Hochschule für Technik und Wirtschaft
Seminar: Wissensmanagement WS 22/23
Dozierende: Sara Reichert, Dr. Daniel Wessolek

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Dokumentation zum Projekt:

Feel Safe with ESP-EYE

Ataullah Shinwari
Christian Willmann
Timon Gödeke
Tobias Schultze

Berlin, den 26.03.2023

Inhaltsverzeichnis

1. Motivation

Laut der Gewerkschaft der Polizei wurden in Deutschland im Jahr 2015 etwa 167.000 Wohnungs- und Hauseinbrüche registriert. 2021 lag die Zahl der Einbrüche weitaus tiefer, bei etwa 54.000. Doch auch durch diese Anzahl konnte 2021 ein Schaden von ca. 180 Millionen Euro verzeichnet werden. Dieser Rückgang innerhalb der Einbruchskriminalität ist vor allem auf die Möglichkeit und Umsetzung der Integration von technischen Sicherheitsvorkehrungen zurückzuführen. Zudem konnte hierdurch ebenfalls das Scheitern bei Einbrüchen von 2015 ca. 42% auf 2021 49% erhöht werden. Auch die polizeiliche Aufklärungsquote konnte von 15% (2015) auf 19% (2021) gesteigert werden. Ein großes Potential ist jedoch trotz der Erhöhung weiterhin zu verzeichnen. (vgl. BKA, 2022)
Eine Möglichkeit, dieses Potential auszuschöpfen, ist die Installation von Videoüberwachungssystemen. Jedoch sind solche Sicherheitssysteme oft sehr kostspielig und dementsprechend nicht flächendeckend einsetzbar. Zudem werden die Daten oft auf den Servern der Anbieter abgespeichert, wodurch eine Weiterverwendung privater Inhalte nicht auszuschließen ist. Aufgrund der herrschenden Aspekte steigt das Verlangen einiger Verbraucher*Innen nach kostengünstigeren sowie datenschutzfreundlichen Alternativen. (vgl. Konopka, 2014)
Durch diese Motivation wird innerhalb des vorliegenden Projekts versucht, durch den Einsatz von Development Boards, wie dem Arduino Nano 33 BLE und dem ESP-EYE sowie der dazu passenden AI- und IOT-Software, eine Alternative zu schaffen. Die Umsetzung soll dabei möglichst kostengünstig und im lokalen Netzwerk erfolgen, so dass ein weitreichender Zugang geschaffen wird und möglichst keine Daten an Drittanbieter weitergegeben werden.

Die folgenden Kapitel verdeutlichen den Use Case des Projekts und geben den Leser*Innen eine intuitive Aufbau-/Nachbauanleitung an die Hand, wie eine solche Alternative umgesetzt werden kann.

2. Use Case

Für den Start wird das System individuell von den Nutzer*Innen eingeschaltet. Im besten Fall ist das System über die gesamte Zeit 24/7 eingeschaltet und einsatzbereit. Nach dem Einschalten wird der Eingang des Grundstücks (oder der Wohnung etc.) durch das System überwacht. Hierfür bleibt die Kamera dauerhaft eingeschaltet und steht mit dem Webinterface im drahtlosen lokalen Netzwerk (WLAN) über HTTP-Requests in Kontakt. Betritt nun eine Person das Grundstück und gerät in den Aufnahmebereich der Kamera, wird pro Sekunde, in der sich die Person in dem Bereich aufhält, ein Bild gemacht. Dieses Bild wird im Netzwerk an das jeweilige Endgerät direkt weitergeleitet und an einem individuell einstellbaren Ort gespeichert. Zusätzlich können Personen, welche sich öfter auf dem Grundstück aufhalten, ihr Gesicht im System speichern, so dass die Kamera keine Aufnahmen von ihnen macht, wenn sie sich im Aufnahmebereich befinden sollten.
Der Use Case teilt sich somit in zwei übergeordnete Fragmente auf. Das erste ist die Face Detection, in welcher lediglich jede Art von Gesichtern erkannt und in Form von Aufnahmen abgespeichert werden soll. Das zweite ist die Face Recognition, in welcher das erste Fragment um die Einspeicherung registrierter Personen erweitert wird.

Das Use Case Diagramm auf der rechten Seite verdeutlicht den Ablauf beider Teile noch einmal visuell.

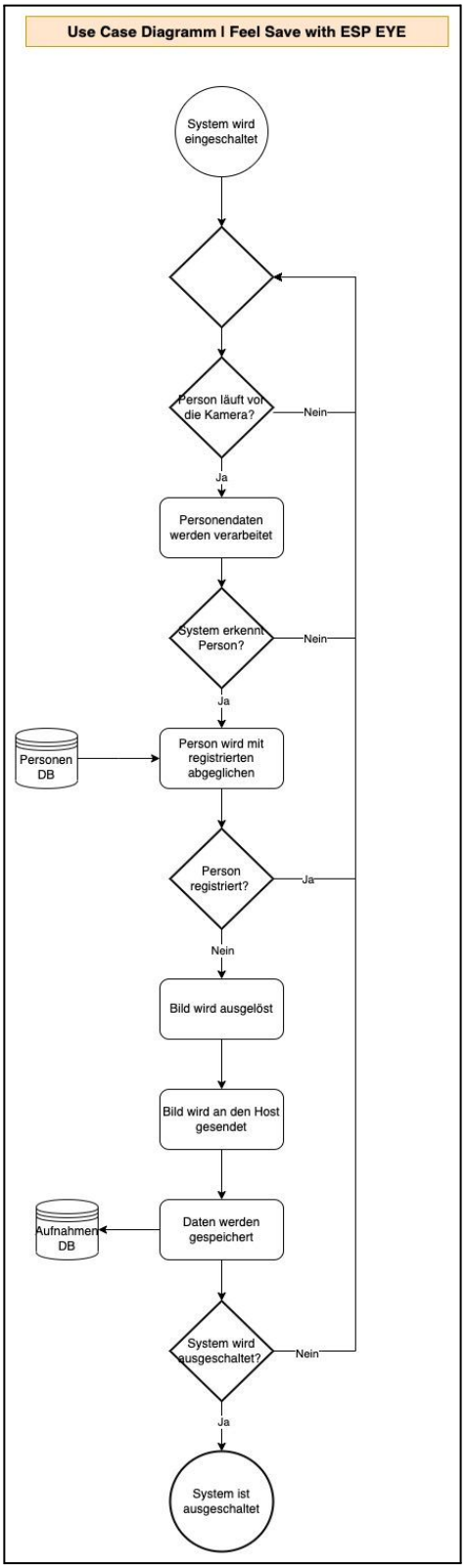

Abb. 1: Use Case Diagramm (Eigene Darstellung)

3. Umsetzung

Im folgenden Abschnitt wird die Umsetzung des Projekts beschrieben. Hierbei werden zwei unterschiedliche Vorgehensweisen dokumentiert. Die erste Variante umfasst die Umsetzung mit dem Arduino BLE 33 Nano und die zweite die Umsetzung mit dem ESP-EYE. Vorab sei gesagt, dass lediglich die Variante mit dem ESP-EYE den oberen Use Case ganzheitlich umsetzen konnte.
Für beide Varianten wird jeweils die benötigte Hard- und Software, der Aufbau/Ablauf, das Ergebnis, die Pro/Cons und das Troubleshooting aufgezeigt.

3.1 Arduino Nano 33 BLE

Der folgende Abschnitt behandelt die Dokumentation über die Umsetzung von Projektteilen mit dem Arduino Nano 33 BLE.

3.1.1 Hard- und Software

Bei der Umsetzung mit dem Arduino werden folgende Hardwarekomponenten benötigt:

- Windows/Linux/Mac PC (innerhalb dieses Projekts wurde mit einem Mac gearbeitet)
- Arduino Nano 33 BLE (Sense wird nicht zwingend benötigt)
- Arduino Tiny Machine Learning Shield
- OV7675 Kamera
- Micro-USB Kabel

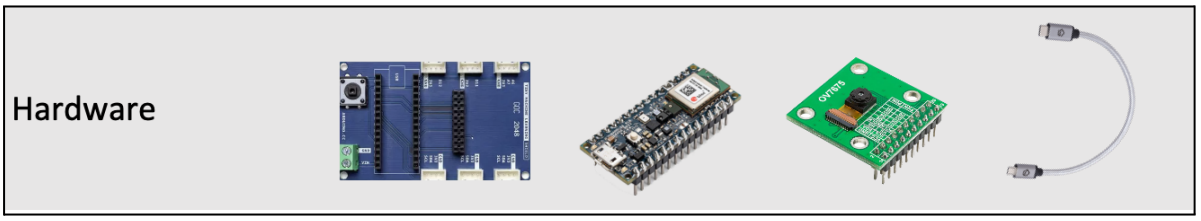

Abb. 2: Arduino Hardwarekomponenten (Eigene Darstellung)

Zusätzlich wird folgende Software benötigt:

- Arduino IDE
- Edge Impulse

Abb. 3: Arduino Softwarekomponenten (Eigene Darstellung)

3.1.2 Aufbau/Ablauf

Für den Aufbau werden folgende Installationen benötigt:

- Arduino IDE
- Arduino CLI
- Edge Impulse CLI
    - Python 3
    - Node.js (mind. v14)

Detaillierte Guides für die Installation über Windows, Mac oder Linux werden über den Link (vgl. Edge Impulse, 2023a) bereitgestellt.

Nach den benötigten Installationen muss der Arduino Nano 33 BLE bei Edge Impulse als Gerät hinzugefügt werden. Hierbei wird der Arduino per doppeltem Klicken auf den Button geflasht und anschließend mit den bereitgestellten Anweisungen über die/das Kommandozeile/Terminal registriert. Wird etwas anderes auf den Arduino überspielt, muss dieses Vorgehen wiederholt werden. Über den Link (vgl. Edge Impulse, 2023a) werden die Kommandozeilen/Terminal Anweisungen bereitgestellt.

Anschließend kann ein Projekt in Edge Impulse erstellt werden. Der grundsätzliche Ablauf besteht aus dem Einspielen von Trainings-/Testdaten, dem Kreieren eines Impulses und dem Deployment.

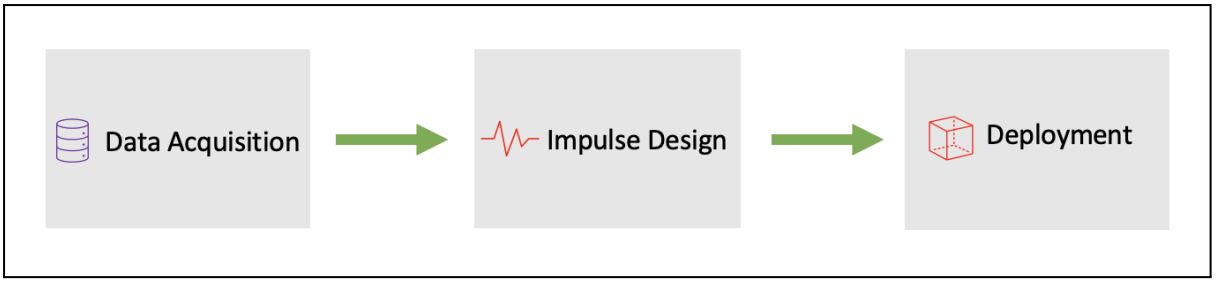

Abb. 4: Arduino Visueller-Ablauf (Eigene Darstellung)

Innerhalb der Data Acquisition ist es sinnvoll, eine umfangreiche Menge an Daten einzuspielen, um das bestmögliche Ergebnis zu erreichen. Endlos viele sollten es jedoch nicht sein, um Overfitting zu vermeiden. Im dokumentierten Beispiel wurden jeweils Fotos von einem Gesicht (Face) und Fotos ohne Gesicht (No Face) aufgenommen und in Trainings- sowie Testdaten auf einen 70/30 Split aufgeteilt.

Anschließend wurde der Impulse kreiert. Für das Erreichen einer möglichst optimalen Genauigkeit, bei moderater Verarbeitungszeit, wurde die Größe der Bilder auf 96x96 reduziert und zusätzlich das Farbschema RGB gewählt.

Im letzten Schritt wurden die Einstellungen für das Modell gewählt. Zusätzlich zu den Grundeinstellungen wurden die Optionen Auto-balance Dataset und Data Augmentation für eine Reduzierung des Overfittings und eine Erhöhung der Trainingszyklen für eine umfassende Genauigkeit gewählt. Beim letztendlichen Modell hat sich das Team für das “MobileNetV1 96x96 0.25 (final layer: 16 neurons, 0.1 dropout)” entschieden, da es mit 131K RAM und 312K ROM die bestmögliche Performance mit der zur Verfügung stehenden Leistung bietet.

Das Modell kommt auf eine Genauigkeit von 80%, wobei mehr Fotos mit Gesichtern (Face) nicht als solche erkannt worden sind, als Fotos ohne Gesicht (NoFace). Die folgende Abbildung zeigt deutlich, dass sich die Daten in zwei grobe Bereiche klassifizieren lassen.

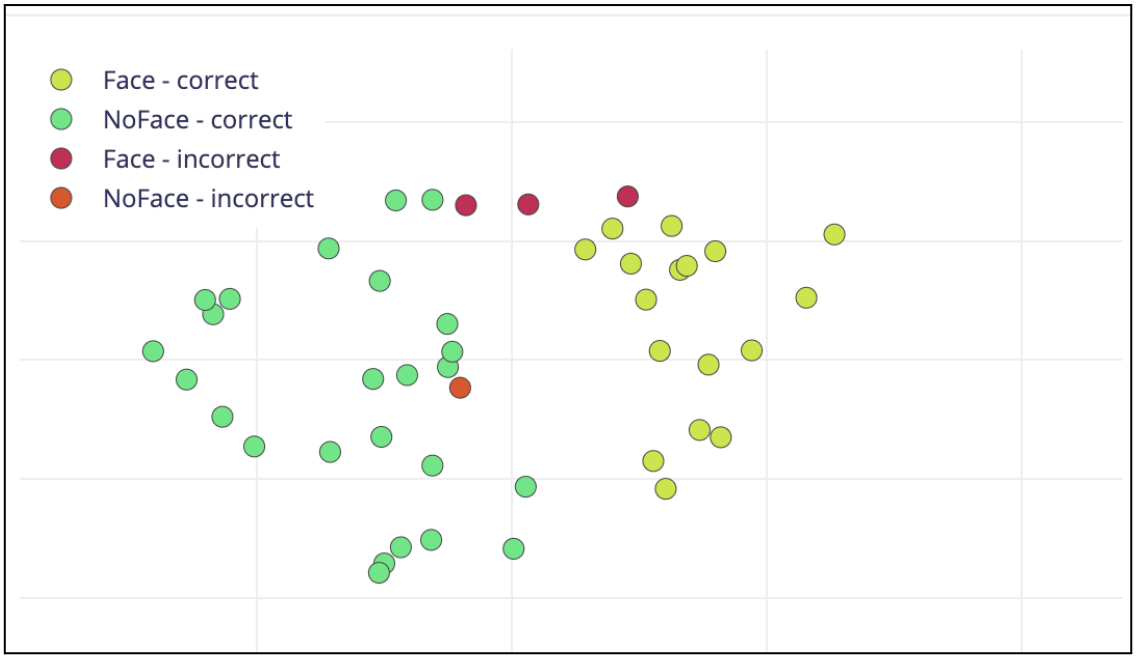

Abb. 5: Arduino Modellgüte Diagramm (Eigene Darstellung)

Über die Live Classification bietet Edge Impulse zusätzlich die Möglichkeit, ein neues Bild aufzunehmen und dieses durch das Modell auf die Features Face/NoFace testen zu lassen.

Im letzten Schritt - dem Deployment - kann das Modell in die gewünschte Umgebung exportiert werden. Innerhalb dieses Projektes wurde dafür die Arduino IDE gewählt. Eine ausführliche Anleitung vom Exportieren der Library als ZIP-Datei, bis hin zur Ausführung des Codes in der Arduino IDE wird über den folgenden Link (vgl. Edge Impulse, 2023b) bereitgestellt. Zu beachten ist, dass der Code nach der Überführung der Library über den Pfad File > Examples > Name_des_Edge_Impuls_Projekts > nano_ble33_sense > nano_ble33_sense_camera abrufbar ist, auch wenn lediglich der Arduino Nano 33 BLE (ohne die Sense Funktionalitäten) genutzt wird.

3.1.3 Ergebnis

Wie Anfangs schon erwähnt, ist es mit dem Arduino Nano 33 BLE nicht möglich, eine Face Recognition durchzuführen. Die Face Detection konnte jedoch mit dem über Edge Impulse erstellten und angelernten Modell in den Grundzügen verwirklicht werden. Nachdem der Code über die Arduino IDE auf die Hardware überspielt wurde, nimmt die OV7675 Kamera alle zwei Sekunden ein Bild auf und überprüft, mit welcher Genauigkeit es sich dabei um ein Gesicht/kein Gesicht handelt. Die Speicherung und Anzeige der Bilder wurde in diesem Beispiel ausgelassen, da dies über die Arduino IDE nur sehr umfassend und umständlich

implementierbar ist. Diese Schritte werden im folgenden Abschnitt mit dem Einsatz des ESP-EYE umgesetzt.
Trotz der geringen Kameraauflösung sowie der relativ schwachen Hardware, ist das System vor allem bei Aufnahmen aus nächster Nähe relativ genau. Ein Ausschnitt der Ausgabe ist in der unten aufgeführten Abbildung zu erkennen. Um das unten abgebildete Ergebnis einsehen zu können, muss das korrekte Board (Arduino Nano 33 BLE) selektiert werden und der Serial Monitor auf einen Upload Speed von: 115200 eingestellt werden.

```
14:41:06.622 -> Starting inferencing in 2 seconds...
14:41:08.632 -> Taking photo...
14:41:11.436 -> Predictions (DSP: 15 ms., Classification: 760 ms., Anomaly: 0 ms.):
14:41:11.436 ->     Face: 0.81641
14:41:11.436 ->     NoFace: 0.18359
14:41:11.436 ->
14:41:11.436 -> Starting inferencing in 2 seconds...
14:41:13.448 -> Taking photo...
14:41:16.257 -> Predictions (DSP: 15 ms., Classification: 760 ms., Anomaly: 0 ms.):
14:41:16.257 ->     Face: 0.76953
14:41:16.257 ->     NoFace: 0.23047
14:41:16.257 ->
14:41:16.257 -> Starting inferencing in 2 seconds...
```

Abb. 6: Arduino Ergebnisausgabe in der IDE (Eigene Darstellung)

3.1.4 Troubleshooting

Innerhalb der Entwicklung mit Edge Impulse werden die Nutzer*Innen von der Software sehr intuitiv durch die Ausarbeitung eines Modells geführt. Auch die dafür benötigten Installationen sind umfassend in den oben aufgeführten Links dokumentiert, so dass das Troubleshooting erst nach der Überspielung des Modells in die Arduino IDE beginnt. Vor allem das Anzeigen sowie das Speichern der Bilder ist über die IDE nur sehr umständlich realisierbar. Zusätzlich ist es, wie oben schon erwähnt, mit der Leistung der Hardware nicht möglich, ein umfassendes ML-Modell von Edge Impulse zu nutzen oder gar eine Face Recognition durchzuführen.
Durch diese Einschränkungen wurde sich innerhalb des Projekts dementsprechend auch auf die Umsetzung mit dem ESP-EYE fokussiert, so dass kein umfassendes Troubleshooting für die Entwicklung über die Arduino IDE vermittelt werden kann.

3.1.5 Pros/Cons

Die Umsetzung mit dem Arduino Nano 33 BLE und Edge Impulse ist vor allem für Nutzer*innen empfohlen, welche noch kein umfassendes Verständnis in der Entwicklung von ML-Modellen besitzen und eher Low- bzw. No-Code entwickeln wollen. Die grafischen

Darstellungen sowie Hilfestellungen und Erklärungen von Edge Impulse geben selbst Einsteiger*Innen einen guten Überblick und eine klare Struktur bei der Entwicklung eines Modells. Zusätzlich bietet Edge Impulse viele Erleichterungen, wie bspw. die Integration eines Handys, um Trainings- und Testdaten in das Modell zu überspielen, um somit das Modell schnell mit ausreichenden Daten anzureichern. Auch der anschließende Export des Modells in die Arduino IDE ist intuitiv gestaltet, so dass das Modell zügig über den Arduino Nano 33 BLE gestartet werden kann, wodurch den Nutzer*innen schnell ein Erfolgsgefühl übermittelt wird.

Die Nachteile der Umsetzung des Projekts mit dem Arduino Nano 33 BLE konzentrieren sich jedoch sehr stark auf die zu schwachen Hardware-Komponenten sowie die unintuitive Umsetzung weiterer Prozessschritte innerhalb der Arduino IDE, so dass das Projekt mit diesen Komponenten nicht vollständig umsetzbar ist, sondern eher als Einführung in die Möglichkeiten von Arduino und Edge Impulse fungiert.

4.2 ESP-EYE

Der folgende Abschnitt behandelt die Dokumentation über die Umsetzung des gesamten Use Cases mit dem Einsatz des ESP-EYE. Die Motivation liegt hierbei auf einer drahtlosen Umsetzung einer Face Detection und Recognition mit den im folgenden Abschnitt aufgeführten Bestandteilen.

4.2.1 Hard- und Software

Bei der Umsetzung mit dem ESP-EYE werden folgende Hardwarekomponenten benötigt:

- Windows/Linux/Mac PC (innerhalb dieses Projekts wurde mit einem Mac gearbeitet)
- ESP-EVE DevKit (mit ESP32 Chip)
- POWER BANK (welche durchgängig einen geringen Stromfluss abgeben kann)
- Micro-USB Kabel

Abb. 7: ESP-EYE Hardwarekomponenten (Eigene Darstellung)

Zusätzlich wird folgende Software benötigt:

- Visual Studio Code inkl. Python (die Umgebung ist beliebig wählbar, in diesem Projekt wurde sich für Visual Studio Code entschieden)
- Arduino IDE

Abb. 8: ESP-EYE Softwarekomponenten (Eigene Darstellung)

Außerdem kommen verschiedene Repositories aus ähnlichen Umsetzungen zum Einsatz:

- Webserver für die Kamera (Link) (vgl. Espressif, 2022)
- Face Recognition (Link) (vgl. Geitgey, 2022)

4.2.2 Aufbau/Ablauf

Für den Aufbau werden folgende Installationen benötigt:

- Python 3.3+
- Brew (oder ein ähnlicher Paketmanager)
- Open CV2
- cmake
- face_recognition
- cmake ninja dfu-util

Die folgende Einrichtung des ESP-EYE muss sowohl für die Umsetzung der Face Detection als auch der Face Recognition durchlaufen werden. Es wird präzise die Vorgehensweise der Installationen/Einrichtungen erläutert oder auf ausgewählte Guides verwiesen.

Die Installation/Konfiguration des ESP-EYE erfolgt im wesentlichen in zwei Schritten:

1. Einrichtung des ESP32 (Microcontroller auf dem ESP-EYE)
2. Installation der Python-Bibliotheken für wahlweise die Face Detection oder Recognition (oder beide).

Einrichtung des ESP-EYE

Die Einrichtung des ESP-EYE basiert auf folgender Dokumentation: Link (Mac/Linux) (vgl. Espressif, 2023)

Im ersten Schritt müssen verschiedene Libraries über den Paketmanager Brew installiert werden:

```
brew install cmake ninja dfu-util
brew insall ccache
brew install python3
```

Um im zweiten Schritt die Anwendung über den ESP-EYE ausführen zu können, werden die aufgelisteten Softwarebibliotheken von Espressif benötigt. Diese liegen im ESP-IDF Repository und müssen geklont werden. Dafür wird in das Installationsverzeichnis navigiert und das Repo mit den folgenden Anweisungen (im Terminal/in der Kommandozeile) in das Verzeichnis *~esp/esp-idf* geklont:

```
mkdir -p ~/esp
cd ~/esp
git clone -b v5.0.1 --recursive https://github.com/espressif/esp-idf.git
```

Anschließend müssen weitere Tools für den ESP-EYE wie Compiler, Debugger und weitere Python-Pakete installiert werden:

```
cd ~/esp/esp-idf
./install.sh esp32
```

Nachdem die Tools installiert sind, müssen diese in die Umgebungsvariable *PATH* eingetragen werden. Dafür stellt ESP-IDF ein weiteres Skript zur Verfügung, welches über den folgenden Befehl aufgerufen werden kann:

```
. $HOME/esp/esp-idf/export.sh
```

1. Nach dem Aufruf sollte folgender Befehl ausgeführt sowie in das Shell-Profil (.profile, .bashrc, .zprofile) einfügen werden:
   `alias get_idf='. $HOME/esp/esp-idf/export.sh'`
2. Im letzten Schritt wird die Konfiguration über den Terminal-Neustart aktualisiert

Jetzt kann über den Befehl *get_idf* die Umgebung eingerichtet oder aktualisiert werden.

Ab diesem Punkt sind alle Voraussetzungen erfüllt, um ein Projekt auf dem ESP-EYE zu starten, zu erstellen, zu flashen und die Geräteausgabe überwachen zu können.

Projekt anlegen und einrichten

Im folgenden Schritt wird beschrieben, wie ein Projekt angelegt und eingerichtet werden kann. Eine gute Orientierung bietet das Beispielprojekt "CameraWebServer" für die Verwendung der ESP32-Kamera mit der Arduino IDE. Das Setup umfasst mehrere Komponenten. Die ESP32-Kamera wird über den GPIO-Port an den ESP32 angeschlossen und mit Hilfe der Arduino IDE programmiert. Anschließend kann der ESP-EYE über eine USB-Schnittstelle an einen Computer angeschlossen werden, auf welchem die Arduino IDE installiert ist. Das Projekt beinhaltet die benötigten Bibliotheken und Header-Dateien sowie Beispielcode, um eine Verbindung zur Kamera herzustellen und eine Webserver-Schnittstelle zu implementieren. Der Webserver ermöglicht es, die Kamera über eine lokale IP-Adresse zu steuern und Bilder oder Videos zu streamen. Anschließend kann das Projekt in andere Anwendungen integriert werden, indem der Code angepasst/erweitert wird, um zusätzliche Funktionalitäten zu implementieren.

Der folgende Abschnitt beinhaltet die Schritte für den Aufbau des ESP32-Cam-Webserver-Beispiels:
Hinweis: Vor dem Durchlauf des Beispiels sollten die erforderlichen Bibliotheken (siehe Schritt zwei im Setup) installiert sein.

1. Im ersten Schritt muss das ESP32-Cam-Webserver-Beispiel von GitHub heruntergeladen werden: (Link) (vgl. Espressif, 2022)
2. Anschließend muss in der Arduino IDE das ESP32-Board-Paket installiert werden. Hierfür wird über die "Datei" in die "Voreinstellungen" navigiert. Im Feld "Zusätzliche Boardverwalter-URLs" kann nun die folgende URL in das Feld "Zusätzliche Boardverwalter-URLs" eingefügt werden: URL (vgl. Espressif, 2022)
3. Im dritten Schritt wird das Board ausgewählt. Hierfür muss über "Werkzeuge" -> "Board" navigiert und das "ESP32 Wrover Module" als Board ausgewählt werden. Die Einstellungen für "Flash Size" und "Partition Scheme" sollten zudem wie folgt eingestellt werden:
    - Flash Size: "4 MB (32 Mb)"
    - Partition Scheme: "Huge APP (3MB No OTA/1MB SPIFFS)"

4. Nach der Wahl des Boards muss über "Werkzeuge" -> "Port" der COM-Port ausgewählt werden, an welchem der ESP-EYE angeschlossen ist.
5. Nach der Einrichtung des Boards und des Ports kann der ESP-EYE an den Computer angeschlossen werden, um die Wifi-Informationen in die Variablen im File CameraWebServer.ino einzufügen. Anschließend müssen diese auf die Hardware überspielt werden.
6. Nach der Einrichtung des WLANs wird der ESP-EYE an eine Stromquelle angeschlossen. Zu beachten ist, dass die Stromversorgung stabil und ausreichend ist.
7. Über den seriellen Monitor in der Arduino IDE kann nun auf die genutzte IP-Adresse des ESP-EYE zugegriffen werden. Nach der Eingabe der IP-Adresse in einem Webbrowser kann anschließend auf die Live-Übertragung sowie verschiedene Funktionen des ESP-EYE zugegriffen werden.
8. Nach der erfolgreichen Übertragung können die Beispielcode-Dateien bearbeitet werden, um den Code an die spezifischen Anforderungen anzupassen und Änderungen an der Webserver-Oberfläche vorzunehmen. Um die Änderungen zu übernehmen, muss lediglich der geänderte Code auf den ESP-EYE überspielt werden.

Nach diesem Durchlauf sollte es möglich sein, die nun folgenden Beispiele zu integrieren.

Installation der Python-Bibliotheken für die Face Detection/Recognition

Die folgenden Installationsschritte können über den Link nachvollzogen werden: Link. (vgl. Geitgey, 2022)

Im ersten Schritt müssen die Libraries installiert werden:

```
brew install cmake
pip3 install face_recognition
```

Nach der Installation können zwei verschiedene Kommandozeilen-Programme benutzt werden:

- face_recognition (Erkennen von bestimmten Gesichtern)
- face_detection (Erkennen von Gesichtern)

Die folgenden Abschnitte erläutern die jeweiligen Vorgehensweisen für die Detection sowie die Recognition.

Face Detection

Die folgende Abbildung beschreibt die grundlegende Vorgehensweise bei der Umsetzung der Face Recognition innerhalb dieses Projekts.

Abb. 9: ESP-EYE Visueller-Ablauf Face Detection (Eigene Darstellung)

Innerhalb des ersten Schrittes werden die schon vortrainierten Classifier initialisiert. In diesem Projekt wird sich durch die intuitive Integration auf die Classifier von OpenCV beschränkt. Die von Paul Viola und Michael Jones entwickelten Modelle sind durch die Verteilung der Features auf verschiedenen Ebenen hinweg durchaus performant und exakt. (vgl. OpenCV, 2023)

Beim Testen ist es jedoch trotzdem oft vorgekommen, dass vor allem in heterogenen Bereichen Gesichter an Stellen erkannt worden sind, wo gar keine waren. Um diesem Problem entgegenzuwirken, wurden die Filter durch Schleifen einander gelegt. Dadurch stand bspw. dem Classifier für die Augenpartie lediglich der Ausschnitt des Gesichts zur Verfügung, welcher durch den ganzheitlichen Gesichts-Classifier erkannt worden ist. Durch diese Verschachtelung konnte innerhalb des Projekts eine geringere Fehleranfälligkeit bei Bildern ohne Gesicht erzielt werden.

Ziel der Verschachtelung ist es, dass erst, wenn alle Classifier ein Gesicht erkannt haben, das Bild über den Befehl `cv2.imwrite` im dafür zugehörigen Verzeichnis gespeichert wird. Damit nicht für jedes Bild des Streams (20 Bilder pro Sekunde) ein Bild abgespeichert wird, wird bei einer Erkennung eines Gesichts lediglich jedes zwanzigste Bild gespeichert.

Face Recognition

Die folgende Abbildung beschreibt die grundlegende Vorgehensweise bei der Umsetzung der Face Recognition innerhalb dieses Projekts. Zu erkennen ist, dass der Prozess lediglich um die “Speicherung bekannter Gesichter" und die "Übereinstimmung der Gesichter prüfen” im Vergleich zu der Face Detection erweitert worden ist.

Abb. 10: ESP-EYE Visueller-Ablauf Face Recognition (Eigene Darstellung)

Im ersten Schritt wird ein Ordner mit einem Bild von jedem Gesicht, welches erkannt werden soll, erstellt.
Der folgende Codeabschnitt zeigt die Verknüpfung der Anwendung mit den Bildern der bekannten Gesichter.

```
name_person = face_recognition.load_image_file("Pfad des Ordners")
name_face_encoding = face_recognition.face_encodings(name_person) [0]
```

Im zweiten Schritt wird der Eye Cascade Classifier initialisiert (im nächsten Absatz erläutert) und anschließend die Gesichtserkennung für das eingespielte Videomaterial durchlaufen. Zudem werden die Namen der gespeicherten Personen festgelegt.

Wird ein Gesicht erkannt, wird dieses auf die Übereinstimmung mit den bekannten abgeglichen. Ist dies der Fall, wird der Name der Person unter dem Gesicht angezeigt. Wird die Person nicht erkannt, erscheint das Wort “Unknown” unter dem Gesicht der Person. Zusätzlich zu den Face Recognition Classifiern, wird das Gesicht nun über den Eye Cascade Classifier erneut identifiziert. Hierdurch konnte im Versuch die Wahrscheinlichkeit einer fehlerhaften Gesichtserkennung minimiert werden.

Wird das unbekannte Gesicht auch über den Eye Cascade Classifier als Gesicht identifiziert, wird jede Sekunde (alle 20 Bilder, da die Aufnahme 20 Bilder pro Sekunde beträgt) ein Bild des unbekannten Gesichts in dem dafür vorher erstellten Ordner abgelegt. Durch eine Random-ID als Name, werden die Bilder im Ordner nicht überschrieben. Über den Befehl: `cv2.imwrite` werden die Bilder gespeichert.

4.2.3 Ergebnis

Innerhalb des Projekts konnten mit dem ESP-EYE sowohl die Face Detection als auch die Recognition umgesetzt werden. Nachdem das Projekt über den ESP-EYE angelegt worden ist und die WLAN Verbindung mit dem Code über die Arduino IDE eingerichtet werden konnte, wurde über den jeweiligen Code die Detection/Recognition via Python gestartet.

Durch das Ineinanderlegen der Classifier konnte innerhalb der Detection eine hohe Genauigkeit der Erkennung erzielt und vor allem die fehlerhafte Erkennung reduziert werden. Etwas ungenauer wurde die Erkennung jedoch bei schlechteren Lichtverhältnissen, was auf die geringe Kameraqualität des ESP-EYE zurückzuführen ist. Die folgende Abbildung zeigt ein vom Algorithmus erkanntes Gesicht. Die Abbildung veranschaulicht noch einmal deutlich, wie die Classifier ineinander agieren.

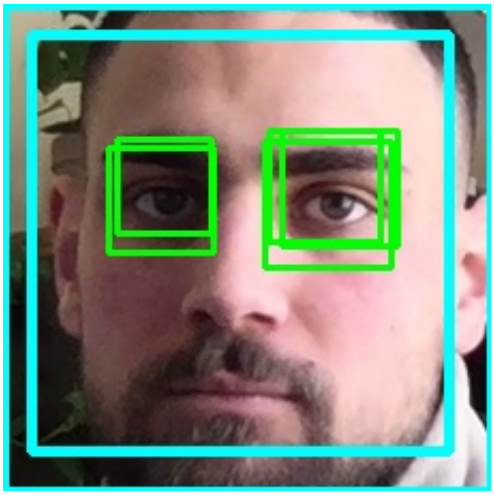

Abb. 11: ESP-EYE Ergebnis Face Detection (Eigene Darstellung)

Durch den Zusatz einzelner Classifier konnte neben der Detection auch die Genauigkeit der Recognition verbessert werden. Nach der WLAN Einrichtung ist es somit möglich, innerhalb der gesamten Netzumgebung den ESP-EYE mit Face Recognition aktiv laufen zu lassen. Zudem können beliebig viele Gesichter eingespeichert werden.
Eine Erkennung mehrerer Gesichter in einem Bild (wie in der unteren Abbildung) ist jedoch mit dem ESP-EYE nicht möglich, da die Qualität der Kamera sowie die Performance des ESP32 auch hier zu schlecht sind. Um dies trotzdem umsetzen zu können, kann die `video_capture` auf `cv2.VideoCapture(0)` gesetzt werden, so dass auf die Webcam des Computers zugegriffen werden kann.

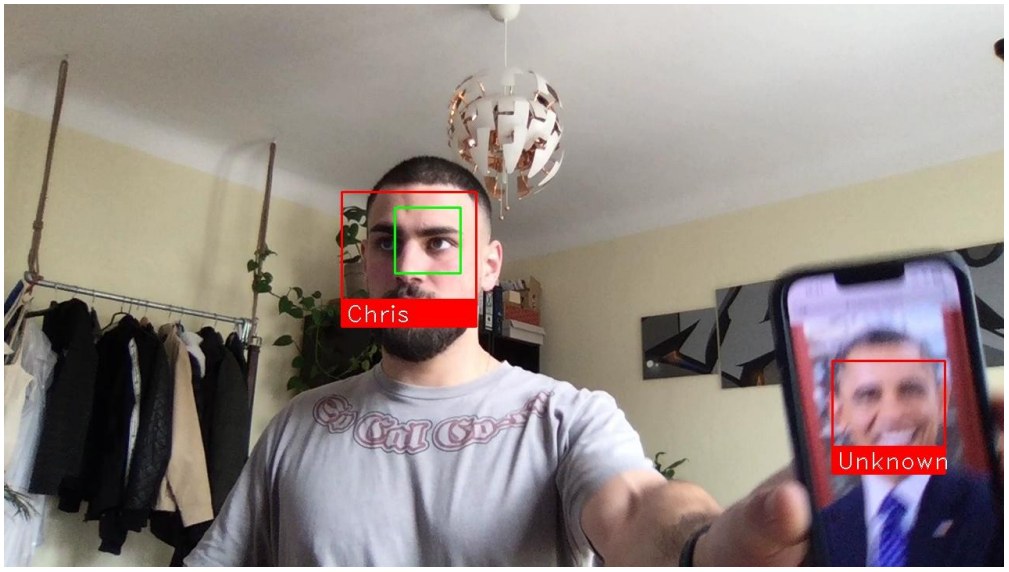

Abb. 12: ESP-EYE Ergebnis Face Recognition (Eigene Darstellung)

4.2.4 Troubleshooting

Bei der Erstellung des Projekts können einige Probleme auftreten, die wie folgt gelöst werden können.
Vor allem innerhalb der Installation von Espressif und dessen Toolchain kann es zu diversen Fehler kommen. Im Setupprozess des ESP-EYE sind einige Fehler aufgetreten, welche am besten durch ein handelsübliches Ausschlussverfahren der verschiedenen Möglichkeiten behoben werden können. Um viele Installationsfehler zu vermeiden, ist es wichtig, den ESP-EYE nicht direkt zu flashen, sondern mit der schon vorinstallierten Software zu arbeiten.
Trotzdem können Probleme auftreten, die lediglich mit dem Neustart (kein flashen) des ESP-EYE behoben werden können.

Weiter ist wichtig, für das Setup des WIFI's, nicht das Espressif Menü für die Konfiguration zu nutzen, sondern die entsprechenden Werte in die Variablen des Files "CameraWebServer.ino" einzusetzen.

Nach der Einrichtung des ESP-EYE hilft es vor allem für die Genauigkeit der Detection die Classifier ineinanderzulegen. Hierbei müssen alle innenliegenden Klassifizierungen, wie Augen- und Mundklassifizierungen, innerhalb der äußeren Gesichts-Classifier liegen. Durch diese Verschachtelung können vor allem einzelne Augen und Mund-Erkennungen außerhalb eines Gesichtes verbannt und dadurch die Genauigkeit erhöht werden.

Innerhalb der Recognition ist es zudem ratsam, die Bilder des aufgenommenen Streams zu verkleinern, um eine gute Performance aufrechtzuerhalten. Dabei wird die Dimension der Bilder auf ein Viertel reduziert. Für die spätere Speicherung können die Bilder wieder hochskaliert werden.

Bezüglich der Speicherung ist es zudem sinnvoll, dass bei einer hohen Bildrate durch die Videoübertragung nicht jedes einzelne Bild mit einem (unbekannten) Gesicht gespeichert wird. Hierfür wurde ein Zähler eingerichtet, wodurch nur bei jedem 20ten Bild der Befehl zur Speicherung weitergegeben wird.

4.2.5 Pros/Cons

Im Vergleich zum Arduino Nano 33 BLE inkl. OV7675, besitzt der ESP-EYE mit ESP32-Chip eine bedeutend bessere Kameraauflösung sowie eine spürbar bessere Performance. Zudem wird keine Platine zum Zusammenstecken der Hardwarekomponenten benötigt.
Vor allem durch Examples wie dem CameraWebServer-Example (auf welchem auch dieses Projekt aufgebaut ist), wird deutlich, dass der Einsatzzweck des ESP-EYE im Gegensatz zum Breiten Einsatzfeld des Arduinos eher spezifisch auf die Face Detection/Recognition abzielt, wodurch er vor allem für dieses Projekt besser geeignet ist. Hinzu kommt ebenfalls die intuitive Einrichtung des WLANs, wodurch der ESP-EYE schnell drahtlos eingesetzt werden kann. Auch die angelernten Classifier von OpenCV sind eine gute Lösung, schnell ML-Modelle integrieren zu können.

Im Gegensatz zu der Einbindung von Edge Impulse, ist das Projekt mit dem ESP-EYE jedoch deutlich aufwändiger und weniger für Code-Neulinge geeignet, da es sich um eine Full Code Lösung handelt. Mit einer gewissen Erfahrung in Python ist es jedoch gut umsetzbar. Zudem ist die Leistung des ESP32-Chips relativ gering. Gerade für die Recognition sollte mind. der ESP-32-S3-Chip eingesetzt werden, um den Algorithmus direkt über das Board laufen zu lassen. Auch die Qualität der eingebauten Kamera sollte für einen umfangreichen Einsatz verbessert werden.
Alles in allem bietet der ESP-EYE jedoch eine gute Basis für das Projekt.

5. Fazit/Ausblick

Der Arduino Nano 33 BLE und der ESP-EYE sind jeweils eine gute Basis, um sich mit der Programmierung von Hardwarekomponenten auseinander setzen zu können. Vor allem der ESP-EYE überzeugt in diesem Projekt mit seinem deutlich besseren Chip und der höheren Kameraauflösung. Durch diese Aspekte bietet er sich im Gegensatz zum Arduino für die Face Recognition wesentlich besser an. Zu den bereitgestellten Anwendungs/Code Beispielen einzelner Bibliotheken, an denen sich vor allem Anfänger*Innen der Hardwareprogrammierung orientieren sollten, bieten die Anbieter OpenCV und Edge Impulse die Möglichkeit, intuitiv ML-Modelle in den Algorithmus mit einzubinden. Sollte das Projekt in einem größeren Umfang erweitert werden, könnten jedoch bspw. die Classifier von OpenCV durch eigene ersetzt werden.
Zudem würde es sich anbieten, im weiteren Verlauf das Projekt um bspw. die Integration von einem Lichtsystem auf dem Grundstück (welches bei einem fremden Gesicht eingeschaltet wird) oder auch Alarmanlagen sowie eine Integration des Notrufs zu erweitern. Für weitere Erweiterungen ist lediglich die Kreativität und Umsetzungsfähigkeit der Nutzer*Innen gefragt. Das Projekt “Feel Safe with ESP-EYE” bietet dafür allzeit eine gute Grundlage für den Aufbau oder auch die Erweiterung von bestehenden/neuen Projekten.

6. Unterstützende Literatur

In diesem Abschnitt wird die zum Projekt unterstützende Literatur aufgelistet. Aus der anfänglichen Sichtung dieser ist ebenfalls die Entwicklung des Use Cases entstanden. Die vollständigen Links und Titel werden im Literaturverzeichnis aufgeführt.

Der Beitrag von Navya Santhoshi Akkisetti behandelt die Implementierung eines auf Gesichtserkennung basierenden Türschlosssystems, welches mit Arduino und der OpenCV-Bibliothek entwickelt wurde. Das System ist eine Alternative zur Passwortüberprüfung und ermöglicht die Verwendung eindeutiger Gesichter für die Überwachung des Heimzugangs. Der Gesichtserkennungsalgorithmus wird unter Verwendung des Haar-Kaskadenalgorithmus und des LBPH-Algorithmus implementiert, um ein klares Bild des eingegebenen Gesichts zu erhalten. Wenn das vom System erkannte Gesicht mit der Datenbank übereinstimmt, wird die Tür entriegelt, andernfalls wird sie verriegelt. (Akkisetti, 2022)

Das Paper von Miry erklärt eine vielversprechende Technologie, in welcher insbesondere die Biometrie für die Gesichtserkennung verwendet wird. In diesem Paper wird ein Echtzeit-Gesichtserkennungssystem mit einer Webcam als Eingabegerät und einem Arduino-Mikrocontroller als Ausgabegerät implementiert. Die Programmierung erfolgt mit MATLAB, wodurch eine große Auswahl an Algorithmen bereitgestellt wird. Zur Merkmalsextraktion wird die Wavelet-Transformation in Verbindung mit der Hauptkomponentenanalyse verwendet, um Datenredundanz zu vermeiden. Das System wurde mit echten Bildern getestet, die von Webcams aufgenommen wurden und zeigt vielversprechende Ergebnisse. (Miry, 2016)

In der Arbeit von Patnaik, Prasad, Rao, Rathnala, Rayan und Sura, geht es darum, Lösungen mit verbesserter Leistung und minimierten Nachteilen zu entwickeln, die sich aus der zunehmenden Verwendung von intelligenten und tragbaren elektronischen Geräten ergeben. Hierbei wurde die Merkmalsextraktion als wichtigste Phase für den Erfolg eines Gesichtserkennungssystems definiert. Verschiedene Implementierungsmöglichkeiten in diesem Projekt hängen von der verwendeten Programmiersprache oder dem verwendeten Algorithmus ab, wie z. B. MATLAB, OpenCV, Visual Basics C#, Viola-Jones-Algorithmus usw.. Die Kernfunktionalität bleibt jedoch gleich. In dieser Arbeit wird die Gesichtserkennung in drei Stufen implementiert. Phase eins umfasst das Erkennen von Gesichtern und das Sammeln von Bild-IDs. Phase zwei umfasst das Trainieren des Erkennens und das Isolieren von Objekten von Interesse und die letzte Phase umfasst das Gruppieren und Speichern in XML-Dateien. (Rathnala et al., 2022)

7. Literaturverzeichnis

Akkisetti, N. S. (2022): Door Lock System Based on Face Recognition Using Arduino, in Social Science Research Network.
Link: https://doi.org/10.2139/ssrn.4236516 (Aufruf: 25.03.2023).

BKA (2022): Polizeiliche Kriminalstatistik (PKS), PKS-Tabellen im Jahr 2021.
Link: https://www.bka.de/DE/AktuelleInformationen/StatistikenLagebilder/PolizeilicheKriminalstatistik/pks_node. (Aufruf: 25.03.2023).

Edge Impulse (2023a): Arduino Nano 33 BLE Sense.
Link: https://docs.edgeimpulse.com/docs/development-platforms/officially-supported mcu-targets/arduino-nano-33-ble-sense (Aufruf: 25.03.2023).

Edge Impulse (2023b): Arduino library.
Link: https://docs.edgeimpulse.com/docs/deployment/arduino-library (Aufruf: 25.03.2023).

Espressif (2022): arduino-esp32, Github Repository.
Link: https://github.com/espressif/arduino-esp32/tree/master/libraries/ESP32/examples/Camera/CameraWebServer (Aufruf: 25.03.2023).

Espressif (2023): Standard Toolchain Setup for Linux and macOS, in ESP-IDF Programming Guide.
Link: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/linux-macos-setup.html (Aufruf: 25.03.2023).

Geitgey, A (2022): face_recognition, Github Repository.
Link: https://github.com/ageitgey/face_recognition (Aufruf: 25.03.2023).

Konopka, B. (2014), Bildqualität ist nicht alles. Optik & Photonik, 9: 29-31.
Link: https://doi.org/10.1002/opph.201400053 (Aufruf: 25.03.2023).

Miry, A. H. (2016): Real Time Face Recognition Based on Matlab and Arduino Microcontroller, in Journal of Al-Ma'moon College, Volume (o.A.), Issue 27.
Link: https://www.iasj.net/iasj/article/110920 (Aufruf: 25.03.2023).

OpenCV (2023): Cascade Classifier Training, in OpenCV Tutorials.
Link: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
(Aufruf: 25.03.2023).

Patnaik, P. K./Prasad,B./Rao, S. M./Rathnala, P./Rayan, D./Sura, S. R. (2022): Design of an Efficient Face Recognition system using Deep Learning Technique, in International Journal of Electrical and Electronics Research (IJEER), Volume 10, Issue 3.
Link: https://ijeer.forexjournal.co.in/papers-pdf/ijeer-100345.pdf (Aufruf: 25.03.2023).