

COMPUTER  
VISION

HANDBUCH  
FACE**R**



# Inhaltsverzeichnis

1. Einleitung	3
2. Bedienung des Programms	4
2.1 Ein neues Gesicht speichern	4
2.2 Erkennung starten	5
3. Softwarestruktur	6
3.1 Installierte Pakete	6
3.2 Struktur	6
3.2.1 Pseudo Code	6
3.2.2 Flussdiagramm	8
3.2.3 Klassendiagramm	9
4. Haar Cascade	10
5. Eigenfaces	12
6. Fazit	15
7. Quellenverzeichnis	16
8. Credits	17

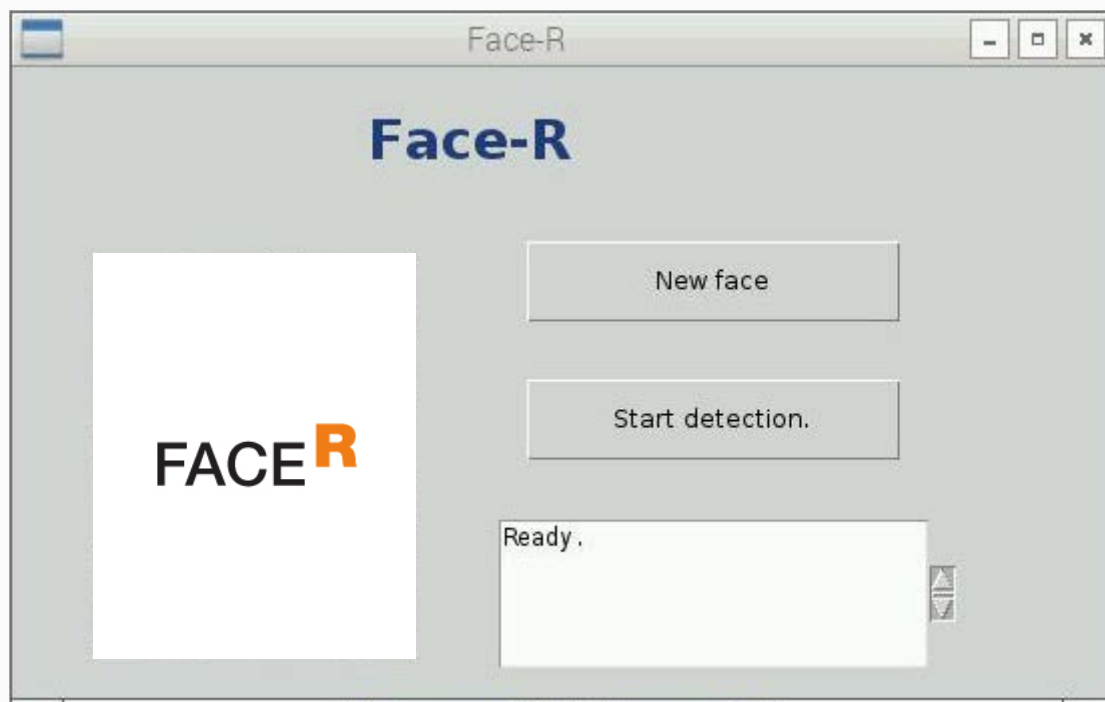
# Einleitung

Die Semesteraufgabe für die BFO „Computervision“ war es, ein Programm umzusetzen, dass Personen mittels einer Kamera erkennt, nach Möglichkeit mit einer Wahrscheinlichkeit, die höher als zufällig ist. Als Hardware wurde ein Raspberry Pi mit einer internen Kamera zur Verfügung gestellt. Die Umsetzung der Gesichtserkennung sollte in der Sprache Python erfolgen. Es handelt sich bei der Aufgabe um eine Gruppenarbeit, die bis zum 17. Juli 2015 abgegeben werden muss.

Auf den folgenden Seiten wird näher auf die Bedienung und technische Umsetzung des Programms eingegangen, sowie auf den zugrunde liegenden theoretischen Hintergrund zur Gesichtserkennung.

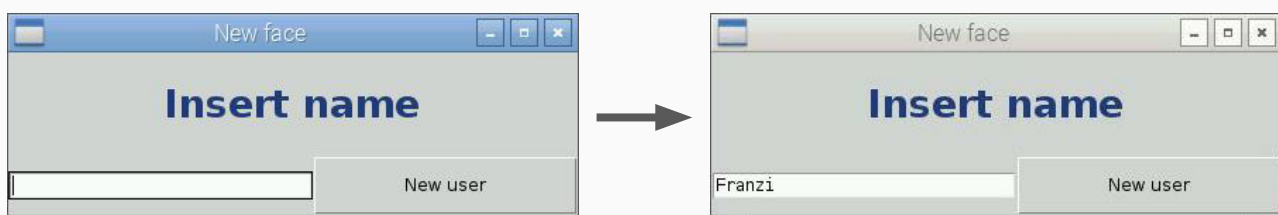
# Bedienung des Programms

Um FaceR erfolgreich benutzen zu können, muss das Programm im ersten Schritt gestartet werden. War der Start erfolgreich, sollte folgende Benutzeroberfläche erscheinen.

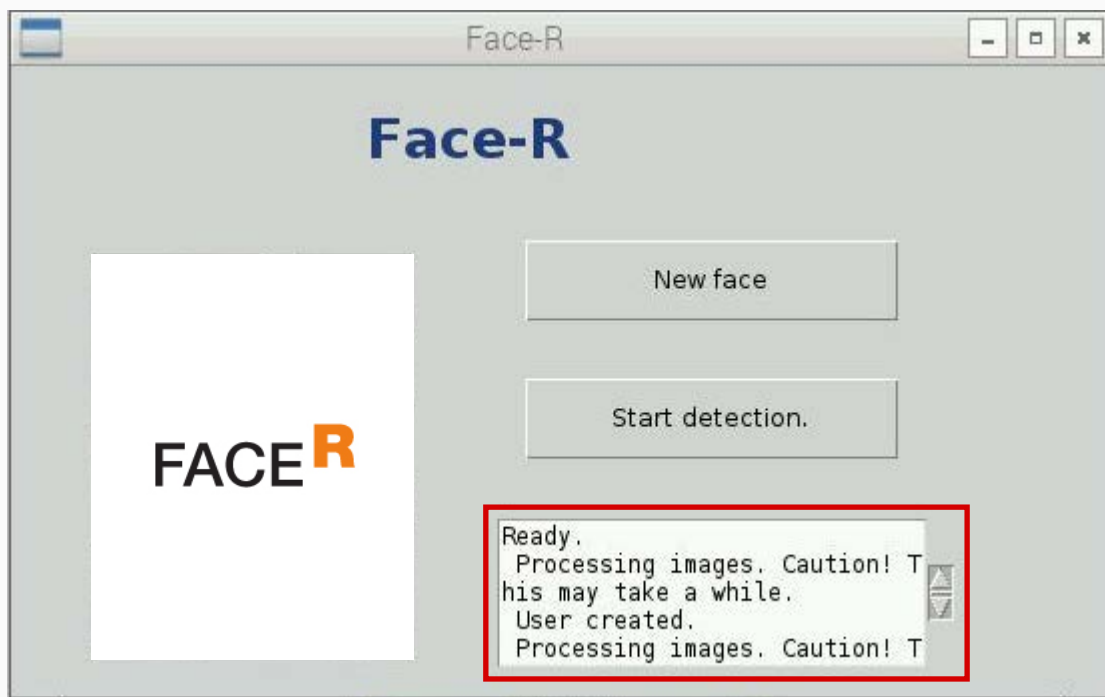


## 2.1 Ein neues Gesicht speichern

Um eine neue Person anzulegen, die erkannt werden soll, wird der Button „New Face“ gedrückt. Daraufhin erscheint ein Dialog, in den der Name des neuen Benutzers eingetragen wird.



Bestätigt man den Namen nun mit dem Button „New User“ wird die Kamera gestartet und es werden 5 Fotos von der Person als Referenzbilder angelegt. Diese werden für die spätere Gesichtserkennung vorbereitet, weswegen der Benutzer einen Moment Geduld aufbringen muss. Während dieser Zeit sind keine anderen Eingaben möglich. Der Benutzer wird durch eine entsprechende Nachricht darauf hingewiesen. Ist der Benutzer schließlich erfolgreich angelegt, wird die Meldung „User created“ ausgegeben.



### 2.2 Erkennung starten

Um die Gesichtserkennung zu starten, wird der Button „Start detection“ gedrückt. Daraufhin startet die Kamera erneut. Guckt der Benutzer nun bei ausreichender Beleuchtung (möglichst dieselbe Beleuchtung wie bereits bei den Referenzbildern) und in einem guten Winkel in die Kamera (es empfiehlt sich hier ebenfalls derselbe Gesichtsausdruck bzw. eine ähnliche Haltung des Kopfes), wird er erkannt und sein Name wird in dem Textfenster ausgegeben.



# Systemstruktur

Das System von FaceR setzt sich aus den folgenden Elementen zusammen und wird durch unterstützende Diagramme verdeutlicht.

---

## 3.1 Installierte Pakete

Zur Umsetzung wurden diese zusätzlichen Pakete installiert:

- vorkompilierte Open CV Library 2.4.8
  - Python 2.7
  - tkinter
  - python-opencv
  - numpy
  - Pakete in der Development-Variante (libjpeg8-dev, libtiff4-dev, libjasper-dev, libpng12-dev, libavcodec-dev, libavformat-dev, libswscale-dev, libv4l-dev)
- 

## 3.2 Struktur

### 3.2.1 Pseudo Code

- 1 *Starte das Programm*
- 2 *User-Optionen (Button-Click-Methoden)*
  - 2.1 *Starte Training (Button-Click)*
    - 2.1.1 *Zeige UI an*
    - 2.1.2 *Gebe Namen ein (User-Input)*
    - 2.1.3 *Starte Aufnahme der Bilder (Button-Click)*
      - 2.1.3.1 *Kamera aktivieren*
      - 2.1.3.2 *5 Bilder aufnehmen*
      - 2.1.3.3 *Wiederhole Bild bearbeiten, solange bis alle 5 bearbeitet wurden*
        - 2.1.3.3.1 *Gesicht in Bild finden*
        - 2.1.3.3.2 *Bild beschneiden*
        - 2.1.3.3.3 *Bild in Graustufen umwandeln*
        - 2.1.3.3.4 *.jpg ersetzen durch .pgm*

#### 2.1.4 UI schließen

### 2.2 Starte Erkennung (Button-Click)

#### 2.2.1 Foto aufnehmen

##### 2.2.1.1 Kamera aktivieren

##### 2.2.1.2 Bild aufnehmen

##### 2.2.1.3 Bearbeite Foto

###### 2.2.1.3.1 Gesicht in Bild finden

###### 2.2.1.3.2 Bild beschneiden

###### 2.2.1.3.3 Bild in Graustufen umwandeln

###### 2.2.1.3.4 reference.jpg ersetzen durch reference.pgm

###### 2.2.1.3.5 in Ordner pictures ablegen

#### 2.2.2 Finde Person

##### 2.2.2.1 Berechne Zahl der Bilder, welche zum Vergleich stehen

###### 2.2.2.1.1 Zähle Ordner

###### 2.2.2.1.2 Multipliziere Ordner mit 5

###### 2.2.2.1.3 Wiederhole Bild zu Array hinzufügen, solange bis alle Bilder aus Datenbank im Array enthalten sind

###### 2.2.2.1.3.1 Gehe in Ordner mit Bildern

###### 2.2.2.1.3.2 Gehe in Ordner mit Namen

###### 2.2.2.1.3.3 Wiederhole Bild hinzufügen, solange alle Bilder aus dem Ordner im Array vorhanden sind

###### 2.2.2.1.3.3.1 Hole dir Foto

###### 2.2.2.1.3.3.2 Füge Foto in Array hinzu

##### 2.2.2.1.4 Hole reference.pgm

##### 2.2.2.1.5 Vergleiche reference.pgm mit Bildern im Ordner

###### 2.2.2.1.5.1 Ist mindestens ein reference.pgm da?

###### 2.2.2.1.5.1.1 JA: Vergleiche reference.pgm mit Bildern

###### 2.2.2.1.5.1.1.1 Vergleiche reference.pgm mit Bild im Array, solange alle Bilder mit reference.pgm verglichen

###### 2.2.2.1.5.1.1.2 Bilde Differenz zwischen Bildern und Referenz

###### 2.2.2.1.5.1.1.3 Gib Bild mit geringster Differenz zurück

###### 2.2.2.1.5.1.1.4 Gib Namen aus

###### 2.2.2.1.5.1.1.5 Gib Differenz aus

###### 2.2.2.1.5.1.1.6 Gib Ergebnis aus

###### 2.2.2.1.5.1.1.6.1 Ist Differenz $< 1.7$ ?

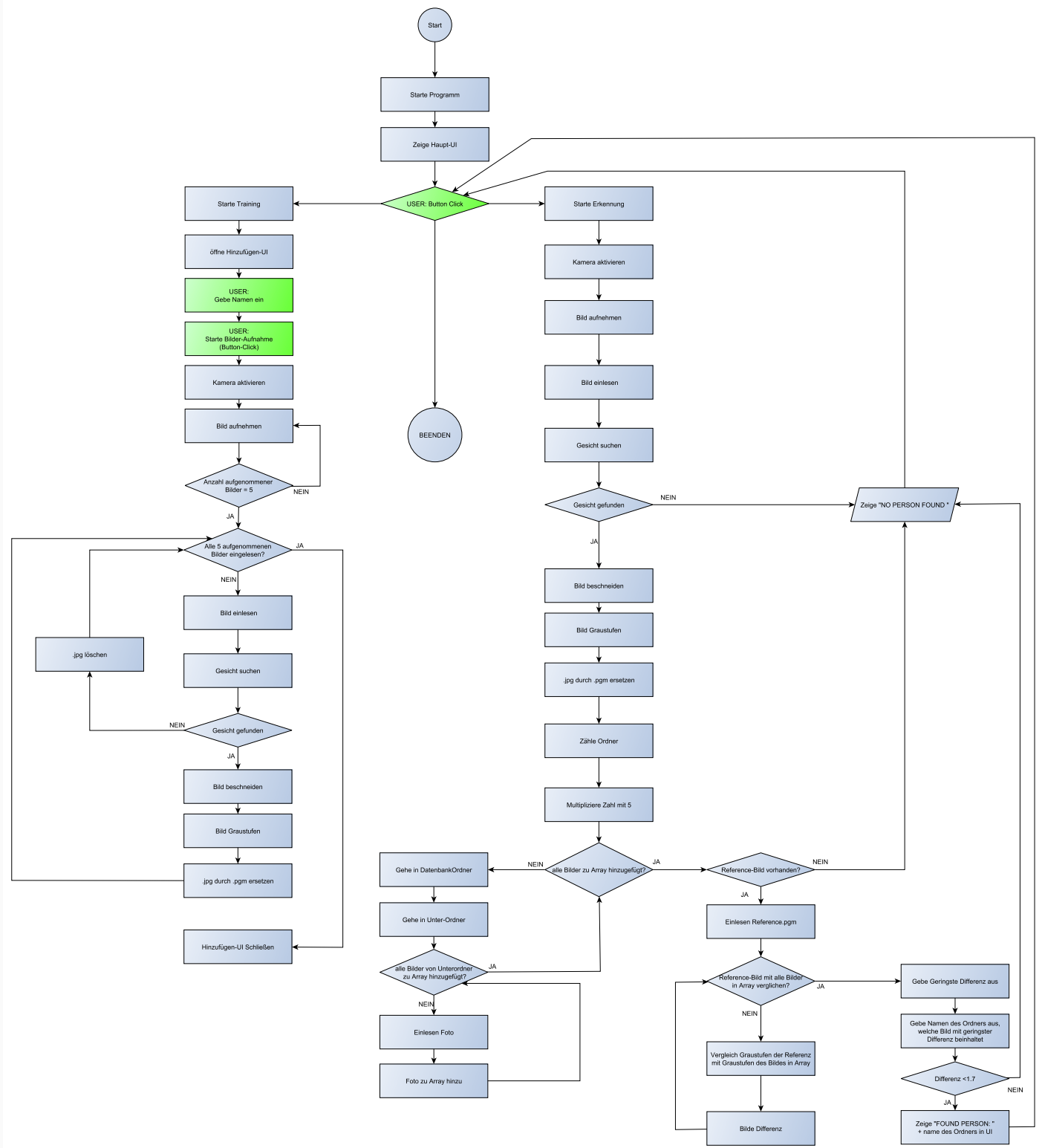
###### 2.2.2.1.5.1.1.6.1.1 JA: Gib „PERSON GEFUNDEN AUS“ + Name

###### 2.2.2.1.5.1.1.6.1.2 NEIN: Gib „KEINE PERSON GEFUNDEN AUS“

###### 2.2.2.1.5.1.2 NEIN: Gib „KEINE PERSON GEFUNDEN AUS“

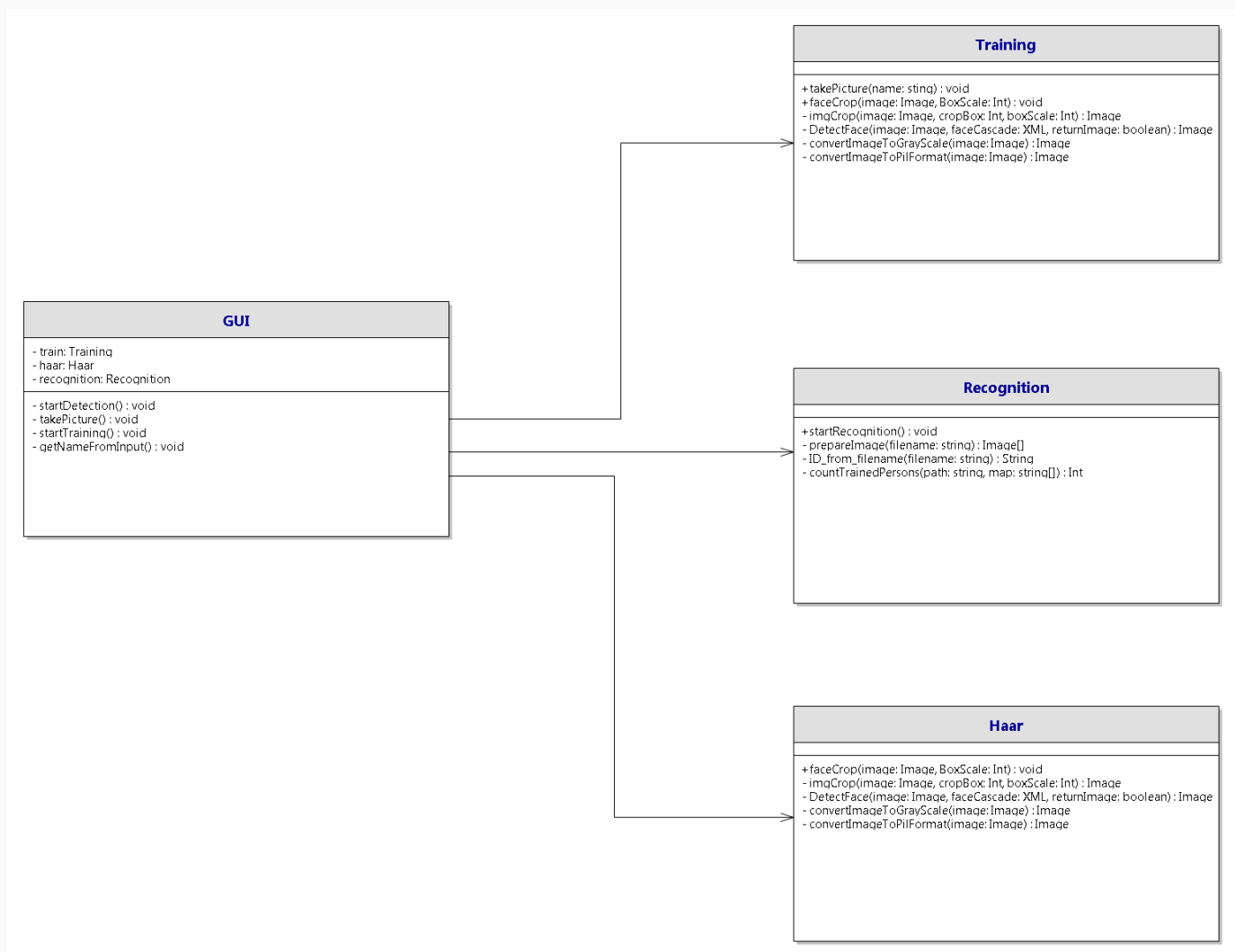
### 3 Programm beenden

## 3.2.2 Flussdiagramm





### 3.2.3 Klassendiagramm

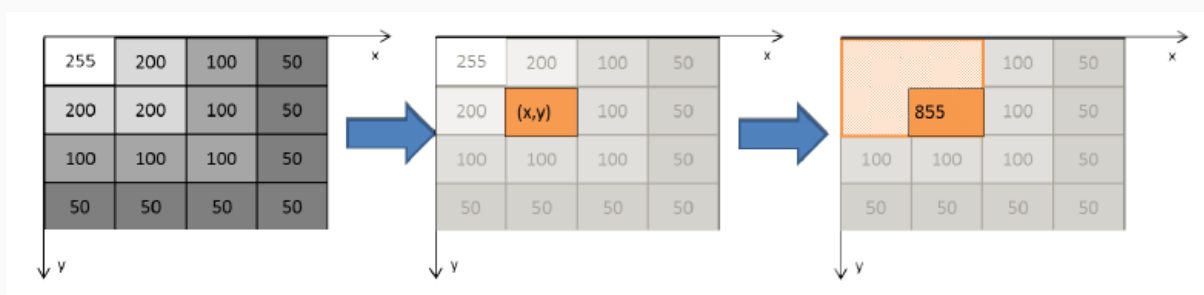


# Haar Cascade

Die Haar Cascade ist ein Algorithmus, mit dem ein Gesicht in einem Bild gefunden werden kann. Er basiert auf der von Paul Viola und Michael Jones entwickelten Methode aus dem Jahr 2001. Sie arbeitet in Echtzeit und wurde für Open CV entwickelt. Es geht dabei darum, dass das Programm auf der Grundlage vieler positiver und negativer Bilder lernt, ein Objekt zu erkennen. Wenn der Funktion eine große Anzahl von Referenzbildern vorliegt, ist sie in der Lage, auch in anderen Bildern das entsprechende Objekt zu finden. Man verwendet diese Methode, um die Rechenintensität zu senken, indem man sich nur auf den Bereich des Bildes konzentriert, welcher für die Gesichtserkennung wichtig ist.

Um den classifier (Sichter) zu „trainieren“ benötigt der Algorithmus eine gewisse Anzahl von positiven Bildern, das bedeutet Bilder mit Gesichtern sowie eine gewisse Anzahl von Bildern ohne Gesichter (negative Bilder). Dazu werden den Bildern bestimmte Eigenschaften entnommen.

Im ersten Schritt wird das Bild in Graustufen konvertiert, da so die Intensitätswerte der einzelnen Pixel besser sichtbar werden. Dazu nutzt man das Konzept des Integralbildes. Es wird ein Rechteck zwischen dem Ursprung und dem aktuellen Pixel aufgespannt und der Wert dieses Pixels ergibt sich aus der Summe aller Pixelwerte, die in dem Rechteck enthalten sind.



Die Verwendung des Integralbildes verkürzt die Rechenzeit deutlich, da alle weiteren Pixelwerte durch Elementaroperationen und den bereits berechneten Werten erschlossen werden können. Deswegen gilt: je heller ein Bereich ist, desto höher ist die Summe des Pixels.

Aus diesen Informationen lässt sich eine Menge interpretieren, da man genau weiß, welche Teile eines Gesichtes heller sind als andere. Diese werden erkannt, indem man bestimmte Rechtecke mit jeweils einer hellen und einer dunklen Seite über das zu erkennende Bild legt.



Nun wird die Differenz der Pixelsummen des jeweils hellen und dunklen Bereichs mit dem zugrunde liegenden Integralbild errechnet. Diese geben Auskunft über die Kanten, indem sie Regionen suchen, die horizontal oder vertikal etwa die gleiche Größe oder Form haben. Zudem werden die diagonalen Summen berechnet und voneinander abgezogen, sowie Linienmerkmale gesucht.

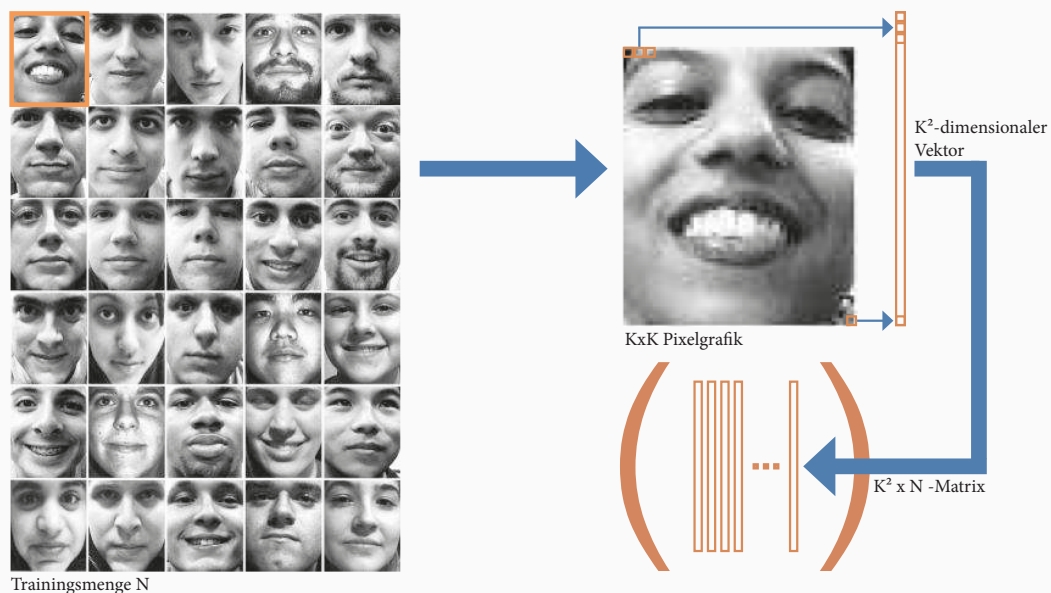
Der nächste Schritt ist die sogenannte AdaBoost-Prozedur. Hier passiert das „Training“ des Classifiers. Das Bild wird mit einem 24 x 24 Pixel großen Unterfenster nach Gesichtsmerkmalen durchsucht. Dadurch entstehen viele „schwache“ Classifiers, die alleine nichts aussagen würden. In der Kombination zu einem classifier jedoch sehr effizient und schnell entscheiden können, wo sich ein Gesicht befindet.

Im letzten Teil passiert die Kaskade. Die eben erwähnten „schwachen“ Classifiers ohne Gesichtsmerkmale werden aussortiert. Es wird davon ausgegangen, dass nur jeder erfolgreiche Classifier durch einen vorhergegangenen Positiven ausgelöst wurde. Dies verkürzt die Rechenzeit, weil somit nur die positiven Unterfenster erhalten bleiben und die negativen weitgehend ignoriert werden können. Dadurch entsteht der finale Classifier, welcher nur aus Classifiers mit relevanten Informationen besteht, nämlich dem eigentlichen Gesicht.

# Eigenfaces

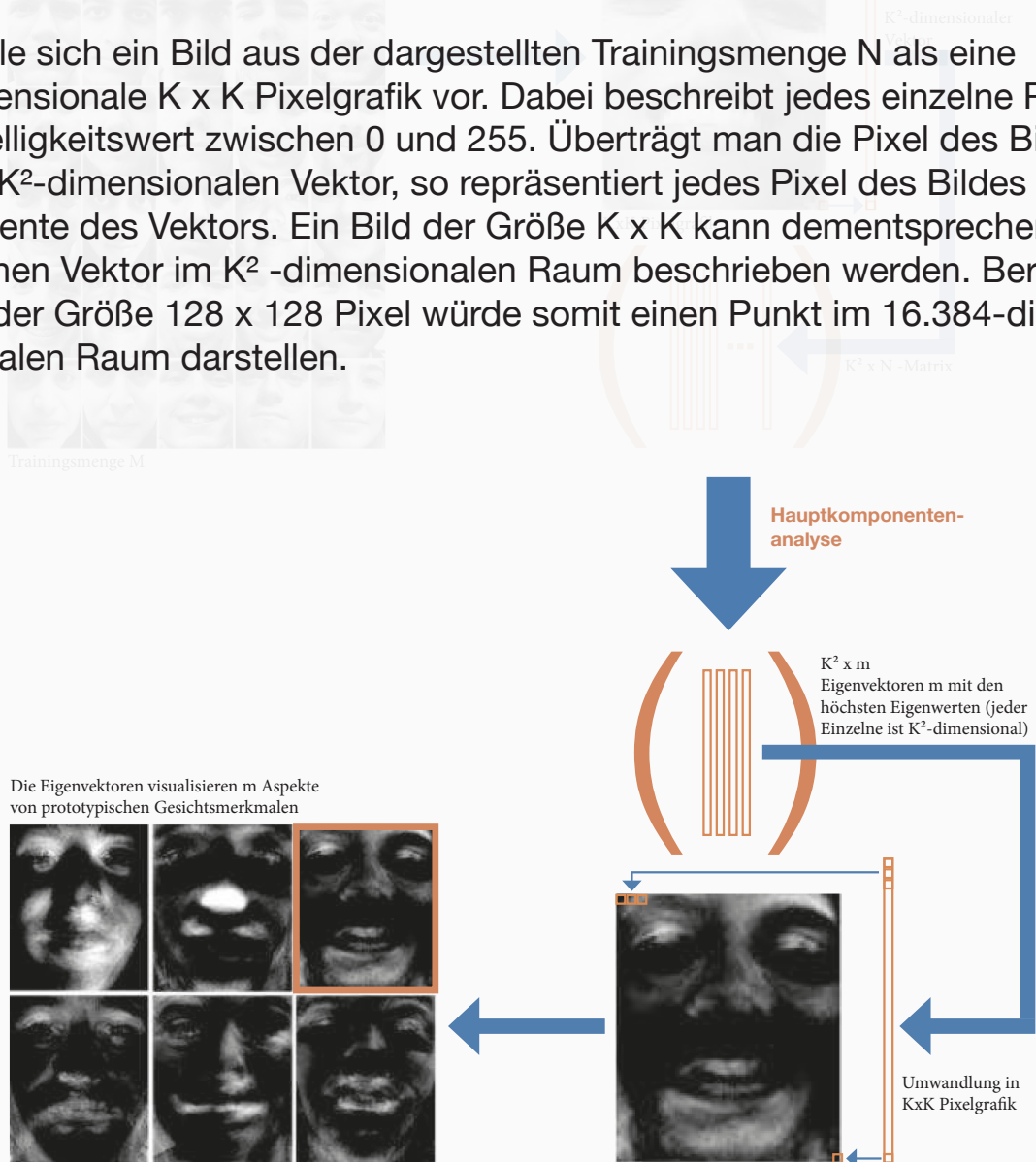
Die Eigenfacemethode nach Kirby und Sirovich (1990) ist ein Verfahren, beruhend auf der Hauptkomponentenanalyse (auch Karhunen-Loève-Transformation genannt) zur Gesichtserkennung. Es betrachtet nicht primär wie andere Methoden die markanten Gesichtsm Merkmale wie Augen, Nase und Mund im Einzelnen, sowie deren relative Lage zueinander, sondern behandelt das Gesicht als Ganzes. Die darin enthaltenen Informationen werden dann so effizient wie möglich verarbeitet, um im Anschluss daran einen Vergleich zu ähnlich codierten Mustern zu ermöglichen.

Um das Verfahren anwenden zu können, muss eine Menge (Trainingsmenge  $N$ ) von verschiedenen Gesichtsbildern bekannter Personen vorliegen. Die spätere Gesichtserkennung funktioniert dabei umso besser, je mehr Variationen an Bildern der einzelnen Personen in der Trainingsmenge vorhanden sind. Die Bilder sollten sich dabei grundsätzlich im Gesichtsausdruck, in der Neigung des Kopfes und den Lichtverhältnissen unterscheiden. So kann eine bessere Identifizierung der Personen unter variierenden Umständen gewährleistet werden. Weiterhin müssen alle Bilder eine einheitliche Größe aufweisen und in Graustufenbilder in das Portable Graymap (PGM) Format umgewandelt werden, sodass jedes einzelne Pixel einen Wert zwischen 0 und 255 annimmt. Diesen Prozess übernimmt die oben beschriebene Haar Cascade. Aus der Trainingsmenge  $N$  errechnet sich das Durchschnittsbild, welches



ebenfalls für die Hauptkomponentenanalyse benötigt wird. Ist diese Vorarbeit geleistet, kann mit der Berechnung der Eigenfaces begonnen werden.

Man stelle sich ein Bild aus der dargestellten Trainingsmenge  $N$  als eine zweidimensionale  $K \times K$  Pixelgrafik vor. Dabei beschreibt jedes einzelne Pixel einen Helligkeitswert zwischen 0 und 255. Überträgt man die Pixel des Bildes in einen  $K^2$ -dimensionalen Vektor, so repräsentiert jedes Pixel des Bildes eine Komponente des Vektors. Ein Bild der Größe  $K \times K$  kann dementsprechend durch einen Vektor im  $K^2$ -dimensionalen Raum beschrieben werden. Bereits ein Bild der Größe  $128 \times 128$  Pixel würde somit einen Punkt im 16.384-dimensionalen Raum darstellen.



Aufgrund der Ähnlichkeit von Gesichtern (jedes gesunde Gesicht verfügt generell über zwei Augen, eine Nase und einen Mund) kommt man zu dem Resultat, dass sich die Punkte (Vektoren) nicht beliebig in diesem hochdimensionalen Raum verteilen, sondern konzentriert an einer Stelle liegen. Die Punkte lassen sich durch diese Anballung, einem relativ kleinen Unterraum, dem Gesichtsraum (face space), beschreiben.

Bei der Behandlung aller Pixelgrafiken auf dieselbe Weise ergibt sich eine  $K^2 \times N$ -Matrix, die die Helligkeitswerte aller vorhandenen Bilder der Trainingsmenge  $N$  beinhaltet. Jede Spalte der Matrix symbolisiert dabei eine Pixelgrafik. Die erhaltene Matrix bildet die Grundlage für die Hauptkomponentenanalyse.

Die Hauptkomponentenanalyse hat die Erhaltung von Merkmalen, die die Trainingsmenge am besten beschreiben, zum Ziel. Folglich müssen diejenigen Vektoren gefunden werden, die die Verteilung der Pixelgrafiken im gesamten Bildraum bestmöglich repräsentieren. Diese Vektoren werden Eigenfaces genannt und bilden die Basis des Gesichtsraums (face space). Subtrahiert man das Durchschnittsgesicht, um die jeweiligen Merkmale jedes Individuums mit der größten Varianz herauszufiltern, erhält man als Resultat die Eigenfaces.

Andersherum setzt sich schließlich das Originalbild aus dem Durchschnittsbild und bestimmten Anteilen verschiedener Eigenvektoren zusammen. Die einzelnen Eigenfaces unterscheiden sich hierbei in ihren ganz bestimmten Merkmalen.



# Fazit

Nach unseren anfänglichen Startschwierigkeiten wegen einer defekten SD-Karte konnten wir das Programm innerhalb der vorgesehenen Zeit funktionsfähig umsetzen. Als gewöhnungsbedürftig zeigten sich unsere ersten Erfahrungen mit Python, wobei sich gerade die Einrückungen des Codes als tückisch herausstellten. Schwierigkeiten ergaben sich besonders im Umgang mit dem Raspberry PI, der aufgrund seiner Leistungsschwäche zu einigen Performanceeinbußen führte. Nach wie vor muss man sich an einigen Stellen (wie in der Bedienungsanleitung beschrieben) etwas gedulden, man wird jedoch in den meisten Fällen mit einer erfolgreichen Erkennung seines Gesichts belohnt.

Voraussetzung dafür ist die Aufnahme der fünf Referenzbilder, welche sich leicht im Ausdruck des Gesichts und der Neigung des Kopfes unterscheiden sollten. Dabei spielt auch das Umgebungslicht eine entscheidende Rolle, denn die Gesichtserkennung mittels Eigenfaces arbeitet, wie bereits im Kapitel „Eigenfaces“ beschrieben, mit den Ähnlichkeiten der Hauptkomponenten der Gesichter, die durch die verschiedenen Helligkeitswerte im Bild charakterisiert sind. Daher werden bei einer geringen Menge an Referenzbildern möglichst gleiche oder sehr ähnliche Lichtverhältnisse empfohlen, um eine präzise Erkennung zu gewährleisten.

Die Probleme, die die Eigenface-Methode mit sich bringt, lassen sich mehr oder weniger durch beispielsweise Belichtungskorrekturen und Histogrammabgleiche beheben, erfordern aber für eine noch präzisere Erkennung als „einer Wahrscheinlichkeit, die mehr als zufällig ist“, auch mehr Rechenleistung der Hardware. Insgesamt haben wir den Aufgabenumfang trotz geringer Rechenleistung erfüllt und präsentieren mit Stolz unser Programm „FaceR“, welches meistens zuverlässig Gesichter erkennt und diese den richtigen Personen zuordnet.

# Quellenverzeichnis

OpenCV-Python Tutorials:

[http://opencv-python-tutroals.readthedocs.org/en/latest/py\\_tutorials/py\\_objdetect/py\\_face\\_detection/py\\_face\\_detection.html](http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html) [Stand: 03.07.2015]

Boffo, Sandra:

Assistenzsysteme mit Emotionserkennung. Prototypische Realisierung mit Betrachtung der ethischen Dimension

[ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart\\_fi/STUD-2450/STUD-2450.pdf](ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/STUD-2450/STUD-2450.pdf) [Stand: 03.07.2015]

Lavrenko, Victor:

PCA 10: eigen-faces

[https://www.youtube.com/watch?v=\\_IY74pXWIS8](https://www.youtube.com/watch?v=_IY74pXWIS8) [Stand: 06.07.2015]

Kopf, Stephan; Oertel, Alexander:

Gesichtserkennung in Bildern und Videos mit Hilfe von Eigenfaces

[http://ub-madoc.bib.uni-mannheim.de/1100/1/TR\\_2005\\_008\\_FaceRecognition.pdf](http://ub-madoc.bib.uni-mannheim.de/1100/1/TR_2005_008_FaceRecognition.pdf) [Stand: 06.07.2015]



An diesem Projekt arbeiteten:

19972/u27924	Franziska Schulz
19796/u27787	Jennifer Kolz
19797/u27786	Sebastian Fischer
19974/u27922	Mirco Hüneke

### ▲ Hochschule Harz

Hochschule für angewandte Wissenschaften