SHKOLLA AUSTRIAKE

Diplomarbeit

Bildverarbeitung

Autor: Egli Hasmegaj

Inhaltsverzeichnis

Kapitel 1

Bildverarbeitung

Diese Diplomarbeit besteht aus vielen unterschiedlichen Modulen, die um bestimmte Ziele bzw. Aufgaben zu lösen gut aufgeteilt sind. Ein sehr wichtiger Teil dieses Projektes ist der Bildverarbeitungsteil. In dem folgenden Abschnitt der Ausarbeitung wird genau erklärt und beschrieben wie diese Aufgabe gelöst wurde und was dafür verwendet wurde.

1.1 Allgemeines

In diesem Allgemeinen Abschnitt werden die verwendeten Technologien und Bibliotheken beschrieben.

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 zu Windows ermöglicht volle Kontrolle über Updates und Upgrades und dadurch könnten komplexe Aufgaben einfacher erlediget werden. Alles wird leicht und bequem durch die Konsole eingegeben. So wurden die Entwicklung und das Testen der genutzten Algorithmen und andere Technologien erleichtert. [5]

Zusätzlich ist Raspberry Pi als Backup System verwendet worden dies, funktioniert auch mit einem lauffähigen Linux Betriebssystem (Debian). Der Raspberry Pi ist klein, funktioniert aber wie ein normaler Rechner und ist kostengünstig. Für technische/elektronische Projekte wie das Betreffende, ist es perfekt geeignet. Für die Implementierung des Codes wurde hauptsächlich die Programmiersprache Python verwendet. Python ist eine höhere Programmiersprache. Dies bedeutet dass es näher an menschlichen Sprachen ist. Also ist ein in Python geschriebener Code sehr leicht von einem Mensch zu lesen, zu verwalten und zu warten.

Es bietet zahlreiche Module, mit seiner großen und robusten Standardbibliothek an, von denen man beruhigt, abhängig vom Bedarf, auswählen kann. Sehr leicht kann

man in der Dokumentation der Python-Standardbibliothek nachsehen um sich besser mit den Funktionalitäten auszukennen. [8]

"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."[1]
Die Versionierung der Software Änderungen erfolgte durch Git. Es hat sich als nützlich herausgestellt, da die Arbeit dadurch zwischen die Projektmitgliedern sehr gut koordiniert und verwaltet wurde.

1.1.2 Frameworks und Bibliotheken

Das Schlüsselwort von unserem Projekt war das Framework bzw. die Bibliothek „OpenCV". OpenCV ist eine open-source Bibliothek für Computer Vision. Ganz allgemein kann sie als eine Bibliothek für Bildverarbeitung betrachtet werden. Sie beschäftigt sich unter anderem mit der Manipulation und Analyse von Bildern, die Analyse von denen und um die Bestimmung von Muster bzw. Objekte, für verschiedenen Zwecken. Ganz berühmte Anwendungsgebieten sind Gesichtserkennung und Stereo Vision. Stereo Vision bedeutet die Extrahierung von Informationen aus einem Bild in eine 3-dimensionale 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. [4]

SciPy hat sich als nützlich, beim Lösen von gewisse mathematische Angaben und bei zusätzliche Installationen von Bibliotheken herausgestellt. SSciPy ist eine kostenlose, Open-Source-Python-Bibliothek für wissenschaftliches und technisches Rechnen."[10]
Dies sind einige der Kernpakete von SciPy die für das Projekt verwendet werden.

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

Dlib ist ein so genanntes Toolkit, das Algorithmen für Machine Learning und Tools zum Erstellen komplexer Software zur Lösung von Probleme der realen Welt beinhaltet. Es wird vielseitig eingesetzt, in Robotik, bei Mobilgeräte und unter anderem in Computer Vision.[7] Es wurde bei der Extrahierung der Gesichtsmerkmalen gebraucht. Das Logo von Dlib ist auf das 1.1 ersichtlich.

Abbildung 1.1: dlib logo [7]

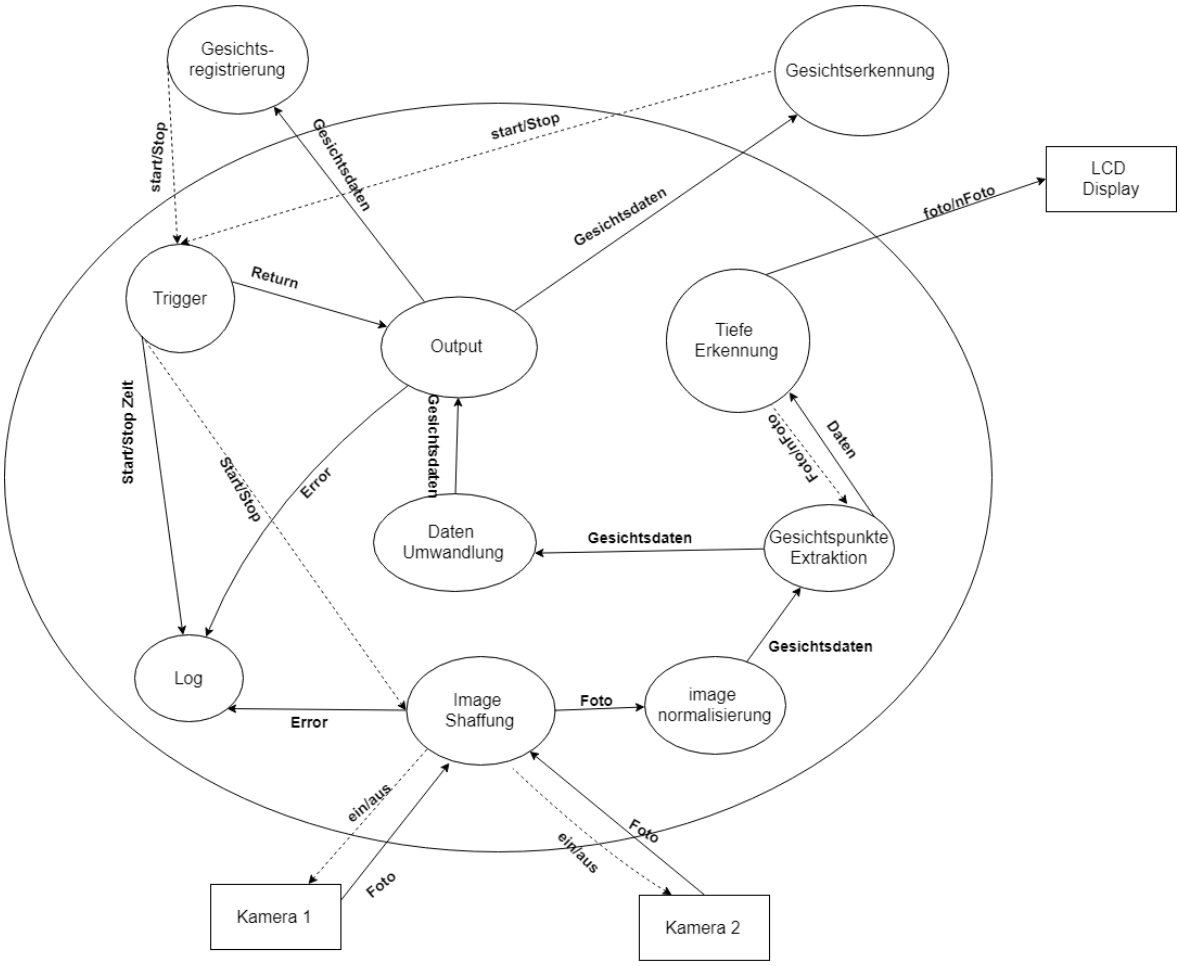


Abbildung 1.2: Bildverarbeitung Structed Design

1.2 Technische Lösungen

Bei der Gesichtsregistrierungs und Gesichtserkennungs Diplomarbeit war es eine schwierige und herausfordernde Aufgabe, sie sorgfältig zu planen, um die Effektivität bei der Arbeit zu steigern und das Endprodukt zufriedenstellend zu machen. Wichtig war es die Ziele richtig zu setzen und sie gut abzugrenzen, damit es zu keinen Lücken bei der Implementierung kommt. Aus diesem Grund musste die Arbeitsteilung gut geregelt werden, damit jedes Teammitglied sich auf bestimmte Modul der Arbeit konzentrieren konnte. Es wurde auch berücksichtigt, dass jedes Teammitglied das machte was ihm/ihr am besten gefällt und was ihm/ihr am leichtesten fällt. Die Planung der betreffenden Bereiche der Projektarbeit ist durch die Methode „Structed Design“ erfolgt. Structed Design ist eine Erweiterung von der „Big Picture“ Methode, die dazu dient, ein technisches System mit ihren Schnittstellen von außen grob zu beschreiben. Es ist in verschieden Ebenen unterteilt, von außen beginnend. Nun wird die erste Ebene der Structed Design von der Bildverarbeitungsteil dargestellt.

1.2.1 Lösungsweg - Structed Software design

Legende:

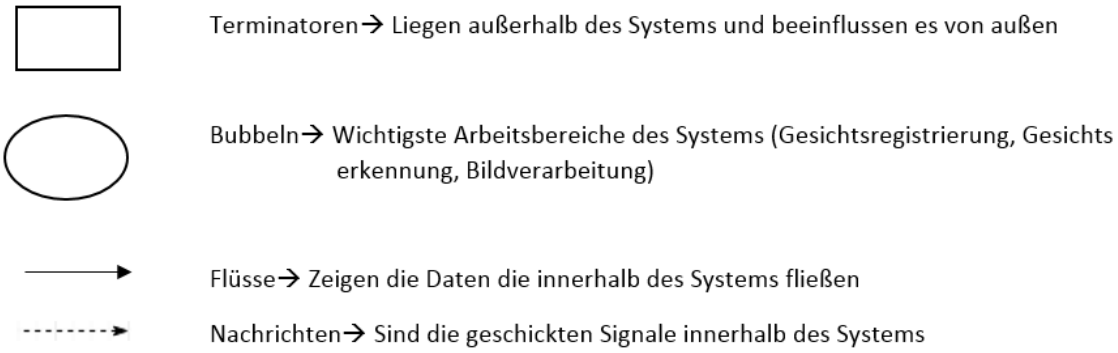


Abbildung 1.3: Erste Ebene Erklärung

Wie es in der Abb.1.2 ersichtlich ist, ist der Bildverarbeitungsteil in diese Einheiten unterteilt:

1. Bildaufnahme
2. Bild Normalisierung
3. Gesichtschlüsselpunkten Extrahierung
4. Log
5. Output
6. Trigger

Die Schnittstellen nach außen sind das Gesichtserkennungsmodul und das Gesichtsregistrierungsmodul. Diese schicken ein Signal an den Trigger der dann das Bildaufnahmemodul initialisiert. Unabhängig von welchen dieser beiden Schnittstellen die Anfrage kommt, macht das Bildaufnahmemodul mit den zwei Kameras ein Foto.

Implementierungsnah wurde nur eine Kamera verwendet, da die Stereo Vision Probleme gegeben hat. Nachdem das Foto gemacht wird folgt die Image Normalisierung bzw. das „Image Processing“. Wenn ein Bild zu klein ist, wird die Größe angepasst, wenn ein Gesicht verdreht ist wird es gerade rotiert. Mehr dazu wird im Normalisierungabschnitt erklärt.

Danach folgt die Erkennung von Gesichtern, das Ausschneiden von denen und die Extrahierung der Gesischtsschlüsselpunkte. Sie werden dann zum Abgleich oder Registrierung je nach Bedarf, geschickt.

Bei der Umsetzung wurden bestimmte Bereiche miteinander verknüpft, was zu Folgenden führte: was bei der Planung steht, stimmt nicht vollig mit der Methoden zur Umsetzung überein.

1.2.2 Gesichtserkennung

Das erste Ziel, dass bei der Gesichtserkennung erfüllt worden war ist das Finden von Gesichtern, also die Feststellung, wo sich die Gesichter im Bild befinden.

Dafür ist der vorgefertigter Klassifikator *„Haar Cascade Classifiers“* benutzt worden. Gesichtserkennung 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 erfolgt ist. Die Arbeit heißt „Rapid Object Detection using a Boosted Cascade of Simple Features”. [9]

Unten wird eine kurze technische Übersicht über diese Klassifikator gegeben.

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 Merkmale Haare benutzt. Auf Abbildung 1.4 ist dies dargestellt. Jedes Haar Merkmal ist nichts anders als ein Wert, den man durch das Subtrahieren der Summe der Pixel unter dem weißen Rechteck von der Summe der Pixel unter dem schwarzen Rechteck erhält.

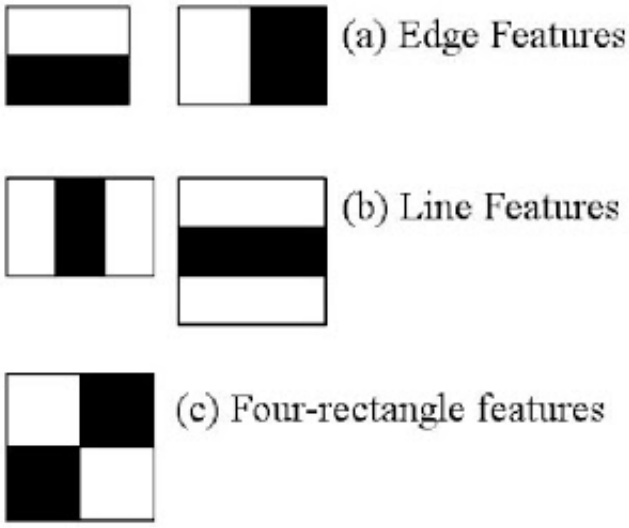


Abbildung 1.4: Haar Features[9]

Man muss aufpassen, da nicht alle Merkmale von Nutzen sein könnten. Man kann zum Beispiel deutlich auf Abbildung 1.5 sehen, wie die Nase viel heller als die Region bei den Augen ist. Die Selektion von den besten Merkmalen wird durch den Adaboost Algorithmus berechnet.

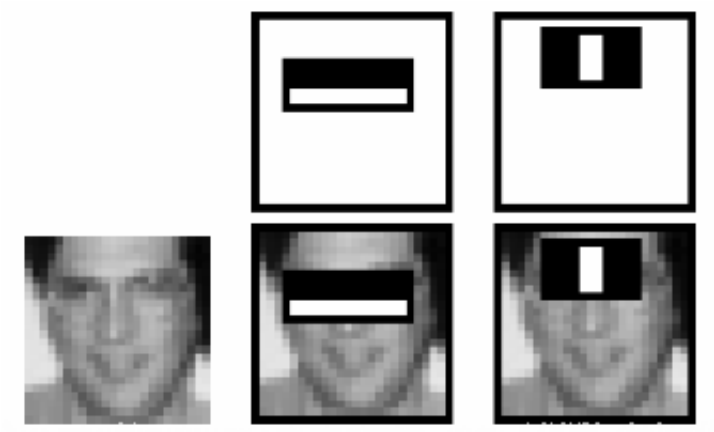


Abbildung 1.5: Haar

Mit dieser Absicht wenden wir alle Funktionen auf alle Trainingsbilder an. Für jedes Merkmal wird der beste Schwellenwert ermittelt, der die Gesichter in positive und negative klassifiziert.

Umsetzung in Code

In Listing 1.1 ist ein Code Abschnitt der Gesichtserkennung dargestellt.

```
1  face_cascade =
2  cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
3
4  image = cv2.imread('gesicht.jpg')
5
6  gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
7
8  faces = face_cascade.detectMultiScale(gray,1.1,4)
9
10 for(x, y, w, h) in faces:
11
12 cv2.rectangle(image, (x, y), (x+w, y+h), (255,0,0), 2)
13
```

Listing 1.1: Kern Code für Gesichtsdetektion

Es wurde die trainierte Klassifikator-XML-Datei (haarcascade_frontalface_default.xml) die sich im GitHub-Repository von OpenCV befindet, verwendet. In den Zeilen **1** und **2** wurde sie geladen und in die Variable face_cascade gespeichert. In den Zeile **4** wird das Bild von den Funktion *imread* geladen. Die Erkennung funktioniert nur bei Graustufenbildern. Daher wurde das Farbbild in Graustufen umgewandelt, wie in der Zeile **6** ersichtlich. Der Funktion *detectMultiScale* in der Zeile **8** erkennt die Gesichter im Bild. Diese Funktion ist sehr wichtig und braucht 3 Argumente – das Eingabebild, der Skalierungsfaktor (scaleFactor) und die Zahl *minNeighbours*.
scaleFactor gibt die an, um wie viel das Bild vergrößert bzw. verkleinert wird.

minNeighbours gibt an, wie viele Nachbarrechtecke jedes Rechteck haben muss. Dieser Parameter bestimmt die Qualität der erkannten Gesichter: Ein höherer Wert führt zu weniger Erkennungen, jedoch zu einer höheren Qualität. Es ist bei uns nicht so entscheidend viele Gesichter zu erkennen, sondern die Qualität. Es wird der Wert 4 benutzt.
In Zeile **10** wird durch die Gesichter *faces* iteriert.
x und y entsprechen den Koordinaten vom Bild, w, h bezeichnen die Breite und Höhe *width*, *height*.
Schließlich wird in Zeile **12** mit den Funktion *cv2.rectangle* ein Rahmen auf dem Gesicht gezeichnet.
Die Funktion bekommt diese Parameter: *image* für die Bildeingabe, x, y beschreiben den Startpunkt, w, h den Endpunkt, die Farbe und die Dicke der Rahmen werden in den 2 letzten Parameter angegeben.
Dieser Code wurde so umgeändert, dass er in der Lage ist, durch mehrere *Predictors* bzw. Klassifikatoren, die Genauigkeit der Gesichtserkennung zu erhöhen.

Auf Abbildung 1.6 sieht man deutlich ein erkanntes Gesicht.

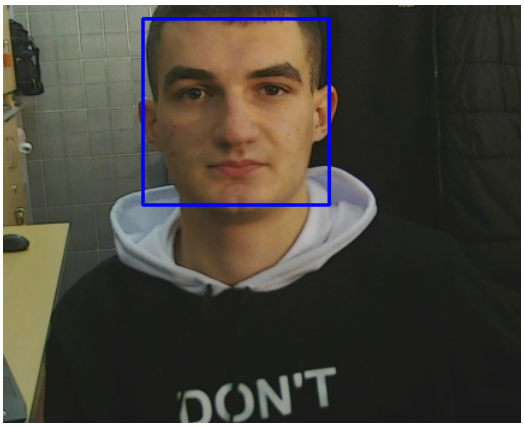


Abbildung 1.6: Output: Box auf Gesicht

1.2.3 Normalisierung

In dieser Kapitel wird ein sehr wichtiger Bereich, die sogenannte Normalisierung eines Bildes, genau erklärt.

Was ist Normalisierung?

Normalisierung in *Computer Vision* ist der Prozess der Bereitstellung von Daten, bei dem "falsche"Daten, die die Genauigkeit unseres Systems beschädigen können, korrigiert werden
Normalisierung heißt in unserem Fall Ausrichtung. Die Ausrichtung der Gesichter, die im Bild "falsch"positioniert sind.
Also entspricht Normalisierung dem Prozess in dem man zuerst die geometrische Struktur von Gesichtern in digitalen Bildern identifiziert und dann versucht eine maßgebliche

Ausrichtung des Gesichts basierend auf Skalierung und Rotation zu erhalten.

Methoden für Normalisierung

Es gibt viele Methoden mit denen man Bilder normalisieren könnte. Einige von denen basieren auf pre-definierten 3D-Modellen und transformieren dadurch die Eingabebilder, sodass die Gesichtsschlüsselpunkte der Eingabebild mit denen von dem 3D Modell übereinstimmen.
Die in dieser Diplomarbeit verwendete Methode, eine eher einfachere, verlässt sich nur auf die Gesichtspunkte selbst, um eine normalisierte Darstellung des Gesichts durch Affine- und Ähnlichkeits-Transformation zu erhalten. Diese Methode wurde deswegen implementiert, weil sie sehr effizient ist.

Warum Normalisierung?

Der Grund warum in dieser Arbeit Normalisierung von Daten durchgeführt wird, ist dass viele Gesichtserkennungsalgorithmen, einschließlich unserem, von der Anwendung dieser Ausrichtung sehr profitieren. Es wird dadurch die Präzision der Gesichtserkennung erhöht.

Implementation

Die hier verwendete Methode für die Normalisierung, wie oben kurz erwähnt wurde, wendet Ähnlichkeitstransformation bei zwei Paaren entsprechender Punkte an. Die Punkten sind die von Dlib extrahierten Gesichtsmerkmale, siehe Kapitel 1.2.5.

OpenCV benötigt in diesem Fall 3 Punkte zur Berechnung der Ähnlichkeitsmatrix. Wir nehmen somit als dritten Punkt, den dritten Punkt eines gleichseitigen Dreiecks mit diesen beiden gegebenen Punkten an.

Was eine Ähnlichkeitstransformation ist, lässt sich mathematisch beschreiben. Eine Ähnlichkeitstransformation ist eine Transformation wie z.b.: Reflexion, Rotation oder Translation.

Wenn eine Figur durch eine Ähnlichkeitstransformation transformiert wird, wird ein Bild erstellt, das der ursprünglichen Figur ähnlich ist. Mit anderen Worten, zwei Figuren sind ähnlich, wenn eine Ähnlichkeitstransformation die erste Figur zur zweiten Figur transformiert.

Ein solches Prinzip ist auch in unsere Methode eingesetzt worden. Es wird eine zweites, dem ersten Bild ähnliches Bild, erstellt.

```
1
2 def similarityTransformMat(initialPoints, destinationPoints):
3 ...
```

```
4 tform = cv2.estimateAffinePartial2D(np.array([initialPoints]),
5   np.array([destinationPoints]))
6 return tform[0]
7
```

Listing 1.2: Implementation Normalisierung

Im Code Abschnitt 1.2 wird die Methode gezeigt, die als Parameter die Anfangs- und die Zielpunkte nimmt und aus den beiden jeweils einen dritten Punkt berechnet.

Sie werden dann in Arrays gespeichert und aus denen wird die Ähnlichkeitstransform von den in OpenCV eingebetteter Funktion *cv2.estimateAffinePartial2D* berechnet.

Ausgabe ist eine 2D affine Transformation, also eine 2x3 Matrix, oder eine leere Matrix, wenn die Transformation nicht geschätzt werden konnte.
"Die Funktion schätzt eine optimale affine 2D-Transformation mit 4 Freiheitsgraden, die auf Kombinationen aus Translation, Rotation und gleichmäßiger Skalierung beschränkt sind. Verwendet den ausgewählten Algorithmus für eine robuste Schätzung."[4]

Die nächste Funktion macht die Ausrichtung des Gesicht im Bild. Es bekommt ein, von der Funktion *cv2.imread* eingelesenes Bild, die gewünschte Größe, und die Dlib Gesichtsmerkmale als Parameter.

```
1   def faceAlign(image, size, faceLandmarks):
2   (h, w) = size
3   initialPoints = []
4   destinationPoints = []
5
6   initialPoints = [faceLandmarks[36], faceLandmarks[45]]
7
8   destinationPoints = [(np.int(0.3*w), np.int(h/3)), (np.int(0.7*w),
9   np.int(h/3))]
10
11  similarityTransform = similarityTransformMat(initialPoints,
     destinationPoints)
12
13  faceAligned = np.zeros((image.shape), dtype=image.dtype)
14
15  faceAligned = cv2.warpAffine(image, similarityTransform, (w, h))
16
17  return faceAligned
18
19
```

Listing 1.3: Gesichtsausrichtung

Im Code Abschnitt 1.3 in Zeile **6** wird die Position der linken Ecke des linken Auges und der rechten Ecke des rechten Auges von dem Eingabebild genommen.
In Zeile **8** wird die Position der linken Ecke des linken Auges und der rechten Ecke des rechten Auges im ausgerichtete Bild berechnet.
In Zeile **10** und **11** wird die Ähnlichkeitstransformation durch die vorher erstellten Funktion *similarityTransformMat* berechnet.

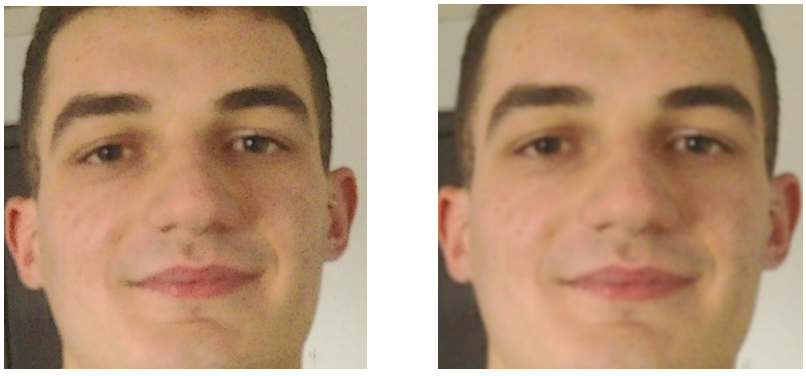
In Zeile **12** wird das ausgerichtete Gesicht in einem Tupel gespeichert.
Schließlich wird die Ähnlichkeitstransformation durch die Methode *cv2.warpAffine* angewendet.
Diese Methode bekommt das Tupel vom Bild, *similarityTransform*, und die Größen vom Bild als Parameter. Das Ergebnis wird dann von den Funktion zurückgegeben. (Zeile **16**)

Eine affine Transformation ist jede Transformation, die die Kollinearität (d. H. Alle auf einer Linie liegenden Punkte, die anfänglich nach der Transformation noch auf einer Linie liegen) und die Entfernungsverhältnisse (z. B. der Mittelpunkt eines Liniensegments bleibt der Mittelpunkt nach der Transformation) bewahrt. Eine affine Transformation wird auch als Affinität bezeichnet.[2]
Zusätzlich wurde noch eine Glättungsfilter angewendet. Dies geschieht durch die Funktion *cv.2filter2d*, die ein Bild mit dem Kernel faltet.
Sie bekommt folgende Parameter: *output* - Zielbild.
kernel - Faltungskern (oder vielmehr ein Korrelationskern), eine einkanalige Gleitkommamatrix.
ddepth - gewünschte Tiefe des Zielbildes.

Der Kernel ist auf Code abschnitt 1.4 genau sichtbar.

```
1  kernel = np.ones((5,5),np.float32)/25
2
3  output = cv2.filter2d(output,-1, kernel)
4
```

Listing 1.4: Glättungsfilter



(a) Nicht normalisiert (b) Normalisiert

Abbildung 1.7: Unterschied: normalisiert, nicht normalisiert

(a) Nicht normalisiert (b) Normalisiert

Abbildung 1.8: Unterschied: normalisiert, nicht normalisiert

Auf Abbildungen 1.8 und 1.7 sieht man ganz klar die Unterschiede die die Normalisierung in Bildern verursacht.

1.2.4 Zuschneiden von Gesichter

Nachdem Gesichter gefunden werden, muss man daraus eigene Bilder machen, die dasExtrahieren der Schlüsselpunkte erleichtern.

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

Listing 1.5: Code Abschnitt: Gesicht Zuschneiden

In Listing 1.5 sieht man folgendes:

nrFace bestimmt die Anzahl der Gesichter, die von den Kaskadenklassifikator gefunden wurden. Nur wenn dieser Wert größer als 0 ist soll das Programm weiterlaufen. Es wird durch jedes Gesicht iteriert.

Zeilen **1-6** berechnen das Zentrum vom neuen Bild und in Zeile **7** wird ein neues Bild mit den berechneten Parametern angelegt.
Mit imwrite wird das Bild gespeichert (Zeile **12**). Auf Abb.1.9 sieht man deutlich nur

das Gesicht.

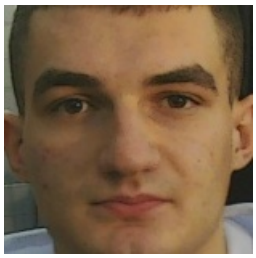


Abbildung 1.9: Output: Zugeschnittenes Bild

1.2.5 Gesichtsschlüsselpunkte Extraktion

Nun kommt ist es zu einem sehr wichtigen Punkt meines Teils der Diplomarbeit, die Extraktion Gesichtsschlüsselpunkte.
Dazu wurden die Gesichtsmerkmale *„Facial Landmarks“* von dlib verwendet.
Die Gesichtsmarkierungen werden verwendet, um Bereiche des Gesichts zu finden und darzustellen, wie zum Beispiel: die Augen, die Augenbrauen, die Nase, den Mund und den Kiefer.
Wie macht man das? Wie erkennt man Gesichtsmerkmale?
Das ist eine Teilmenge des Problems der Formvorhersage.
Bei einem vorgegebenen Eingabebild (und normalerweise einem ROI[1], die das interessierende Objekt angibt) versucht ein Formvorhersager, wichtige Punkte entlang der Form zu lokalisieren.
Dieser Formvorhersager, der im Dlib integriert ist wurde von Kazemi and Sullivan in ihrem Paper: *One Millisecond Face Alignment with an Ensemble of Regression Trees* entwickelt.[6]
Dieser Methode werden viele Trainingsdaten zur Verfügung gestellt, damit sie ein Kombination von Regressionskurven trainiert, um die Positionen der *Facial Landmarks* direkt aus den Pixelintensitäten selbst zu berechnen.[6]

Dadurch werden unsere gewünschten 68 Gesichtsmarkierungen mit hohen Vorhersagbarkeit extrahiert und als x, y Koordinaten weitergegeben. Zusätzlich ist es auch so, dass diese Koordinaten nicht von den Dimensionen des Bildes abhängig sind. Die Indizes der 68 Koordinaten sind in der Abb.1.10 dargestellt:

[1]Region of interest, Region von Interesse

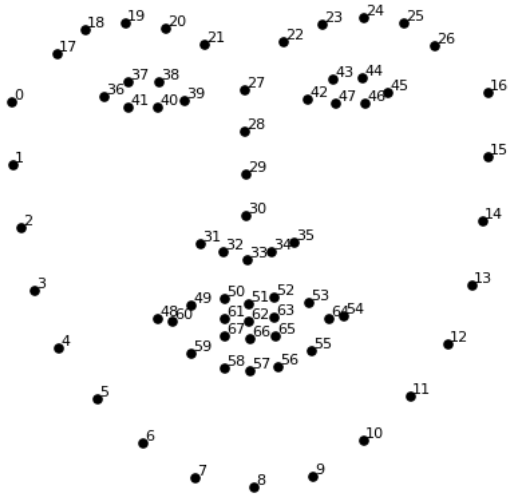


Abbildung 1.10: Gesichtsschlüsselpunkte [6]

Nachdem der Prediktor geladen wird, speichern wir die Gesichtsschlüsselpunkte in einem Formobjekt mit 68(x,y) Koordinaten.
Es wird hier nicht die *resize* Funktion verwendet, weil das Bild 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. Auf Abbildung 1.11 sieht man die extrahierten Daten von einem Gesicht.

0. 0.425 0.1359375
1. 0.42916666666666664 0.1625
2. 0.4375 0.190625
3. 0.44583333333333336 0.215625
4. 0.46041666666666664 0.2390625
5. 0.48333333333333334 0.259375
6. 0.5125 0.2734375
7. 0.5479166666666667 0.2859375
8. 0.5833333333333334 0.2875
9. 0.61875 0.2828125
10. 0.65 0.2703125
11. 0.675 0.253125
12. 0.6916666666666667 0.23125
13. 0.7 0.20625
14. 0.7041666666666667 0.1796875
15. 0.7104166666666667 0.153125
16. 0.7125 0.125
17. 0.44583333333333336 0.11875
18. 0.4625 0.103125
19. 0.4875 0.0984375
20. 0.5166666666666667 0.1
21. 0.5416666666666666 0.1078125
22. 0.5895833333333333 0.10625
23. 0.6166666666666667 0.0984375
24. 0.6458333333333334 0.09375
25. 0.6729166666666667 0.096875
26. 0.6916666666666667 0.1125
27. 0.56875 0.1265625
28. 0.5708333333333333 0.1453125
29. 0.5729166666666666 0.1640625
30. 0.5729166666666666 0.184375
31. 0.5479166666666667 0.196875
32. 0.5625 0.1984375
33. 0.5770833333333333 0.2
34. 0.5895833333333333 0.1984375
35. 0.6041666666666666 0.1953125
36. 0.47708333333333336 0.1328125
37. 0.49375 0.1265625
38. 0.5125 0.125
39. 0.53125 0.1328125
40. 0.5125 0.1375
41. 0.49375 0.1375
42. 0.6083333333333333 0.13125
43. 0.625 0.121875
44. 0.64375 0.121875
45. 0.6604166666666667 0.128125
46. 0.6458333333333334 0.1328125
47. 0.6270833333333333 0.134375
48. 0.525 0.2265625
49. 0.54375 0.2203125
50. 0.5625 0.2171875
51. 0.5770833333333333 0.21875
52. 0.5916666666666667 0.215625
53. 0.6104166666666667 0.21875
54. 0.6291666666666667 0.2234375
55. 0.6125 0.23125
56. 0.5958333333333333 0.2359375
57. 0.5791666666666667 0.2375
58. 0.5645833333333333 0.2359375
59. 0.5458333333333333 0.2328125
60. 0.5333333333333333 0.2265625
61. 0.5625 0.225
62. 0.5791666666666667 0.2265625
63. 0.59375 0.225
64. 0.6229166666666667 0.2234375
65. 0.59375 0.2234375
66. 0.5791666666666667 0.225
67. 0.5645833333333333 0.2234375

Abbildung 1.11: Gesichtsdaten

Die Merkmale werden in einer numpy Array gespeichert für die Speicherung in der Datenbank.

1.3 Herausforderungen, Probleme und deren Lösung

Die größte Herausforderung lag bei der Planung der Software und den Methoden die zum Extrahieren von den Gesichtsschlüsselpunkten dienten.
Es hat mich viel Zeit gekostet, bis ich eine geeignete Lösung gefunden habe und das hat mir 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 ihnen gefunden werden. Also ist viel Zeit beim Installieren und Konfigurieren investiert worden. Ein Grund dafür ist die mangelnde Erfahrung mit den verwendeten Technologien.

Viele Sachen, wie z.b.: die Einteilung der Arbeit, die genaue Spezifikationen, die sehr grobe Planung, war am Beginn auch wegen den Kommunikationslücken unklar.

Als das Projekt weiter fortschritt, hat sich der Bedarf an Genauigkeit und Präzision auch enorm gesteigert. Es hat immer wieder Fälle beim Finden von Gesichter gegeben, in denen die fertig gestellten neuronale Netze nicht so gut funktioniert haben .

Es war keine gute Idee sich nur von einem Prädiktor abhängig zu machen. Es war deutlich dass, die Einsetzung von mehreren neuronalen Netzen Erhöhung der Treffsicherheit ergeben hätte.

Eine einfache Normalisierung genügte auch nicht die Umwandlung der Bilder in Graustufe), sondern es war eine komplexere Lösung dafür notwendig.
Bis das funktioniert hat, hat es viel Zeit und Mühe gekostet. Dafür waren auch relativ fortgeschrittene mathematische Kenntnisse nötig. Dabei habe ich viel Try and Error angewendet.

Es hat auch Schwierigkeiten beim Qualitätsmanagement gegeben. Die Erfahrung fehlte, somit waren gewisse Maßstäbe und Standarts auch nicht bekannt.

1.3.1 Lösungen

Während der Arbeit habe ich viel recherchiert und mich genau über alles Mögliche informiert.

Es wurde viel herumprobiert und experimentiert. Es wurde auch sehr viel getestet, damit die Ziele auch qualitativ erfüllt wurden.
Die Kommunikation hat sich steigendem mit Bedarf über die Zeit stark verbessert und dadurch sind auch die Unklarheiten abgeklärt worden.

1.4 Qualitätssicherung, Controlling

Die Qualität wurde durch verschiedene Methoden gesichert. Eine von denen war die Methode 5xWarum.

„Fünf warum“ ist eine iterative Methode, die Fragen als Basis hat und die Beziehungen zwischen Fragen und Problemen.
Es geht hier um die Verschachtelung der Ursachen und das Herausfinden von ihnen durch iterative Fragetechnik, da viele Probleme nicht nur eine einzige Ursache haben. Die Methode ruft jedes Mal eine andere Folge von Fragen auf.[3]

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[2].“

1. **Was ist überhaupt ein „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 passiert das, 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 zurückgibt nicht in kp gespeichert werden.

   ODER

   Das Paket in dem die FAST Algorithmus drinnen ist ist nicht (richtig) installiert worden. Warum? Alle nötigen Pakete wurden in einer gesamten Installation geholt und FAST war nicht da.

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

   Die Daten die fast.detect zurückgibt, 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?**

[2]Features from Accelerated Segment Test

Es wurde nicht manuell gegeben weil, die Methode fast.detect nicht gut recherchiert wurde. Man sollte mehr darüber in die Dokumentation nachschauen.

Konklusion
Durch die 5x Warum Methode ist es zu den Ergebnis gekommen: Es musste ja mehr untersucht werden bevor man eine solche Methode implementiert. Die „5xWarum“ Methode hat dabei geholfen, dass die eigentliche Ursache des Problems herausgefunden wurde.
Die nächste Schritte sind: entweder eine andere Methode, die schon installiert ist, verwenden oder die benötigten Pakete für FAST manuell installieren.

1.5 Ergebnisse

Ich habe ungefähr 180 Stunden Zeit ins gesamt für die Diplomarbeit investiert und folgendes erreicht.

- Gesichtserkennung und das Zuschneiden von Gesichter wurden fertig implementiert.
- Bilder sind normalisiert worden.
- Gesichtsschlüsselpunkte wurden extrahiert.

1.6 Evaluierung und Resümee

In diesem Kapitel geht es um die Reflexion, Evaluierung der Ergebnisse. Was von der Planung abgewichen hat usw.

1.6.1 Planung vs. Realisierung

Es wurde am Anfang so gedacht, dass die Klassifikatoren und die neuronale Netze alle selbst gemacht werden müssten. Über die Zeit wurde mir klar, dass so was zu kompliziert und fast unmöglich war. Die Genauigkeit der Gesichtserkennungsalgorithmen könnte nie erreicht werden, wenn es vom Scratch erarbeitet wurde. Geplant waren auch gewisse Ziele, die durch den Diplomarbeit-Mitgliedern gemischt worden waren. Z.b.: Ein Ziel von mir war die Vektorumwandlung, die später so geschätzt wurde, dass es eigentlich zu Rei's Teil gehörte.

1.6.2 Lessons Learned

Im technischen Aspekt habe ich viel gelernt. Diese Diplomarbeit hat viele Kenntnisse mit sich gebracht. Zahlreiche neue Bibliotheken wurden verwendet. Eine ganz unbekannte Framework ist kennen gelernt worden. Ich habe viel Erfahrung über Computer

Vision bekommen. Da meine Aufgaben zum Teil sehr anspruchsvoll waren, habe ich viel Zeit und Mühe für die Realisierung investiert. Soft Skills wurden auch dazugewonnen. Ich habe erlernt wie wichtig Teamwork und Kommunikation ist. Wenn ich die ganze Diplomarbeit noch einmal machen würde, wäre vieles anders sein.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

Aus dem Netz

[1] *Git*. URL: https://git-scm.com/docs.

[2] Eric W. Weisstein. *Affine Transformation*. MathWorld. URL: http://mathworld.wolfram.com/AffineTransformation.html.

Der ganze Rest

[3] *5x Warum*. https://www.quality.de/lexikon/5xwarum/. [Online; accessed 2019]. 2017.

[4] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[5] EDUCBA. *Differences Between Linux vs Windows*. https://www.educba.com/linux-vs-windows/. [Online; accessed 2019]. 2017.

[6] Vahid Kazemi und Josephine Sullivan. "One millisecond face alignment with an ensemble of regression trees". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014), S. 1867–1874.

[7] Davis E. King. "Dlib-ml: A Machine Learning Toolkit". In: *Journal of Machine Learning Research* 10 (2009), S. 1755–1758.

[8] Mindfire solutions. *Python: 7 Important Reasons Why You Should Use Python*. https://medium.com/@mindfiresolutions.usa/python-7-important-reasons-why-you-should-use-python-5801a98a0d0b/. [Online; accessed 2019]. 3/10/2017.

[9] Paul Viola und Michael Jones. "Robust Real-time Object Detection". In: *International Journal of Computer Vision*. 2001.

[10] Pauli Virtanen u. a. "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python". In: *arXiv e-prints*, arXiv:1907.10121 (Juli 2019), arXiv:1907.10121. arXiv: 1907.10121 [cs.MS].