# Gefahrenmonitoring im Straßenverkehr für Radfahrer mithilfe von Objekt Detektion und einem Raspberry Pi

Herberto Werner
Department Technik und Informatik, HAW Hamburg,
Berliner Tor 7, 20099 Hamburg

28 Februar 2021

**Zusammenfassung**

Das vorliegende Paper untersucht die Funktionalität einer Gefahrenmonitoring-Anwendung für Radfahrer auf dem Raspberry Pi mithilfe von Objekt Detektion. Ziel ist es, eine Basis für die Weiterentwicklung des Gefahrenmonitorings mithilfe von Objekt Detektion zu schaffen und die Idee weiterzuvermitteln. Es wurden für das Training eigens aufgenommene Datensätze mit der Raspberry Pi Cam sowie Google Open Images Datensätze mit den zwei Klassen Auto und Fahrrad verwendet. Dabei werden drei Modelle mit verschiedenen Trainingsdatensätzen und gleichem Testdatensatz trainiert und verglichen. Das Modell mit den eigenen Datensätzen hatte die höchsten Werte bei der Precision sowie beim Recall und wurde aufgrund dessen für den Test der Funktionalität der Anwendung im Straßenverkehr gewählt. Die Ergebnisse aus der Testfahrt zeigen, dass die Anwendung eine mangelnde Funktionalität der Überwachung darstellt und keine zuverlässige Überwachung aufgrund fehlender Datensätze bietet. Kleine Objekte wurden nicht erkannt und Inkonsistenzen der Zeichnung von Begrenzungsrahmen in Echtzeit erschwerten die Gefahrendetektion und -überwachung. Jedoch wurden potenzielle Gefahrensituationen zum Teil vom Modell erkannt.

**Keywords:** Gefahrenmonitoring, Radfahrer, Objekt Detektion

## 1 Einleitung

Die Zahl der Radfahrer als auch die Zahl der Verkehrsunfälle von Radfahrern durch Fahrten innerorts und außerorts ist in den letzten 10 Jahren stark gestiegen [6]. Um diesem Trend entgegenzuwirken, bietet das Gefahrenmonitoring für Radfahrer mithilfe von Objekt Detektion einen möglichen Lösungsansatz. Im Straßenverkehr hat Objekt Detektion auf Kamerabasis in den letzten Jahren weitreichende Fortschritte durch die Architekturen „You Only look once“(YOLO), MobileNet [10] und „Single-Shot-Detectors“(SSD) [12] erzielt. Für Bilder mit geringer Auflösung zeichnen sie sich durch schnelle Inferenz und gute Performance aus. MobileNet weist darüber hinaus kompakte Modelle bei gleichzeitig schneller Inferenz für mobile oder eingebettete Systeme auf. Dadurch ist es möglich, die Objekt Detektion im Straßenverkehr in vielen Anwendungsszenarien einzusetzen. Neben den klassischen Anwendungen wie autonomes Fahren oder der Detektion von Straßenschildern gibt es die Überwachung von Radfahrern durch Überwachungskameras [8], eine Straßenschäden-Detektion und Klassifikation [13] sowie die Klassifikation, Objekt Detektion und das Tracking von sich bewegenden Objekten mithilfe multipler, teurer Sensoren wie Radar, Lidar und Kamera [7]. Jedoch gab es bisher keine Gefahrenüberwachung für Radfahrer durch ein kostengünstiges, eingebettetem System.
Das vorliegende Paper konzentriert sich auf eine Gefahrenüberwachung für Radfahrer mithilfe von Objekt Detektion und eines Raspberry Pi's. Ziel ist es, die Funktionalität der Objekt Detektion mit dem Raspberry Pi im Straßenverkehr zu untersuchen. Hierfür wird im folgenden Abschnitt als erstes der Aufbau der Hardware sowie der Aufbau für die Datenerfassung und Testfahrt im Straßenverkehr beschrieben. Anschließend werden die verwendeten Datensätze betrachtet. Es wird zwischen eigenen, annotierten Datensätzen durch die Raspberry Pi Kameraaufnahmen und Google Open Images unterschieden. Für das Training werden drei Modelle mit verschiedenen Trainingsdatensätzen und gleichen Testdaten mit den Klassen Auto und Fahrrad trainiert, um diese später bei den Ergebnissen vergleichen und bewerten zu können. Nach dem Training wird die Objekt Detektion auf dem Raspberry Pi betrachtet, da hierbei mithilfe des Flächeninhalts der Begrenzungsrahmen der erkannten Objekte die LED-Ampel angesteuert wird. Die LED-Ampel symbolisiert den aktuellen Gefahrenstatus für den Radfahrer. Anhand der Ergebnisse wurde das Modell mit den eigenen Datensätzen für die Testfahrt gewählt. Die Ergebnisse bzw. Bildaufnahmen werden zuletzt auf die Funktionalität der Gefahrendetektion im Straßenverkehr bewertet und diskutiert.

# 2 Methode

In diesem Abschnitt werden die Methoden und Schritte für die Umsetzung des Projektes beschrieben, dessen Ergebnisse im darauf folgenden Abschnitt dargestellt und erläutert werden. Alle Schritte von Konfiguration bis zum Training basieren auf der Dokumentation TensorFlow Object Detection API [17]. Die Anwendung des Modells und der Aufbau der Hardware ist in einem Video [3] zu sehen, welches die Erfassung der Autos und Fahrräder zeigt.

## 2.1 Hardware

### 2.1.1 Aufbau und Verbindungen

Folgende Hardwarekomponenten werden für den Aufbau des Gefahrenmonitoring mithilfe von Objekt Detektion und einem Raspberry Pi benötigt:

- Raspberry Pi 4 mit 2GB Ram und als Betriebssystem „Raspberry OS“
- Coral USB Beschleuniger
- LED Ampel Modul (3,3V - 5V)
- Powerbank 20Ah
- 1x USB-C Kabel und 1x 3.0 USB-Kabel für Versorgung und Datenübertragung
- 4x Flexible Drahtbrücken

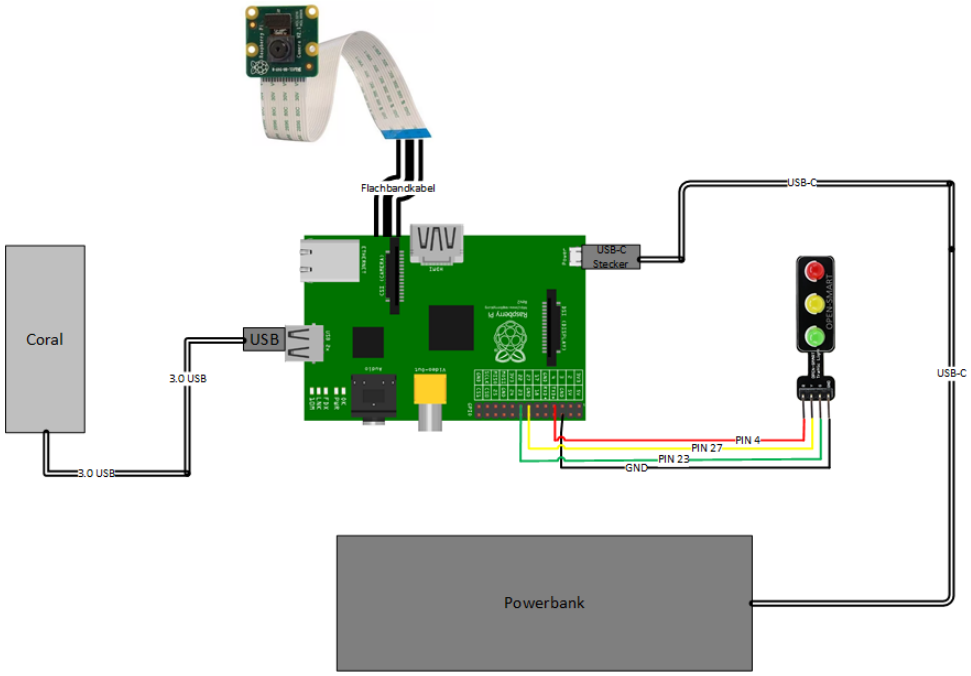


Abbildung 1: Verbindungen und Verkabelung zwischen den Hardwarekomponenten

Anhand der Abbildung 1 sieht man, dass eine Powerbank, welche eine Kapazität von 20000mAh besitzt, mit dem Raspberry Pi 4 über einen USB-C Kabel verbunden ist. Diese Verbindung dient der Spannungsversorgung des Raspberry Pi's. Der Coral USB Beschleuniger ist mit einem 3.0 USB-Kabel mit dem Raspberry Pi verbunden. Dieser dient der Beschleunigung der Inferenz auf dem Raspberry Pi. Der Coral USB Beschleuniger besitzt einen leistungsfähigen Spezialchip (TPU), welcher es ermöglicht, die Inferenz Tensor Flow Lite Modelle für Echtzeit-Anwendungen für Künstliche Intelligenz zu beschleunigen [9] [15]. Ein Flachbandkabel ist an die Raspberry Cam angebunden, welche in Echtzeit den Straßenverkehr erfasst. Über vier flexible Drahtbrücken wird eine LED-Ampel über die verschiedenen Pins angesteuert und mit Spannung versorgt.

### 2.1.2 Hardware Setup für die Datenerfassung im Straßenverkehr

Für die Objekt Detektion wurden eigene Datensätze im Straßenverkehr gesammelt. Durch die Hardware in Abschnitt 2.1.1 ergibt sich folgender Aufbau für die Datenerfassung im Straßenverkehr (siehe Abbildung 2)

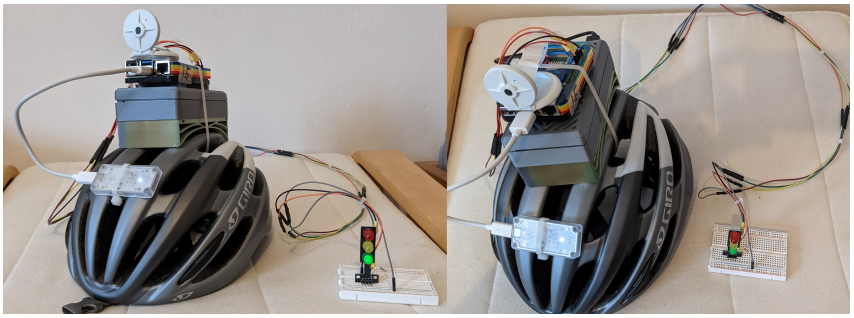

Abbildung 2: Hardware Setup für die Datenerfassung im Straßenverkehr

Wie in Abbildung 2 zu sehen, befinden sich die Powerbank (Spannungsversorgung) und die Raspberry Pi mit Cam auf dem Helm. Durch die hohe Position der Cam ist eine weite Sicht bzw. eine größere Entfernung für das Monitoring im Straßenverkehr möglich. Autos können zum Beispiel bei Abbiegevorgängen schneller erfasst und überwacht werden. Der Neigungswinkel der Cam liegt zwischen 1-5°. Die LED-Ampel wird am Handgelenk befestigt. Somit kann sie dem Radfahrer den aktuellen Gefahrenstatus sowie das Sicherheitsrisiko durch Autos und andere Fahrräder des Straßenverkehrs signalisieren.

## 2.2 Datensätze

In diesem Abschnitt wird zwischen zwei Datensätzen unterschieden. Diese werden als Input für das Netz im Training eingesetzt. Im ersten Datensatz sind die aufgenommenen Datensätze mit der Raspberry Cam. Der zweite Datensatz besteht aus heruntergeladenen, gelabelten Bildern mit Begrenzungsrahmen aus dem Google Open Images Datensatz.

### 2.2.1 Aufgenommene Datensätze mit der Raspberry Pi Cam

Für die Aufnahme des Straßenverkehrs wird die PiCamera Bibliothek [11] verwendet. Hierfür wird ein Python Skript für die Datensammlung ausgeführt. Das Skript nimmt den Straßenverkehr als Videostream auf. Die Auflösung wurde für die Aufnahme auf 640x480 reduziert, da der Raspberry Pi eine begrenzte Speicherkapazität besitzt. Beim Videostream trat das Problem auf, dass stark, unerkennbare Verzerrungen auftraten (siehe Abbildung 3). Objekte wie Radfahrer, Autos oder Fußgänger waren auf diesen Bildaufnahmen nicht zu erkennen. Jedoch war es möglich, die benötigten, unverzerrten Bilder aus den Videos zu extrahieren.


Abbildung 3: Verzerrte Aufnahmen durch kontinuierliche Bildaufnahmen

### 2.2.2 Annotation der eigenen Datensätze

Nach der Datenaufnahme werden anhand der Videos Screenshots aufgenommen. Auf den Screenshots sind die Autos und/oder Fahrräder im Straßenverkehr enthalten. Die Autos und Fahrräder werden auf den Bildern für das Training mit Begrenzungsrahmen gelabelt. Es sind für die Warnung vor anderen Radfahrern nur die Fahrräder und nicht die Personen auf den Rädern gelabelt worden, da die Google Open Images Datenbank keine Bilder von Radfahrern enthält. Für die spätere Identifikation der Objekte werden Begrenzungsrahmen gezeichnet. Die Details der Bilder mit den Koordinaten der Rahmen werden als Pascal VOC Format gespeichert. Es müssen möglichst alle Autos/Fahrräder auf einem Bild einen Begrenzungsrahmen erhalten, da sonst die Performance und Genauigkeit eines Netzes (siehe Abbildung 5) sinkt [18]. Die Begrenzungsrahmen sind auf der Abbildung 4 zu

finden. Autos und Fahrräder sind die beiden einzigen Klassen für die Objekt Detektion. Die Aufnahmen sind nach Abbildung 4 zwischen bewölkt und sonnig zu unterscheiden. Es konnten insgesamt 422 Bilder mit einer Größe von 640x480 RGB im bewölkten und/oder sonnigen Wetter aus den Videos durch Screenshots extrahiert werden. Eine Schwierigkeit ist hier, dass durch die Wetterbedingungen verschiedene Licht- und Schattenverhältnisse entstehen. Daher ist es entscheidend, nicht nur Datensätze im bewölkten Wetter aufzunehmen, sondern auch im sonnigen, wolkenlosen Wetter. Dadurch kann das Modell im Gebrauch über verschiedene Lichtverhältnisse generalisieren. Der Fokus der eigenen Datensätze lag auf der Qualität der Bilder. Dies bedeutet, dass möglichst unterschiedliche Aufnahmen aus verschiedenen Winkeln, Lichtverhältnissen und Fahrrad- und Autotypen aufgenommen wurden.

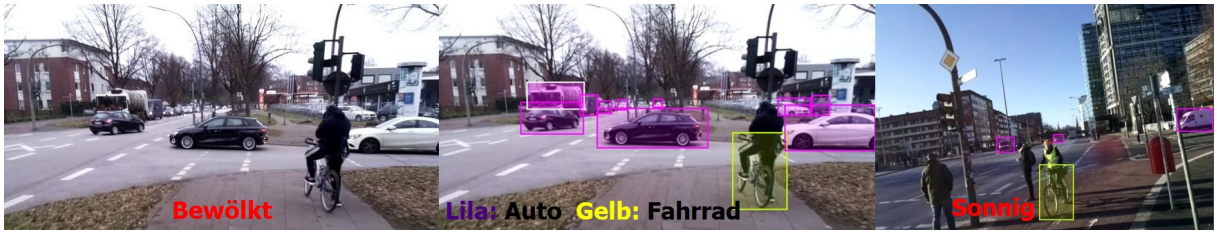


Abbildung 4: Datenaufnahme mit und ohne Begrenzungsrahmen in bewölktem und sonnigem Wetter

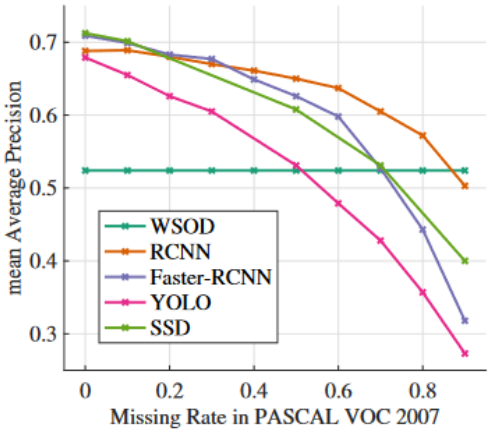


Abbildung 5: Entnommen aus [18] und zeigt die Performance (Mean Average Precision) der Netze bei fehlenden, gezeichneten Rahmen

### 2.2.3 Datensätze von Google Open Images und Probleme

Neben den aufgenommenen Datensätzen mithilfe der Raspberry Cam wurden Datensätze von Google Open Images verwendet. Google Open Images verfügt über ca. 1,9 Millionen Bilder mit 600 Objektklassen und 16 Millionen Begrenzungsrahmen [4]. Für das Training werden jeweils 3000 Auto- und Fahrraddatensätze mit bereits vorhandenen Begrenzungsrahmen heruntergeladen. Diese Datensätze stammen aus der Plattform „Flickr“. Es stellte sich bei den Google Open Images als Problem heraus, dass viele Koordinaten der Begrenzungsrahmen über die Bildkoordinaten (xmax, xmin, ymax, ymin) hinausgingen. Die Problematik sieht man anhand Abbildung 6 Bild 1. Im späteren Verlauf führte dies zu Problemen beim Training der Modelle. Ein weiteres Problem mit den Datensätzen von Google Open Images stellte dar, dass auf den Auto-Datensätzen nur Autos gelabelt wurden und Fahrräder ignoriert wurden. Die Problematik sieht man an Abbildung 6 Bild 1. Anhand dieser Probleme wurde Roboflow für die Bearbeitung aller Google Open Images gewählt. Roboflow ist eine Plattform, welche ein Toolset für die Datenvorverarbeitung wie Filter, Datenaugmentation bis zum Trainieren eines Modell anbietet. Durch den Upload aller Google Open Images Datensätze auf Roboflow [2] wurden alle Begrenzungsrahmen der gelabelten Bilder korrigiert. Neben der Korrektur wurden die Bilder ebenfalls auf 416x416 RGB skaliert. Dieser Ansatz hat die Datenvorverarbeitung und die Vorbereitung für das Training beschleunigt.

### 2.2.4 Mischung der Datensätze

In dem dritten Datensatz befinden sich sowohl die selbst aufgenommenen Bilder als auch die von Google Open Images. Diese wurden ebenfalls bei Roboflow hochgeladen. Die Google Open Images Datensätze wurden hier ebenfalls korrigiert und auf 416x416 RGB skaliert.


Abbildung 6: In den Google Open Images Datensätzen sind nicht alle Begrenzungsrahmen, wie in Bild 1 zu sehen, gezeichnet worden. Hinzu kommt, dass viele Begrenzungsrahmen über ymax oder xmax, wie in Bild 2 zu sehen, hinausgehen

## 2.3 Konfiguration des eigenen Objekt-Detektors

Für das Training müssen zwei wichtige Vorkonfigurationen für einen eigenen Objekt-Detektor vorgenommen werden.

1. **Label Map**
   Es wird eine Label Map erstellt, die den beiden Klassen Äutoünd Fahrradëine ID zuordnet
2. **Trainings-Pipeline**
   Mit der Trainings-Pipeline „pipeline.config “wird der gewollte Ablauf des Trainings für Tensorflow beschrieben. Die Batchsize wird auf 12 gesetzt. Ein weiterer Parameter, welcher konfiguriert wird, ist die Zahl der Epochen. Die Zahl der Epochen wird auf 100000 gesetzt.

Neben der Erstellung der Label Map und Konfiguration der Pipeline werden csv-Dateien aus den Trainings- und Testdaten erstellt. Diese enthalten die Informationen eines Bildes wie Höhe, Breite, Klasse und die Bildkoordinaten (ymax, ymin, xmax, xmin) der Begrenzunsgrahmen. Die CSV zusammen mit den Bildern und den Begrenzungsrahmen werden in das von Tensorflow lesbare binäre Format TFrecord umgewandelt. Die TFRecords werden für die Pipeline, welche das Modell für das Training konfiguriert, verwendet. Die aufgeführten Probleme bei Abbildung 6 führen dazu, dass ebenfalls die CSV-Dateien und TFRecords die falschen Koordinaten durch Begrenzungsüberschreitungen erhalten. Die Folge daraus ist, dass das Training anhand der falschen Argumenten aus den TFRecords fehlschlägt.

## 2.4 Training mit SSD MobileNet

Als Basis für das Training wird eine Single-Shot-Detektor (SSD) Architektur mit dem vortrainierten Netz „ssd_mobilenet_v1 _quantized_300x300_coco14“gewählt. Single-shot detectors (SSD) sind für Objekt Detektion in Echzeit konzipiert [12]. Die Lokalisierung und die Klassifikation der Begrenzungsrahmen in einem Bild geschieht im gleichen Moment. Diese Methode ist schnell und eignet sich daher für die Detektion im Straßenverkehr [12]. Aus dem Modelzoo [16] wird SSD MobileNetV1 Quantized verwendet. MobileNet ist mit dem COCO, Kitti, Open Images, AVA v2.1, iNaturalist und Snapshot Serengeti Datensatz trainiert und als SSD Architektur für mobile oder eingebettete Anwendungen wie die Objekt Detektion auf dem Raspberry Pi geeignet. Das vortrainierte Modell ist quantisiert und benutzt statt 32-Bit Floatwerte 8-Bit Integerwerte im neuronalen Netz. Dadurch wird das Modell kompakter und kann durch eine TPU beschleunigt werden [14]. Die Inputgröße des neuronalen Netzes liegt bei 300x300 RGB [10]. Durch die pipeline.config werden die Datensätze von einer Größe von 640x480 und 416x416 auf 300x300 als Input runterskaliert. Die Datensätze durchlaufen 28 Convolutional Layer mit einer Batchnormalisation und der Aktivierungsfunktion ReLu. Am Ende des Netzes befindet sich der Softmax Layer. Hier findet die Klassifikation zwischen der Klasse Auto und Fahrrad mit den Werten 0 oder 1. Das fertige Modell muss auf den Raspberry Pi transferiert und für die Egde TPU kompiliert werden. Dieser Prozess wird in Abschnitt 2.6 beschrieben.

## 2.5 Modelle

Es wurden drei Modelle mit verschiedenen Datensätzen und mit dem gleichen Netz „ssd_mobilenet_v1_quantized _300x300_coco14“ trainiert. Der Testdatensatz ist bei allen Modellen gleich, um die Modelle später vergleichen zu können. Im zweiten und dritten Modell gehören alle eigenen Datensätze zum Testdatensatz. Anhand des Testdatensatzes werden die Modelle auf den Daten im Straßenverkehr getestet. Das soll die Funktionalität für den Gebrauch im Straßenverkehr prüfen.

1. Modell mit eigenen Datensätzen (422), davon 80 % Trainingsdaten und 20 % Testdaten
2. Modell mit Google Open Images (6000), davon 80 % Trainingsdaten und 20 % Testdaten
3. Modell mit Google Open Images Daten und eigene Datensätze, davon 80 % Trainingsdaten und 20 % Testdaten. Bei den Testdaten sind alle eigene Datensätzen (422) als Testdaten mit Google Open Images verwendet worden.

## 2.6 Objekt Detektion auf dem Raspberry Pi

Das fertig trainierte Modell wird für den Raspberry Pi in ein TensorFlow Lite (tflite) Modell umgewandelt. Mithilfe von Python3 und OpenCV wird ein Skript verwendet, welches das tflite modell in ein neues tflite modell kompeliert, das mit der Egde TPU kompatibel ist. Das Skript basiert auf „detect_webcam.py “[5]. Das Skript zeichnet Boxen und Labels um einen Videostream. Mit der Coral Edge TPU wird die Inferenzzeit von 1 FPS auf 8-10 FPS bei Videostreams beschleunigt.
Der folgende Codeausschnitt 1 des Skriptes zeigt den Loop über die Detektionen und Ansteurung der LED-Ampel. Für die Aufnahme im Straßenverkehr werden kontinuierliche Bildaufnahmen gemacht (Siehe Zeile 14 bis 15 von Code 1). Hierfür wird ein aktueller Zeitstempel und eine hochzählende Rahmennummer für einen Bildrahmen verwendet. Ein Video würde die Grenze der Speicherkapazität des Raspberry Pi schnell erreichen. Für die Ansteuerung der LED-Ampel werden die Koordinaten der Begrenzungsrahmen der erkannten Objekte im Straßenverkehr verwendet. Zeile 8 von Code 1 zeigt, dass hierfür die Fläche mithilfe der xmax, xmin, ymax, ymin Koordinaten berechnet wird. Die berechnete Fläche ändert sich je nach Entfernung des erkannten Objektes. Je näher ein Auto oder ein Fahrrad sich vor dem Helm befindet, desto größer werden die Begrenzungsrahmen gebildet. Daraus folgend haben diese einen größeren Flächeninhalt. Mit diesem Ansatz wird die LED-Ampel (Grün, Gelb, Rot) bei Zeile 20 bis 38 von Code 1 angesteuert. Bei der Objekt Detektion auf dem Straßenverkehr haben die Farben der LED-Ampel folgende Definitionen:

1. **Grün**
   Weite Entfernung eines Autos bzw. Fahrrads. Es besteht keine Gefahr.
2. **Gelb**
   Mittlere Entfernung eines Autos bzw. Fahrrads. Achtung, achte auf Autos und Fahrräder.
3. **Rot**
   Geringe Entfernung eines Autos bzw. Fahrrads. Es besteht Gefahr! Ein Unfall oder gefährliche Situation ist sehr wahrscheinlich.

```
1  # Loop ueber alle Detektionen und zeichne ein Begrenzungsrahmen um die Detektion, wenn
       die Detektionssicherheit ueber Minimum Threshold liegt. Postive ab 50 Prozent
2      for i in range(len(scores)):
3          if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):
4
5              # Gebe Begrenzungsrahmenkoordinaten zurueck and zeichne Begrenzungsrahmen
6              ...
7              # Berechne Flaecheninhalt des Begrenzunsrahmen
8              rectangleSize = (xmax-xmin) * (ymax-ymin)
9              # Zeichne Label
10             ...
11
12             #Speichere die kontinuierlichen Bildaufnahmen
13             cv2.imwrite(f'{now}_frame_{frameNum}.png', frame)
14             frameNum +=1
15
16             ## Zustand Gruen: Skript laeuft
17             green_pin.on()
18
19             #Status: ROT (Nahe Distanz)
20             if rectangleSize >= 250000:
21                 green_pin.off()
22                 yellow_pin.off()
23                 red_pin.on()
24                 rectangleSize = 0
25
```

```
            #Status: GELB (Mittlere Distanz)
            elif rectangleSize < 250000 and rectangleSize >= 100000:
                red_pin.off()
                green_pin.off()
                yellow_pin.on()
                rectangleSize = 0

            #Status: GRUEN (Weite Distanz)
            elif rectangleSize < 100000:
                yellow_pin.off()
                red_pin.off()
                green_pin.on()
                rectangleSize = 0
```

Code 1: Ansteuerung der LED-Ampel und Loop über die Detektion

# 3 Ergebnisse

In diesem Abschnitt werden die drei trainierten Modelle verglichen und ausgewertet. Es werden die Unterschiede und Gemeinsamkeiten betrachtet. Für den Vergleich der drei Modelle wird eine Metrik-Tabelle „Coco Evaluation Metric“ der trainierten Modelle verwendet [1]. Aus der Tabelle 1 können die Average Precision und Recall Werte für die einzelnen Objektgrößen entnommen werden. Für die Auswertung der Modelle wird vor allem der Average Recall betrachtet. Der Average Recall ist hier wichtig, da das Monitoring auf dem Straßenverkehr keine Gefahren übersehen darf. Die Kosten für eine übersehene Gefahr ist hoch, da der Radfahrer nicht vor einer Gefahr gewarnt würde und zu Schaden kommen könnte. Aus der Spalte des Average Recalls hat das erste trainierte Modell aus den selbst erstellten Datensätzen bei allen Objektgrößen die höheren Werte im Vergleich zu den anderen Modellen. Die Average Recall Werte werden in allen Modellen mit steigender Objektgröße größer. Dies bedeutet, dass kleine Objekte bei allen Modellen kaum oder sogar gar nicht erkannt werden. Im Gegensatz dazu können alle Modelle große Objekte besser erkennen. Aufgrund der höheren Werte der Average Precision und Recall wird für die Testfahrt im Straßenverkehr das erste Modell mit den eigenen Daten gewählt.

| **Datensatz für das Training und Modell** | **Area** | **Average Precision** | **Average Recall** |
|---|---|---|---|
| Modell 1(Eigener Datensatz) | small | 0,021 | **0,02** |
| | medium | 0,235 | **0,286** |
| | large | 0,536 | **0,58** |
| Modell 2(Google Open Images) | small | 0,001 | 0,009 |
| | medium | 0,057 | 0,118 |
| | large | 0,341 | 0,476 |
| Modell 3(Eigener Datensatz und Google Open Images) | small | 0,003 | 0,033 |
| | medium | 0,094 | 0,203 |
| | large | 0,45 | 0,558 |

Tabelle 1: Average Precision und Average Recall der drei trainierten Modelle

### 3.0.1 Auswertung der Bilder

Für die Auswertung der Bilder wird zwischen den Bildern vom Straßenverkehr und dem Tensorboard unterschieden. Das Tensorboard ist ein Feature von TensorFlow, welches verschiedene Trainings- und Auswertungsmetriken überwacht und visualisiert. Abbildung 7 zeigt die Ergebnisse vom Tensorboard. Es wird die Performance der trainierten Netze hinsichtlich der detektierten Objekte und Genauigkeit verglichen. In der ersten Spalte sind die Detektionen durch das jeweilige Modell zu sehen. In der zweiten Spalte sind die händischen Annotationen zu sehen. Das Bild 7 zeigt, dass das erste Modell die meisten Detektionen und höchsten Genauigkeiten vorweist. Auf der anderen Seite erkennt das zweite Modell gar kein Objekt auf der Straße. Das erste und dritte Modell unterscheiden sich in der Genauigkeit und Anzahl der erkannten Objekte. Andere Bilder zeigen ähnliche Ergebnisse und bestätigen die Unterschiede in den Average Precision und Recall Werten der Tabelle 1.

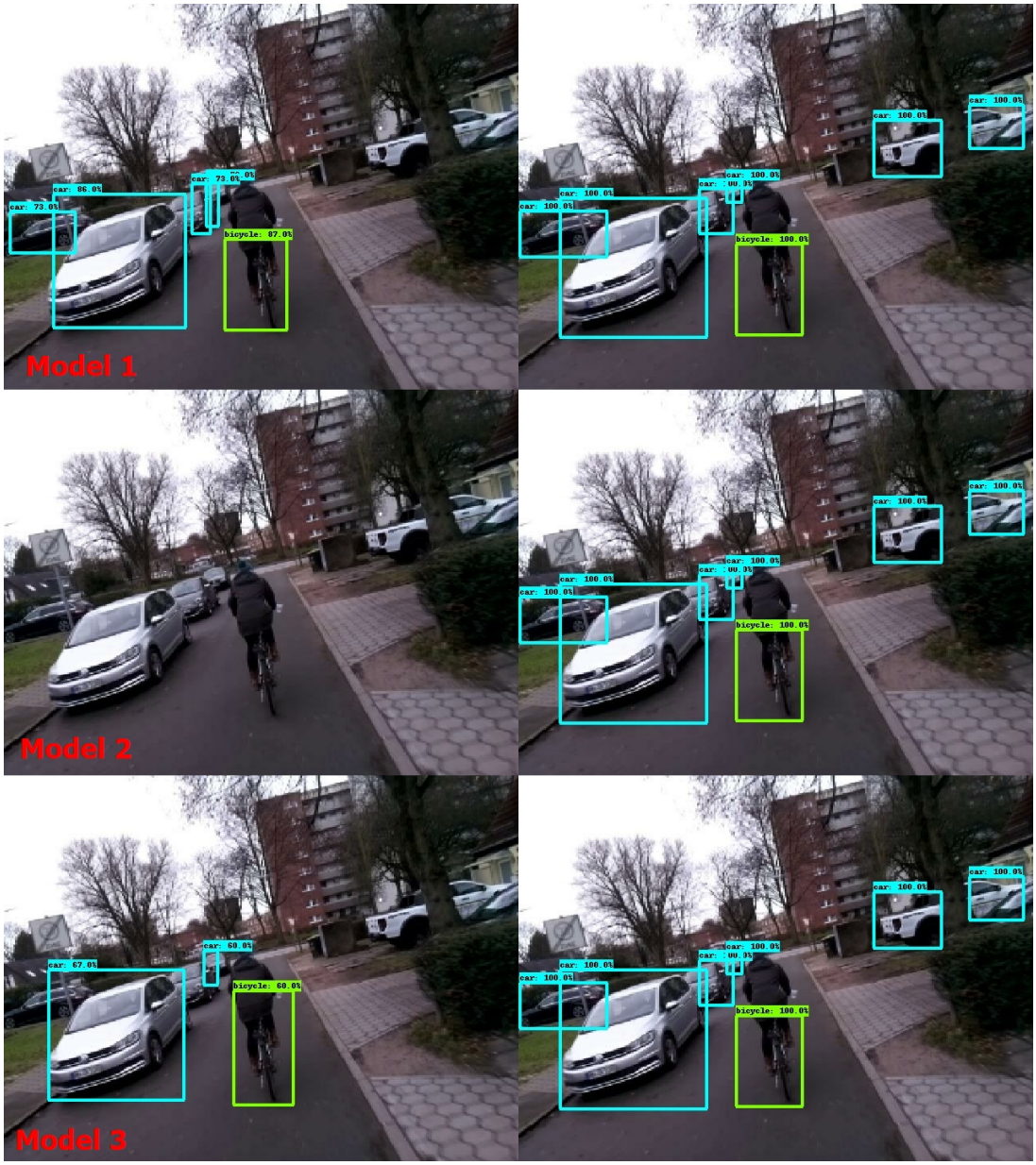


Abbildung 7: In diesem Bild sieht man den Unterschied der erwarteten, händischen Annotation und der wirklichen Detektion der trainierten Modelle

Für die Aufnahme der Bilder im Straßenverkehr wurde anhand der Ergebnisse (Tabelle 1 und Abbildung 7) das erste Modell genommen. Die Aufnahme der Bilder erfolgte nur mit 8-10 FPS, weil die aufgenommenen Bilder auf dem Raspberry gespeichert wurden, um sie im Nachhinein analysieren zu können. Folgende Bilder (siehe Abbildungen 8 und 9) zeigen die Ergebnisse und die Probleme des Modells im Straßenverkehr auf. Die Genauigkeiten der erkannten Objekte lagen bei 72, 87 oder 94 Prozent. Diese variierten je nach Entfernung und Objekt. In Abbildung 8 in der ersten Bildreihe von Abbildung 8 sieht man, dass trotz der Einschränkungen der Raspberry Pi Cam weit bis mittelweit entfernte und große Objekte erkannt wurden. Häufig wurden Autos mit großer Genauigkeit erkannt. Der Gefahrenstatus war in diesen Bildsituationen Grün. Die LED-Ampeln bzw. der Gefahrenstatus wechselten häufig von Grün auf Gelb. Nahe bis sehr nahe Objekte (0-2m), welche durch den Gefahrenstatus Rot repräsentiert werden, wurden oft nicht erkannt. Abbildung 8 Bildreihe 1 und 2 zeigen verschiedene Ampelphasen/Gefahrenstatus. Bild 2 von Bildreihe 2 zeigt eine Situation für den Gefahrenstatus Rot. Hierbei handelte es sich um ein parkendes Auto auf einer Fahrradstraße. Vorbeifahrende Autos, die jeweils an dem Helm mit dem Raspberry Pi vorbei fuhren, veranlassten viele Wechsel des Gefahrenstatus von Grün zu Gelb. Diese Wechsel sah man auch beim Warten an einer Ampel, wo viele Autos (siehe Bildreihe 2 und Bild 1 von Abbildung 8) vorbeifuhren.


Abbildung 8: Hier sind die verschiedenen Ampelphasen und die Entfernungen der Detektionen aufgezeigt

**Probleme**
Bei allen Bildern war zu beobachten, dass es eine Inkonsistenz der gezeichneten Begrenzungsrahmen gab. Bei längerem Fokussieren eines Objektes wurde nur kurzzeitig das Objekt erkannt und ein Begrenzungsrahmen gezeichnet. Dadurch wechselten die Farben des Gefahrenstatus sehr oft. Neben den Autos wurden auch Fahrräder erkannt. Diese wurden meistens bei mittlerer bis naher Distanz detektiert. Bild 3 von Bildreihe 2 von Abbildung 8 zeigt eine Detektion mit kurzer Distanz. Hier trat das Problem auf, dass Fahrräder bei allen Distanzen nicht als Gefahr erkannt worden sind. Der Gefahrenstatus blieb hierbei meistens auf Grün. Hinzu kommt, dass Fahrräder siehe Bild 2 von Abbildung 9 bei mittlerer Distanz oft nicht erkannt werden. Die Fahrräder sind von der Objektgröße klein und können vom Modell nicht einer Klasse zugeordnet werden. Auch die Beine der Fußgänger (siehe Bild 1 von Abbildung 9 oder Kinderwagen (siehe Bildreihe 2 und Bild 3 von Abbildung 8) werden öfters als Fahrräder erkannt. Ein weiteres Problem, dass bei der Auswertung der Bilder von der Testfahrt auffällig war, ist dass bei 1 von 40 Bildern zufällige Begrenzungsrahmen an zufälligen Positionen des Bildes gezeichnet worden sind. Bild 3 von Abbildung 9 zeigt einen Teil eines Baumes mit einem Begrenzungsrahmen mit dem Label Auto.


Abbildung 9: Diese Abbildung zeigt die Probleme auf, die bei der Auswertung der Bilder aufgetreten sind

# 4 Diskussion

In diesem Abschnitt wird die Fragestellung nach der Funktionalität der Anwendung auf dem Raspberry Pi diskutiert. Die Kriterien für eine funktionierende Anwendung sollten eine hohe Genauigkeit die Detektion aller Autos und Fahrräder in allen Winkeln und Entfernungen sein. Man kann sagen, dass das aktuelle, trainierte Modelle keine zuverlässige Überwachung liefert und dem Radfahrer keine Sicherheit gewährleisten kann. Mit dem trainierten Modell konnten trotzdem einige gefährliche Situationen auf dem Straßenverkehr erkannt werden, sodass unachtsame Radfahrer gewarnt werden. Ein Beispiel (siehe Abbildung 10) zeigt eine gefährliche Situation auf der Testfahrt. In dieser Situation wurde man vor dem Rechtsabbieger durch den gelben Gefahrenstatus gewarnt.
Betrachtet man jedoch das gewählte Modell näher gibt es eine Inkonsistenz bei der Zeichnung von Begrenzungsrahmen bei erkannten Objekten und eine mangelnde Funktionalität der Überwachung durch falsche oder sogar keine Detektionen. Das lässt sich damit begründen, dass aufgrund der geringen Datensätze nicht alle Lichtverhältnisse, Formen von Autos und Fahrrädern, Straßenverhältnisse- und umgebungen im Straßenverkehr abgedeckt sind. Das bildet sich auch beim Average Recall in allen Objektgrößen ab. Zum anderen kann die Berechnung des

Flächeninhalts und Werte für die LED-Ansteuerung hinsichtlich der Genauigkeit noch verbessert werden. Mithilfe eines Modells mit höheren Recall Werten können die Abfragewerte für die LED-Ansteuerung verfeinert werden. Jedoch lässt sich anhand der Ergebnisse bei Abschnitt 3 sagen, dass das aktuelle Modell eine Grundbasis für die Weiterentwicklung schafft. Mehr Datensätze von Autos, Fahrrädern und anderen Straßenumgebungen und zusätzliche Datenaugmentation für andere Lichtverhältnisse, Winkel und Farben können die Gefahrendetektion als Anwendung erheblich verbessern.


Abbildung 10: Das Bild ist ein Ausschnitt von der Testfahrt mit der Reaktion der Gefahrendetektion Anwendung in einer potenziell gefährlichen Situation

# 5 Fazit und Ausblick

Alles in allem zeigt das vorliegende Paper, dass eine Gefahrendetektion für Radfahrer am besten mit eigens gesammelten Daten möglich ist. Dennoch ist deutlich geworden, dass eine Objekt-Detektion-Anwendung auf dem Straßenverkehr auf einen sehr großen Datensatz für die Funktionalität trotz der Hardware Einschränkungen angewiesen ist.

Der Vergleich zwischen den Modellen mit den verschiedenen Datensätzen hat gezeigt, dass alle Modelle kleine oder zu nahe Objekte im Straßenverkehr mit einer sehr geringen Genauigkeit oder sogar gar nicht erkennen. Im Gegensatz dazu werden große Objekte wie Autos mit höherer Genauigkeit erkannt. Außerdem konnte der Vergleich zeigen, dass das trainierte Modell mit den eigenen Datensätzen die höchsten Genauigkeiten beim Average Recall für alle Objektgrößen hatte.

Insgesamt kann man daher darauf schließen, dass die Gefahrendetektion-Anwendung eine mangelnde Funktionalität auf dem Straßenverkehr aufweist. Es kann keine zuverlässige Überwachung liefern und dem Radfahrer keine Sicherheit gewährleisten. Jedoch wurden Gefahrensituationen zum Teil vom Modell erkannt und daher schafft das vorliegende Paper eine Basis für die Weiterentwicklung der Gefahrendetektion für Radfahrer. Mehr Datensätze von Autos, Fahrrädern und anderen Straßenumgebungen sowie zusätzliche Datenaugmentation für andere Lichtverhältnisse, Winkel, Farben und eine bessere Kamera können die Gefahrendetektion als Anwendung erheblich verbessern.
Als Ausblick kann man die Gefahrendetektion in einem vernetzten, urbanen Bereich wie „Smart City“ einsetzen. Durch den Datenaustausch von Position, Geschwindigkeit und Gefahrenstatus der Radfahrer ist es möglich, Unfälle oder gefährliche Situationen im urbanen Bereich vorherzusagen und ein Warnsystem zu schaffen. Das würde das öffentliche Zusammenleben allgemein schützen und vor allem kann man durch den aktuellen Fahrradboom potentielle Unfälle verhindern.

# Literatur


[1] *Coco Evaluation Docs.* – URL https://cocodataset.org/#detection-eval

[2] : *Roboflow Plattform.* – URL https://roboflow.com

[3] : *Youtube Link zum Video.* – URL https://youtu.be/AM_MD52Al8c

[4] *Google Open Images*, 2019. – URL https://storage.googleapis.com/openimages/web/factsfigures_v5.html

[5] *TFLite Detect_Webcam Skript*, 2019. – URL https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/TFLite_detection_webcam.py

[6] Bundesamt, Statistisches: Kraftrad- und Fahrradunfälle im Straßenverkehr 2019 / Statistisches Bundesamt (Destatis). URL https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Verkehrsunfaelle/Publikationen/Downloads-Verkehrsunfaelle/unfaelle-zweirad-5462408197004.pdf?__blob=publicationFile, Juli 2020. – Forschungsbericht

[7] Chavez-Garcia, Ricardo O. ; Aycard, Olivier: Multiple Sensor Fusion and Classification for Moving Object Detection and Tracking / IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 17. 2016. – Forschungsbericht

[8] Dahiya, Kunal ; Dinesh Singh, C ; Mohan, Krishna: Automatic Detection of Bike-riders without Helmet using Surveillance Videos in Real-time / Department of Computer Science and Engineering, Indian Institute of Technology, Hyderabad, India. 2016. – Forschungsbericht

[9] Google: *Google Coral USB Accelerator Docs*, 2019. – URL https://coral.ai/docs/accelerator/datasheet/

[10] Howard, Andrew G. ; Zhu, Menglong ; Chen, Bo ; Kalenichenko, Dmitry ; Wang, Weijun ; Weyand, Tobias ; Andreetto, Marco ; Adam, Hartwig: MobileNets: Efficient Convolutional Neural Networks for Mobile VisionApplications / Google. 2017. – Forschungsbericht

[11] Jones, Dave: *Picamera Docs*, 2013. – URL https://picamera.readthedocs.io

[12] Liu, Wei ; Anguelov, Dragomir ; Erhan, Dumitru ; Szegedy, Christian ; Reed, Scott ; Fu, Cheng-Yang ; Berg, Alexander C.: SSD: Single Shot MultiBox Detector / Google and University of Michigan and Zoox Inc and UNC Chapel Hill. 2016. – Forschungsbericht

[13] Maeda, Hiroya ; Sekimoto, Yoshihide ; Seto, Toshikazu ; Kashiyama, Takehiro ; Omata, Hiroshi: Road Damage Detection and Classification Using Deep Neural Networks with Smartphone Images / Institute of Industrial Science, The University of Tokyo. 2018. – Forschungsbericht

[14] Pytorch (Veranst.): *Pytorch Docs Quantization.* 2019. – URL https://pytorch.org/docs/stable/quantization.html

[15] Reichelt Elektronik (Veranst.): *Reichelt - Elektronik Beschreibung RPI AI Coral USB.* – URL https://www.reichelt.de/de/de/raspberry-pi-google-coral-usb-accelerator-rpi-ai-coral-usb-p287989.html?r=1

[16] Tensorflow (Veranst.): *TensorFlow 1 Detection Model Zoo.* 2020. – URL https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md#coco-trained-models

[17] Vladimirov, Lyudmil: *TensorFlow Object Detection API 1.14*, 2018. – URL https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/tensorflow-1.14/index.html

[18] Xu, Mengmeng ; Bai, Yancheng ; Ghanem, Bernard: Missing Labels in Object Detection / The IEEE Conference on Com-puter Vision and Pattern Recognition (CVPR) Workshops. 2019. – Forschungsbericht