SHKOLLA AUSTRIAKE

# Diplomarbeit

# Bildverarbeitung

Autor: Egli Hasmegaj

# Inhaltsverzeichnis

# Kapitel 1

# Bildverarbeitung

## 1.1 Allgemeines

Diese Diplomarbeit besteht aus vielen unterschiedlichen Modulen, die um bestimmte Ziele bzw. Aufgaben zu lösen gut aufgeteilt sind.
Sehr wichtiger Teil dieses Projekts ist die Bildverarbeitungsteil.
In der folgenden Abschnitt der Ausarbeitung wird es genau erklärt und beschrieben wie diese Aufgabe gelöst wurde und was dafür verwendet war.
Für die Umsetzung wurden folgende Technologien gebraucht.

### 1.1.1 Entwicklungsumgebung und Technologien

Die gewählte Entwicklungsumgebung war grundsätzlich eine virtuelle Maschine die Linux auf einem Windows System geboten hat.
Linux im Gegensatz von Windows ermöglicht volle Kontrolle über Updates und Upgrades und dadurch könnten komplexe Aufgaben einfacher erlediget werden.
Alles wird leicht und bequem durch Konsole eingegeben.
So wurden die Entwicklung, das Testen von den genutzten Algorithmen und anderen Technologien vielmals erleichtert. [**linx**]

Zusätzlich ist Raspberry Pi als Backup System verwendet worden das auch mit einem lauffähigen Linux Betriebssystem (Debian) funktionierte.
Raspberry pi ist zwar klein, aber funktioniert ganz gut wie ein normaler Rechner und ist kostengünstig. Für technische/elektronische Projekte wie das betreffende, ist es perfekt geeignet.
Für die Implementierung der Code wurde hauptsächlich die Programmiersprache Python verwendet.
Python ist eine allgemeine Programmiersprache auf hohem Niveau. Dies bedeutet dass es näher an menschlichen Sprachen und weiter von Maschinensprachen ist.
Also ist ein in Python geschriebener Code sehr leicht von einem Mensch zu lesen, zu verwalten und zu warten.

Es bietet zahlreiche Modulen durch seine große und robuste Standardbibliothek an, von denen man ruhig, abhängig vom Bedarf, auswählen kann. Sehr leicht kann es in der Dokumentation der Python-Standardbibliothek nachgesehen werden um sich besser mit den gezielten Funktionalitäten auszukennen.
[**why˙python**]

"Git ist ein freies und Open Source verteiltes Versionskontrollsystem, das entwickelt wurde, um alles von kleinen bis zu sehr großen Projekten mit Geschwindigkeit und Effizienz abzuwickeln."[**Git1**]
Die Versionierung der ganzen Software Änderungen erfolgte durch Git.
Es hat sich nützlich erweist indem die Arbeit dadurch zwischen die Projektmitgliedern sehr gut koordiniert und verwaltet wurde.

### 1.1.2 Frameworks und Bibliotheken

Schlüsselwort von unserem Projekt war die Framework bzw. die Bibliothek „OpenCV".
OpenCV ist eine open-source Bibliothek für Computer Vision. Also, ganz allgemein kann sie als eine Bibliothek für Bildverarbeitung betrachtet werden.
Sie kümmert sich unter anderem um die Manipulation von Bildern, die Analyse von denen und um die daraus bestimmte Muster bzw. Objekte, für verschiedenen Zwecken einzusetzen. Ganz berühmte Anwendungsgebieten sind Gesichtserkennung und Stereo Vision.
Stereo Vision bedeutet die Extrahierung von Informationen aus einem Bild in einer 3-dimensionalen Ebene.
OpenCV wurde unter anderen Bibliotheken aufgrund seiner vielen guten Eigenschaften und seiner Flexibilität ausgewählt. Es umfasst hunderte von Computer-Vision-Algorithmen und besteht aus strukturierten Einheiten bzw. Module die als feststehende Bibliotheken implementiert sind.
Es ist Cross-Platform, in C/C++ geschrieben und unterstützt auch Python.
[**opencv˙library**]

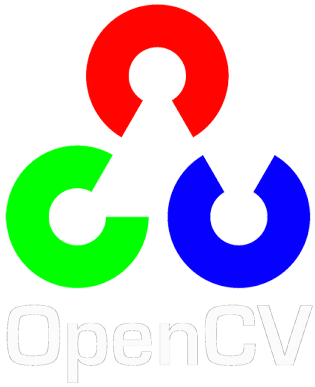


Abbildung 1.1: OpenCV logo

[**OpencvLogo**]

SciPy hat sich deswegen von Nutzen erweist, um gewisse mathematische Angaben zu lösen und bei zusätzliche Installationen von Bibliotheken zu helfen.
SSciPy ist eine kostenlose und Open-Source-Python-Bibliothek für wissenschaftliches und technisches Rechnen."
[**2019arXiv190710121V-scipy**]
Dies sind einige der Kernpakete von SciPy die nutzbar für das Projekt waren:

NumPy legt die Basis für das wissenschaftliche Rechnen mit Python.
Es unterstützt große mehrdimensionale Arrays und Matrizen. Es enthält viele Mathematische Funktionen auf hoher Ebene um die Arrays zu bearbeiten.
NumPy kann also als leistungsfähiger mehrdimensionaler Behälter für generische Daten verwendet werden.
Es können beliebige Datentypen definiert werden.


Abbildung 1.2: Numpy logo

[**Numpy**]

Dlib ist ein so genanntes Toolkit, die Algorithmen für Machine Learning und Tools zum Erstellen komplexer Software zur Lösung von der realen Welt getauchte Probleme, beinhaltet.
Es wird überall eingesetzt, in Robotik, Mobilgeräte und unter anderem in Computer Vision auch.[**dlib**]
Es wurde prinzipiell um bei der Extrahierung der so genannten „Facial Landmarks" gebraucht.

Abbildung 1.3: dlib logo

[**dlib**]

## 1.2 Technische Lösungen

Bei der Gesichtsregistrierung und Gesichtserkennung Diplomarbeit war es die schwierigste und herausforderndste Aufgabe, sie sorgfältig zu planen, um die Effektivität der Arbeit zu steigen und das Endprodukt zufriedenstellend zu machen.

Wichtig war es die Ziele richtig zu setzen und sie gut abzugrenzen damit es in keine Lücken bei der Implementierung führte.

Aus diesem Grund musste die Arbeitsteilung gut geregelt werden, damit jedes Teammitglied an einem bestimmten Modul der Arbeit konzentrieren konnte.
Es wurde auch das berücksichtigt, dass jedes Teammitglied das machte was ihm/ihr am besten gefällt und was ihm/ihr am leichtesten fälle.

Die Planung der betreffende Bereich der Projektarbeit hat durch die Methode „Structed Design“ erfolgt.
Structed Design ist eine Erweiterung von der „Big Picture“, die dazu dient, ein technisches System mit ihren Schnittstellen mit außen grob zu beschreiben.
Es ist in verschieden Ebenen unterteilt, von außen beginnend.

Unten wird die erste Ebene der Structed Design von der Bildverarbeitungsteil dargestellt.

### 1.2.1 Lösungsweg - Structed Software design

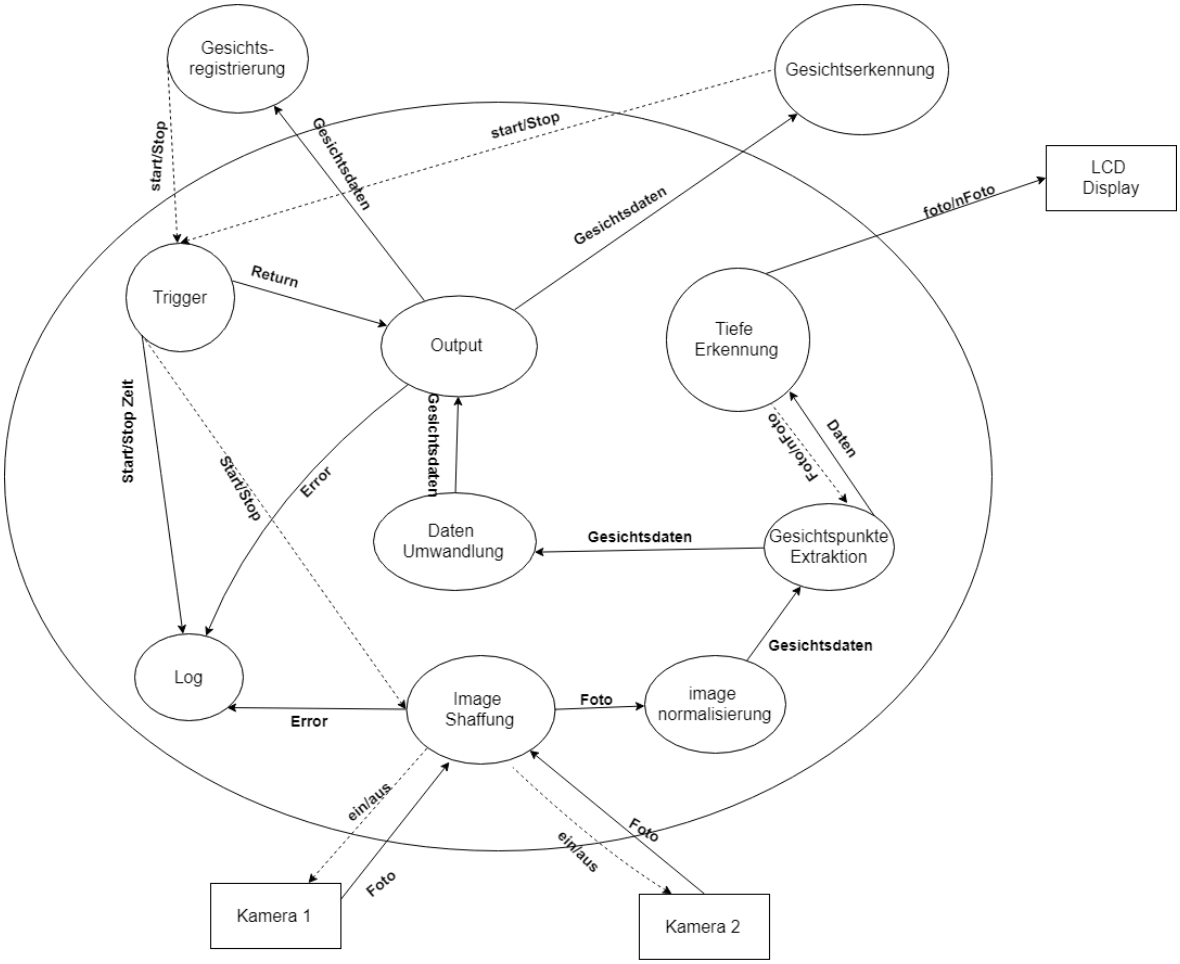


Abbildung 1.4: Bildverarbeitung Structed Design

Wie es in der Abb.1.4 sichtbar ist, ist die Bildverarbeitungsteil in diese Einheiten unterteilt:

1. Image Schaffung
2. Image Normalisierung
3. Gesichtschlüsselpunkten Extrahierung
4. Log
5. Output
6. Trigger

Die Schnittstellen von außen sind die Gesichtserkennungsmodul und die Gesichtsregistrierungsmodul.
Diese schicken eine Anforderung an den Trigger der dann die Image Schaffung Modul initialisiert. Unabhängig von welchen von diesen beiden der Bedarf hatte, macht die Image Schaffung aus der zwei Kameras ein Foto.
Implementierungsnah wurde bis jetzt nur eine Kamera verwendet, da die Stereo Vision noch nicht funktional ist.
Nachdem das Foto gemacht wird folgt die Image Normalisierung bzw. die „Image Prozessing“.
Sei ein Bild ist zu klein wird die Größe angepasst, seien die Gesichter rotiert werden sie so gemacht dass sie gerade rotiert. Affines und perspektivisches Warping wird falls notwendig geführt. Farben werden nach Bedarf auch angepasst usw.
Danach erfolgt die Erkennung von Gesichtern, die Ausschneidung von denen und die Extrahierung der Gesischtsschlüsselpunkte.
Sie werden dann in Vektoren umgewandelt und zur Abgleich oder Registrierung je nach welche Signal der Trigger bekommt, geschickt.

Hinsicht: Bei der Umsetzung wurden bestimmte Bereiche mit einander verknüpft, was zu Folgendes führte: was bei der Planung steht, stimmt nicht vollig mit der Methoden zur Umsetzung überein. Weitere Unterschiede werden im Punkt 13b. genauer beschrieben.

### 1.2.2 Face Detection

Das erste Ereignis, dass erfüllt werden muss bei der Gesichtserkennung ist das Finden von Gesichtern, also die Feststellung, wo die Gesichter in das Bild befinden.
Dafür ist hier die „Haar Cascade Classifiers“ pre-trained Classifier gebraucht worden.

Gesichtsdetektion durch „Haar“ Merkmale ist eine sehr effektive Methode die von Paul Viola und Michael Jones entwickelt wurde indem sie Merkmale gruppiert haben und die Erkennung von diesen dadurch schneller gemacht.

Die Arbeit heißt „Rapid Object Detection using a Boosted Cascade of Simple Features”.

Es geht hier um Maschinelles Lernen, wie diese Kaskadenfunktionen durch viele negative(Bilder ohne Gesichter) und positive(Bilder mit Gesichter) Bilder trainiert wurden um Objekte zu erkennen.

In diesem Fall arbeiten wir selbstverständlich mit Gesicht Objekten. Dafür wurden die Haar Merkmale benutzt. Auf Abbildung 2 sind sie dargestellt. Jedes Haar Merkmal ist nichts anders als ein Wert, durch das Subtrahieren der Summe der Pixel unter dem weißen Rechteck von der Summe der Pixel unter dem schwarzen Rechteck, erhalten.
Diese Summen berechnet man durch integrale Bilder, die zur Vereinfachung von den

Summenberechnungen dient.

[**Viola01robustreal-time**]

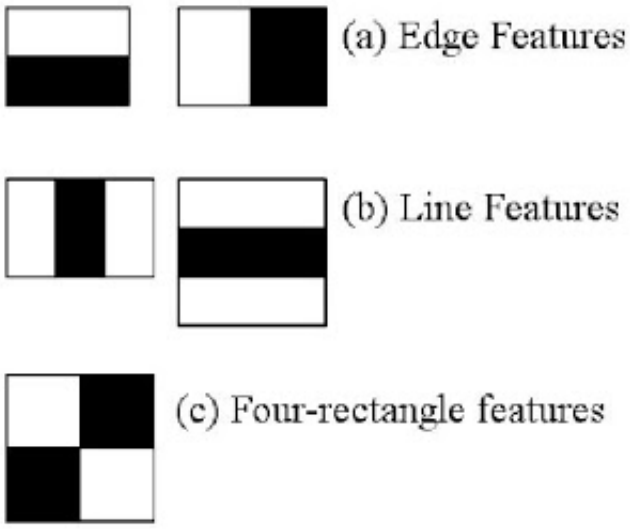


Abbildung 1.5: Haar Features

[**Viola01robustreal-time**]

Man muss aufpassen, da nicht alle Merkmale vom Nutzen sein können. Man kann es zum Beispiel deutlich auf Abbildung 3 sehen, wie die Nase viel heller als die Region von der Augen ist. Die Selektierung von den besten Merkmalen wird durch das Adaboost Algorithmus berechnet.

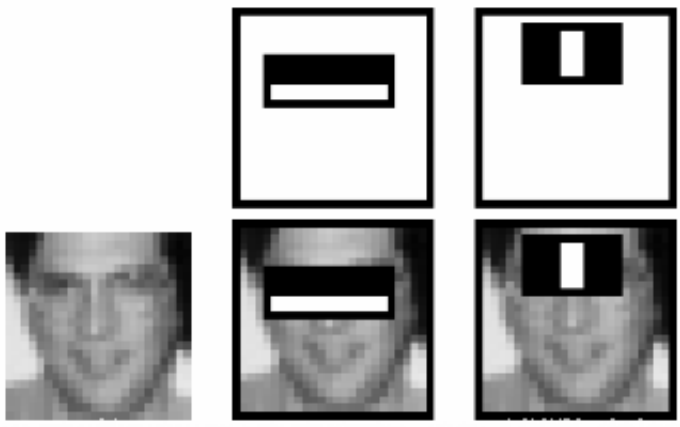

Abbildung 1.6: Haar Einsetzung

[**Viola01robustreal-time**]

Mit dieser Absicht setzen wir alle Funktionen auf alle Trainingsbilder ein. Für jedes Merkmal wird der beste Schwellenwert ermittelt, der die Gesichter in positive und

negative klassifiziert.

### Umsetzung in Code

Durch die Verwendung von classifiers die OpenCV bietet, setzen wir das um.

```
//haar cascade classifier laden
1  face_cascade =
    cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

2  gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

3  faces = face_cascade.detectMultiScale(gray,1.1,4)

4  for(x, y, w, h) in faces:
//parameter: image, startpunkt, endpunkt(w..width, h..height), farbe, Dicke
    der Rahmen
5  cv2.rectangle(image, (x, y), (x+w, y+h), (255,0,0), 2)
```

//opencv und numpy importieren
//Bild als input einfügen
//Bild einlesen
//Bild in „gray scale“ umwandeln wodurch die Rechenleistung reduziert wird. //(Da es keine Farben zuhanden sind).
//Parametern: src(source image), dst(destination image), code(conversion code)
//Hier benutzt man die Cascade Classifier um Gesichter zu finden.
//Parameter:
//image, objects, scaleFactor = 1.1, minNeighbors = 3, flags = 0, minSize = new cv.Size(0, 0), maxSize = new cv.Size(0, 0))
//Zeichenrahmen für das Gesicht zeichnen.
//Bild zeigen durch ‘image’ Fenster
//Press any key zum schließen vom Fenster

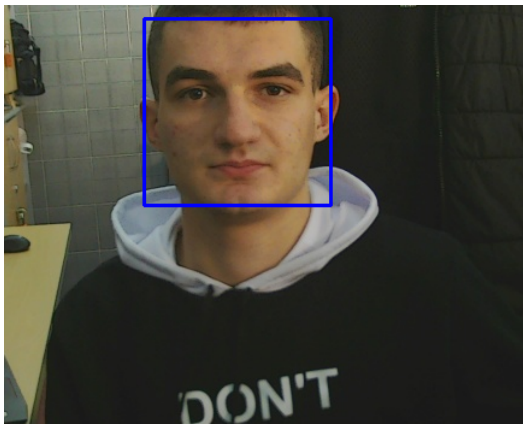


Abbildung 1.7: Box on face: Aron

Die Funktionalität des Code ist genau auf Abb.1.7 ersichtlich.

### 1.2.3 Face Detect and crop

Nachdem Gesichter gefunden werden, braucht man daraus eigene Bilder machen, die zur Erleichterung von der Extrahierung der Schlüsselpunkte helfen.

nrFace bestimmt die Anzahl von Gesichter von der cascade classifier gefunden. Nur wenn es größer als Null ist soll die Logik von der Crop Funktion weiterlaufen.

```
if nrFace > 0:
for face in faces:
for(x, y, w, h) in faces:
1  r = max(w, h) /2
2  centerx = x + w /2
3  centery = y + h /2
4  nx = int(centerx - r)
5  ny = int(centery - r)
6  nr = int(r * 2)
7  faceimg = image[ny:ny+nr, nx:nx+nr]
filenam = input("Give new filename for cropped photo: \n")
image2 = cv2.imwrite(filenam,faceimg)
elif nrFace <= 0:
print("no faces found")
```

Zeilen 1-6 berechnen das Zentrum von Bild neu und in Zeile 7 wird ein neues Image mit den berechneten Parametern angelegt.
Mit imwrite wird das Image gespeichert. Auf Abb.1.8 sieht man deutlich nur das Gesicht von Aron.

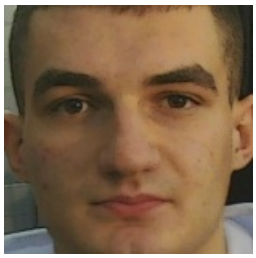

Abbildung 1.8: Aron Gesicht

### 1.2.4 Facial Landmarks Extraction

Endlich ist es zu dem wichtigsten Punkt meines Teils der Diplomarbeit gekommen, die Gesichtsschlüsselpunkte Extraktion.

Dazu wurden die „Facial Landmarks“ von dlib verwendet.
Sie, Gesichtsmarkierungen, werden verwendet, um Bereiche des Gesichts zu finden und darzustellen, wie zum Beispiel: die Augen, die Augenbrauen, die Nase, den Mund, den Kiefer.

Wie macht man das? Wie erkennt man Landmarken?
Das ist eine Teilmenge des Problems der Formvorhersage.
Bei einem vorgegebenen Eingabebild (und normalerweise einer ROI[1], die das interessierende Objekt angibt) versucht ein Formvorhersager, wichtige Punkte entlang der Form zu lokalisieren.
Dieser Formvorhersager die im dlib integriert ist wurde von Kazemi and Sullivan in ihrem Paper: One Millisecond Face Alignment with an Ensemble of Regression Trees entwickelt.

Dieser Methode werden viele Trainingsdaten hinzugefügt womit es ein Kombination von Regressionsbäumen trainiert wird um die Positionen der „Facial Landmarks“ direkt aus den Pixelintensitäten selbst zu beurteilen.
[**Kazemi2014OneMF**]

Dadurch werden unsere gewünschten 68 Gesichtsmarkierungen mit hohen Vorhersagbarkeit extrahiert und als x,y Koordinaten weitergegeben. Die Indizes der 68 Koordinaten sind in der Abb.1.9 dargestellt:

[1] Region of interest

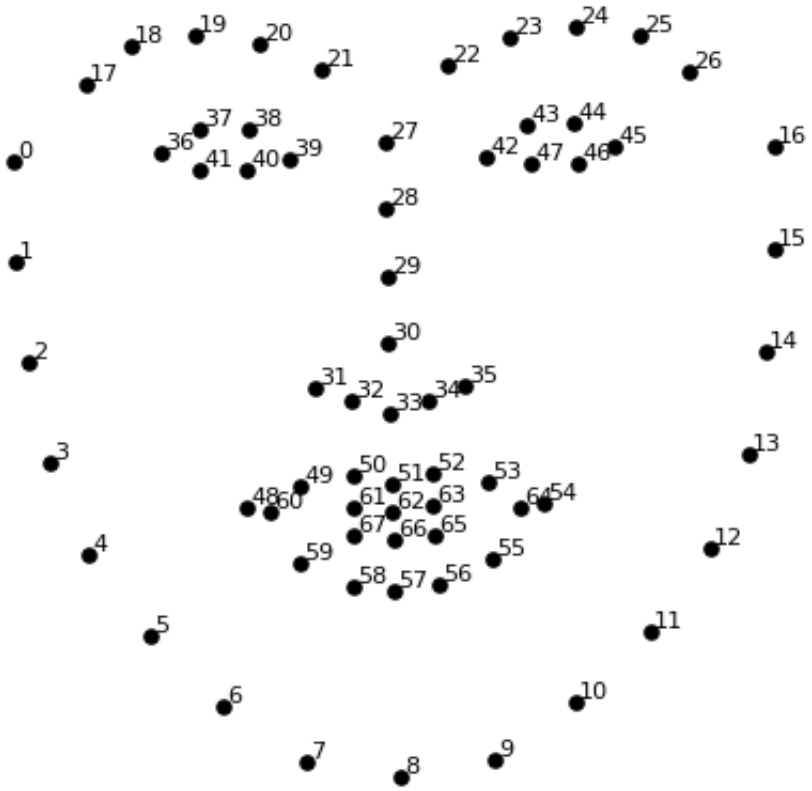


Abbildung 1.9: Facial Landmarks

[**Kazemi2014OneMF**]

Nachdem wir der „Predictor“ geladen haben speichern wir es in einem Formobjekt die die 68(x,y) Koordinaten enthält.
Es wird hier die resize Funktion nicht verwendet, weil es an Qualität verliert und es rechenintensiver ist, je größer das Eingabebild wird.
Jede Koordinate läuft in einer Schleife durch und entspricht dem spezifischen Gesichtsmerkmal im Bild.
Die Merkmale werden in einer numpy array gespeichert für den Zweck der Speicherung in der Datenbank.

Nur die Positionen von diesen Merkmalen alleine reichen bei Gesichtserkennung nicht. Deswegen wurden die relative Positionen bzw. Abstände zwischen Gesichtsmerkmalen als Vektoren genommen und betrachtet um die Qualität und die Präzision der Erkennung zu verbessern/verbessern.

*Die Zahlen beziehen sich auf die oben genannten Indizes.

1. Rechtes Auge Höhe: V1 = 41 – 37
2. Rechtes Auge Breite: V2 = 39 – 36
3. Linkes Auge Höhe: V3 = 46-44

4. Linkes Auge Breite: V4 = 45-42
5. Rechte Augenbraue Breite: V5 = 21 – 17
6. Linke Augenbraue Breite: V6 = 26 – 22
7. Rechte augenecke mit mittlerer rechter Augenbraue: V7 = 36 – 19
8. Linkes augenecke mit mittlerer linker Augenbraue: V8 = 45 – 24
9. Nasespitze mit Mundzentrum: V9 = 45 -24
10. Linke Augenecke mit linker Mund Ecke: V10 = 54 – 45
11. Rechte Augenecke mit rechter Mund ecke: V11 = 48 – 36
12. Nase Höhe: V12 = 33 – 27
13. Nase Breite: V13 = 35 – 31

## 1.3 Herausforderungen, Probleme, und wie wurden sie gelöst

Die größte Herausforderung lag bei der Überlegung von der Software und die Methoden die zum Extrahieren von den Gesichtsschlüsselpunkten dienten.
Es hat mir viel Zeit gedauert bis ich eine geeignete Lösung gefunden habe und das hat viel Stress gemacht.
Eine weitere Herausforderung war das Verknüpfen von den Entwicklungsumgebungen und die Kooperation zwischen den Teammitgliedern.
Die verwendete Systeme waren all zu unterschiedlich und es konnte keine Standardisierung zwischen denen geben. Also ist viel Zeit beim Installieren und Konfigurieren investiert worden. Ein Grund dafür ist die mangelte Erfahrung mit den neuen Technologien.
Viele Sachen waren im Beginn auch unklar wegen die Kommunikationslücken und auch die sehr grobe Planung hat nicht geholfen.

Lösungen:

Während der Arbeit habe ich viel recherchiert und mich genau über alles Mögliche informiert.
Die Kommunikation hat sich mit der Zeit viel verbessert und dadurch wurden auch die Unklarheiten abgeklärt.

## 1.4 Qualitätssicherung, Controlling

Die Qualität wurde durch verschiedene Methoden gesichert. Eine von denen war 5xWarum. “Fünf Warum (5x Warum) ist eine iterative Fragetechnik, mit der die Ursache-Wirkungs-Beziehungen untersucht werden, die einem bestimmten Problem zugrunde liegen. “ [**fmea**] Die aufgetauchten Probleme haben viele Ursachen, die nicht mit dem ersten Blick sichtbar sind. 5x warum hilft durch diese verschachtelten Ursachen das grundlegende Problem zu entdecken.
Eine Problemstellung: „Segmentation fault“ beim Finden von Keypoints mit FAST.“

1. Warum bekommt man überhaupt einen „Segmentation Fault“? Ein Segmentierungsfehler tritt auf, wenn ein Programm versucht, auf einen Speicherort zuzugreifen, auf den es nicht zugreifen darf, oder wenn versucht wird, auf einen Speicherort auf nicht zulässige Weise zuzugreifen (z. B. beim Versuch, an einen schreibgeschützten Speicherort zu schreiben, oder einen Teil des Betriebssystems zu überschreiben).

2. Warum versucht mein Program auf einen Speicherort zuzugreifen, auf den es nicht zugreifen darf?

   Problem ergibt sich in dieser Zeile: “kp = fast.detect(image, None)”

   Wahrscheinlich durften die Daten die fast.detect ergibt nicht in kp gespeichert werden.

3. Warum dürfen die Daten die fast.detect ergibt nicht in kp gespeichert werden ?

   Die Daten die fast.detect ergibt, dürfen nicht in kp gespeichert werden, weil kp kein Array ist.

4. Warum ist kp kein Array?

   Kp ist kein array, weil man es in Python manuell angeben muss.

5. Warum wurde es nicht manuell angegeben?

   Es wurde nicht manuell gegeben weil, die Methode fast.detect nicht gut untersucht/gearbeitet war. Man sollte mehr darüber in die Doku nachschauen. ODER hatte man beim Installation die benötigte Pakete nicht richtig installiert.

## 1.5 Ergebnise

Ich habe ungefähr 100 Stunden Zeit bis jetzt für die Diplomarbeit investiert und folgendes erreicht.
Face Detection wurde fertig implementiert.
Bilder sind halbwegs normalisiert worden.

Gesichtsschlüsselpunkte wurden extrahiert und die Merkmalvektoren wurden angegeben.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literatur