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. [4]

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.
[7]

"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.
[3]

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."[9]
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 [6]

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.[6]
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:

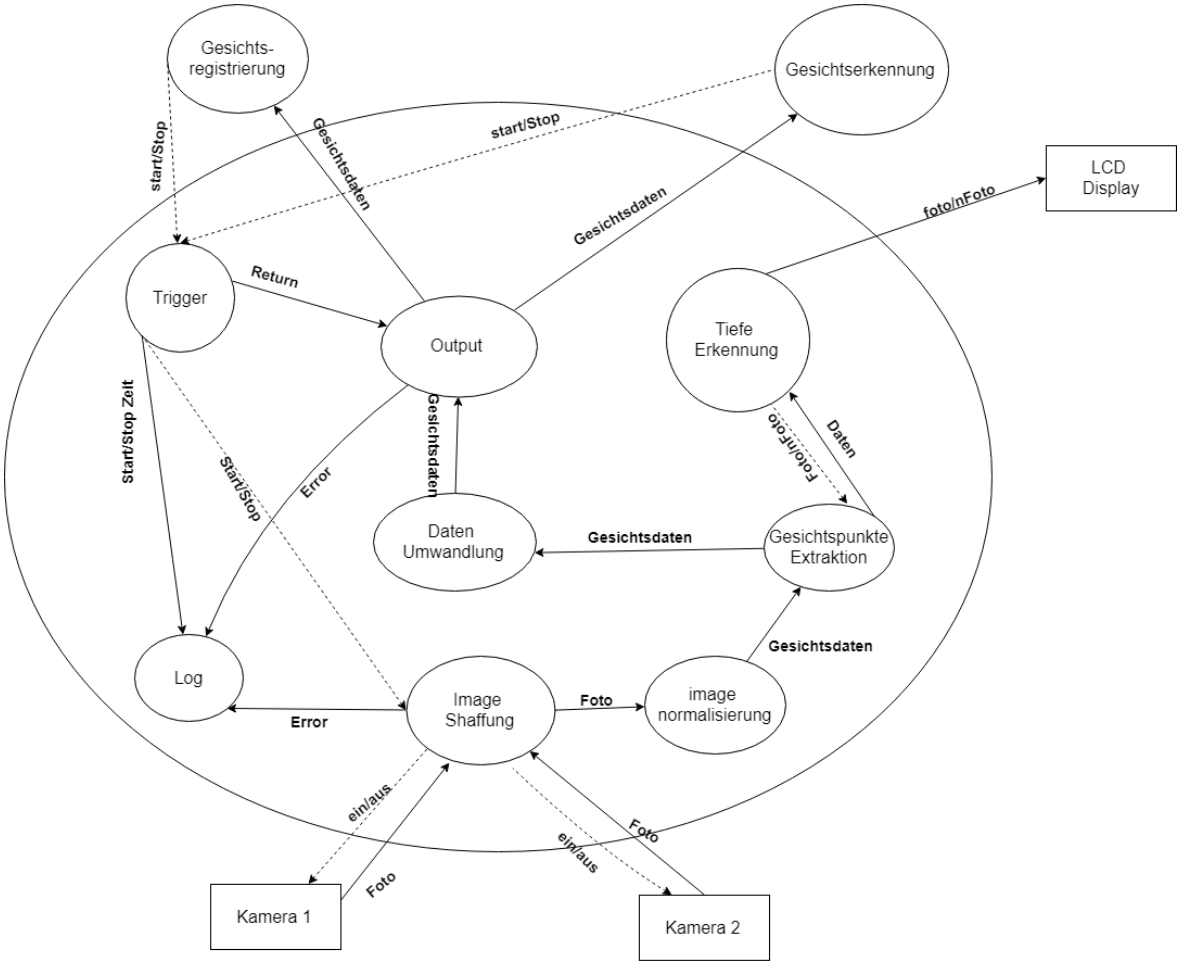

Abbildung 1.2: Bildverarbeitung Structed Design

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 werden muss bei der Gesichtserkennung ist das Finden von Gesichtern, also die Feststellung, wo sich die Gesichter im Bild befinden.
Dafür ist 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 haben.

Die Arbeit heißt „Rapid Object Detection using a Boosted Cascade of Simple Features”. [8]

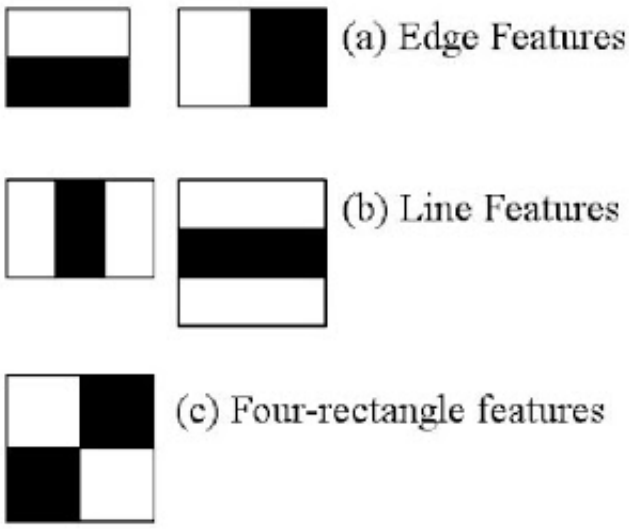

Abbildung 1.3: Haar Features[8]

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

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

```
//haar cascade classifier laden
1       face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_def
```

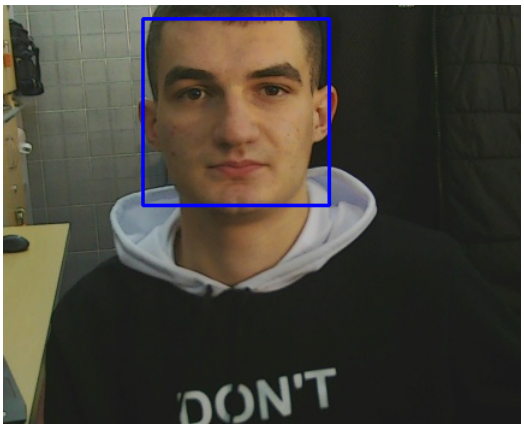

Abbildung 1.4: Box auf Gesicht

```
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, D
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

Die Funktionalität des Code ist genau auf Abb.1.4 ersichtlich.

1.2.3 Normalisierung

1.2.4 Gesichtsdetektion und Zuschneiden

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

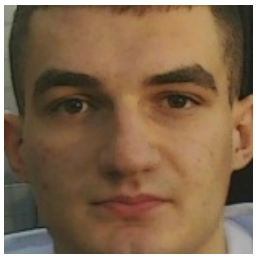

Abbildung 1.5: Abgeschnittenes Gesicht

nrFace bestimmt die Anzahl von Gesichter, die von den cascade classifier gefunden wurden. Nur wenn dieser Wert 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.1.5 sieht man deutlich nur das Gesicht.

1.2.5 Gesichtsschlüsselpunkte Extraktion

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

Dazu wurden die „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 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.[5]

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.[5]

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. dargestellt:.

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

1.2.6 Zusammenfassung

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 waren am Beginn auch unklar wegen den Kommunikationslücken und auch die sehr grobe Planung war problematisch.

[1] Region of interest

Lösungen:

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

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. Es geht hier um die Verschachtelung der Ursachen und die Herausfindung 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. [2] 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 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.

[2]Features from Accelerated Segment Test

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 Ergebnise

Ich habe ungefähr 150 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

Aus dem Netz

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

Der ganze Rest

[2] *5x Warum*. https://www.quality.de/lexikon/5xwarum/. [Online; accessed 2019]. 2017.

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

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

[5] 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.

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

[7] 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.

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

[9] 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].