}$$

errechnet. Mit ihr sind Aussagen über Orientierung zweier Datenpunkte möglich, da für $\gamma = 0°$ (Vektoren sind parallel) $d_{\cos} = 0$ folgt.

### Entscheidungsbäume

*Entscheidungsbäume* (engl. *„decision trees“*) dienen der strukturierten Abbildung von Entscheidungsregeln. Dabei stellt der namengebende *Baum* eine Struktur (einen sogenannten *Graphen*) aus der Graphentheorie dar. Jeder Entscheidungsbaum besteht aus einem *Wurzelknoten* und (in beliebiger Tiefe) weiteren Entscheidungsknoten[7] oder Blättern[8] darunter. In der theoretischen Informatik dient der Baum weiterhin als Datenstruktur. Es wurde hier eine Vielzahl von möglichen Realisierungen (balancierte Bäume, binäre Bäume und viele weitere) eingeführt und algorithmisch betrachtet, so dass Bäume heute die Grundlage der meisten effizienten Speicher- und Suchverfahren darstellen.

Relevant für diese Arbeit sind die in der *Haarkaskade* zum Einsatz kommenden *binary decision trees*, die von Breiman in [9] als CART (engl. *„classification and regression trees“*) eingeführt wurden. In diesem Buch wird zusätzlich die Unterscheidung zwischen *Klassifikationsbäumen* (zur Prognose qualitativer Daten und damit der

[6] Ist $\mathbf{C} = \mathbf{I}$ bedeutet dies, dass die Vektoren nicht korreliert sind, da alle Kovarianzen null sind.

[7] Diese Entscheidungsknoten kann man sich als Frage vorstellen, deren Antwort über weitere Knoten und Blättern gegeben werden.

[8] Blätter stellen hier die endgültige Entscheidung dar.

Klassenzugehörigkeit dieser Daten) und *Regressionsbäumen* (zur Prognose quantitativer Daten) herausgearbeitet. Prinzipiell dienen Entscheidungsbäume immer dem Ziel, die vorhandenen Daten als sogenannte *Kovariablen* möglichst optimal voneinander zu trennen (engl. *„to split“*). Diese Trennung erfolgt unter dem Kriterium der *Entropiereduzierung* über einen Split, wobei die möglichen Splitpunkte über sogenannte *Heterogenitätsmaße* ermittelt werden. Die Heterogenitätsmaße selbst werden dabei über die geschätzte Klassenzugehörigkeit errechnet. Abschließend wird der entstandene Baum durch *Beschneiden* (engl. *„pruning“*) über ad hoc definierte Kriterien (optimale Baumgröße, cost-complexity-pruning und viele andere) auf wesentliche Kriterien verkürzt.
Die größten Vorteile dieses Verfahren stellen die grafische Anschaulichkeit, die Abbildung nichtlinearer Zusammenhänge und auch die Unabhängigkeit der Kovariablen von unterschiedlichen Skalen[9] dar. Nachteilig ist die in A.2.7 beschriebene Anfälligkeit für *Überanpassung* des Klassifikators an die Trainingsdaten, wenn zu viele Trennregeln gefunden werden. Für dieses Problem sind aber Methoden beschrieben, die die Anfälligkeit auf Überanpassung reduzieren oder sogar ganz umgehen. So stellt das *Bagging* [10] eine Methodik bereit, bei der mehrere Bäume aus zufälligen Stichproben der Trainingsdaten erstellt werden und somit die Varianz des Ergebnisses reduziert wird. Sehr gute Ergebnisse werden weiterhin mit sogenannten *random trees* (dt. *„Zufallsbäumen“*) oder auch *random forests* (dt. *„Zufallswälder“*) erzielt. Ein solcher Klassifikator besteht aus verschiedenen voneinander unabhängigen Entscheidungsbäumen, bei deren Training die zu Grunde liegenden Trainingsdaten absichtlich geringfügig verändert wurde. Die endgültige Entscheidung findet dann über eine „Abstimmung“ (engl. *„vote“*) über alle Entscheidungsbäume des Waldes statt. Dieses Verfahren skaliert sehr gut für parallele Verarbeitung, da die Einzelbäume auch auf getrennten Systemen erstellt und berechnet werden können. In dieser Arbeit schließlich kommt das sogenannte *Boosting* zum Einsatz.

### Boosting

**Einleitung** *Boosting* ist ein Verfahrensansatz des maschinellen Lernens, bei dem ein *starker Klassifikator* (oder Lerner) (engl. *„strong classifier/learner“*) aus mehreren *schwachen Klassifikatoren* (engl. *„weak classifier“*) zusammengesetzt wird. Jeder dieser schwachen Lerner stellt dabei einen sogenannten PAC-Lerner (engl. *„probably approximately correct“*) dar [62], der bei binären Problemen mit seinen Entscheidungen nur geringfügig besser als 50% richtig liegen muss. Der schwache Lerner wird dabei intern über einen binären Entscheidungsbaum (siehe A.2.7) realisiert.

[9] vgl. Überdeckung in A.2.7

Ein erster praktikabler Ansatz für Boosting wurde von Freund und Schapire in [21] beschrieben und als Algorithmus unter dem Namen *AdaBoost* (engl. *„adaptive boosting“*) eingeführt. AdaBoosts' größter Vorteil ist es, dass im Gegensatz zu bis dato bekannten Boosting-Verfahren kein Vorwissen über die Genauigkeit des schwachen Lerners benötigt wird, da diese Information adaptiert wird.

**AdaBoost** Boostingalgorithmen werden verwendet, um $T$ schwache Klassifikatoren $h_t$ mit $t \in \{1, ..., T\}$ zu finden. Jeder einzelne dieser schwachen Klassifikatoren erhält anschließend ein „Stimmgewicht“ $\alpha_t$, das seinen Einfluss im starken Klassifikator $H(x)$ zu

$$H(x) = \operatorname{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

bestimmt. AdaBoost verwendet weiterhin ein Datenpunktgewicht $D_t(i)$, das eine Fehlklassifikation eines Datenpunkts für den nächsten schwachen Klassifikator wichtet. Der Algorithmus ist definiert als:

**Algorithmus A.1:** AdaBoost allgemein

**Gegeben** $n$ *Datensätze* $x_1, ..., x_n$ *mit zugeordneten Klassenzugehörigkeiten* $y_1, ..., y_n \in \{-1, +1\}$ ;
**Initialisiere** $D_1(i) = \frac{1}{m}$;
**für** $t = 1, ..., T$ **tue**
  **Wähle** den Klassifikator $h_t = \arg\min_{h_j \in H} \epsilon_j$, der den gewichteten Fehler $\epsilon_j = \sum_{i=1}^{m} D_t(i)$ minimiert, solange $\epsilon_j < \frac{1}{2}$ ist; sonst Abbruch ;
  **Aktualisiere** das Stimmgewicht $\alpha_t = \frac{1}{2} \log\left[\frac{1-\epsilon_t}{\epsilon_t}\right]$ ;
  **Aktualisiere** das Datenpunktgewicht $D_{t+1}(i) = \frac{1}{Z_t} D_t(i) e^{-\alpha_t y_i h_t(x_i)}$, wobei $Z_t$ hier der Normierung aller Datenpunktgewichte dient: $\sum_{i=1}^{n} D_{t+1}(i) = 1$
**Ende**

Eine sehr detaillierte Betrachtung des Boostings liefern Meir und Rätsch in [39]. Die Autoren gehen dabei ausführlich auf die zu Grunde liegende Theorie des Verfahrens ein, betrachten seine Robustheit und führen umfangreiche Untersuchungen zum gemachten Trainingsfehler durch.

### Hinweis zum überwachten Lernen

Einige der beschriebenen Verfahren neigen dazu, sich bei nicht rechtzeitigen Abbrechen des Lernens den Trainingsdaten überanzupassen (engl. *„over-fitting“*) und

damit anstatt des erwünschten Erkennens von Klassen nur die Testdaten zu erkennen. Es hat sich daher folgender Ansatz [8] als Best Practice herausgestellt (vgl. Abbildung A.4):

1. Aufteilen der vorhandenen Daten $D$ in
   - Trainingsdaten $D_L$
   - Testdaten $D_T$
   - optional: Validierungsdaten $D_V$
2. Anlernen eines Klassifikators mit $D_L$
3. optional: Validierung des während des Lernens entstehenden Klassifikators $C'$ mit den nicht zum Trainingsset gehörenden $D_V$
4. Test des Klassifikators $C$ mit $D_T$

Die Testdaten $D_T$ standen dem Klassifikator während der Lernphase nicht zur Verfügung und lassen damit Rückschlüsse auf seine Genauigkeit bei unbekannten Daten zu. Sollte der Klassifikator mit den Testdaten keine befriedigenden Ergebnisse erzielen, so können entweder neue Merkmale zum Klassifikator hinzugefügt werden oder direkt ein neuer Klassifikator gewählt werden.
Die optionale Verwendung eines Validierungsdatensatzes $D_V$ wird zur Steigerung der Effektivität vor allem für Verfahren empfohlen, bei denen der eigentliche Test sehr zeitintensiv ist. Er findet parallel zum Lernen statt und kann im Falle einer negativen Tendenz direkt zum Abbruch des Lernens führen beziehungsweise den Testlauf überflüssig machen.

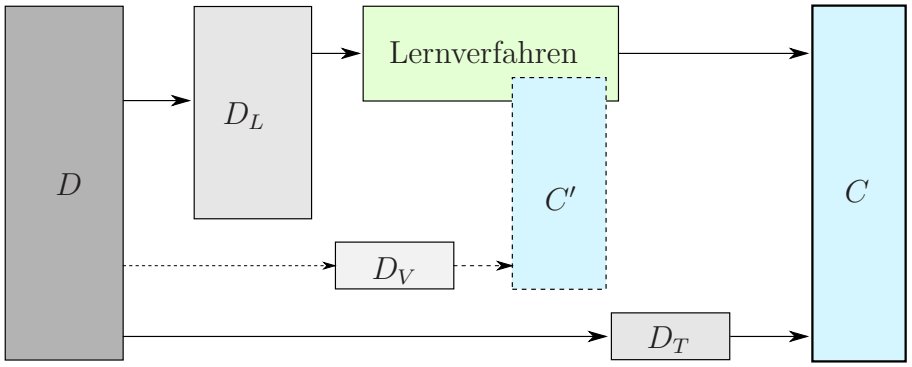



Abbildung A.4: Verfahrensansatz für überwachtes Lernen

## A.3 Optische Grundlagen

Jede Aufgabe in der digitalen Bildverarbeitung beginnt zunächst mit der Aufnahme einer Szene als Bild. Ausgangspunkt jedes Sehens ist immer die Abstrahlung von Licht aus Lichtquellen. Dieses Licht breitet sich ungehindert im Raum aus, bis es auf ein Hindernis trifft. An diesem Hindernis werden die Lichtstrahlen nun reflektiert und zum Teil auch absorbiert. Abhängig vom Verhältnis Reflexion zu Absorption entsteht damit die jeweilige Farbwirkung. Ein Objekt das kein Licht reflektiert ist somit nicht sichtbar („schwarzes Loch“).
In diesem Abschnitt wollen wir uns nun grundlegende Erkenntnisse zur Bildentstehung auf Fotosensorelementen verschaffen und auf mögliche Fehler und Ungenauigkeiten eingehen.

### A.3.1 Kameramodell

**Lochkameramodell** Bei jeder Abbildung mit einer Kamera findet eine Abbildung vom dreidimensionalen Objektraum auf die zweidimensionale Bildebene statt. Dabei findet immer ein Informationsverlust (in Form der Raumtiefe) statt. Für Aussagen über das entstehende Bild benötigen wir ein geeignetes Modell.

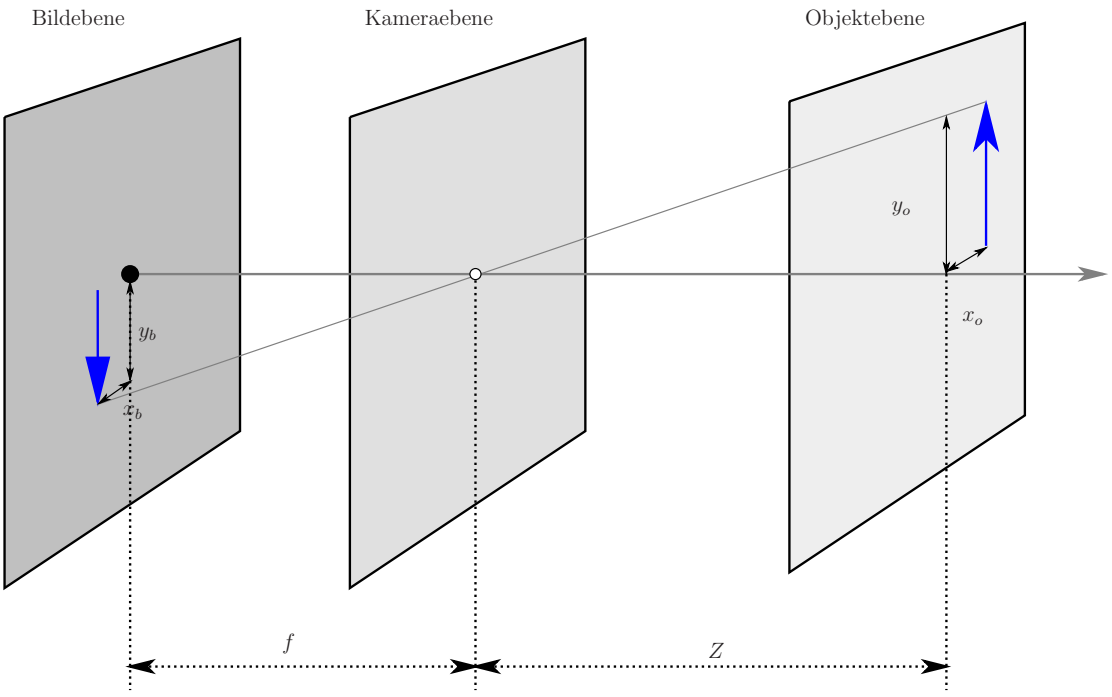



Abbildung A.5: Strahlengang am Lochkameramodell

Das *Lochkameramodell* soll uns hier als erster Angriffspunkt dienen. Es geht von einer Abbildung mit einer infinitesimal kleinen Blende aus. Dabei kann von jedem Raumpunkt aus nur ein einziger Lichtstrahl auf die Bildebene abgebildet werden. Der Lichtstrahl von Punkt $X$ wandert dabei durch die Blende und trifft die Bildebene in Punkt $x$. Wie in Abbildung A.5 leicht ersichtlich ist, sind die Koordinaten $x$ und $y$ auf der Bildebene einfach durch das Verhältnis

$$\frac{x_b}{f} = -\frac{x_O}{Z} \qquad \text{und} \qquad \frac{y_b}{f} = -\frac{y_O}{Z}$$

beschrieben. Da alle Strahlen durch ein Zentrum - die Blende - laufen, können wir die Bilderfassung am Lochkameramodell als eine *perspektivische Projektion* auffassen. Diese sogenannte *Zentralprojektion* ermöglicht es uns theoretisch, Bild- und Kameraebene zu vertauschen (siehe Abbildung A.6). Damit entfällt das lästige Vorzeichen und wir erhalten für die Abbildung eines Objektpunktes $Q$ auf die Bildebene

$$Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \mapsto q = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \end{bmatrix}$$

**Reale Abbildung** Für einen realen Einsatz vereinfacht das Lochkameramodell die Wirklichkeit zu stark. Man benötigt vor allem eine Lösung für die nicht ideale Blendengröße, da gängige CCD-Sensoren erst ab einer Beleuchtungsstärke von 2000 Lux bei Blendenzahl 8 Licht wahrnehmen ([18]). Weiterhin bildet ein reales optisches System ein Objekt nur in einem gewissen Bereich scharf ab. In diesem Zusammenhang wird bei digitalen Bilderfassungssystemen auch von *Tiefenschärfe* gesprochen. Diese Tiefenschärfe beschreibt die maximale akzeptable Unschärfe, die durch die nicht infinitesimal kleinen Sensorelemente (Pixel) des verwendeten Sensors entstehen. Herleitungen dazu sind erneut in [26] und [8] zu finden.

### A.3.2 Kamerakalibrierung

Wie wir im vorhergehenden Abschnitt ermittelt haben, benötigen wir für unser gewähltes Kameramodell mehrere Parameter. Diese Parameter unterteilen sich in *intrinsische Parameter* (der Kamera innewohnende Parameter) und *extrinsische Parameter* (Position und Orientierung der Kamera im Weltkoordinatensystem). Die Bestimmung dieser Parameter ist Aufgabe der *Kamerakalibrierung*. Es existieren hierfür heute mehrere etablierte Verfahren, auf die an dieser Stelle nur kurz verwiesen werden soll: Eine grundlegende Arbeit wurde im Jahr 1987 unter dem Titel *A*

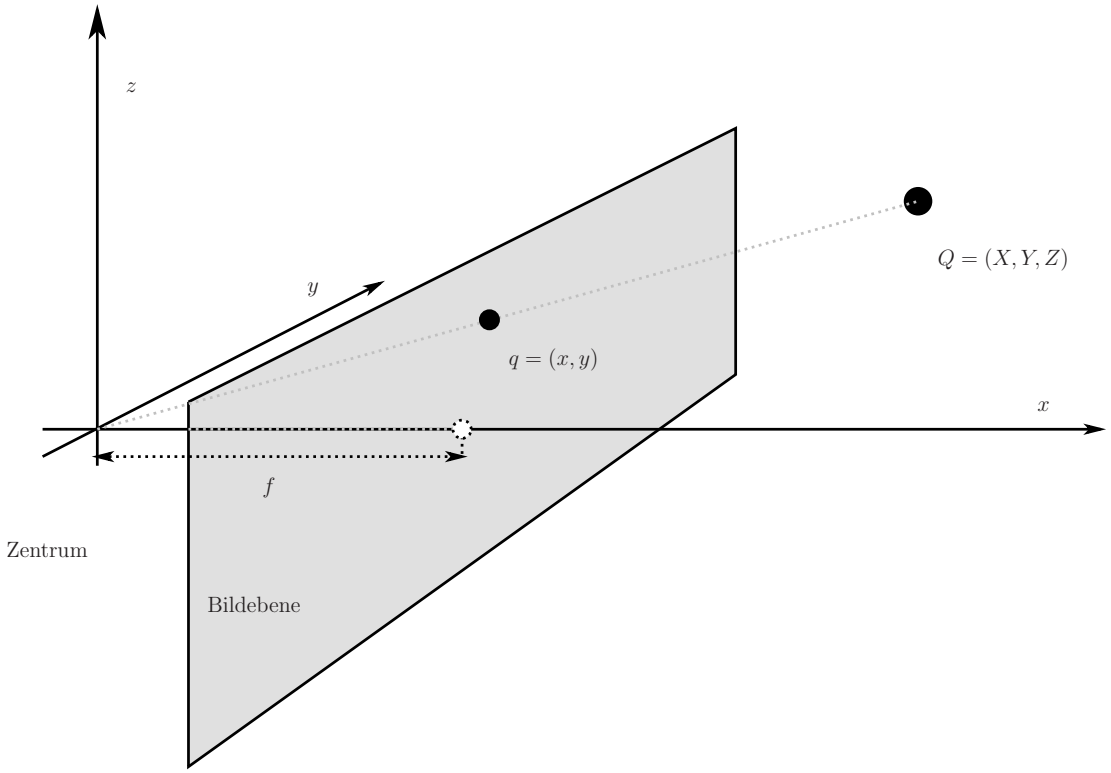



Abbildung A.6: Darstellung einer Zentralprojektion

*versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses* von Roger Tsai veröffentlicht [60]. Darauf aufbauend wurde von Janne Heikkilä im Jahr 1997 unter dem Titel *A Four-step Camera Calibration Procedure with Implicit Image Correction* [25] veröffentlicht. Sein Ansatz wurden direkt in der Standardtoolbox für Kalibrierung in MATLAB implementiert und kann als Standardverfahren angesehen werden.

## A.4 Grundlagen der Bildverarbeitung

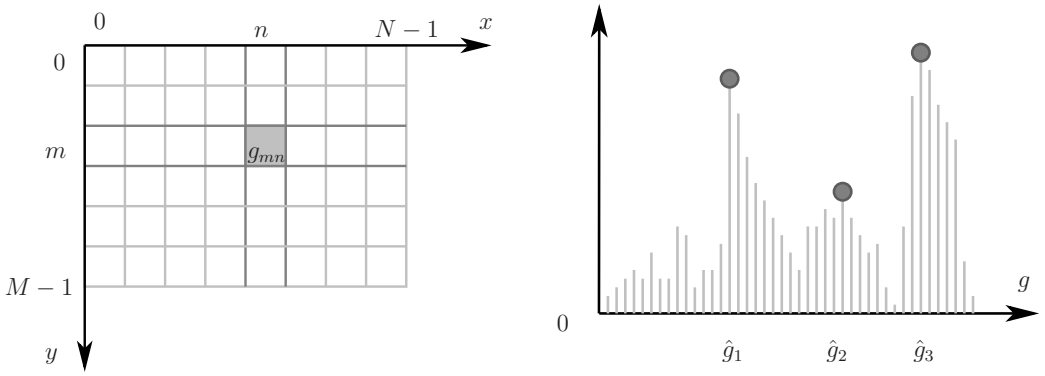
Nachdem wir nun die Grundlagen der Bildentstehung eingeführt haben, wollen wir uns mit der Frage beschäftigen, wie Bilder rechentechnisch erfasst und verarbeitet werden können.

In diesem Abschnitt werden wir ein grundlegendes Verständnis für die algorithmische Arbeit mit Bildern treffen und einige einfache Verfahren der Bildverarbeitung einführen, die wir später in der Software benötigen werden.

### A.4.1 Bildrepräsentation

Jedes zweidimensionale Bild $B$ der Breite $M$ und Höhe $N$ kann als Funktion zweier diskreter Variablen $m$ und $n$ aufgefasst werden. Wir nennen $m$ den Spaltenindex, $n$ den Zeilenindex und definieren

$$\begin{aligned} m &= 0, 1, \ldots, M-1 \\ n &= 0, 1, \ldots, N-1 \end{aligned}$$




(a) Bild als diskrete Funktion

(b) Histogramm eines Bildes mit drei lokalen Maxima

Abbildung A.7: Darstellung eines Bildes als diskrete Funktion, Histogramm eines Bildes

Jedem *Bildelement* $b(x, y)$ (auch *Pixel*, Lehnwort aus dem Englischen für *picture element*) wird dabei in Grauwert $g$ zugeordnet. Dieser Grauwert ist die Repräsentation der Intensität des Lichts an diesem Pixel. Er entsteht durch die Digitalisierung des einfallenden Lichts auf das bilderfassende Element (z.B. den CMOS-Sensor einer Kamera). Typische Kameras benutzen hier eine 8-Bit Digitalisierung, womit sich der Grauwert $g$ zu $g = [2^0 - 1, 2^8 - 1] = [0, 255]$ definiert.
Fassen wir nun die Erfassung der einzelnen Pixel als eine Reihe von Messungen auf[10], können wir den Grauwert als Zufallsvariable $g$ verstehen. Die Wahrscheinlichkeitsdichtefunktion $f(g)$ charakterisiert nun den beobachteten Prozess. Die Häufigkeitsverteilung der Grauwerte (auch *Histogramm*) gibt uns nun Aufschluss über die relative Häufigkeit $f_n(g)$ eines Grauwerts

$$f_n(g) = \frac{f(g)}{A},$$

wobei hier die Gesamtzahl aller Bildpunkte Verwendung findet als $A = M \cdot N$. Wir

[10]Wir betrachten die Bilderfassung hier als stochastischen Prozess.

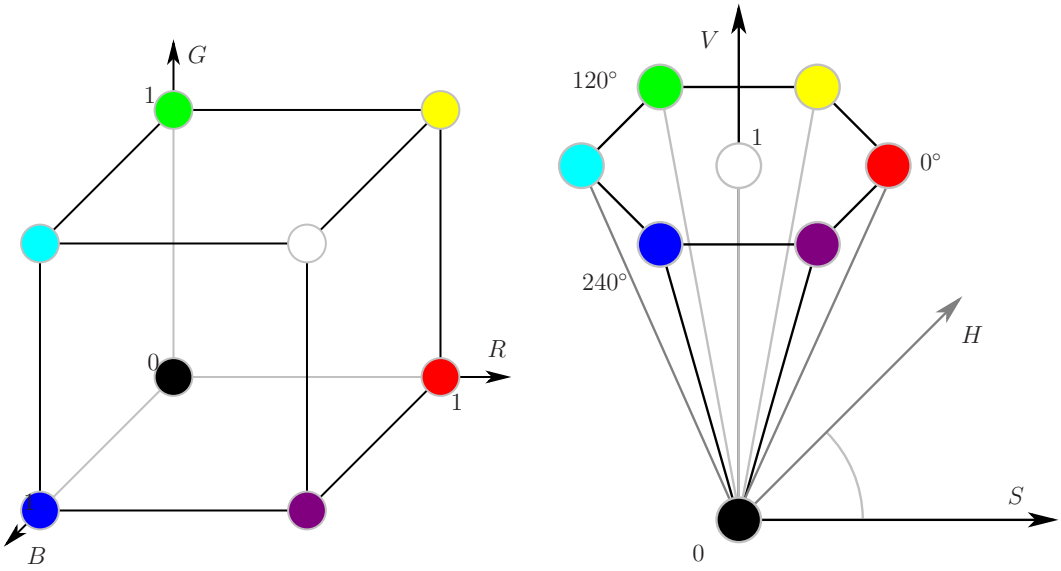
verwenden das Histogramm als Grundlage für viele im Folgenden verwendeten Bildverarbeitungsverfahren (z.B. automatische Bildverbesserung, Segmentierung).
Unsere bisherige Betrachtung kann lediglich zur Darstellung von Graustufenbildern verwendet werden. Farbige Bilder werden erreicht durch das $n$-malige übereinanderlegen von Graustufenbildern, wobei jedem der $n$ Kanäle eine Bedeutung im Sinne eines Farbmodells gegeben wird.

### A.4.2 Farbmodelle

Farbmodelle dienen der Definition einer Nomenklatur zur Festlegung der Farbe eines Pixels. Die Farbinformationen werden dabei für jeden Kanal $k$ und jeden Pixel $a \in A$ separat gespeichert.
Am PC als Standard hat sich der *RGB-Farbraum* etabliert, ein durch additives Mischen von Anteilen der Grundfarben Rot, Grün und Blau definiertes Farbmodell. Es orientiert sich dabei am Farbsehen des Menschen und wird als Farbwürfel in kartesischen Koordinaten dargestellt (siehe dazu Abbildung A.8).
Für den Einsatz in der digitalen Bildverarbeitung ist RGB allerdings nur begrenzt geeignet, da sich Helligkeitsunterschiede extrem auf die Farbrepräsentation auswirken. Hier schafft der *HSV-Farbraum* Abhilfe. HSV steht dabei für *hue* (engl. = Farbton, als Winkel von 0°=Rot, über 120°=Grün und 240°=Blau zurück zu Rot), *saturation* ( = Sättigung, in Prozent von 0%=Neutralgrau bis 100% gesättigte, reine Farbe) und *value* ( = Helligkeit, in Prozent von 0%=keine Helligkeit bis 100%=volle Helligkeit). Treten hier während der Bilderfassung Helligkeitsschwankungen auf, so wirken diese sich primär auf die Helligkeit $V$ aus. Der Farbton $H$ bleibt davon so gut wie unberührt. Damit ist es unter anderem möglich, in einem Bild Objekte nach ihrer Farbe zu finden.

### A.4.3 Bildpyramiden

Bildpyramiden (engl. *„image pyramids“*) stellen eine Sammlung von Bildern verschiedener Auflösung $(G_1, ..., G_n)$ dar, die alle aus einem Originalbild $G_0$ durch Downsampling mit einem Filter erzeugt werden. Die Grundidee dazu wurde von Adelson und Burt in [2] und [1] eingeführt und basiert auf der aus der Signalverarbeitung stammenden Annahme, dass Informationen in Bildern durch verschiedene Frequenzanteile repräsentiert werden, wobei hochfrequente Anteile geringeren Informationsgehalt besitzen. Daher eignen sie sich gut zur recheneffektiven Kompression von Bildern. In der Literatur beschrieben sind vor allem die *Gausspyramide* - zum Downsampling eines Bildes - und die *Laplace-Pyramide* zur Rekonstruktion in einem durch Upsampling erstellten Bild beschrieben.




(a) RGB-Farbmodell

(b) HSV-Farbmodell

Abbildung A.8: Vergleich zwischen RGB- und HSV-Farbmodell

Um unter Anwendung der Gauss-Pyramide aus $G_i$ die Ebene $G_{i+1}$ zu errechnen, wird zunächst eine Faltung mit dem namensgebenden Gauß-Filter $\mathcal{G}$ durchgeführt[11] und anschließend jede gerade Zeile und Spalte aus dem Bild entfernt. Das resultierende Bild ist nun nur noch halb so groß. Dieses Verfahren kann beliebig oft wiederholt werden, wobei ein Bild der Größe 2 px × 2 px das natürliche Abbruchkriterium darstellt.

Um nun aus einem existierenden Bild $G_i$ das in der Pyramide nächstgrößere Bild $G_{i-1}$ zu errechnen, wird die Laplace-Pyramide benötigt. In ihr sind die Schärfeanteile jedes Bilds gesichert.

Beim Vergrößern werden daher die Pixel von $G_i$ in einem leeren Bild doppelter Größe auf die ungeraden Zeilen und Spalten verteilt:

$$\mathrm{PixUp}(I) : I(x, y) \mapsto J(2x + 1, 2y + 1)$$

Die geraden Zeilen und Spalten werden als „fehlende“ Pixel mit Null gefüllt und anschließend über den Gauß-Filter approximiert. Man erhält

$$L_i = G_i - \mathrm{PixUp}(G_{i+1}) * \mathcal{G}$$

[11] Dies entspricht einem Tiefpassfilter in der Signaltheorie.

Dadurch kann ein Bild $G_i^*$ über

$$G_i^* = L_i + \text{PixUp}(G_{i+1}) * \mathcal{G}$$

rekonstruiert werden.

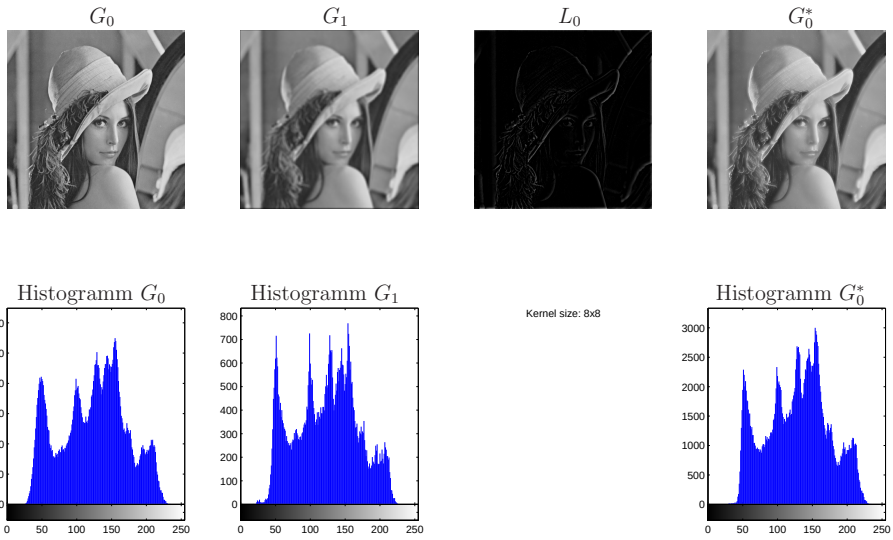



Abbildung A.9: Bildpyramide erster Ordnung mit 8×8 Gauß-Kernel, siehe Quelle C.7

Da viele Bildanalyseverfahren sehr rechenintensiv sind, werden Bildpyramiden heute zumeist für eine „Vorberechnung“ mit kleineren Bildern verwendet (z.B. in der Bildsegmentierung), wobei durch die festen Beziehungen der Pixel der einzelnen Pyramidenstufen untereinander anschließend Rückschlüsse auf das Originalbild möglich sind.

### A.4.4 Histogrammausgleich

Mit *Histogrammspreizung* (auch *Tonwertspreizung*) (engl. *„histogram equalization“* = dt. *„Histogrammausgleich“*) wird ein Verfahren der Kontrastverstärkung für kontrastarme Grauwertbilder beschrieben.
Es wird dabei der Grauwertbereich $\{g_{min}, ..., g_{max}\}$ des Originalbildes durch eine lineare Transformation

$$G_n = G\frac{g - g_{min}}{g_{max} - g_{min}}$$

auf den gesamten Grauwertbereich $\{0, ..., G\}$ abgebildet. Zum Einsatz kommt dieses Verfahren zur Bildoptimierung vor der Anwendung der meisten Filter (z.B. in

der Kantenerkennung) oder Mustererkennungsverfahren (z.B. bei Anwendung der Haarkaskade).

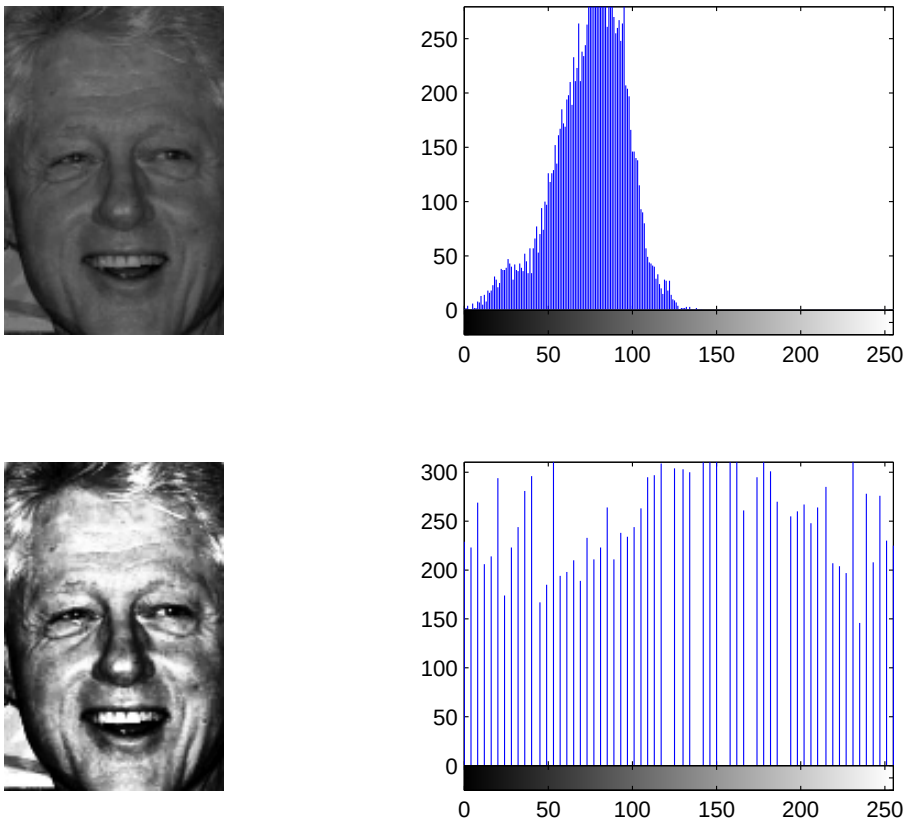



Abbildung A.10: Originalbild mit Histogramm vor und nach Tonwertspreizung, siehe Quelle C.8

Eine Erweiterung des Verfahrens stellt der sogenannte *adaptive Histogrammausgleich* dar, bei dem das Bild zuvor in Bildausschnitte zerlegt wird, von denen jedes ein eigenes Histogramm liefert. Aus diesen Teilhistogrammen wird anschließend das neue Histogramm für das gesamte Bild errechnet und auf das Bild angewendet, was bei sehr dynamischen Szenen eine bessere adaptive Anpassung ermöglicht. Tiefergehende Informationen zum Verfahren und eine Analyse des dadurch geänderten Rauschverhaltens liefert erneut [26].

### A.4.5 Hough-Transformation

Die *Hough-Transformation* führt ein Verfahren ein, um mathematisch beschreibbare Formen[12] schnell in binären Bildern zu finden. Sie eignet sich daher sehr gut, um gefundene Kanten aus Bildern zu extrahieren.

[12] bspw. Geraden und Kreise

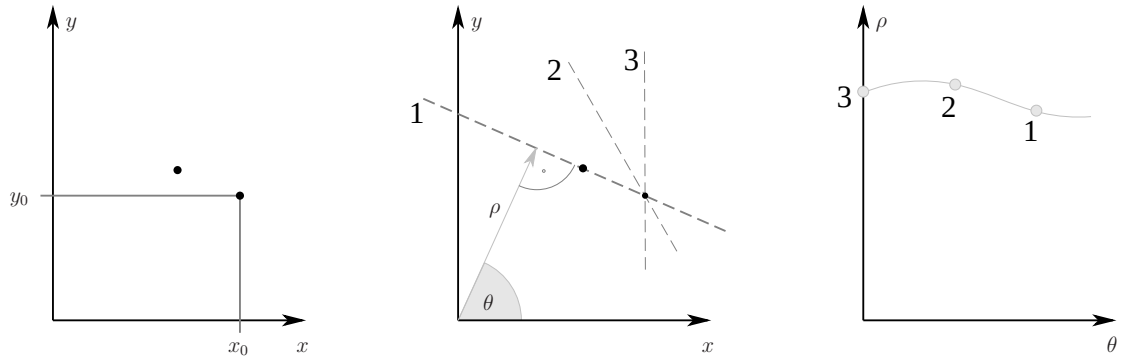



Abbildung A.11: Veranschaulichung der Hough-Transformation

Die Grundlage der Hough-Transformation liefert die Annahme, dass prinzipiell jeder Punkt eines binäres Bildes Teil von beliebig vielen Linien sein kann. Der Hough-Algorithmus bestimmt nun für jeden Punkt die Gerade, deren Normale durch den Koordinatenursprung verläuft, über die Normalform

$$\vec{r} \cdot \vec{n} - c = 0$$

womit unter Verwendung des Skalarprodukts direkt folgt

$$\vec{r} \cdot \vec{n} = \rho \tag{A.5}$$

$$\vec{r} \cdot \vec{n} = |\vec{r}||\vec{n}| \cos\theta \tag{A.6}$$

$$\cos\theta = \frac{\rho}{|\vec{r}||\vec{n}|} \tag{A.7}$$

Anschließend werden der Schnittwinkel mit der $x$-Achse $\theta$ und Abstand zum Koordinatenursprung $\rho$ als Punkt in der sogenannten $\theta$-$\rho$-Ebene (*Parameterraum*) eingetragen. Liegen nun Punkte der $x$-$y$-Ebene auf einer Geraden, so beschreiben sie in der $\theta$-$\rho$-Ebene den gleichen Punkt und werden summiert. Somit stellen sich im Parameterraum Auftrittshäufungen in den Punkten $g_i(\theta_i, \rho_i)$ ein. Über einen Schwellwert wird entschieden, welche dieser „Maxima“ als Geraden übernommen werden. Die gefundenen Geraden ergeben sich dann zu

$$\rho_i = x \cos\theta_i + y \sin\theta_i$$

Dieses Verfahren gilt als das Standardverfahren zur Kantenerkennung und ist in gängigen Bibliotheken (OpenCV) und Programmen implementiert, wobei hier vielfach bereits Optimierungen des Verfahrens eingesetzt werden.

### A.4.6 Kantenerkennung

Eine der wichtigsten Aufgaben der Bildverarbeitung stellt das Finden von Merkmalen im erfassten Bild dar. Grundlegend ist hier zuerst das Finden von *Kanten* im Bild. Modelliert wird eine Kante im Bild als eine Änderung des Grauwerts zweier benachbarter Pixel. Die meisten Kantenerkennungsverfahren verwenden daher die Ableitung zum Finden solcher Kanten (siehe Abbildung A.12).

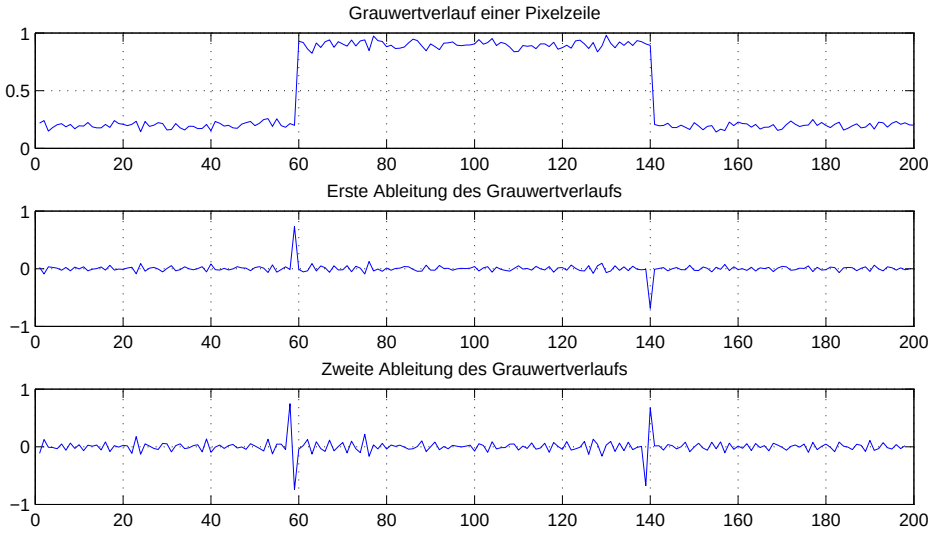



Abbildung A.12: Eindimensionaler Grauwertverlauf (verrauscht) und Ableitungen

Da wir nun aber, wie in A.4.1 eingeführt, ein Bild als diskrete Funktionen zweier Variablen auffassen möchten, müssen wir auch auf diskrete Nachbarschaftsoperationen zurückgreifen und verwenden die bereits eingeführte Faltung.

Eine erste grundlegende Arbeit zu diesem Thema stammt von Sobel und Feldman [58], in der der sogenannten *Sobel-Operator* eingeführt wird. Dieser beschreibt eine Konvolution mit den Sobel-Operatoren $S_x$ und $S_y$, in dessen Folge aus einem Bild $G$ das Gradientenbild $G'$ entsteht

$$\begin{aligned} G'_x &= S_x * G \\ G'_y &= S_y * G \end{aligned}$$

Die Sobel-Operatoren sind hier definiert als Gradient in $x$- und $y$-Richtung

$$S_x = \frac{1}{12}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad S_y = \frac{1}{12}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

In seiner Dissertation *Optimale Operatoren in der Digitalen Bildverarbeitung* konnte Scharr [52] zeigen, dass diese Parameter nicht optimal sind und führt stattdessen die optimalen Koeffizienten

$$S_x = \frac{1}{32}\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \qquad S_y = \frac{1}{32}\begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

ein. Da diese Verbesserung keine Auswirkung auf die Rechenkomplexität hat, verwenden gängige Bildverarbeitungsbibliotheken (z.B. OpenCV) diese Koeffizienten.

Eine Erweiterung des Sobel-Verfahrens stellt die Verwendung des *Laplace-Operators* dar. Dieser ist definiert als

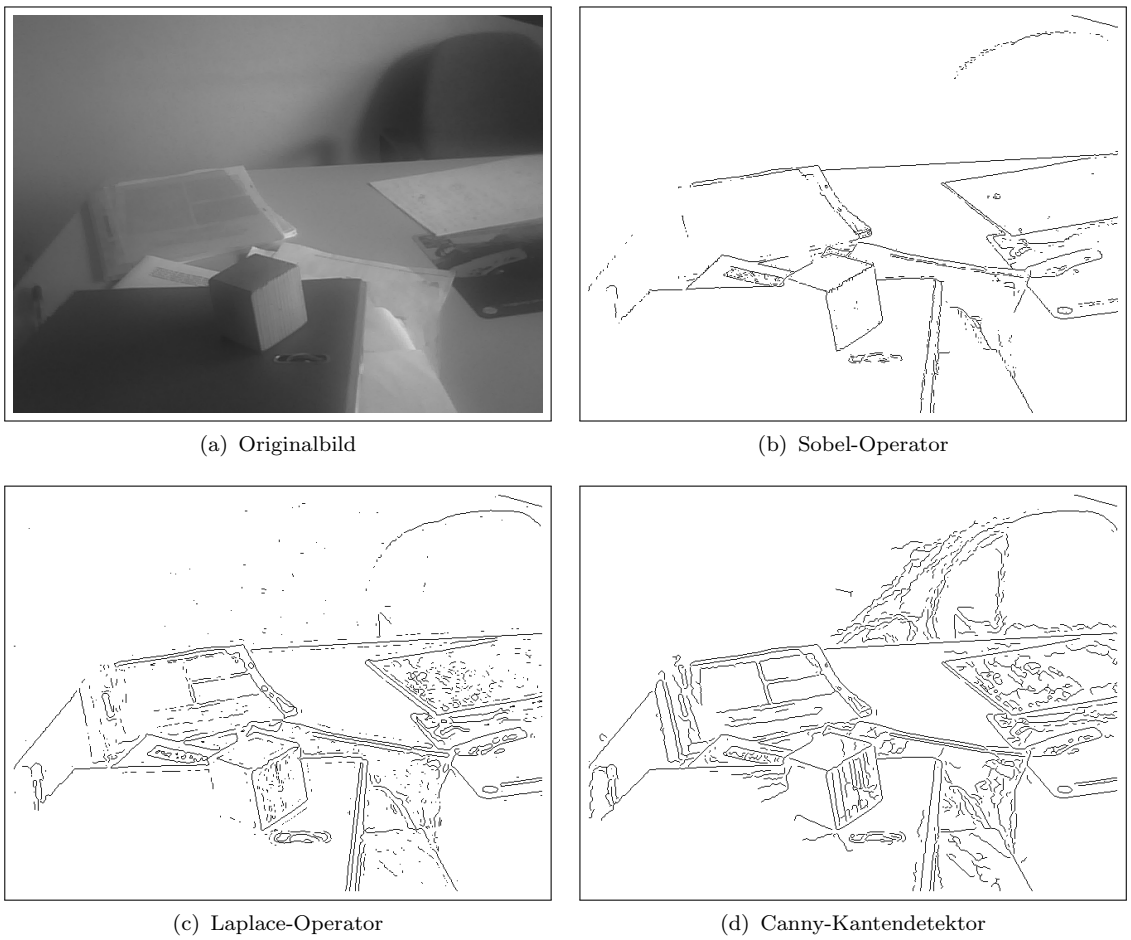
$$\Delta f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad ,$$

stellt also die Verwendung der zweiten Ableitung dar. Wie in Abbildung A.4.1 ersichtlich ist, liegen Kanten genau dann vor, wenn $\Delta f(x,y) = 0$ ist. Allerdings führt das Rauschen des Eingangssignals zu vielen false-positives, was allerdings unter Einbeziehung der ersten (Sobel) Ableitung als Filter gelöst werden kann.

Die bisher beschriebenen Filter heben speziell einzelne Pixelkanten entlang der $x$- und $y$-Richtung hervor. Eine weitere Anforderung an Kanten ist jedoch neben oben genanntem Gradientenverhalten auch die Eigenschaft, sich aus zusammenhängenden Punkten zusammenzusetzen.

Aus diesem Grund stellte Canny in *A computational approach to edge detection* [13] einen weitergehenden Ansatz vor, der die ersten Ableitungen in $x$- und $y$-Richtung in insgesamt vier Richtungsableitungen kombinierte. Punkte, die lokale Maxima in diesen Richtungsableitungen sind, stellen kantenverdächtige Punkte dar. Weiterhin gilt für den *Canny-Kantendetektor*, dass Punkte nur dann Teil einer Kante sind, wenn sie innerhalb einer durch einen oberen und unteren definierten Schwellwerthysterese liegen. Ist der Gradient eines Pixels größer als der obere Schwellwert, so ist er Teil einer Kante. Liegt der Gradient unterhalb des unteren Schwellwerts, fällt er aus der Betrachtung heraus. Hat der Gradient einen Wert innerhalb der Schwellwerte, so kommt er nur dann als Teil einer Kontur in Frage, wenn er in seiner Nachbarschaft

einen Kantenpunkt oberhalb des oberen Schwellwerts hat (also sicher auf einer Kante liegt). Dies verbessert die erkannten Kanten deutlich (vgl. Abbildung A.13). Canny empfiehlt ein Schwellwertverhältnis zwischen 2:1 und 3:1.



(a) Originalbild

(b) Sobel-Operator

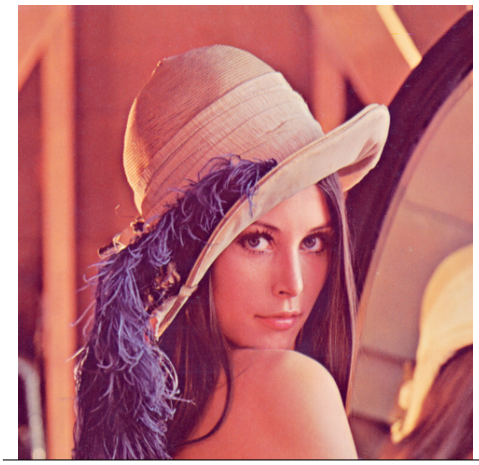
(c) Laplace-Operator

(d) Canny-Kantendetektor

Abbildung A.13: Gegenüberstellung verschiedener Kantenerkennungsverfahren; wenige Kanten in Sobel, „Rauschen“ in Laplace

# B Übersicht über Beispielbilder

Alle verwendeten Bilder sind im Unterverzeichnis `data/` zu finden. Folgende Bilder werden in dieser Arbeit als Referenzen verwendet:



| | |
|---|---|
| **Dateinamen** | `lena.png` |
| **Titel** | *Lena* |
| **Rerenziert als** | lena |
| **Format** | `[FORMAT:lena.png]` |
| **Kommentar** | Frontalaufnahme einer Frau; Größe des Gesichts 170x170px, Gesicht teilweise verdeckt |

---

| | |
|---|---|
| **Dateinamen** | `ss-090402-g20-01ss.jpg` |
| **Titel** | *Aufnahme vom G20 Gipfel 2009* |
| **Rerenziert als** | g20-09 |
| **Format** | `[FORMAT:ss-090402-g20-01ss.jpg]` |
| **Kommentar** | Zehn Personen, vier Frontal-, drei Halbprofil-, drei Profilaufnahmen; verschiedene Hautfarben, nur Männer; Gesichter zwischen 65x65px und 75x75px |

---

| | |
|---|---|
| **Dateinamen** | `NMUN-08-1230025756.jpg` |
| **Titel** | *NMUN Gruppe der TU Chemnitz von 2008* |
| **Rerenziert als** | nmun-08 |
| **Format** | `[FORMAT:NMUN-08-1230025756.jpg]` |
| **Kommentar** | Zwölf Personen, sieben Männer, fünf Frauen; drei Brillenträger; Gesichter teilweise stark gegen Bildebene verdreht |

| | |
|---|---|
| **Dateinamen** | `train-angie-1.png`<br>`train-angie-2.png`<br>`train-angie-3.png`<br>`train-angie-4.png`<br>`train-angie-5.png`<br>`train-angie-6.png`<br>`train-angie-7.png`<br>`train-angie-8.png`<br>`train-angie-9.png` |
| **Titel** | *Trainingsbilder: Angela Merkel* |
| **Rerenziert als** | eigenface-angie |
| **Format** | `[FORMAT:train-angie-1.png]` |

| | |
|---|---|
| **Dateinamen** | `train-ban-1.png train-ban-2.png train-ban-3.png train-ban-4.png train-ban-5.png train-ban-6.png train-ban-7.png train-ban-8.png train-ban-9.png` |
| **Titel** | *Trainingsbilder: Ban Ki Moon* |
| **Rerenziert als** | eigenface-ban |
| **Format** | `[FORMAT:train-ban-1.png]` |

| | |
|---|---|
| **Dateinamen** | `train-bill-1.png train-bill-2.png train-bill-3.png train-bill-4.png train-bill-5.png train-bill-6.png train-bill-7.png train-bill-8.png train-bill-9.png` |
| **Titel** | *Trainingsbilder: Bill Clinton* |
| **Rerenziert als** | eigenface-bill |
| **Format** | `[FORMAT:train-bill-1.png]` |

| | |
|---|---|
| **Dateinamen** | `test-1.png test-2.png test-3.png test-4.png test-5.png test-6.png` |
| **Titel** | *Testbilder der Eigengesichter* |
| **Rerenziert als** | eigenface-test |
| **Format** | `[FORMAT:test-1.png]` |

# C Quellcodeauszüge

## C.1 Realisierung Haarklassifikator in C

```
1  // face detection using OpenCVs haar-cascade
2  //
3  #include "cv.h"
4  #include "cxcore.h"
5  #include "highgui.h"
6  int main(int argc, char* argv[])
7  {
8          clocks[0] = clock();
9          IplImage* img = 0;
10         char* cascadeToUse = "haarcascade_frontalface_alt_tree.xml";
11         CvMemStorage* storage;
12         CvHaarClassifierCascade* cascade;
13         double scaleFactor=1.1;
14         int minNeighbors=2;
15         int flags = 0;
16         int minFaceSize = 0;
17         bool runInteractively = 1;
18         //check arguments
19         //load image
20         img = cvLoadImage(argv[1]);
21         if (!img)
22         {
23                 printf("Could not load image '%s '.",argv[1]);
24                 return 1;
25         }
26         //prepare memory storage
27         //and load the cascade
28         storage = cvCreateMemStorage(0);
29         cascade = (CvHaarClassifierCascade*)cvLoad(cascadeToUse, 0, 0,
               0);
30         //run the detector
31         CvSeq* facesFound = cvHaarDetectObjects(img, cascade, storage,
               scaleFactor, minNeighbors, flags, cvSize(minFaceSize,
               minFaceSize));
32         //show found faces
```

```
33          for (int i = 0; i < facesFound->total; i++)
34          {
35                  //read coordinates
36                  CvRect* r = (CvRect*)cvGetSeqElem(facesFound, i);
37                  //draw rectangle
38                  cvRectangle(img, cvPoint(r->x,r->y), cvPoint(r->x+r->
                        width,r->y+r->height), CV_RGB(255,255,0) );
39          }
40          //output in image
41                  cvNamedWindow("Result");
42                  cvShowImage("Result", img);
43                  cvWaitKey();
44                  cvDestroyAllWindows();
45          //clean up
46          cvReleaseImage(&img);
47          cvReleaseMemStorage(&storage);
48          //ready
49          return 0;
50  }
```

Listing C.1: S:/c/facedetection/facedetection/main.cpp - Quellcode (Auszug): Realisierung Haarklassifikator in C

## C.2 Realisierung Haarklassifikator in IronPython

```
1 from EmguInit import *
2 detector = HaarCascade(cascade)
3 image = Image[Structure.Bgr, Byte](file)
4 im2use = image.Convert[Structure.Gray, Byte]()
5 objectsDetected = im2use.DetectHaarCascade(detector, scalingFactor,
      minNei, flags, Size(minSize, minSize))[0]
6 for obj in objectsDetected:
7         image.Draw(obj.rect, Structure.Bgr(0,255,255), 1)
```

Listing C.2: s:/ironpython/HaarTest.ipy - Quellcode (Auszug): Realisierung Haarklassifikator in IronPython

## C.3 Realisierung ReadImageDatabase in MATLAB

```
1 % [T, C, r, c] = READIMAGEDATABASE (filename, prefix, maxFromClass)
2 %
3 % filename       Image Database Description
4 % prefix         Pathprefix to use for the images to load.
5 % maxFromClass   number of images per class
6 %
7 % T              image vector
8 % C              classnumbers
9 % r              rows
```

```
10 % c              cols
11 %
12 % Reads images given in a structured textfile into memory
13 % and returns a vector containing the images and an vector containing
14 % the classnumbers.
15 function [T, C, r, c] = ReadImageDataBase(filename, prefix,
       maxFromClass)
16     %read the database
17     fid = fopen(filename);
18     t = textscan(fid, '%d %s','CommentStyle','%');
19     fclose(fid);
20     %we have to rebuild the read Classes later as they must correspond
21     %to the loaded images
22     C = [];
23     C_ = t{1};
24     %read images incorporating the prefix
25     im = strcat(prefix, '/', t{2});
26     T = [];
27     for i = 1:size(C_)
28         I = imread(im{i});
29         % reshape into vector
30         [ r, c ] = size(I);
31         t = reshape( double(I), r*c, 1);
32         T = [T t]; %#ok<AGROW>
33         C = [C, C_(i)]; %#ok<AGROW>
34     end
35 end
```

Listing C.3: s:/matlab/eigenfaces/ReadImageDataBase.m - Quellcode (Auszug): Realisierung ReadImageDatabase in MATLAB

## C.4 Realisierung EigenfaceCore in MATLAB

```
1  % [m] = EIGENFACECORE(T)
2  %
3  % T              training set allready preloaded and preprocessed
4  %                [nPixels noImages]
5  % nEigenfaces    top number of eigenfaces to use
6  %
7  % m                  mean image from loaded images [nPixels 1]
8  % A                  centered images (image - mean image)
9  %                    [nPixels noImages]
10 % Eigenfaces         resulting eigenfaces [nPixels noImages-1]
11 % ProjectedImages    projected testimages into face plane
12 %
13 function [m, A, Eigenfaces, ProjectedImages] = EigenfaceCore(T,
       nEigenfaces)
```

```
14      if ~isequal(nargin,2)
15          nEigenfaces = -1;
16      end
17      % preload variables
18      n = size(T, 2);  % number of faces overall
19      % calculate the meanimage over all images in the set
20      m = mean ( T, 2);
21      % center all images
22      A = T - repmat(m, 1, n); %repeat matrix m 1*n times
23      % eigenface approach
24      L = A' * A;
25      [V D] = eig(L);
26      %resort the eigenvalues and eigenvectors
27      [V D] = ResortEigenvaluesEigenvectors(V, D);
28      % determine eigenvectors
29      L_v = [];
30      for i = 1:size(V,2)
31          if isequal(i, nEigenfaces+1)
32              break
33          end
34          %skip to small eigenvalues
35          if (  D(i) > 1  )
36              L_v = [L_v V(:,i)];  %#ok<AGROW>
37          end
38      end
39      % calculate eigenfaces
40      Eigenfaces = A * L_v ;
41      ProjectedImages = Eigenfaces' * A;
42  end
```

Listing C.4: s:/matlab/eigenfaces/EigenfaceCore.m - Quellcode (Auszug): Realisierung EigenfaceCore in MATLAB

## C.5 Realisierung Eigenface Erkennung in MATLAB

```
1  % [n, dist] = EIGENRECOGNITION(I, m, A, Eigenfaces)
2  %
3  % I                 test image to use (must be grayscale image!)
4  % m                 mean image of training set from EIGENFACECORE
5  % A                 centered images from EIGENFACECORE
6  % Eigenfaces        eigenfaces from EIGENFACECORE
7  % ProjectedImages   eigenfaces from EIGENFACECORE
8  %
9  % n             index of the image closest to I
10 % dist          array containing all distances
11 %
12 % Determines
```

```
13 function [n, dist] = EigenRecognition(I, m, A, Eigenfaces,
       ProjectedImages)
14     % vectorize the image and center it
15     [r c] = size(I);
16     I = reshape(I, r * c, 1);
17     I_d = double(I) - m;
18     I_p = Eigenfaces' * I_d;
19     % now calculate the distance to all projected images
20     dist = [];
21     for i = 1:size(ProjectedImages,2)
22         d = ( norm(I_p - ProjectedImages(:,i)) );
23         dist = [ dist d ]; %#ok<AGROW>
24     end
25     % determine the minimum
26     [~,n] = min(dist);
27 end
```

Listing C.5: s:/matlab/eigenfaces//EigenRecognition.m - Quellcode (Auszug): Realisierung Eigenface Erkennung in MATLAB

## C.6 Realisierung FisherfaceCore in MATLAB

```
1  % [m, A, Eigenfaces, V_EF, Fisherfaces, V_FF, ProjectedImagesEF,
2  %                 ProjectedImagesFF] = FISHERFACECORE(T, C, nEigenfaces)
3  %
4  % T                       training set allready preloaded and
5  %                         preprocessed [nPixels noImages]
6  % C                       classnumber corresponding to the trainingset
7  %                         [1 noImages]
8  % nEigenfaces             number of eigenfaces to use
9  %
10 % m                       mean image from loaded images [nPixels 1]
11 % A                       centered images (image - mean image)
12 %                         [nPixels noImages]
13 % Eigenfaces              resulting eigenfaces [nPixels noImages-1]
14 % V_EF                    eigenface vectors
15 % Fisherfaces             resulting fisherfaces
16 % V_FF                    fisherface vectors
17 % ProjectedImagesEF       projected testimages into eigenface plane
18 % ProjectedImagesFF       projected eigenface projection into the
19 %                         fisherface
20 %                         plane
21 %
22 function [m, A, Eigenfaces, V_EF, Fisherfaces, V_FF, ProjectedImagesEF,
       ProjectedImagesFF] = FisherfaceCore(T, C, nEigenfaces)
23     if ~isequal(nargin,3)
24         nEigenfaces = -1;
```

```
25      end
26      %determine the number of classes we have
27      nClasses = size(unique(C), 2);
28      nImages = size(C, 2);
29      % preload variables
30      n = size(T, 2);  % number of faces overall
31      ProjectedImagesEF = [];
32      % calculate the meanimage over all images in the set
33      m = mean(T, 2);
34      % center all images
35      A = T - repmat(m, 1, n); %repeat matrix m 1*n times
36      % eigenface approach
37      L = A' * A;
38      [V D] = eig(L);
39      % resort descending
40      [V D] = ResortEigenvaluesEigenvectors(V, D);
41      % determine eigenvectors
42      V_EF = [];
43      for i = 1:size(V,2)
44          % use only the top nEigenfaces
45          if isequal(i, nEigenfaces+1)
46              break
47          end
48          %skip to small eigenvalues
49          if (  D(i) > 1  )
50              V_EF = [V_EF V(:,i)];  %#ok<AGROW>
51          end
52      end
53      % calculate eigenfaces
54      Eigenfaces = A * V_EF;
55      % use eigenfaces to create projections of images into face space
56      ProjectedImagesEF = Eigenfaces' * A;
57      %mean value in the eigenspace
58      mPCA = mean(ProjectedImagesEF, 2);
59      n = size(ProjectedImagesEF, 1);
60      mPI = zeros(n, nClasses);
61      %within scatter
62      Sw = zeros(n, n);
63      %between scatter
64      Sb = zeros(n, n);
65      %calculate the scatters over all classes
66      for i = 1:nClasses
67          %determine the images of a class
68          PI = ExtractClassImages(ProjectedImagesEF, C, i);
69          %determine the mean projection for this class
70          mPI(:, i) = mean(PI, 2)';
```

```
71          %sum scatter
72          S = zeros(n, n);
73          for j = 1:size(PI,2)
74              %center ProjectedImage
75              D = PI(:,j) - mPI(:, i);
76              % add to scatter
77              S = S + D*D';
78          end
79          %add to within scatter matrix
80          Sw = Sw + S;
81          %calculate between scatter matrix
82          mb = mPI(:,i) - mPCA;
83          Sb = Sb + mb*mb';
84      end
85      %calculate Fishers discriminant basis using all classes
86      % W = argmax\limits_{W}|(W' S_B W)/(W^T S_W W)|
87      [V_FF, v_ff] = eig(Sb, Sw);
88      [V_FF, v_ff] = ResortEigenvaluesEigenvectors(V_FF, v_ff); %#ok<
            NASGU>
89      %TODO: we should sort them descending!
90      Fisherfaces = A * V_EF * V_FF;
91      %now project the resulting eigenface projections again
92      ProjectedImagesFF = V_FF' * ProjectedImagesEF;
93  end
```

Listing C.6: s:/matlab/fisherfaces/FisherfaceCore.m - Quellcode (Auszug): Realisierung FisherfaceCore in MATLAB

## C.7 Bildpyramiden in MATLAB

```
1  %  reconstructed image GOO %%
2  function pyramid(s,g)
3      %%
4      G1 = PixDown(G0,g);
5      L0 = G0 - PixUp(G1,g);
6      G00 = L0 + PixUp(G1,g);
7      imshow(G0);
8      imshow(G00);
9      imshow(G1);
10     imshow(L0);
11     imhist(G0);
12     imhist(G1);
13     imhist(G00);
14 end
15
16 function J = PixUp(I,g)
17     [h w] = size(I);
```

```
18      J = imresize(I,[h*2 w*2],'nearest');
19      J = uint8(conv2(double(J),fspecial('gaussian',g,g),'same'));
20  end
21
22  function J = PixDown(I,g)
23      [h w] = size(I);
24      I = conv2(double(I),fspecial('gaussian',g,g),'same');
25      J = imresize(I,[ceil(h/2) ceil(w/2)],'nearest');
26      J = uint8(J);
27  end
```

Listing C.7: s:/matlab/image-pyramid/pyramid.m - Quellcode (Auszug): Bildpyramiden in MATLAB

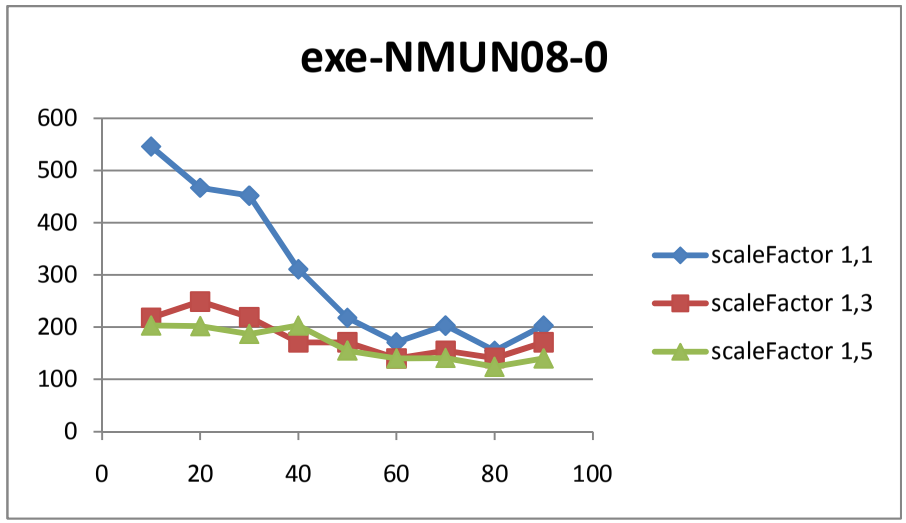
## C.8 Histogrammausgleich in MATLAB

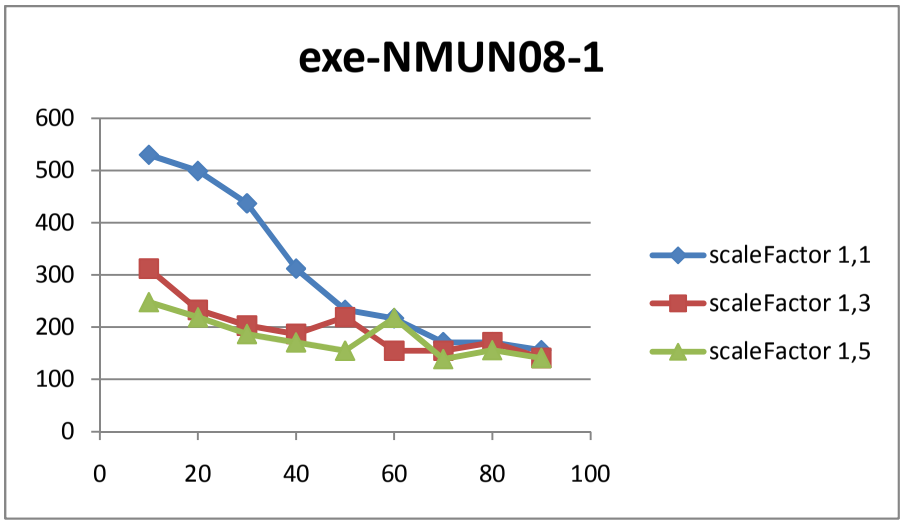
```
1  function histeqsample
2          I = imread('L:\data\train-bill-5.png');
3          imshow(I)
4          imhist(I)
5          Ih = histeq(I);
6          imshow(Ih)
7          imhist(Ih)
8  end
```

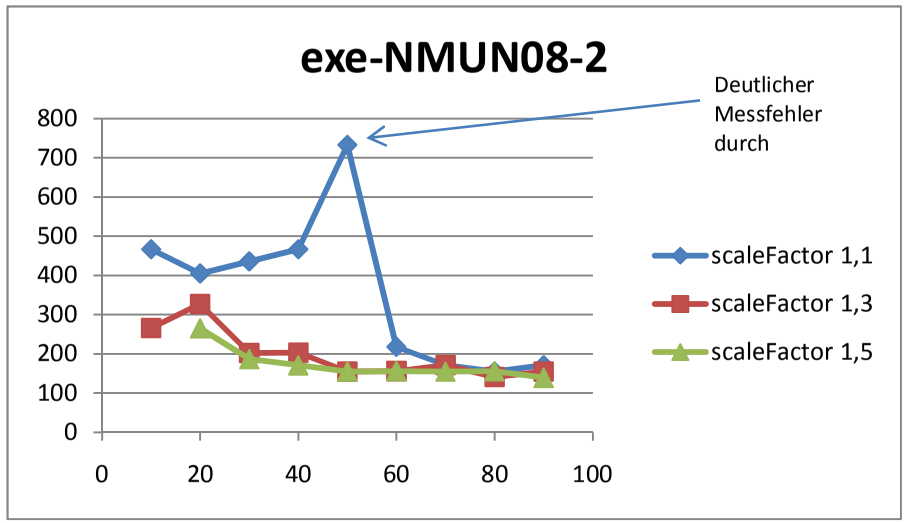
Listing C.8: s:/matlab/hist-eq/histeqsample.m - Quellcode (Auszug): Histogrammausgleich in MATLAB

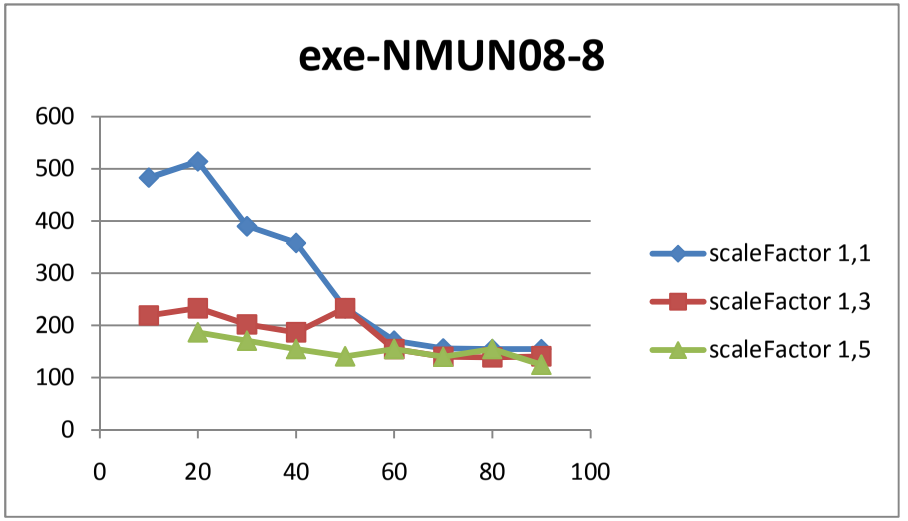
# D Messergebnisse zur Haar-Kaskade

**Abhängigkeit Laufzeit von minSize**

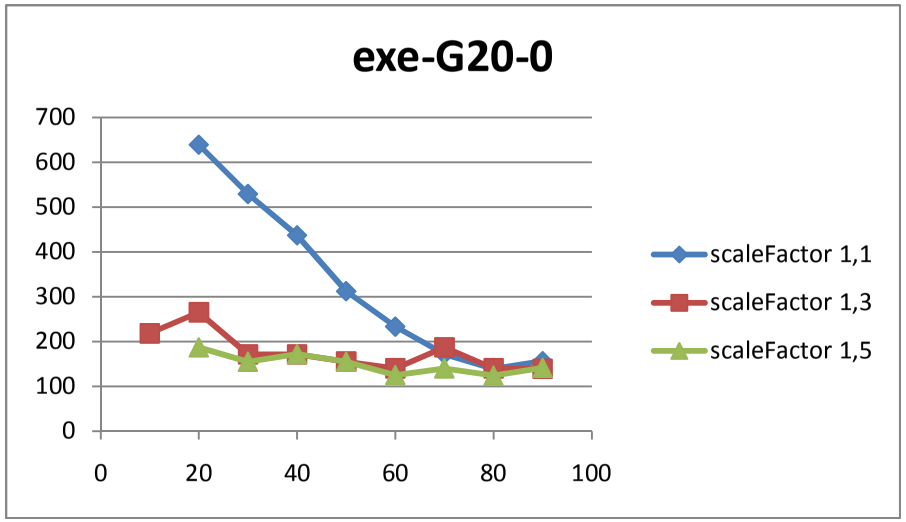


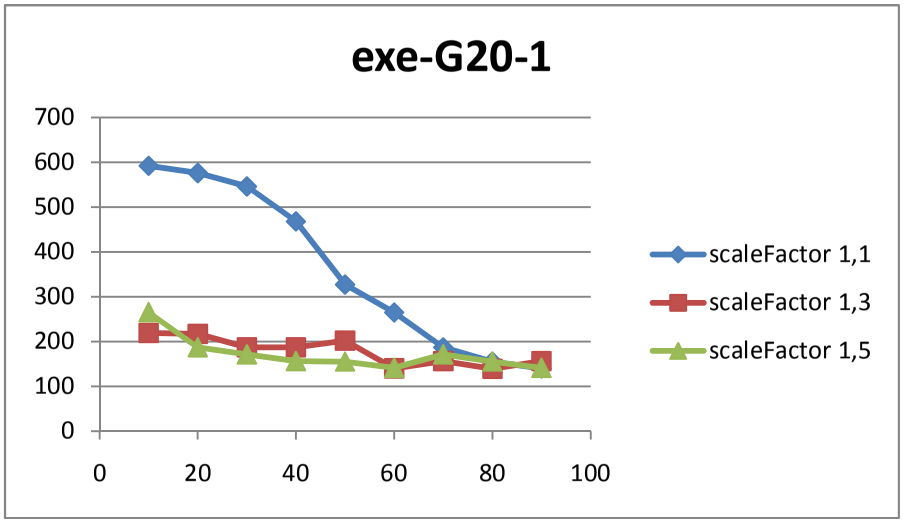



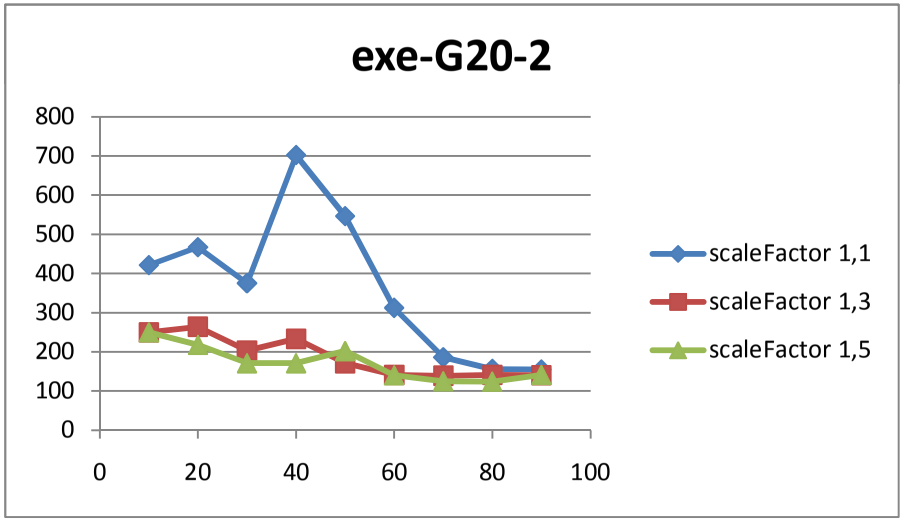



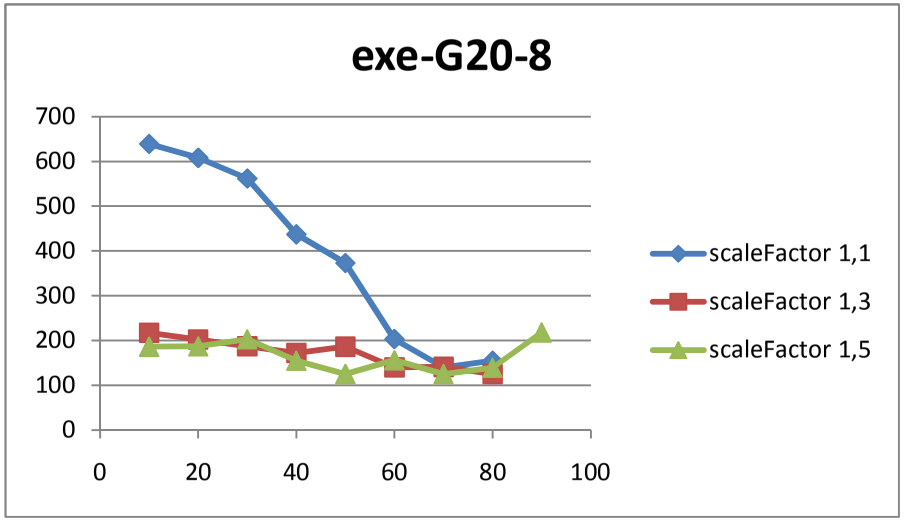



**Abhängigkeit Laufzeit von minSize**

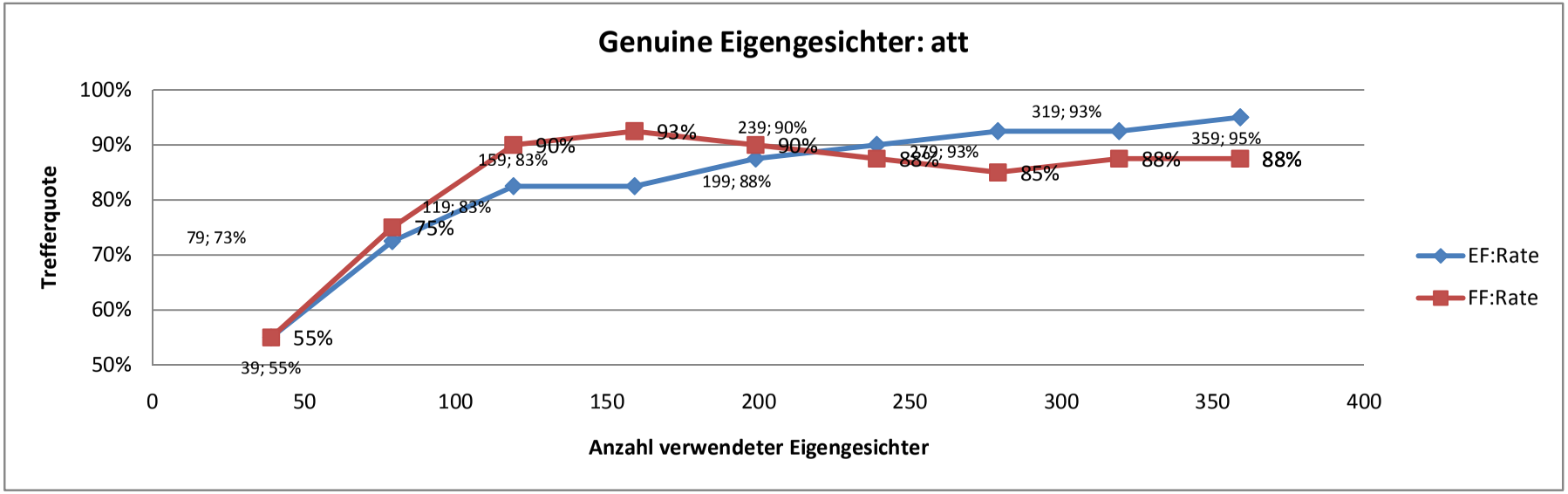


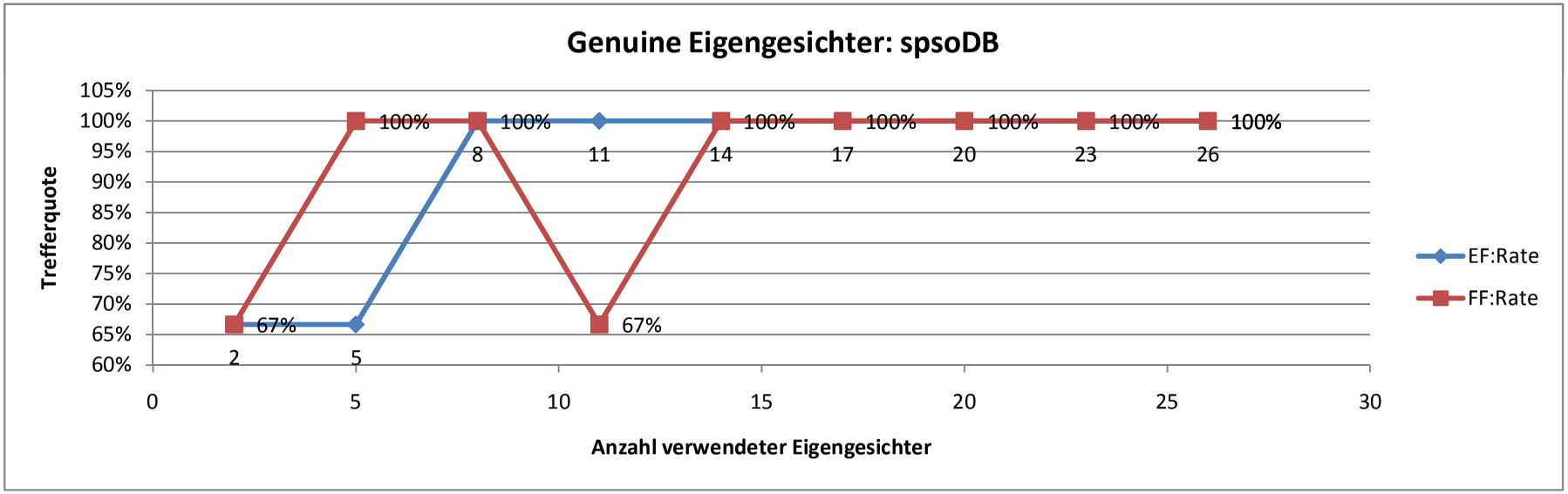



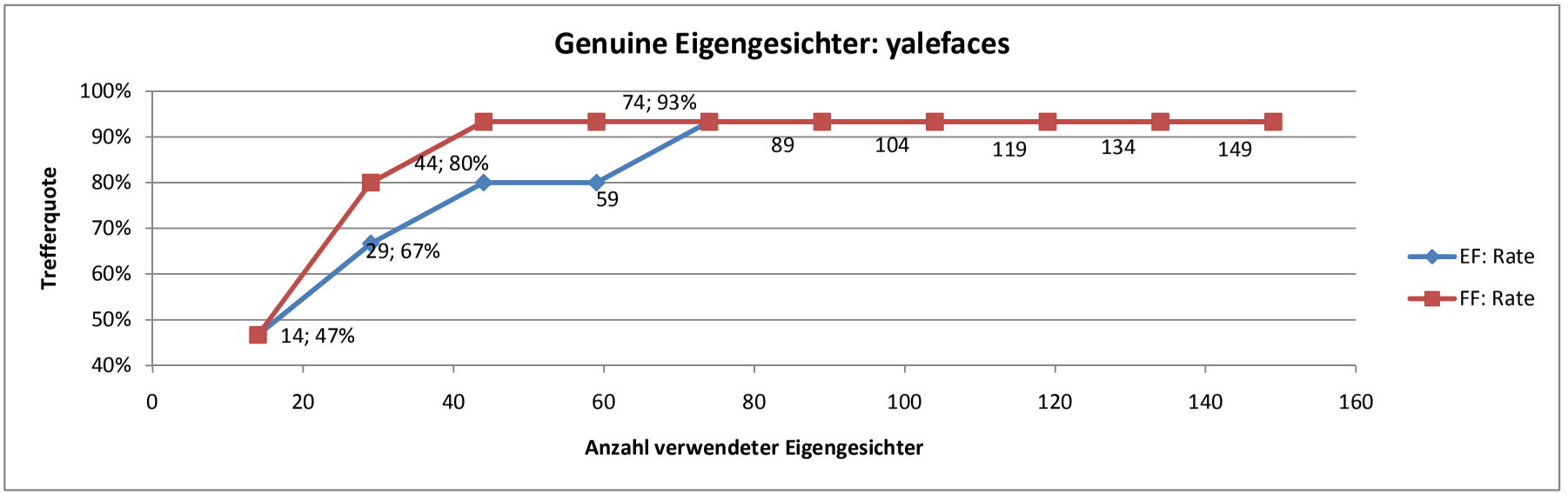



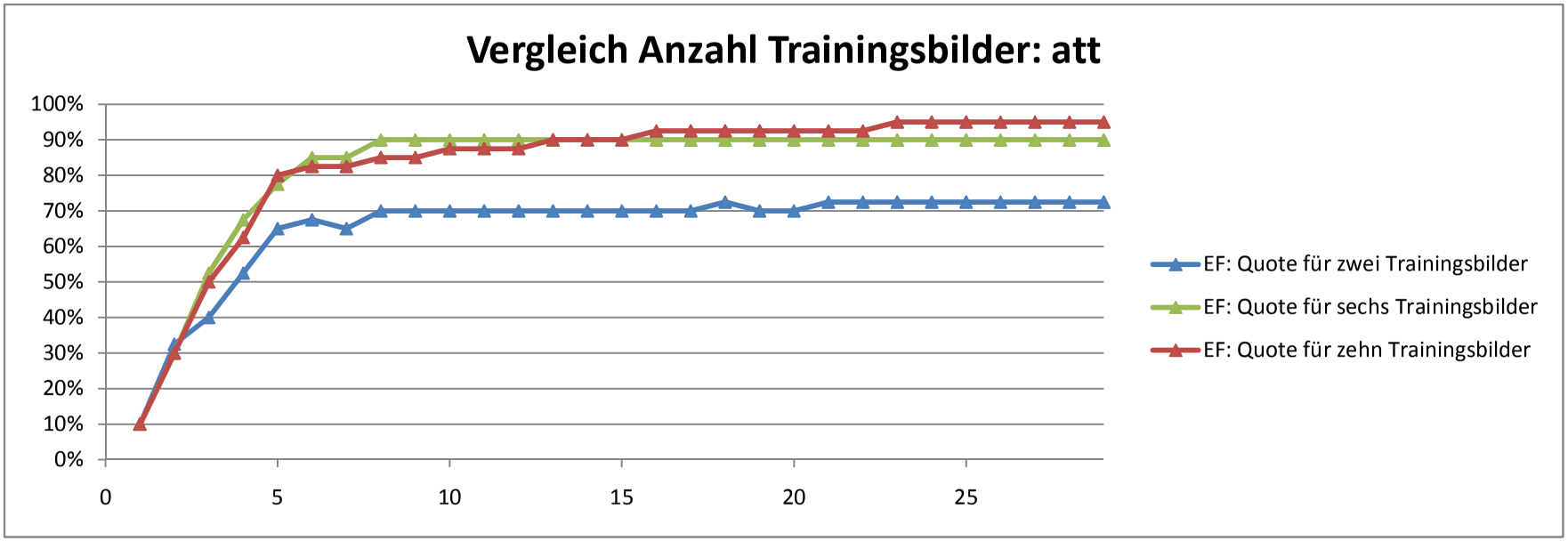



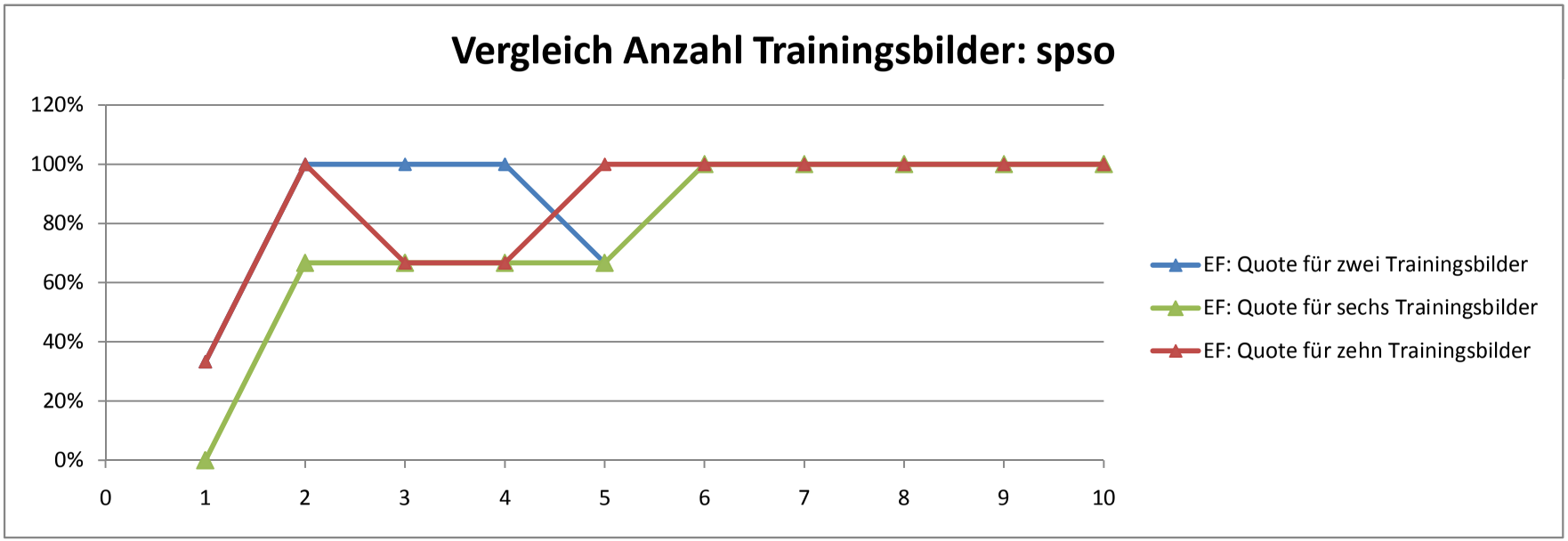
# E Messergebnisse zu Eigenfaces und Fisherfaces

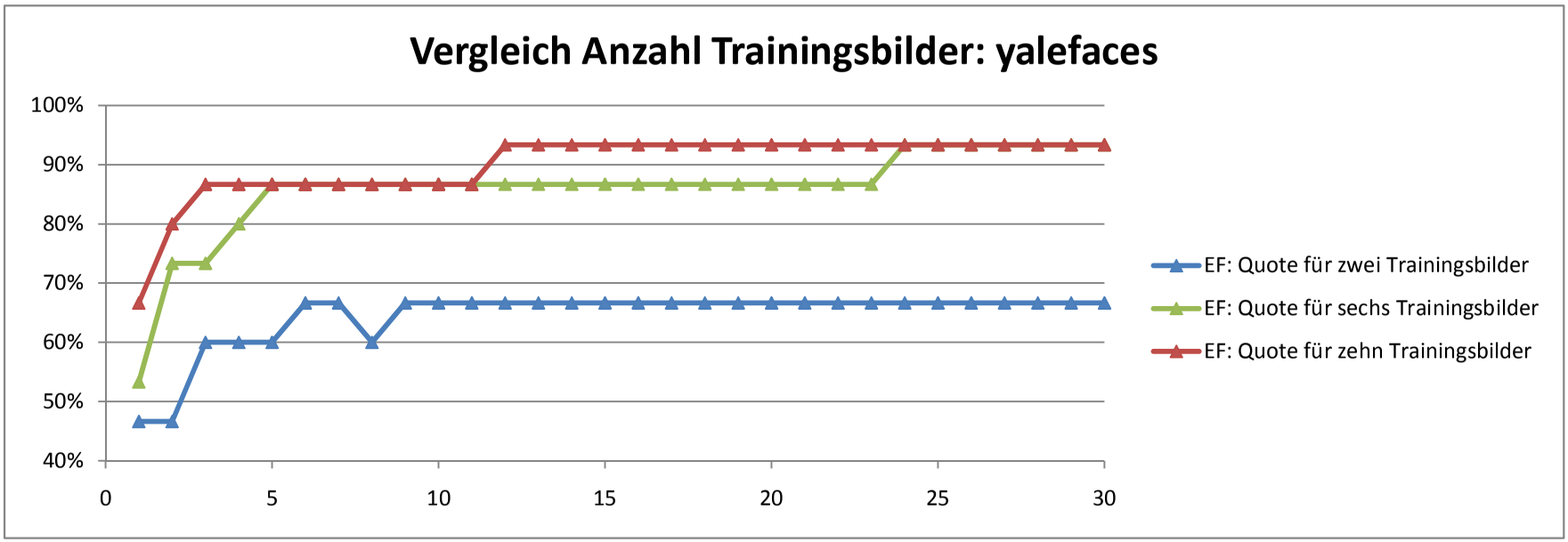

Genuine Eigengesichter: att
Trefferquote
100%
90%
80%
70%
60%
50%
0
50
100
150
200
250
300
350
400
Anzahl verwendeter Eigengesichter
79; 73%
39; 55%
55%
119; 83%
75%
159; 83%
90%
93%
199; 88%
239; 90%
90%
279; 93%
88%
319; 93%
85%
359; 95%
88%
88%
EF:Rate
FF:Rate

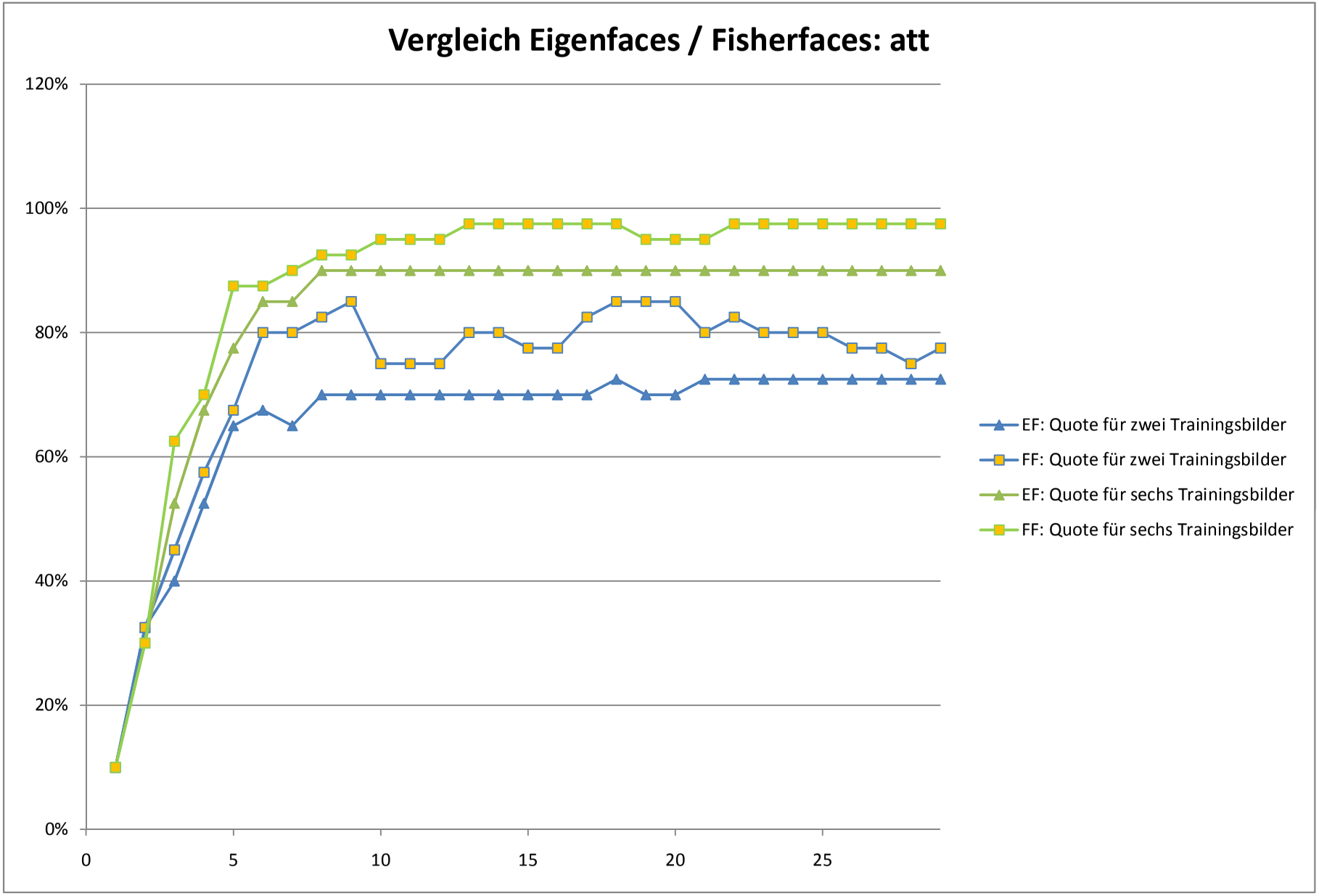

Genuine Eigengesichter: spsoDB
Trefferquote
105%
100%
95%
90%
85%
80%
75%
70%
65%
60%
0
5
10
15
20
25
30
Anzahl verwendeter Eigengesichter
67%
2
100%
5
100%
8
100%
11
100%
14
100%
17
100%
20
100%
23
100%
26
67%
EF:Rate
FF:Rate

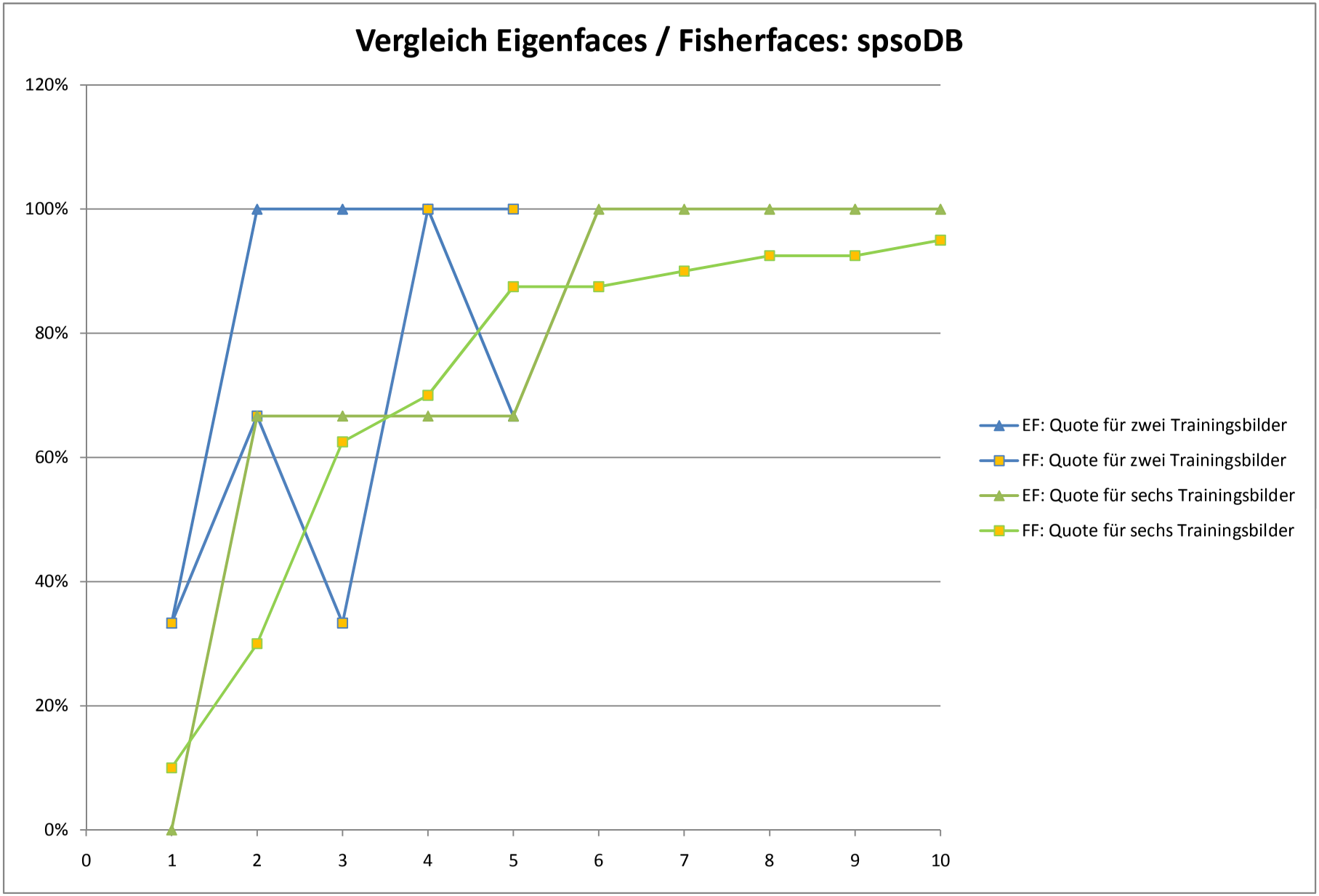

Genuine Eigengesichter: yalefaces
Trefferquote
100%
90%
80%
70%
60%
50%
40%
0
20
40
60
80
100
120
140
160
Anzahl verwendeter Eigengesichter
14; 47%
29; 67%
44; 80%
59
74; 93%
89
104
119
134
149
EF: Rate
FF: Rate


Vergleich Anzahl Trainingsbilder: att
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
5
10
15
20
25
EF: Quote für zwei Trainingsbilder
EF: Quote für sechs Trainingsbilder
EF: Quote für zehn Trainingsbilder


Vergleich Anzahl Trainingsbilder: spso
120%
100%
80%
60%
40%
20%
0%
0
1
2
3
4
5
6
7
8
9
10
EF: Quote für zwei Trainingsbilder
EF: Quote für sechs Trainingsbilder
EF: Quote für zehn Trainingsbilder


Vergleich Anzahl Trainingsbilder: yalefaces
100%
90%
80%
70%
60%
50%
40%
0
5
10
15
20
25
30
EF: Quote für zwei Trainingsbilder
EF: Quote für sechs Trainingsbilder
EF: Quote für zehn Trainingsbilder


Vergleich Eigenfaces / Fisherfaces: att
120%
100%
80%
60%
40%
20%
0%
0
5
10
15
20
25
EF: Quote für zwei Trainingsbilder
FF: Quote für zwei Trainingsbilder
EF: Quote für sechs Trainingsbilder
FF: Quote für sechs Trainingsbilder


Vergleich Eigenfaces / Fisherfaces: spsoDB
120%
100%
80%
60%
40%
20%
0%
0
1
2
3
4
5
6
7
8
9
10
EF: Quote für zwei Trainingsbilder
FF: Quote für zwei Trainingsbilder
EF: Quote für sechs Trainingsbilder
FF: Quote für sechs Trainingsbilder

# F Verwendete Software

## F.1 MATLAB

MATLAB in der benötigten Version 7.9.0 (R2009a) ist als Lizenz an der Professur Robotersysteme vorhanden und auf dem Arbeitsplatz im Labor installiert.

## F.2 Visual Studio 2008 Express Edition

Zur Entwicklung der Software, die in dieser Diplomarbeit beschrieben wurde, wurde Microsoft Visual Studio 2008 Professional aus dem für Studenten kostenlosen Microsoft Academic Alliance Programm verwendet. Alle Quellen sind aber auch mit dem frei verfügbaren Microsoft Visual Studio 2008 Express kompilierbar, das unter http://www.microsoft.com/germany/Express/ heruntergeladen werden kann.

## F.3 NLog

*NLog* ist ein hochflexibles Logging-Framework für .NET-Projekte. OPVS verwendet es, um interne Abläufe in Programmteilen ohne graphische Oberfläche besser debugbar zu machen. NLog ist unter http://www.nlog-project.org/ verfügbar. OPVS verwendet Version 1.0 Refresh .

## F.4 GhostDoc for Visual Studio 2008

*GhostDoc* ist ein Plugin für Visual Studio 2008, das zur automatischen Dokumentation von Klassen in C# und VB.NET-Quellcodes verwendet werden kann. Es ist unter http://submain.com/products/ghostdoc.aspx frei verfügbar; Version 2.5 war während der Erstellung dieser Diplomarbeit aktuell.

## F.5 Sandcastle Help File Builder

*Sandcastle Help File Builder* ist eine Software zur automatischen Quellcodedokumentation für Microsoft Visual Studio 2008 XML-Dokumentationen. Zu finden ist die freie Software in Microsofts Codeplex unter [73]. Zum Zeitpunkt der Erstellung dieser Arbeit ist Version 1.7.0.0 aktuell.

## F.6 IronPython

IronPython ist eine managed-code Implementierung der freien objekt-orientierte Programmiersprache Python. OPVS verwendet sie zur Realisierung der internen Programmlogik in OPVS -Projekten. IronPython wird unter der Open Source-Lizenz Microsoft Public License (Ms-PL) entwickelt und ist im Microsoft CodePlex unter `http://ironpython.codeplex.com/` zu finden. Zum Zeitpunkt der Erstellung dieser Arbeit war die Version 2.6 stabil.

## F.7 OpenCV / Emgu

Zur Realisierung der Bilderfassung und Bilderkennung in OPVS fiel die Wahl auf die ursprünglich von Intel entwickelte Open Computer Vision Library (*OpenCV*), die sich als de facto Standard zur Realtime-Bildverarbeitung entwickelt hat. Eine umfangreiche Dokumentation ist unter `http://opencv.willowgarage.com/wiki/` zu finden. Eine sehr gute Einführung in die Arbeit mit der Bibliothek liefert [8]. Diese Arbeit verwendet Version 2.0a vom 01.10.2009. Um die C-Beispiele übersetzen zu können, müssen zuvor die zu linkenden Bibliotheken (libcv200.lib u.a.) erstellt werden.
Für die Realisierung von OPVS wurde die *managed-code* Variante *Emgu CV* verwendet, die unter `http://www.emgu.com/` heruntergeladen werden kann. Zum Zeitpunkt der Entwicklung war die Version 2.0.1.0 stabil, die intern auf OpenCV 2.0a aufsetzt. Emgu ist unter einer dualen Lizenz freigegeben, die eine nicht-kommerzielle Verwendung an einer Universität ohne Lizenzkosten ermöglicht.

## F.8 CMake

CMake ist ein plattformunabhängiges Buildtool, das aus Vorlagen die jeweilige Projektdatei für die vom Endanwender gewünschte Entwicklungsumgebung erstellt. Es wird von OpenCV zum Build verwendet und muss daher vor Erstellen der Bibliothek

installiert werden. Zum Zeitpunkt der Erstellung der Arbeit war Version 2.6-patch-4 für Windows verfügbar.

## F.9 Subversion

Alle während der Erstellung dieser Arbeit erzeugten Dokumente und Quellcodes wurden in einem zentralen Subversion-Repository unter `https://subversor.hrz.tu-chemnitz.de/svn/VisualServo/repo/` abgelegt und können für freigegebene Nutzer mit ihrem URZ-Login ausgecheckt werden. Freigeschaltene Benutzer sind `gzs` und `spso`, wobei mit Abgabe dieser Arbeit die Verwaltungsrechte auf `gzs` übergeben werden.

*Subversion* (auch SVN abgekürzt) ist eine Software zur zentralen Versionsverwaltung für Softwareprojekte. SVN ermöglicht es auf einfache Weise, Änderungen an Dokumenten nachzuverfolgen und beliebige historische Versionen einer Datei wiederherzustellen[1]. Die Speicherung dieser Versionen erfolgt dabei nicht lokal beim Anwender, sondern zentral auf einem Subversion-Server. Jede Änderung auf dem Server[2] wird dabei vom Anwender mit einem Kommentar versehen. Damit eignet sich SVN nicht nur gut zur Dokumentation der Entwicklung einer Software, sondern auch zum zentralen Backup.

Subversion selbst ist eine Software für die Kommandozeile, die mit dem Befehl `svn` aufgerufen wird. Zum komfortablen Zugriff auf Subversion-Repositories steht unter `http://tortoisesvn.tigris.org/` die Software *TortoiseSVN* für Windows zur Verfügung. TortoiseSVN integriert sich nahtlos in den Windows Explorer und stellt alle verfügbaren Kommandos per Kontextmenü zur Verfügung. Als gute graphische Oberfläche für Linux hat sich *kdesvn* herausgestellt, das in allen gängigen Distributionen als Paket verfügbar ist.

Weiterführende Informationen zum Thema Versionsverwaltung mit Subversion können im sehr empfehlenswerten Buch „Versionskontrolle mit Subversion“ [20] gefunden werden, das mittlerweile in 3. Auflage in deutscher Sprache vorliegt.

## F.10 LaTeX

*LaTeX* ist eine Erweiterung des Textsatzsystems TeX von Donald E. Knuth, das vor allem zum Setzen wissenschaftlicher Texte verwendet wird. LaTeX ist für so gut wie

[1] Subversion führt eine eigene „Sprache“ ein und definiert jede Änderung als „Revision“, einen automatisch inkrementierten Zähler.
[2] In Subversion ist hierfür der Begriff `commit` gebräuchlich.

jede Plattform übersetzt worden und somit universell einsetzbar. Diese Diplomarbeit wurde mit MiKTEX 2.7 (http://miktex.org) unter Windows übersetzt, als Editor wurde TEXnicCenter 2.0alpha (http://www.texniccenter.org) eingesetzt. Das Literaturverzeichnis wurde automatisch mit BibTEX erzeugt, welches die Literaturinformationen aus der mit JabRef (http://jabref.sourceforge.net) gepflegten Datenbank verwendet.

## F.11 Enterprise Architect

Enterprise Architect ist eine Softwaremodellierungslösung von Sparks Systems. EA ermöglicht eine einfache modellgetriebene Softwareentwicklung und unterstützt den Benutzer von Anforderungsaufnahme über Use-Case- und Komponenten-Modell bis hin zu Klassenmodellierung und integrierter Quellcodeerstellung. Weiterhin bietet es gute Unterstützung für Reverse-Modelling, bei dem vorhandener Code in Klassenmodelle überführt werden kann und eignet sich somit sehr gut zur Dokumentation der entstandenen Software.
Enterprise Architect greift bei der Darstellung auf die standardisierte *Unified Modelling Language* (*UML*) zurück, durch deren Verwendung das gesamte Softwareprojekt nachvollziehbar dokumentiert wurde. Eine sehr gute deutschsprachige Einführung in UML liefert [41]. Für die Erstellung der Modelle für diese Diplomarbeit wurde EA Version 7.5 eingesetzt. Eine Demoversion kann unter [80] für Windows und Linux heruntergeladen werden.

# G Verwendete virtuelle Maschine

Im Rahmen der Entwicklung der in dieser Diplomarbeit beschriebenen Software wurde eine virtuelle Maschine erstellt, die die gesamte, für die Entwicklung benötigte Software enthält und damit eine einfache Weiterentwicklung ermöglicht. Erstellt wurde die virtuelle Maschine mit *VMware Workstation*. Zum Ausführen der virtuellen Maschine ist auf dem einzusetzenden Rechner die Software *VMware Player* zu installieren, welche frei zum Download für Windows und Linux bereit steht[1]. Mit *VirtualBox* steht eine weitere Virtualisierungslösung für Windows und Linux zur Verfügung[2], die die entstandene virtuelle Festplatte der virtuellen Maschine ebenso booten kann. VirtualBox steht als OpenSource-Software unter der GNU General Public License und kann frei verwendet werden.
Die entstandene virtuelle Maschine ist im Projekt-AFS dieser Diplomarbeit im Unterverzeichnis `vm/` zu finden und sollte vor ihrer Verwendung auf ein lokales Laufwerk kopiert werden.

[1] http://www.vmware.com/de/products/player/
[2] http://www.virtualbox.org/

# Literaturverzeichnis


[1] Adelson, E. H. ; Anderson, C. H. ; Bergen, J. R. ; Burt, P. J. ; Ogden, J. M.: Pyramid Methods in Image Processing. In: *RCA Engineer* 29 (1984), S. 33–41

[2] Adelson, Edward H.: Image Data Compression With The Laplacian Pyramid. In: *In Proceedings of the conference on pattern recognition and image processing*, IEEE Computer Society Press, 1981, S. 218–223

[3] Ahonen, Timo ; Hadid, Abdenour ; Pietikainen, Matti: Face Description with Local Binary Patterns: Application to Face Recognition. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 28 (2006), Nr. 12, S. 2037–2041. http://dx.doi.org/http://dx.doi.org/10.1109/TPAMI.2006.244. – DOI http://dx.doi.org/10.1109/TPAMI.2006.244. – ISSN 0162–8828

[4] Baeten, Johan ; De Schutter, Joris: *Integrated Visual Servoing and Force Control*. Springer, 2004. – ISBN 3–540–40475–9

[5] Bartlett, M. S. ; Movellan, J. R. ; Sejnowski, T. J.: Face recognition by independent component analysis. In: *IEEE Transactions on Neural Networks* 13 (2002), November, Nr. 6, S. 1450–1464. http://dx.doi.org/10.1109/TNN.2002.804287. – DOI 10.1109/TNN.2002.804287

[6] Bay, H. ; Tuytelaars, T. ; Van Gool, L.: Surf: Speeded up robust features. In: *Lecture notes in computer science* 3951 (2006), 404. http://bit.ly/Surf06

[7] Belhumeur, P. N. ; Hespanha, J. P. ; Kriegman, D. J.: Eigenfaces vs. Fisherfaces: recognition using class specific linear projection. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19 (1997), July, Nr. 7, S. 711–720. http://dx.doi.org/10.1109/34.598228. – DOI 10.1109/34.598228

[8] Bradski, Gary ; Kaehler, Adrian: *Learning OpenCV - Computer Vision with the OpenCV Library*. O'Reilly Press, 2008. – ISBN 978–0–596–51613–0. – in Safari Books Online: http://proquest.safaribooksonline.com/9780596516130

[9] Breiman, L. ; Friedman, J. ; Olshen, R. ; Stone, C. ; Wadsworth (Hrsg.): *Classification and Regression Trees*. Wadsworth, 1984

[10] Breiman, Leo: Bagging Predictors / University of California. 1994 (421). – Forschungsbericht. – http://bit.ly/Breiman96

[11] Burg, Klemens ; Haff, Herbert ; Wille, Friedrich: *Mathematik für Ingenieure*. B.G. Teubner Stuttgart, 1989

[12] Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. In: *Data mining and knowledge discovery* 2 (1998), Nr. 2, 121–167. http://bit.ly/Burges98SVM

[13] Canny, J.: A computational approach to edge detection. In: *IEEE Transactions of Pattern Analysis and Machine Intelligence* 8 (1986), 679-714. http://bit.ly/Canny86

[14] Chang-yeon, Jo: Face Detection using LBP features / Standford. 2008 (CS 229). – Final Project Report. – -

[15] Chen, Li-Fen ; Liao, Hong-Yuan M. ; Ko, Ming-Tat ; Lin, Ja-Chen ; Yu, Gwo-Jong: A new LDA-based face recognition system which can solve the small sample size problem. In: *Pattern Recognition* 33 (2000), Nr. 10, 1713 - 1726. http://dx.doi.org/DOI:10.1016/S0031-3203(99)00139-9. – DOI DOI: 10.1016/S0031–3203(99)00139–9. – ISSN 0031–3203

[16] Cootes, T.F. ; Taylor, C. J. ; Cooper, D. H. ; Graham, J.: *Training models of shape from sets of examples*. 1992

[17] Ernesti, Johannes ; Kaiser, Peter: *Python - Das umfassende Handbuch*. Galileo Computing, 2008

[18] Fardi, Dr. B.: *Vorlesungsskript Bildkommunikation*. 2005

[19] Fisher, R. A.: The use of multiple measurements in taxonomic problems. In: *Annals of Eugenics* 7 (1936), Nr. 2, S. 179–188

[20] Fitzpatrick, Brian W. ; Pilato, C. M.: *Versionskontrolle mit Subversion, 3. Auflage*. O'Reilly, 2009. – http://www.oreilly.de/catalog/subvers3ger/

[21] Freund, Yoav ; Schapire, Robert E.: A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting. In: *Journal of Computer and System Sciences* 55 (1997), Nr. 1, 119-139. http://www.cse.ucsd.edu/~yfreund/papers/adaboost.pdf

[22] FRH: *Face recognition homepage - Algorithms*. 2009. – http://www.face-rec.org/algorithms/

[23] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995

[24] Hadid, Abdenour ; Pietikäinen, Matti ; Ahonen, Timo: A Discriminative Feature Space for Detecting and Recognizing Faces. In: *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on* 2 (2004), S. 797–804. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/CVPR.2004.9. – DOI http://doi.ieeecomputersociety.org/10.1109/CVPR.2004.9. – ISSN 1063–6919

[25] Heikkilä, Janne ; Silvén, Olli: A Four-step Camera Calibration Procedure with Implicit Image Correction. In: *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, IEEE Computer Society, 1997, 1106

[26] Jähne, Bernd: *Digitale Bildverarbeitung*. Bd. 6., überarbeitete und erweiterte Auflage. Springer, 2005

[27] Kaiser, Gerald ; Birkhäuser (Hrsg.): *A Friendly Guide to Wavelets*. Birkhäuser Boston, 1994

[28] Kim, D. ; Dahyot, R.: Face components detection using SURF descriptors and SVMs. In: *Machine Vision and Image Processing Conference, 2008. IMVIP'08. International*, 2008, 51–56

[29] Lahres, Berhard ; Raýman, Gregor: *Objektorientierte Programmierung - Das umfassende Handbuch*. Galileo Computing, 2009. – isbn: 978-3-8362-1401-8

[30] Lanitis, A. ; Hill, A. ; Cootes, T.F. ; C.J.Taylor: Locating Facial Feautes using Genetic Algorithms. In: *Proceedings of the 27th International Conference on Digital Signal Processing, Limassol (Cyprus)*, 1995, 520–525

[31] Lienhart, Rainer ; Maydt, Jochen: An Extended Set of Haar-like Features for Rapid Object Detection. In: *IEEE ICIP 2002* 1 (2002), September, 900-903. http://www.lienhart.de/ICIP2002.pdf

[32] Liu, Chengjun ; Wechsler, Harry: Enhanced Fisher Linear Discriminant Models for Face Recognition. In: *International Conference on Pattern Recognition* 2 (1998), 1368. http://dx.doi.org/http:

//doi.ieeecomputersociety.org/10.1109/ICPR.1998.711956. – DOI http://doi.ieeecomputersociety.org/10.1109/ICPR.1998.711956. – ISSN 1051–4651

[33] Liu, Chengjun ; Wechsler, Harry: Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition. In: *IEEE Transactions On Image Processing* 11 (2002), April, Nr. 4, 467–476. http://dx.doi.org/10.1109/TIP.2002.999679. – DOI 10.1109/TIP.2002.999679

[34] Liu, Ke ; Cheng, Yong-Qing ; Yang, Jing-Yu: Algebraic feature extraction for image Recognition based on an optimal discriminant criterion. In: *Pattern Recognition Society* 26 (1993), Nr. 6, 903–911. http://dx.doi.org/10.1016/0031-3203(93)90056-3. – DOI 10.1016/0031–3203(93)90056–3

[35] Lotlikar, Rohit ; Kothari, Ravi: Fractional-Step Dimensionality Reduction. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 22 (2000), Nr. 6, 623–627. http://dx.doi.org/http://dx.doi.org/10.1109/34.862200. – DOI http://dx.doi.org/10.1109/34.862200. – ISSN 0162–8828

[36] Lu, Juwei ; Plataniotis, N. ; Venetsanopoulos, Anastasios: Face Recognition Using LDA-based Algorithms. In: *IEEE Transactions On Neural Networks* 14 (2003), 195-200. http://dx.doi.org/10.1.1.91.4281. – DOI 10.1.1.91.4281

[37] Lu, Xiaoguang: *Image Analysis for Face Recognition.* 2003

[38] Martinez, Aleix M. ; Kak, Avinash C.: PCA versus LDA. In: *IEE Transactions on Pattern Analysis and Machine Intelligence* 23 (2001), S. 228–233

[39] Meir, R. ; Ratsch, G.: An introduction to boosting and leveraging. In: *Lecture Notes in Computer Science* 2600 (2003), 118–183. http://dx.doi.org/10.1.1.102.6483. – DOI 10.1.1.102.6483

[40] Moghaddam, Baback ; Pentland, Alex: Probabilistic Visual Learning for Object Representation. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 19 (1997), Nr. 7, 696-710. http://bit.ly/Moghaddam97

[41] Oestereich, Bernd: *Analyse und Design mit UML 2.3: Objektorientierte Softwareentwicklung.* 9., aktualisierte und erweiterte Auflage. Oldenbourg, 2009 http://amazon.de/o/ASIN/3486588559/. – ISBN 9783486588552

[42] Osuna, E. ; Freund, R. ; Girosi, F. u. a.: Training support vector machines: an application to face detection. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1997, 130–136

[43] Papageorgiou, C.P. ; Oren, M. ; Poggio, T.: A general framework for object detection. In: *Sixth International Conference on Computer Vision, 1998.*, 1998, 555-562

[44] Pearson, Karl: On Lines and Planes of Closest Fit to Systems of Points in Space. In: *Philosophical Magazin* 2 (1901), Nr. 6, 559-572. http://bit.ly/PearsonKLT01

[45] Penev, Penio S. ; Sirovich, Lawrence: The Global Dimensionality of Face Space. In: *In: Proceedings of the 4th Intl. Conference on Automatic Face and Gesture Recognition, IEEE CS*, 2000, 264–270

[46] Phillips, P. J. ; Moon, Hyeonjoon ; Rizvi, Syed A. ; Rauss, Patrick J.: *The FERET Evaluation Methodology for Face-Recognition Algorithms*. 1999

[47] Phillips, P.J. ; Wechsler, H. ; Huang, J. ; Rauss, P.: The FERET database and evaluation procedure for face recognition algorithms. In: *Image and Vision Computing Journal* 16 (1998), Nr. 5, S. 295–306

[48] Rowley, H. A. ; Baluja, S. ; Kanade, T.: Neural network-based face detection. In: *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR '96*, 1996, S. 203–208

[49] Rowley, Henry A. ; Baluja, Shumeet ; Kanade, Takeo: Neural Network-Based Face Detection. In: *IEE Transactions on Pattern Analysis and Machine Intelligence* 20 (1998), January, Nr. 1, 23-36. http://bit.ly/Rowley98

[50] Ruderman, Daniel L. ; Bialek, William: Statistics of natural images: Scaling in the woods. In: *Phys. Rev. Lett.* 73 (1994), Aug, Nr. 6, 814–817. http://dx.doi.org/10.1103/PhysRevLett.73.814. – DOI 10.1103/PhysRevLett.73.814

[51] Sanderson, Arthur C. ; Weiss, Lee: Adaptive Visual Servo Control of Robots. In: *Robot Vision* 1 (1983), S. 107–116

[52] Scharr, Hanno: *Optimale Operatoren in der Digitalen Bildverarbeitung*, Ruprecht-Karls-Universität Heidelberg, Diss., 2000. http://bit.ly/Scharr00

[53] Seo, Naotoshi: *Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features)*. Internet. http://note.sonots.com/SciSoftware/haartraining.html. Version: 2009

[54] Shakhnarovich, Gregory ; Moghaddam, Baback: Face Recognition in Subspaces / Mitsubishi Electric Research Laboratories. Version: May 2004. http://bit.ly/Shakhnarovich04. Ed. Springer-Verlag, May 2004 (2004-041). – Forschungsbericht. – -

[55] Sharkas, M. ; Elenien, M.A.: Eigenfaces vs. fisherfaces vs. ICA for face recognition; a comparative study. In: *Signal Processing, 2008. ICSP 2008. 9th International Conference on*, 2008, S. 914–919

[56] Siciliano, Bruno ; Sciavicco, Lorenzo ; Villani, Luigi ; Oriolo, Giuseppe: *Robotics - Modelling, Planning and Control*. Springer, 2009

[57] Sirovich ; Kirby: Low-dimensional procedure for characterization of human faces. In: *Journal of Optical Society* 4 (1987), 519-524. http://dx.doi.org/10.1364/JOSAA.4.000519. – DOI 10.1364/JOSAA.4.000519

[58] Sobel, I. ; Feldman, G.: *A 3x3 Isotropic Gradient Operator for Image Processing*. 1968. – vorgestellt während eines Vortrags im Stanford Artificial Project (1968), nicht veröffentlicht

[59] Swets, D. L. ; Weng, J. J.: Using discriminant eigenfeatures for image retrieval. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18 (1996), August, Nr. 8, S. 831–836. http://dx.doi.org/10.1109/34.531802. – DOI 10.1109/34.531802

[60] Tsai, Roger: A Versatile Camera Calibration Techniaue for High-Accuracy 3D Machine Vision Metrology Using Off-the-shelf TV Cameras and Lenses. In: *IEEE Journal of Robotics and Automation* RA-3, Nr. 4 (1987), 323-344. http://bit.ly/Tsai87

[61] Turk, Matthew ; Pentland, Alex: Eigenfaces for Recognition. In: *Journal of Cognitive Neuroscience* 3 (1991), 71-86. http://www.face-rec.org/algorithms/PCA/jcn.pdf

[62] Valiant, Leslie: A Theory of the Learnable. In: *Communications of the ACM* 27 (1984), Nr. 11, 1134-1142. http://bit.ly/Valiant84

[63] Viola, Paul ; Jones, Michael: Robust Real-time Object Detection. In: *International Journal of Computer Vision* 57(2) (2001), 137-154. http://bit.ly/Viola01

[64] Viola, Paul ; Jones, Michael: Fast Multi-view Face Detection. In: *TR2003-96* 96 (2003), -. http://www.merl.com/papers/docs/TR2003-96.pdf

[65] Walnut, David F. ; Birkhäuser (Hrsg.): *An Introduction To Wavelet Analysis*. Birkhäuser Boston, 2002

[66] Wikipedia: *Lambertsches Gesetz — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Lambertsches_Gesetz&oldid=65625065. Version: 2009. – [Online; Stand 13. November 2009]

[67] Wolpert, David H. ; Macready, William G.: No Free Lunch Theorems for Search / Santa Fe Institute. 1995 (SFI-TR-95-02-010). – Forschungsbericht. – http://bit.ly/Wolpert95nofree

[68] Yang, G. ; Huang, T.S.: Human face detection in a complex background. In: *Pattern recognition* 27 (1994), Nr. 1, 53–63. http://bit.ly/Yang94

[69] Yang, Ming-Hsuan ; Kriegman, D. J. ; Ahuja, N.: Detecting faces in images: a survey. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (2002), Januar, Nr. 1, S. 34–58. http://dx.doi.org/10.1109/34.982883. – DOI 10.1109/34.982883

[70] Zhang, Taiping ; Fang, Bin ; Tang, Yuan Y. ; Shang, Zhaowei ; Xu, Bin: Generalized Discriminant Analysis: A Matrix Exponential Approach. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 40 (2010), Februar, Nr. 1, S. 186–197. http://dx.doi.org/10.1109/TSMCB.2009.2024759. – DOI 10.1109/TSMCB.2009.2024759

[71] Zhang, ZhenQiu ; Li, MingJing ; Li, S. Z. ; Zhang, HongJiang: Multi-view face detection with FloatBoost. In: *Proc. Sixth IEEE Workshop on Applications of Computer Vision (WACV 2002)*, 2002, 184–188

[72] Zhao, W. ; Corporation, Zhao S. ; Chellappa, R. ; Phillips, P. J. ; Rosenfeld, A.: Face Recognition: A Literature Survey. In: *ACM Computing Surveys* - (2003), 399-458. http://www.face-rec.org/interesting-papers/General/zhao00face.pdf

# Linkverzeichnis

[73] CodePlex. Homepage des projekts SANDCASTLE im microsoft codeplex. http://www.codeplex.com/Sandcastle.

[74] FDB. At&t "database of faces". (früher Olivetti Research Labs Database).

[75] FDB. Color feret database. Portions of the research in this paper use the FERET database of facial images collected under the FERET program, sponsored by the DOD Counterdrug Technology Development Program Office.

[76] FDB. Sheffield face database (former umist).

[77] FDB. Yale face database.

[78] FDB. Yale face database b. siehe auch [GeBeKr01].

[79] FRH. Face recognition homepage - databases. http://www.face-rec.org/databases/.

[80] SparxxSystems. Download 30 tage-testversion von enterprise architect. http://www.sparxsystems.de/enterprise-architect/download-trial/.

# Index