GEBÄUDELEITSYSTEM FÜR FEUERWEHREINSÄTZE

JUST

Zeppelin Jugendstiftung

Schule: Bildungszentrum Parkschule Kressbronn (Realschule)

Erarbeitungsort: Schülerforschungszentrum Friedrichshafen

KURZFASSUNG

Bei Feuerwehreinsätzen in großen Gebäuden, wie beispielsweise Schulen, müssen die Rettungskräfte jeden einzelnen Raum nach vermissten Personen absuchen um diese zu finden. Die unsystematische Suche kann unter Umständen viel Zeit in Anspruch nehmen. Mit unserem Projekt wollen wir die Suche von Personen in einem Gebäude vereinfachen. Dies erreichen wir mit Webcams, die im Brandfall mithilfe eines neuronalen Netzes Personen automatisch erkennen. Die Rettungskräfte werden anschließend mithilfe von LED-Streifen gezielt zu diesen Räumen geführt.

Milan Steinbach Pit Reichler

Klasse: 10

Inhalt

1. Einleitung

1.1. Ideenfindung

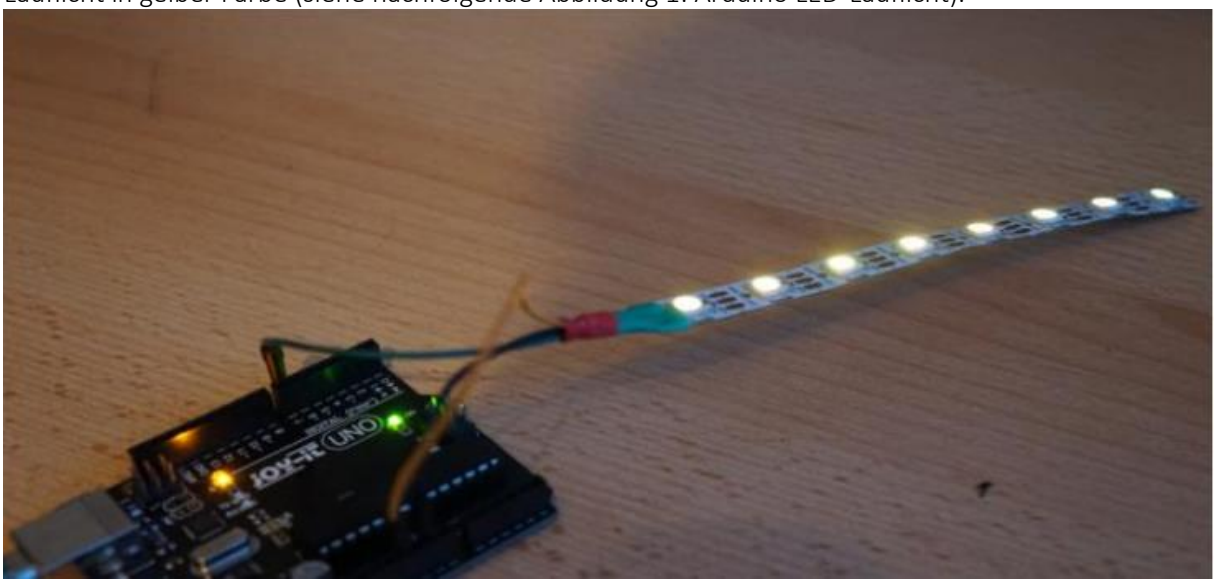
Die Idee entstand durch die vermehrten Fehlalarme in unserer Schule (Bildungszentrum Parkschule Kressbronn). Dabei musste die Feuerwehr immer das ganze Haus durchsuchen, um nach Personen zu schauen, die sich zum Beispiel noch in Klassenzimmern aufhalten. Beim Zuschauen kam uns die Idee, das Gebäudeleitsystem so aufzurüsten, so dass es den Einsatzkräften Zeit spart und automatisch erkennt, in welchen Räumen sich noch Personen aufhalten.

1.2. Unser Lösungsansatz

Durch unser vorheriges Projekt “LED-Gebäudeleitsystem”, welches zur Findung von Räumen in öffentlichen Gebäuden, wie zum Beispiel Rathäusern, Schulen und Kinos verwendet wurde, hatten wir bereits eine gute Grundlage für unsere neue Idee. In dieser Projektarbeit stellen wir unsere Lösung vor, welche man zukünftig in den oben genannten Gebäuden verwenden könnte.
Doch was benötigten wir nun um dieses Projekt umzusetzen? Zum einen eine Möglichkeit Personen zu erkennen und zum anderen ein Leitsystem, damit die Einsatzkräfte auch sicher zu diesem Ziel geführt werden. Das Leitsystem hatten wir bereits, also fehlte uns nur noch eine Lösung für die Personenerkennung.
Auf TensorFlow, ein Hilfsmittel für Objekterkennung, sind wir gestoßen, als wir im Internet nach einer Möglichkeit für die Objekterkennung von selbst bestimmten Gegenständen gesucht haben.
Mithilfe von TensorFlow Lite erstellten wir dann ein neuronales Netz einer Spielzeugfigur und konnten dieses dann mit einem Raspberry und einer Kamera abfragen. Falls nun ein “Mensch” auf dieser Kamera erkannt wird, leiten LED-Laufbänder einen Weg vom Eingang zum jeweiligen Raum.

2. Aufbau des Gebäudeleitsystems (Modell)

Wir benötigten einen LED-Streifen, welcher einzeln adressierbare RGB-LEDs besitzt. Hierfür eignet sich u.a. der LED-Stripe WS2812, der drei Pins zur Nutzung verwendet (Vcc, GND, Data-in). Die LEDs werden mit einem Raspberry 4 angesteuert. In einem ersten Test programmierten wir ein einfaches Lauflicht in gelber Farbe (siehe nachfolgende Abbildung 1: Arduino LED-Lauflicht).



Da in einem Gebäude aber auch Kreuzungen und Abzweigungen existieren, versuchten wir das Problem anhand einer Modellkreuzung zu simulieren. Wir nahmen eine Spanplatte und klebten 4 LED-Streifen mit insgesamt 40 Dioden zu einer rechtwinkligen Kreuzung, verlöteten diese in Reihe und verbanden sie mit Jumper-Kabeln mit dem Arduino (siehe Abb. 3).

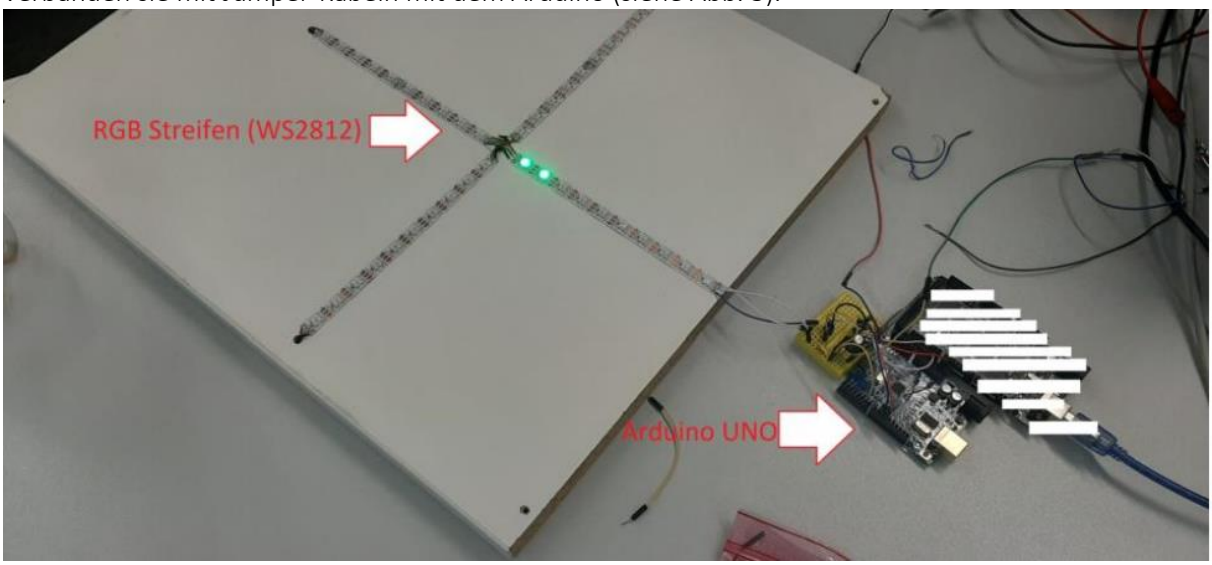


Abb. 2: Kreuzung mit 40 LEDs

Alle Komponenten arbeiten mit 5 V. Da der Strom des Raspberrys nicht ausreicht, um die LED-Streifen zu versorgen, nutzen wir ein Steckdosennetzteil mit einer Ausgangsspannung von 5 V und 4.0 A, um die LED-Streifen zu betreiben. Zusätzlich haben wir zur Erhöhung der Lebensdauer der LEDs einen Kondensator mit 100 µF an die externe Stromquelle und einen Widerstand von 330 Ω an den Datenpin der LED-Streifen verschaltet. Das Schema der Kreuzung Abb. 3, wurde mit der Freeware Fritzing erstellt.

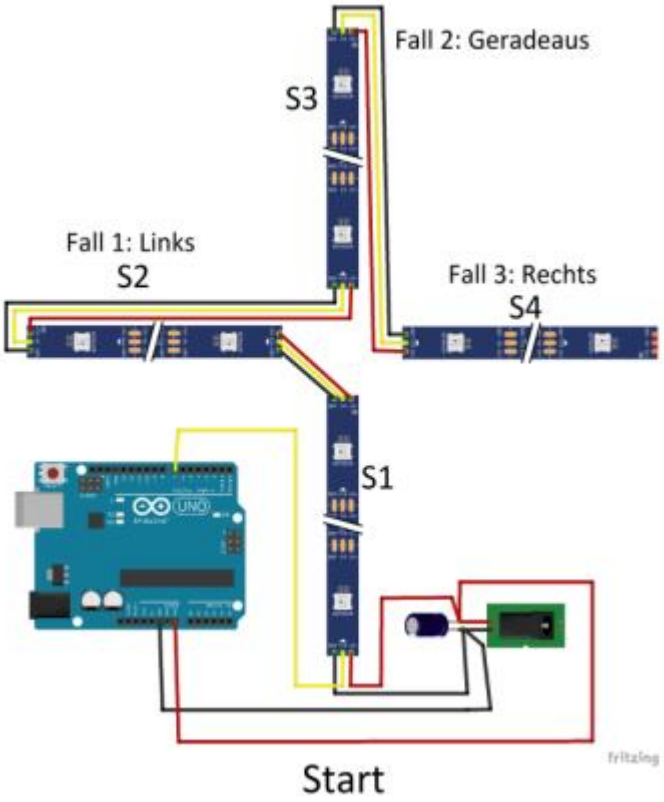


Abb. 3: Kreuzung Schema

2.1. Umsetzung automatisiertes Stockwerk

Nachdem der Test mit der Kreuzung erfolgreich verlief, waren wir bereit, ein Stockwerk in Modellgröße aus Holz zu bauen.
Das Modell besteht aus acht Räumen sowie einen Haupteingang. Für die Umsetzung fräste unser Betreuer, Herr Burkhard Mau, mit einer Oberfräse 12 mm breite und 8 mm tiefe Kanäle in das Holz, damit wir die LED-Streifen verbauen und anschließend eine Polymethylmethacrylat-Platte auflegen und anschrauben konnten. Wenn sich im Brandfall eine Person in einem Raum befindet, so erstellt der Mikrocontroller mit dem LED-Streifen ein Lauflicht bis zu der Tür des Raumes und beleuchtet den Boden des Raumeinganges.

Abb. 4: Diffuses Licht dank milchigem PMA-Glas

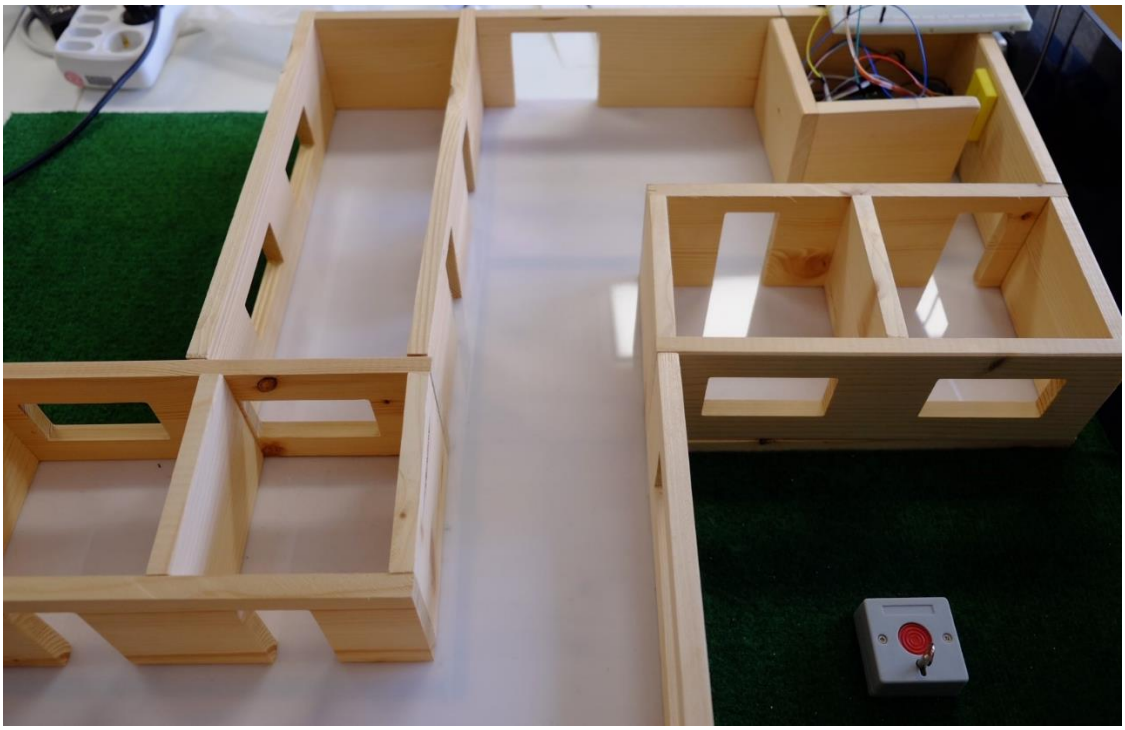


Abb. 5: Stockwerkmodell

2.2. RGB-Dioden

Eine RGB-Diode ist eine Kombination aus drei LEDs mit verschiedenen Farben. Es sind drei separate LEDs, die in einem Gehäuse verbaut sind. Eine RGB-Diode kann verschiedene Farben ausgeben, indem sie die drei Grundfarben rot, grün und blau mischt und auch deren Intensität ändert. Da die LEDs sehr nahe aneinander liegen, kann unser Auge die einzelnen Farben nicht sehen und erkennt nur die Farbkombination (additive Farbmischung, d.h. „rot“ + „grün“ + „blau“= „weiß“). Deshalb gibt es vier Leitungen, eine Leitung für jede der drei Farben und eine gemeinsame Kathode oder Anode, abhängig vom RGB-LED-Typ. Zur Einstellung der Intensität jeder LED kann ein PWM-Signal (Pulsweitenmodulation) verwendet werden. Das Diagramm zeigt, wie man die Farben kombinieren kann.

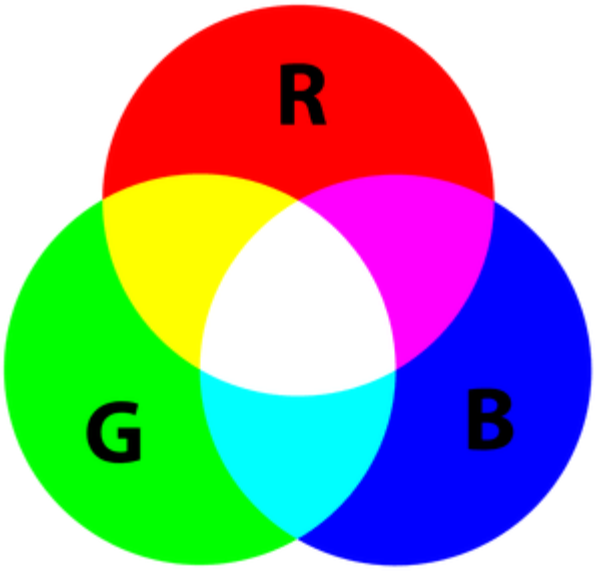


Abb. 6: RGB Diagramm

2.3. Aufbau und Funktionsweise

Wir verwenden einen WS2812 Streifen mit fertig montierten digitalen LEDs, die einen integrierten Controller besitzen. Die Rückseite des Streifens ist mit Klebefilm versehen, so ist ein einfaches Montieren möglich.
Die einzelnen LEDs können an den Verbindungsstellen einfach mit einer Schere getrennt und einzeln verdrahtet werden. Bei den WS2812 LEDs handelt es sich um adressierbare RGB-Dioden. Sie verfügen über einen integrierten Controller. Damit genügt ein Datensignal vom Raspberry-Board und die LEDs leuchten in verschiedenen Farben und in verschiedener Intensität. Für die Intensität der Farbwerte rot, grün und blau werden die Zahlen 0 – 255 verwendet. Die folgende Grafik verdeutlicht die Verwendung der Zahlen (vgl. Hermann).

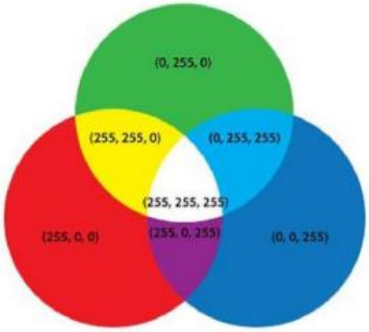


Abb. 7: RGB-Grafik

Jede LED mit dem WS2812 Controller verfügt über vier Anschlüsse. 5V+, GND, DI und DO. DI steht für Data IN und DO für Data OUT. Man kann also auch mehrere Strips hintereinanderschalten, max. 530 LEDs können mit nur einem Datenpin angesteuert werden. Ein Raspberry besitzt 26 davon, daher wären theoretisch ca. 13600 LEDs ansteuerbar.
Das folgende Schema zeigt, wie ein LED-Streifen an den Arduino angeschlossen werden kann:

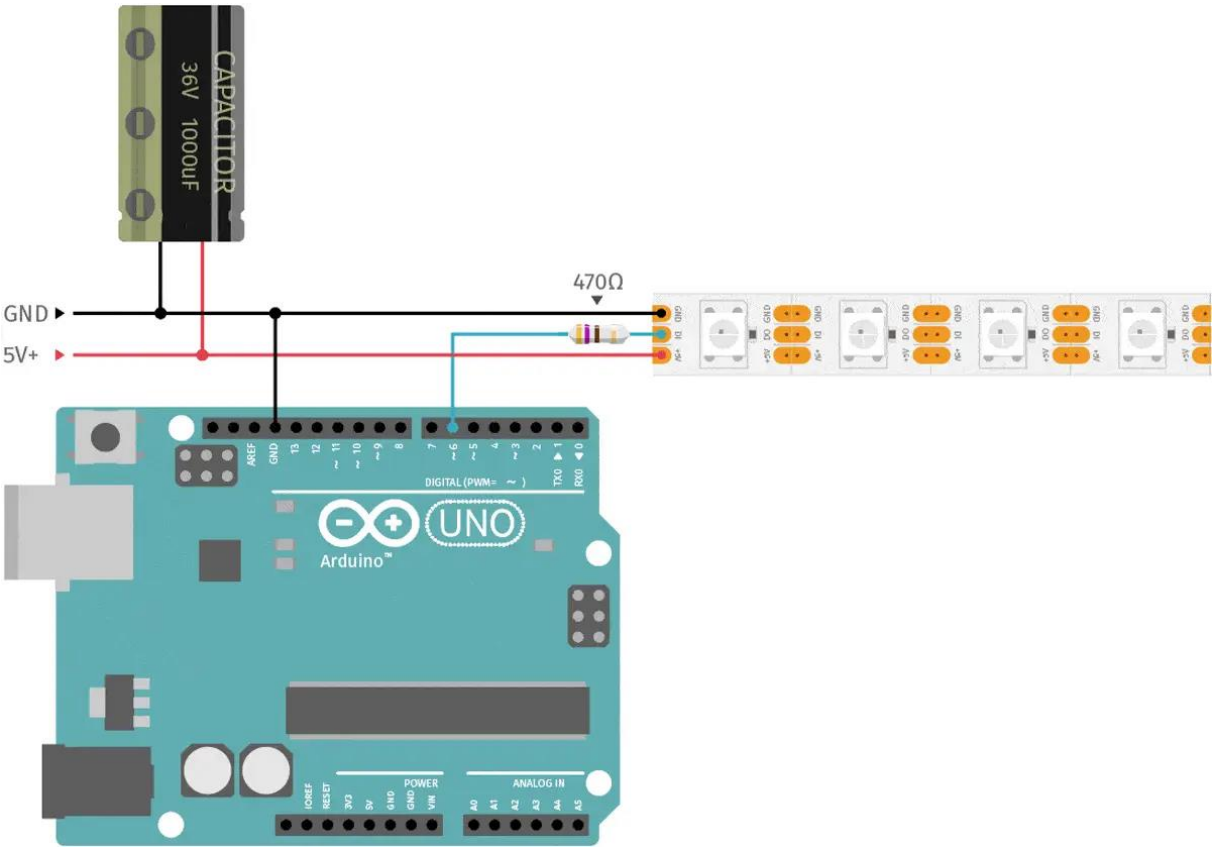


Abb. 8: Anschluss einer WS2812
Wie das Schema zeigt, sollten zwei Dinge beachtet werden. Der Datenpin sollte über einen Widerstand mit dem Anschluss DI verbunden werden, das verlängert die Lebensdauer des Streifens. Des Weiteren ist es bei der Verwendung von mehr als 20 LEDs sinnvoll, eine externe Stromversorgung durch ein Netzgerät zu verwenden.

3. Erkennung von Playmobilfiguren

3.1. Was ist TensorFlow(TF)?

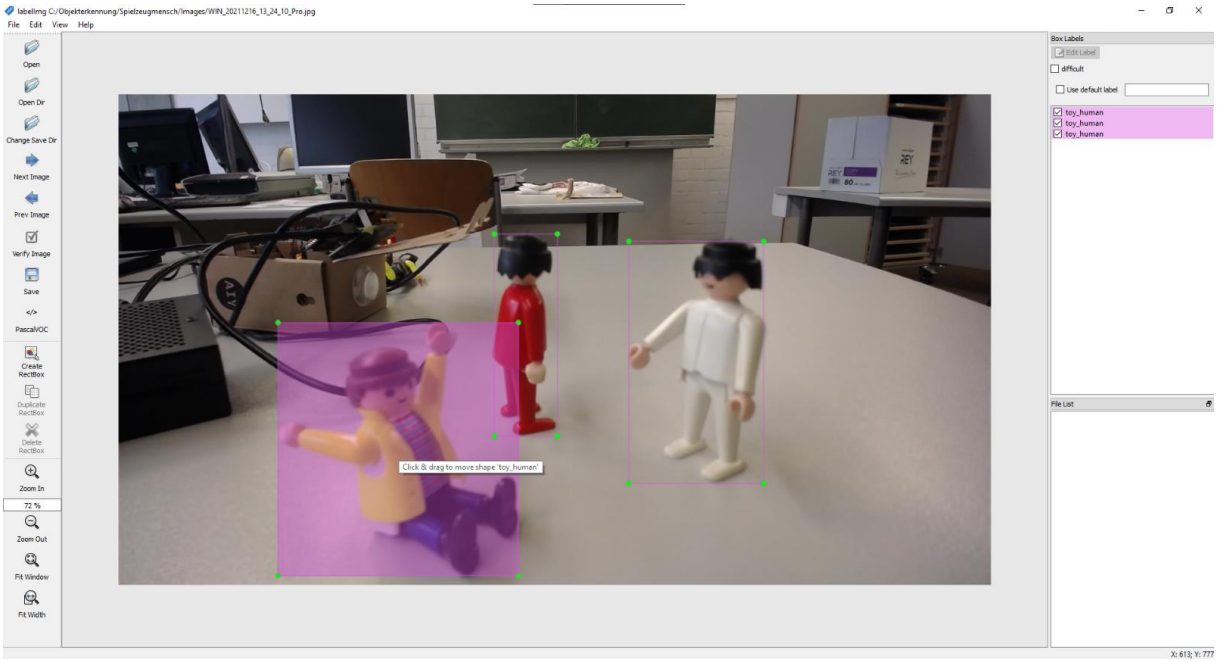
„TensorFlow ist eine End-to-End-Open-Source-Plattform für maschinelles Lernen. Es verfügt über ein umfassendes, flexibles Ökosystem aus Tools, Bibliotheken und Community-Ressourcen, mit denen Forscher den Stand der Technik im Bereich ML vorantreiben und Entwickler auf einfache Weise ML-basierte Anwendungen erstellen und bereitstellen können.“ (Tensorflow)

3.2. Fotografieren von Figuren

Als erstes benötigten wir eine Anzahl von Bildern, um unser neuronales Netz zu trainieren. Damit dieses Modell möglichst flexibel ist, verwendeten wir 10 verschiedene Playmobilfiguren und versuchten aus vielen verschiedenen Winkeln und Positionen Fotos zu machen.

3.3. Labelling von den Bildern

Damit TensorFlow weiß, an welcher Position die Figuren auf den vorher erstellten Bildern ist, mussten wir zuerst diese Fotos labeln. Das bedeutet, dass wir einen Rahmen um diese Figuren ziehen und dann die Position, wo dieser Rahmen zu finden ist, in einer XML- Datei speichern mussten. Des Weiteren gibt man diesem Label auch noch einen Namen. In einem Bild können auch mehrere Figuren auftreten, das heißt, man kann in einer Label-Datei mehrere Figuren einrahmen und somit Speicherplatz sparen. Um diese Label zu erstellen, verwendeten wir ein Programm namens “Labelimg”, welches wie folgt aussieht:



Auf diesem Bild (Abb. 9) sieht man 3 Figuren in einem Bild gelabelt, mit dem Namen ”toy_human”.

3.4. Bilder und Label in ein TFlite Modell umwandeln

Wir hatten nun also die Bilder und die XML-Dateien. Um diese Informationen in einem Modell zu verarbeiten, verwendeten wir ein Python Skript welches uns ein “.tflite” Modell ausgab.
Das erstes gaben wir den Ordner an, in welchem wir unsere Label und Bilder gespeichert haben.

```
# Bilder- und Labelordner festlegen
train_directory = r"/home/user/Desktop/Tensorflow/Spielzeugmensch/train"
```

Abb. 10

Als zweites mussten wir die Komplexität des Modells festlegen. Für diese stellt uns TensorFlow fünf Modell-Typen zu Verfügung: “efficientdet_lite0” bis “efficientdet_lite4”.

```
# Modell Komplexität festlegen (0-4)
spec = model_spec.get('efficientdet_lite4')
```

Abb. 11

Die Auswirkung auf Genauigkeit, Dateigröße und Geschwindigkeit der Typen kann man in diesen Grafiken sehen:

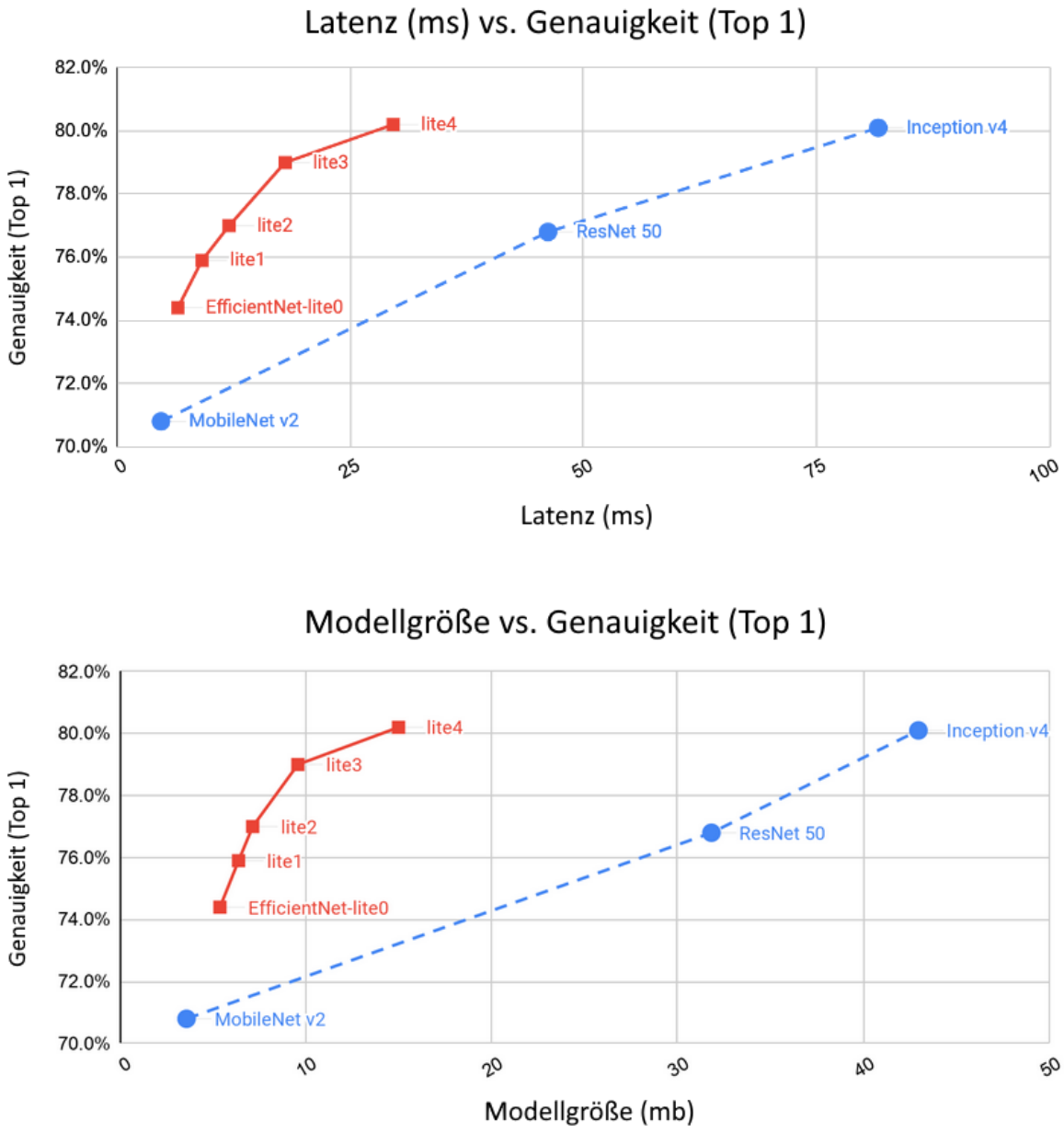


Abb. 12: Vergleich von verschiedenen Modellen

Der dritte Schritt war nun, die Bilder und Label zu importieren.

```
# Label und Bilder importieren
train_data = object_detector.DataLoader.from_pascal_voc(train_directory, train_directory, label_map={1: "toy_human"})
```

Abb. 13

Jetzt konnten wir mit unseren Daten und Einstellungen ein Modell erstellen.

```
# Das Modell erstellen
model = object_detector.create(train_data,
                               model_spec=spec,
                               batch_size=8,
                               train_whole_model=True,
                               validation_data=train_data)
```

Abb. 14

Zuletzt exportierten wir unser Modell in einen Ordner unserer Wahl.

```
# Export the Model
model.export(export_dir=r"/home/user/Desktop/Tensorflow/Spielzeugmensch")
```
Abb. 15

3.5. Wie viele Label brauchen wir?

Um nicht hunderte Bilder zu labeln, was sehr viel Zeit in Anspruch nimmt, führten wir ein Experiment durch, bei dem wir die Erkennung von Playmobilfiguren mit verschiedenen neuronalen Netzen testeten, die mit verschiedenen Anzahlen von Labeln trainiert wurden.

3.5.1. Was wir erreichen wollen

Wir wollten eine effiziente Erkennung der Playmobilfiguren für unsere finale Anwendung finden, die möglichst wenige Label braucht.

3.5.2. Was wir vermuten

Wir waren der Meinung, dass sich ein Punkt findet, an welchem eine gewünschte Genauigkeit erreicht wird, ohne zu viele Label zu verwenden und dass sich ein großer Unterschied, gerade mit den ersten Labelanzahlen zeigt. Des Weiteren vermuteten wir, dass ab einer bestimmten Anzahl von Labeln die Genauigkeit sich nicht mehr spürbar verändert und deshalb ab dort diese Modelle uninteressant für uns wären.

3.5.3. Versuchsaufbau

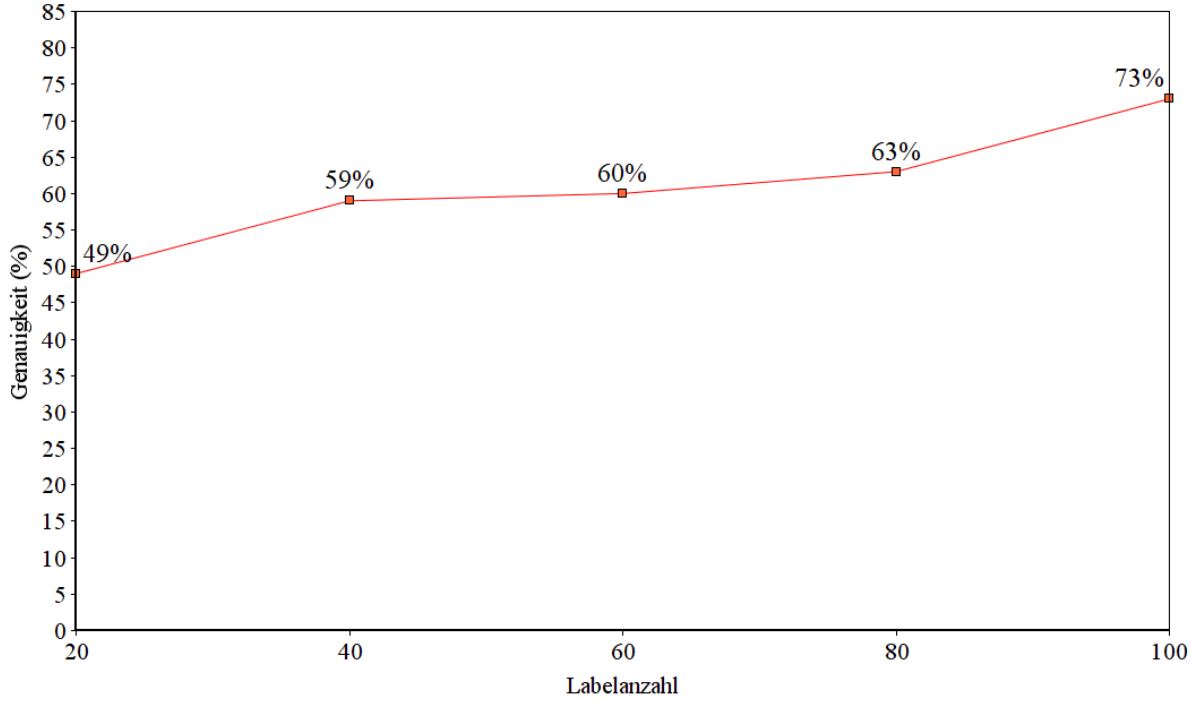
Für unseren Versuch verwendeten wir Modelle mit jeweils 20, 40, 60, 80 und 100 Labeln. Wir verwendeten jedes Mal dieselben Bilder und fügen immer 20 weitere hinzu. Die Netze wurden dann mit den gleichen 27 Testbildern überprüft. Diese Bilder haben insgesamt 100 Figuren, womit wir dann sehen konnten, wie oft die verschiedenen Netze diese Playmobilfiguren erkennen. Für unsere Modellkomplexität verwendeten wir EfficientNet-4 für die höchstmögliche Leistung. Bei diesem Versuch gilt eine Figur erst als erkannt, wenn das Modell mindestens eine 60%ige Übereinstimmung ermittelt.

3.5.4. Auswertungsaufbau

Wir berechneten aus allen erkannten oder nicht erkannten Figuren den Durchschnitt und verglichen damit die verschiedenen neuronalen Netzwerke. Aus unserem Ergebnis erstellten wir ein Diagramm über die Genauigkeit der Figurenerkennung in Abhängigkeit der verwendeten Anzahlen an Labeln.

3.5.5. Unsere Ergebnisse

Wie vermutet gab es zwischen 20 und 40 Labeln eine große Differenz, wie man in unserem Diagramm (Abb. 16) sehen kann:



Zwischen 40 und 80 Labeln gab es wiederum kaum eine spürbare Erhöhung. Was uns aber überraschte, war der Anstieg bei 80 auf 100 Labeln, welcher 10% Punkte betrug. Das lag vermutlich daran, dass die letzten 20 Label eine Ansicht auf die Figuren von oben zeigte, womit das Modell deutlich mehr Figuren aus diesem Winkel erkannte.
Was wir daraus mitnehmen können ist, dass nicht nur die Anzahl der Bilder eine große Rolle spielt, sondern auch die verschiedenen Winkel und Positionen, aus welcher man seine Bilder fotografiert.

3.6. TFlite Modell mithilfe des Raspberry zur Erkennung verwenden

Dank unseres vorherigen Versuches wussten wir bereits, wie man ein möglichst effizientes Modell erstellt, also konnten wir unser Modell nun vollständig nutzen. Damit es aber mit unserem Leitsystem verwendbar ist, benötigten wir eine Möglichkeit, die Objekterkennung auf möglichst kleinem Raum anzuwenden. Und dafür verwendeten wir einen Raspberry 4, welcher gewissermaßen ein “Mini-Computer" ist und sich somit perfekt für unseren Zweck eignet. Der Raspberry hat zusätzlich eine Kamera angeschlossen, damit wir einen Input für unsere Objekterkennung haben. Da Objekterkennung aber oft ziemlich rechenaufwändig ist, verwenden wir einen Coral Co-Prozessor, welcher für TensorFlow Lite optimiert ist, um unsere Erkennung zu beschleunigen.
Auf dem Raspberry verwenden wir wieder ein Python Skript, welches als erstes unser Modell importiert.

```
# Tensorflow Modell importieren
interpreter = Interpreter(model_path="/home/user/Desktop/Tensorflow/Model.tflite",
                          experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
```

Abb. 17

Als zweites starten wir die Kamera.

```
# Kamera initialisieren
videostream = VideoStream(resolution=(imW,imH),framerate=30).start()
```

Abb. 18

Jetzt beginnen wir mit der Hauptschleife, welche jedes Mal folgende Funktionen wiederholt:

1. Ein Bild von der Kamera holen
2. Die eigentliche Objekterkennung ausführen, ohne grafischen Output
3. Einen Rahmen um erkannte Objekte zeichnen, falls das Modell eine Sicherheit von 50% hat
4. Label von den erkannten Objekten zeichnen
5. Das Bild dem Benutzer anzeigen

```
# Hauptschleife für Objekterkennung
while True:

    # Bild aus Videostream erfassen
    frame1 = videostream.read()

    # Eigentliche Erkennung mit unserem Bild als Input durchführen
    set_input_tensor(interpreter, input_data)
    interpreter.invoke()

    # Erkennungsergebnisse abrufen
    boxes = get_output_tensor(interpreter, 1) # Koordinaten von erkannten Objekten
    classes = get_output_tensor(interpreter, 3) # Labelnamen von erkannten Objekten
    scores = get_output_tensor(interpreter, 0) # Zuverlässigkeit von erkannten Objekten
   # Über alle erkannten Objekte durchlaufen und bei 50% Sicherheit zeichnen
   for i in range(len(scores)):
       if ((scores[i] > 0.5) and (scores[i] <= 1.0)):

           # Rahmen um erkanntes Objekt zeichnen
           ymin = int(max(1,(boxes[i][0] * imH)))
           xmin = int(max(1,(boxes[i][1] * imW)))
           ymax = int(min(imH,(boxes[i][2] * imH)))
           xmax = int(min(imW,(boxes[i][3] * imW)))

           cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

           # Label anzeigen
           object_name = labels[classIndex]
           label = '%s: %d%%' % (object_name, int(scores[i]*100))
           labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)
           label_ymin = max(ymin, labelSize[1] + 10)
           cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10),
                         (xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED)
           cv2.putText(frame, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)
    # Bilder pro Sekunde anzeigen
    cv2.putText(frame,'FPS: {0:.2f}'.format(frame_rate_calc),(30,50),
                cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,0),2,cv2.LINE_AA)

    # Unser fertiges Bild anzeigen
    cv2.imshow('Object detector', frame)
```

Abb. 19 bis 21

Das fertige Ergebnis sieht so aus:

Abb. 22

3.7. Funktioniert eine Objekterkennung in einem Brandfall?

Um die Funktionstüchtigkeit unseres Gebäudeleitsystem für Feuerwehreinsätze beim Brandfall auf echte Personen übertragen zu können, fanden wir im Internet ein bereits trainiertes neuronales Netzwerk von TensorFlow, welches wir für unseren Versuch verwendeten. Keine Informationen konnten wir allerdings darüber finden, ob bei Rauchentwicklung die Objekterkennung überhaupt noch funktioniert. Um die Zuverlässigkeit der Objekterkennung bei Rauchentwicklung sicherzustellen, entschieden wir uns dies experimentell zu testen.
Wir vermuteten, dass die Software bereits bei geringer Rauchentwicklung die Personen nicht mehr sicher erkennen würde.
Für unser final angelegtes Experiment versuchten wir möglichst realitätsnahe Rahmenbedingungen zu schaffen. In einem typischen Unterrichtsraum mit weißen Wänden stellten wir eine Nebelmaschine drei Metern entfernt von zwei Versuchspersonen auf. Rechts von der Nebelmaschine befand sich die Kamera des Raspberry 4, welche auf einem Stativ fixiert war. Die Kamera war auf die Versuchspersonen gerichtet und gab die Bilddaten an den Raspberry zur Verarbeitung weiter. Der Raspberry 4 erkannte mithilfe des Google Coral USB Accelerator die Personen und gab die Bilder an den Monitor weiter.

Bei eingeschaltetem Licht und ohne Rauch konnte die Software uns sehr gut (ca.90%) erkennen.

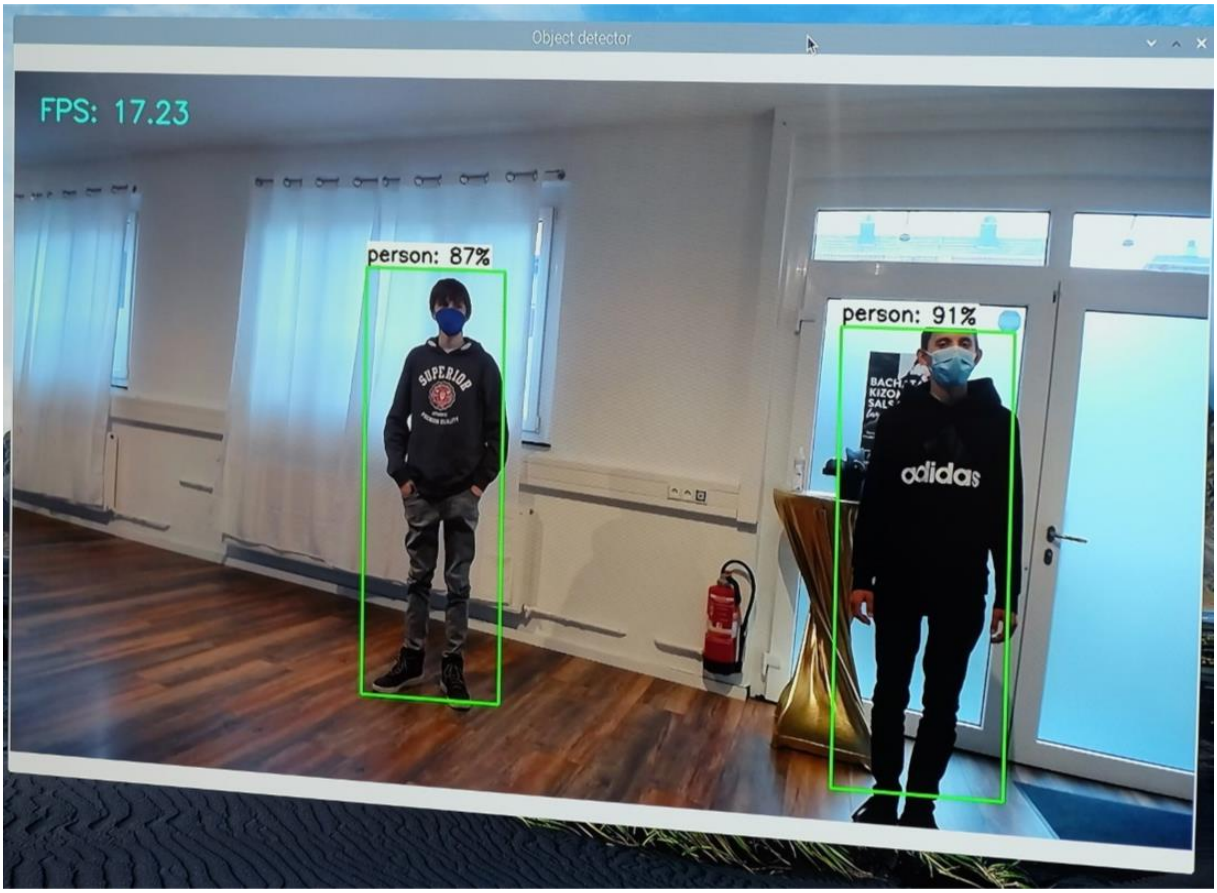


Abb. 23

Bei sehr wenig Rauch war die Erkennungsrate immer noch sehr hoch. Die Software erkannte uns nur um 3% schlechter.

Das änderte sich erst, als zum Beispiel die Füße durch den Rauch nicht mehr sichtbar waren. Die prozentuale Erkennungsrate verringerte sich auf 73%.

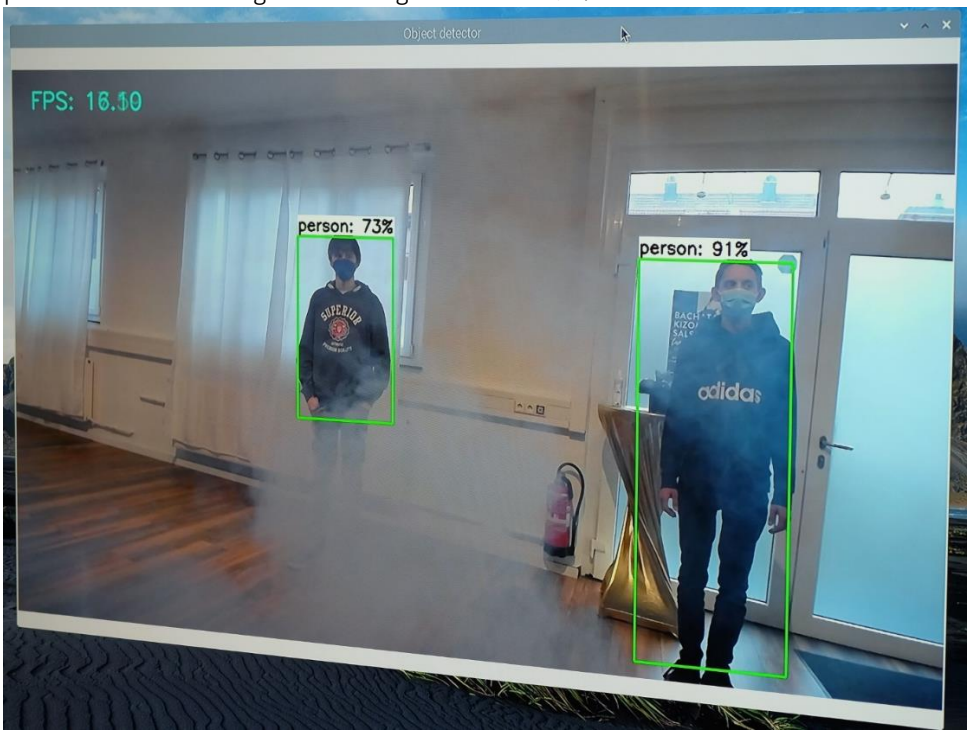


Abb. 24

Als der Nebel den Kontrast stark verringerte, wurden die Personen schlechter erkannt (etwa 70%).

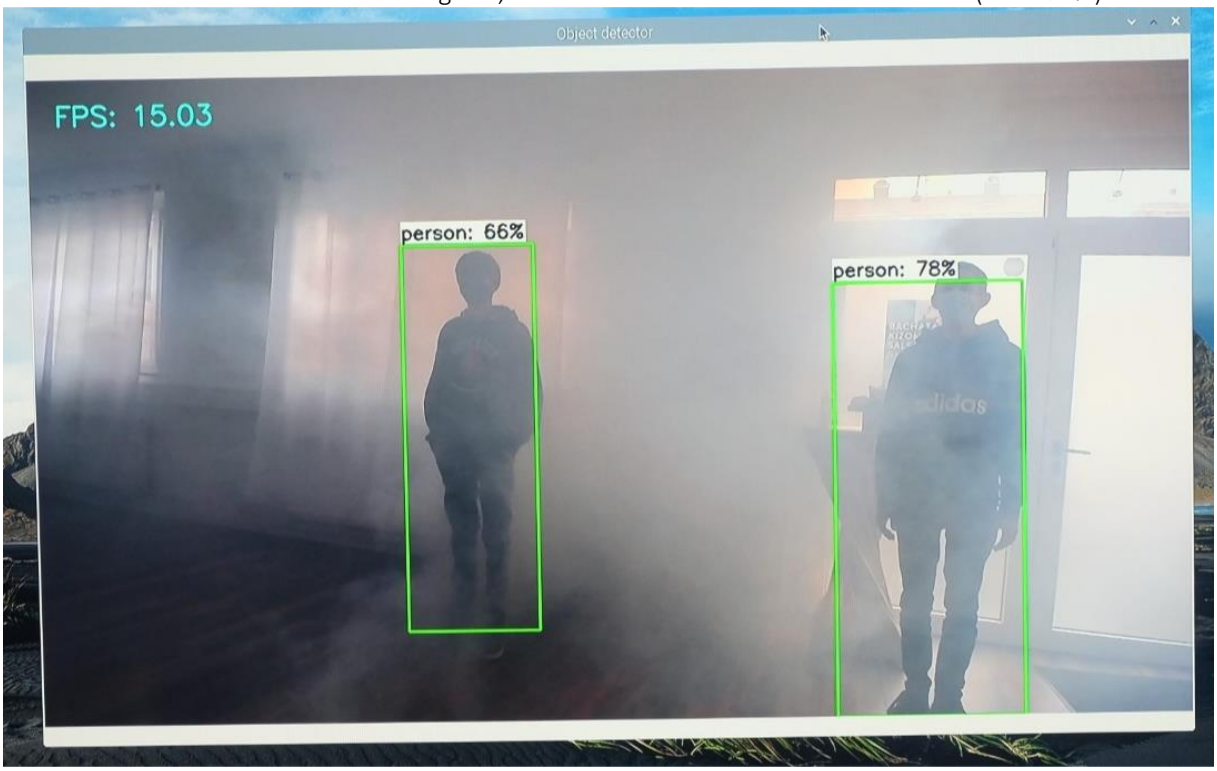


Abb. 25

Bei ausgeschaltetem Licht und geringem Nebel konnte die Software die Personen gut erkennen, da der Hintergrund weiß war und durch die Fenster genügend Licht einfiel.
Bei unserem Experiment stellten wir fest, dass die beiden Testpersonen bei einem dunklen Hintergrund ohne Fenster nicht mehr zuverlässig erkannt werden konnten.

Abb. 26

Insgesamt stellten wir fest, dass die Personenerkennung von TensorFlow die beiden Testpersonen sehr gut bei leichtem Nebel oder Rauch erkennen konnte. Nur bei sehr viel Rauch (Personen optisch kaum noch wahrnehmbar) oder bei fehlendem Kontrast, konnte keine Identifikation mehr stattfinden. Da die Erkennung mit hellem Hintergrund besser funktioniert, empfehlen wir weiße Wände und Fenster in Gebäuden, in denen das System eingebaut werden soll.

4. Literaturverzeichnis

Hermann, Stefan: WS2812 – Der einfachste Weg, viele LEDs mit Arduino steuern
https://starthardware.org/viele-leds-mit-arduino-steuern-ws2812/ [Letzter Zugriff: 5.01.2020]

TensorFlow, Warum TensorFlow
https://www.tensorflow.org/ [Letzter Zugriff: 29.12.2020]

5. Abbildungsverzeichnis