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 Projektes ist der Bildverarbeitungsteil.
In dem folgenden Abschnitt der Ausarbeitung wird es genau erklärt und beschrieben wie diese Aufgabe gelöst wurde und was dafür verwendet wurde.
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. [5]

Zusätzlich ist Raspberry Pi als Backup System verwendet worden das auch mit einem lauffähigen Linux Betriebssystem (Debian) funktioniert.
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 allgemeine Programmiersprache auf hohem Niveau. 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 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.
[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 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 das 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 eine 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.
[4]

SciPy hat sich nützlich, beim Lösen von gewisse mathematische Angabenund bei zusätzliche Installationen von Bibliotheken zu helfen.
SSciPy ist eine kostenlose und Open-Source-Python-Bibliothek für wissenschaftliches und technisches Rechnen."[10]
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.

Dlib ist ein so genanntes Toolkit, das Algorithmen für Machine Learning und Tools

Abbildung 1.1: dlib logo [7]

zum Erstellen komplexer Software zur Lösung von der realen Welt getauchte Probleme, beinhaltet.
Es wird vielseitig eingesetzt, in Robotik, Mobilgeräte und unter anderem in Computer Vision.[7]
Es wurde prinzipiell bei der Extrahierung der so genannten „Facial Landmarks“ gebraucht.

1.2 Technische Lösungen

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

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

Aus diesem Grund musste die Arbeitsteilung gut geregelt werden, damit jedes Teammitglied sich auf einen 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 fällt.

Die Planung der betreffenden Bereiche der Projektarbeit hat 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 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

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

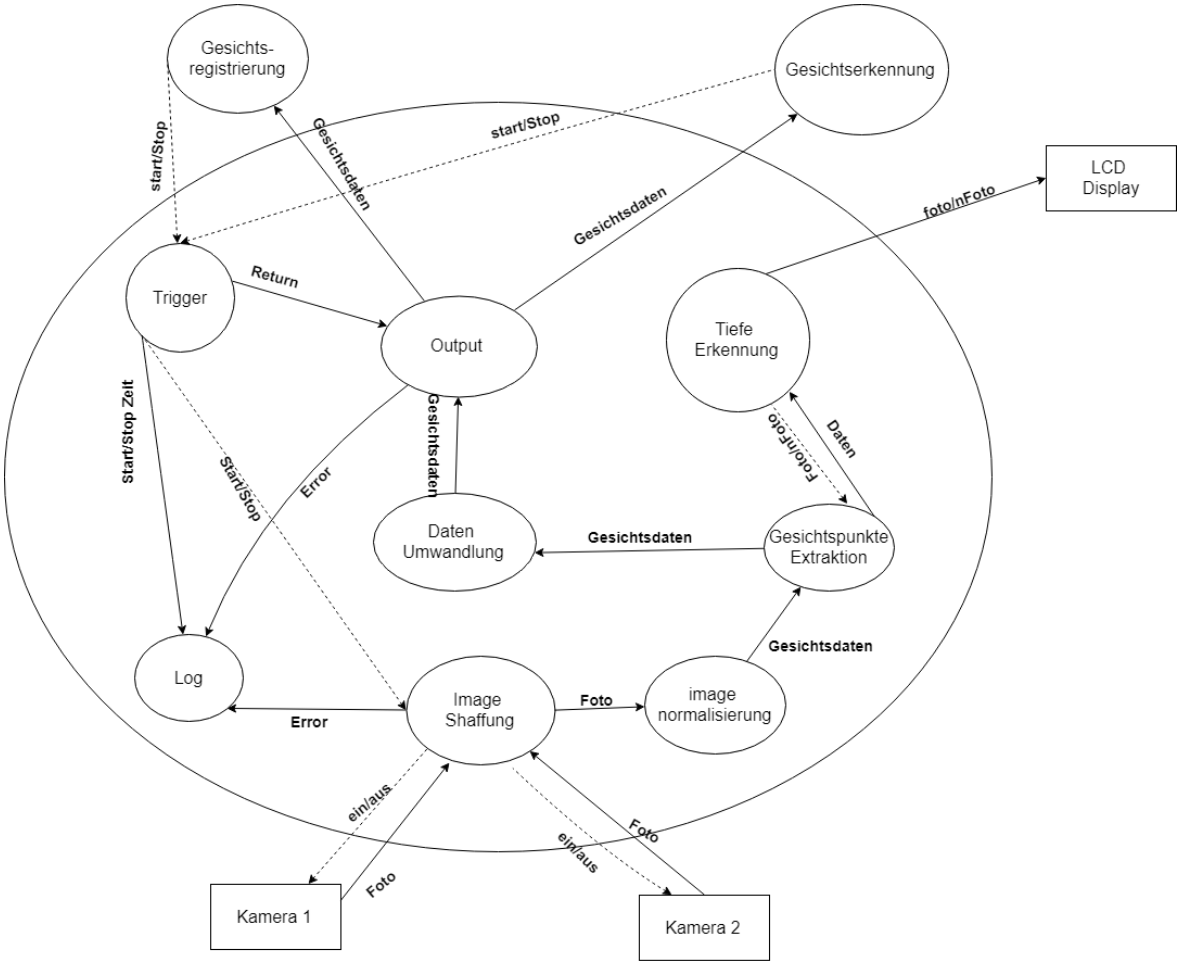


Abbildung 1.2: Bildverarbeitung Structed Design

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

Die Schnittstellen von außen sind das Gesichtserkennungsmodul und die Gesichtsregistrierungsmodul.
Diese schicken eine Anforderung an den Trigger der dann die Bildaufnahme Modul initialisiert. Unabhängig von welchen von diesen beiden Schnittstellen die Anfrage kommt, macht das Bildaufnahme Modul 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. das „Image Prozessing“.
Wenn ein Bild zu klein ist wird die Größe angepasst, wenn ein Gesicht verdreht ist wird es gerade rotiert. Mehr dazu wird in dem Normalisierung Abschnitt erklärt.
Danach folgt die Erkennung von Gesichtern, das Ausschneiden von denen und die Extrahierung der Gesischtsschlüsselpunkte.
Sie werden dann zur Abgleich oder Registrierung je nach Bedarf, bereitet und geschickt.

Hinweis: 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 Gesichtsdetektion

Das erste Ereignis, dass erfüllt worden war bei der Gesichtserkennung ist das Finden von Gesichtern, also die Feststellung, wo sich die Gesichter im Bild befinden.
Dafür ist der „Haar Cascade Classifiers“ vorgefertigter Klassifikator benutzt 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 haben.

Die Arbeit heißt „Rapid Object Detection using a Boosted Cascade of Simple Features”. [9]

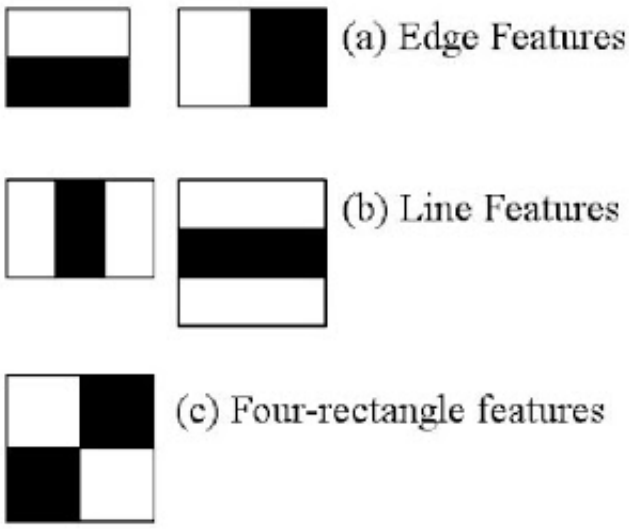


Abbildung 1.3: Haar Features[9]

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 die Haar Merkmale benutzt. Auf Abbildung 1.3 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, erhält.
Diese Summen berechnet man durch integrale Bilder, die zur Vereinfachung zu Summenberechnungen dient.

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

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

Unten ist ein Code Abschnitt der Gesichtsdetektion dargestellt.

```
face_cascade =
  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)
```

Listing 1.1: Kern Code für Gesichtsdetektion

Es wurde die trainierte Klassifikator-XML-Datei (haarcascade_frontalface_default.xml) die im GitHub-Repository von OpenCV zu befinden ist, herunterladen.
In den Zeilen **1** und **2** wurde sie geladen und in die Variable face_cascade gespeichert.
In die Zeile **4** wird das Bild vom den Funktion *imread* gelesen.
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 bf erkennt die Gesichter im Bild. Dieser Funktion ist sehr wichtig und braucht 3 Argumente – das Eingabebild, der Skalierungsfaktor (scaleFactor) und *minNeighbours*.

scaleFactor gibt die an, um wie viel wird das Bild vergrößert bzw. verkleinert. Und *minNeighbours* gibt an, wie viele Nachbarn jedes Kandidatenrechteck haben muss, um es beizubehalten. 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, ist hier der Wert 4 eingesetzt.

In Zeile 10 beginnend, wird es durch die Gesichter *faces* iteriert. x und y entsprechen die Koordinaten vom Bild, w, h bezeichnen die Breite und Höhe *width*, *height*.

Schließlich wird in Zeile **12** mit dem Funktion *cv2.rectangle* ein Rahmen auf dem Gesicht gezeichnet.
Der Funktion bekommt diese Parameter: *image* für die Bildeingabe, x, y ergeben den Startpunkt, w, h den Endpunkt, die Farbe und die Dicke der Rahmen werden in den 2 letzten Parameter gegeben.

Dieser Stammcode wurde so umgeändert, dass es in der Lage ist, durch mehrere *Predictors* bzw. Klassifikatoren, die Genauigkeit der Gesichtsdetektion zu erhöhen.

Auf Abbildung 1.4 sieht man deutlich das erkannte Gesicht.

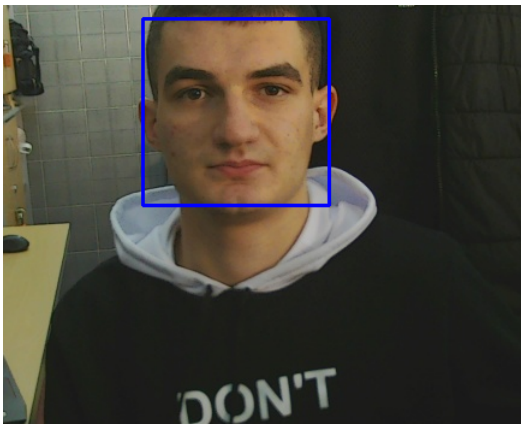


Abbildung 1.4: Output: Box auf Gesicht

1.2.3 Normalisierung

In dieser Kapitel wird ein sehr wichtiger Bereich, der sogenannte Normalisierung der Daten, genau erklärt.

Was ist Normalisierung?

Normalisierung in *Computer Vision* ist der Prozess der Bereitstellung von Daten, die Korrektur der "falschen"Daten, die die Genauigkeit unseres Systems beschädigen können.
Normalisierung heißt anders auch Ausrichtung. Die Ausrichtung der Gesichter, die im Bild "falsch"positioniert sind.
Also entspricht Normalisierung den Prozess in dem man zuerst die geometrische Struktur von Gesichtern in digitalen Bildern identifiziert, das Versuch, eine maßgebliche Ausrichtung des Gesichts basierend auf Skalierung und Rotation zu erhalten.

Methoden für Normalisierung

Es gibt viele Methoden mit dem man Bilder normalisieren könnte. Einige von denen basieren auf irgendwelche pre-definierte 3D-Modelle und transformieren dadurch die Eingabe Bilder, dass die Gesichtsschlüsselpunkte der Eingabe Bild die von dem 3D Modell übereinstimmen.
Die in dieser Diplomarbeit verwendete Methode, eine eher einfachere, verlässt sich nur auf die Gesichtspunkte selbst, um eine normalisierte Rotation und Skalen Darstellung des Gesichts durch Affine und Ähnlichkeit Transformation zu erhalten. Diese Methode wurde deswegen implementiert, weil es sehr effizient ist und genau gut zu dieser Diplomarbeit passt.

Warum Normalisierung?

Der Grund warum in dieser Arbeit Normalisierung von Daten durchgeführt wird, liegt genau in der Tatsache, dass viele Gesichtserkennungsalgorithmen einschließlich unserer,

von der Anwendung von dieser Ausrichtung viel profitieren können. Es wird dadurch der Präzision der Gesichtserkennung erhöht.

Implementation

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

OpenCV benötigt aber 3 Punkte zur Berechnung der Ähnlichkeitsmatrix. Wir nehmen somit als dritten Punkt, der dritte Punkt des gleichseitigen Dreiecks mit diesen beiden gegebenen Punkten an.

Was eine Ähnlichkeitstransformation lässt die Mathematik Theoremen beschreiben. Eine Ähnlichkeitstransformation ist eine der mehreren starre Transformationen wie z.b.: Reflexion, Rotation oder Translation gefolgt von einer Dilatation.

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 trägt.

Solches Prinzip ist auch in unsere Methode eingesetzt. Es wird eine zweite, dem ersten Bild ähnlichen Figur, erstellt.

```
1
2 def similarityTransformMat(initialPoints, destinationPoints):
3   ...
4   tform = cv2.estimateAffinePartial2D(np.array([initialPoints]), np.
     array([destinationPoints]))
5   return tform[0]
6
```

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 doch die Ähnlichkeitstransformation von den in OpenCV eingebetteter Funktion *cv2.estimateAffinePartial2D* berechnet.

Ausgabe ist eine 2D affine Transformation, also eine 2x3 Matrix oder 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, genau die wichtigste, macht die Ausrichtung der Gesicht im Bild. Es bekommt ein, von Funktion *cv2.imread* gelesenes 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),
      np.int(h/3))]
9
10   similarityTransform = similarityTransformMat(initialPoints,
      destinationPoints)
11
12   faceAligned = np.zeros((image.shape), dtype=image.dtype)
13
14   faceAligned = cv2.warpAffine(image, similarityTransform, (w, h))
15
16   return faceAligned
```

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 den 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 eben die Ähnlichkeitstransformation durch den vorher erstellten Funktion *similarityTransformMat* berechnet.

In Zeile **12** wird das ausgerichtete Gesicht in einem Tupel gespeichert.

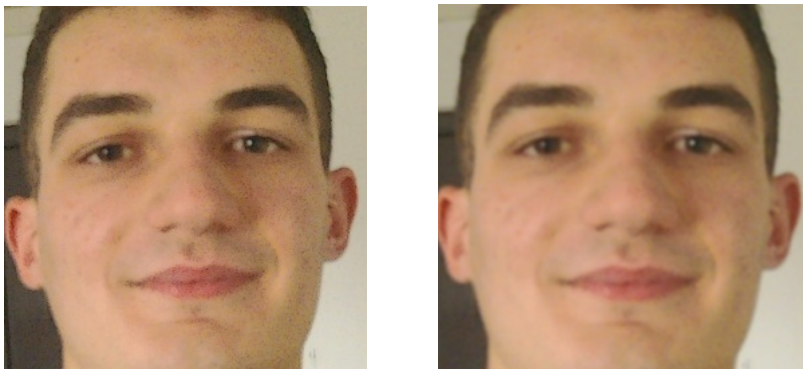
Schließlich wird die Ähnlichkeitstransformation durch die Methode *cv2.warpAffine* angewendet.
Diese Methode bekommt den Tupel vom Bild, *similarityTransform*, und die Größen von Bild als Parameter. Das wird dann vom Funktion zurückgegeben. (Zeile **16**)

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 als Parameter ein Bild bekommt und der die Daten zur Filter. 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)
```

Listing 1.4: Glättungsfilter



(a) Nicht normalisiert (b) Normalisiert

Abbildung 1.5: Unterschied: normalisiert, nicht normalisiert

(a) Nicht normalisiert (b) Normalisiert

Abbildung 1.6: Unterschied: normalisiert, nicht normalisiert .2

Auf Abbildungen 1.6 und 1.5 sieht man ganz klar die Unterschiede die die Einsetzung der Normalisierung in Bilder verursacht.

1.2.4 Zuschneiden von Gesichter

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

```
if nrFace > 0:
  for face in faces:
    for(x, y, w, h) in faces:
      r = max(w, h) /2
      centerx = x + w /2
      centery = y + h /2
      nx = int(centerx - r)
      ny = int(centery - r)
      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

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

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 (Zeile **12**). Auf Abb.1.7 sieht man deutlich nur das geschnittene Gesicht.

Abbildung 1.7: Output: Abgeschnittenes Gesicht

1.2.5 Gesichtsschlüsselpunkte Extraktion

Nun kommt ist es zu dem wichtigsten Punkt meines Teils der Diplomarbeit, die Gesichtsschlüsselpunkte Extraktion.

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 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.

[1]Region of interest, Region von Interesse

Dieser Methode werden viele Trainingsdaten hinzugefügt womit es ein Kombination von Regressionskurven trainiert wird um die Positionen der *Facial Landmarks* direkt aus den Pixelintensitäten selbst zu beurteilen.[6]

Dadurch werden unsere gewünschten 68 Gesichtsmarkierungen mit hohen Vorhersagbarkeit extrahiert und als x, y Koordinaten weitergegeben. Zusätzlich ist es auch so gemacht worden, dass diese gefundene Koordinaten den Dimensionen vom Bild nicht abhängig sind. Die Indizes der 68 Koordinaten sind in der Abb.1.8 dargestellt:

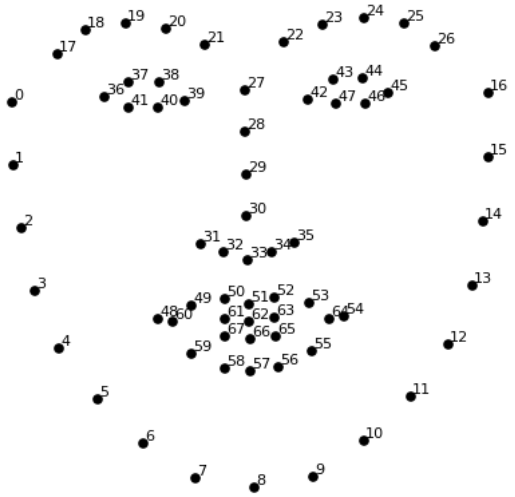


Abbildung 1.8: Gesichtsschlüsselpunkte [6]

Nachdem wir der „Predictor“ geladen haben speichern wir sie in einem Formobjekt mit 68(x,y) Koordinaten.
Es wird hier nicht die resize Funktion 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.

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 1.9: Gesichtsdaten

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

1.3 Herausforderungen, Probleme und deren Lösung

Die größte Herausforderung lag bei der Planung von der Software und die Methoden die zum Extrahieren von den Gesichtsschlüsselpunkten dienten.
Es hat mich viel Zeit gekostet 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 ihnen gefunden werden. Also ist viel Zeit beim Installieren und Konfigurieren investiert worden. Ein Grund dafür ist die mangelnde Erfahrung mit den neuen Technologien.

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

Als das Projekt weiter entwickelte, hat die Frage nach Genauigkeit und Präzision auch enorm gestiegen. Es hat immer wieder Fälle gegeben in dem die fertig gestellten neuronale Netze nicht so gut funktioniert haben beim Finden von Gesichter.

Es war keine gute Idee, nur von einem Prädiktor *Predictor* sich abhängig zu machen. Herausforderung war es deutlich, die Einsetzung von mehreren neuronalen Netzen und die Erhöhung der Treffsicherheit.

Eine einfache Normalisierung genügte auch nicht (die Umwandlung der Bilder in Graustufe), sondern es war eine komplexere Lösung dafür viel notwendiger und brauchbar. Bis das fertig geschrieben wurde, bis es richtig gut funktionierte, hat es viel Zeit und Bemühung gekostet. Dafür waren auch relativ fortgeschrittene mathematische Kenntnisse verlangt. Dabei habe ich viel "Try und Errorëingewendet.

Es hat auch Schwierigkeiten bei der Verwaltung von Qualität gegeben. Die Erfahrung fehlte, somit waren gewisse Maßstäbe, Standarte 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 mit Bedarf während der 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 die Ursachen und die Probleme. Es geht hier um die Verschachtelung der Ursachen und das Herausfinden von denen 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

[2]Features from Accelerated Segment Test

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 ergibt 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 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 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 haben dabei geholfen dass die eigentliche Ursache des Problems herausgefunden worden ist.
   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.
Gesichtsdetektion und das Zuschneiden von Gesichter wurden fertig implementiert.
Bilder sind ausführlich normalisiert worden.
Gesichtsschlüsselpunkte wurden extrahiert.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

Aus dem Netz

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

[2] Eric 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].