



Duale Hochschule Baden-Württemberg Mannheim

## Projektbericht Integrationsseminar

SmartGarage

Studiengang Wirtschaftsinformatik

Data Science

Verfasser:

Andreas Edte

Kilian Ebi

Luca Fennuciu

Miguel Sarasa-y-Zimmermann

Matrikelnummern:

6715309

9803923

2512023

4151972

Kurs:

WWI19DSB

Bearbeitungszeitraum:

16.12.2021 – 27.02.2022

# Ehrenwörtliche Erklärung

Wir versichern hiermit, den vorliegendenen Projektbericht selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Mannheim, 19.02.2022

Ort, Datum

Team SmartGarage

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>IV</b>
<b>Tabellenverzeichnis</b>	<b>V</b>
<b>Abkürzungsverzeichnis</b>	<b>V</b>
<b>Kurzfassung (Abstract)</b>	<b>VI</b>
<b>1 Motivation</b>	<b>1</b>
<b>2 Grundlagen</b>	<b>2</b>
2.1 Projektumfang . . . . .	2
2.2 Hardware . . . . .	3
2.2.1 Raspberry Pi 3 . . . . .	3
2.2.2 Luxonis OAK-D . . . . .	4
2.2.3 Ultraschallsensor . . . . .	4
2.2.4 RFID . . . . .	6
2.3 Software . . . . .	7
2.3.1 Programmiersprache . . . . .	7
2.3.2 Darknet und YOLO v3 . . . . .	7
2.3.3 OCR . . . . .	8
2.3.4 SSH . . . . .	8
2.3.5 Betriebssystem . . . . .	8
2.3.6 Tunneling-Software . . . . .	9
2.3.7 Weboberfläche . . . . .	9
<b>3 Kennzeichenerkennung</b>	<b>11</b>
3.1 Datenbeschaffung . . . . .	11
3.2 Modelltraining . . . . .	11
3.3 Geometrische Lösung . . . . .	12
3.4 OCR . . . . .	13
3.5 Evaluation der Pipeline . . . . .	16
3.5.1 Cropping . . . . .	16
3.5.2 OCR . . . . .	17
3.5.3 Laufzeit . . . . .	19
3.5.4 Zwischenfazit . . . . .	21

<b>4 Alternative Öffnungsmöglichkeiten</b>	<b>22</b>
4.1 Sprachassistenten-Steuerung . . . . .	22
4.2 Weboberfläche . . . . .	23
4.3 RFID . . . . .	23
4.4 Ultraschallsensor . . . . .	24
4.5 Öffnungssimulation . . . . .	24
4.6 Steckplan . . . . .	25
<b>5 Risiken und Limitierungen</b>	<b>28</b>
<b>6 Wirtschaftliche Aspekte</b>	<b>29</b>
<b>7 Fazit</b>	<b>30</b>
7.1 Zusammenfassung . . . . .	30
7.2 Ausblick . . . . .	30
<b>Anhang</b>	
<b>A Ablaufplan</b>	<b>32</b>
<b>B Schaltplan</b>	<b>34</b>
<b>C Steckplan</b>	<b>35</b>
<b>D Versuchsaufbau</b>	<b>37</b>
<b>E Fehlermeldungen</b>	<b>39</b>
<b>Literaturverzeichnis</b>	<b>40</b>

# Abbildungsverzeichnis

3.1	Wortdetektion . . . . .	14
3.2	Evaluation mit fuzzy ratio . . . . .	17
3.3	Evaluation mit partial ratio . . . . .	18
3.4	Evaluation mit eigener Funktion . . . . .	19
4.1	Steckplan . . . . .	26
4.2	Versuchsaufbau . . . . .	27
A.1	Programmablauf . . . . .	33
B.1	Schaltplan . . . . .	34
C.1	Steckplan groß . . . . .	36
D.1	Foto Aufbau 1 . . . . .	37
D.3	Foto Aufbau 3 . . . . .	38
D.2	Foto Aufbau 2 . . . . .	38
E.1	Fehlermeldung Blobkonvertierung . . . . .	39

# Abkürzungsverzeichnis

<b>DSGVO</b>	Datenschutz-Grundverordnung
<b>GPIO</b>	General Purpose Input/Output
<b>IoT</b>	Internet of Things
<b>OCR</b>	Optical Character Recognition
<b>RFID</b>	Radio-Frequency Identification
<b>SFTP</b>	Secure File Transfer Protocol
<b>SSH</b>	Secure Shell
<b>VPU</b>	Vision Processing Unit

# Kurzfassung (Abstract)

Dieser Projektbericht behandelt die Umsetzung des Projektes *SmartGarage* einer Gruppe von Wirtschaftsinformatikstudenten an der dualen Hochschule Baden-Württemberg - Mannheim. Ziel des Projektes ist die Entwicklung einer smarten Garagentorsteuerung, die Kennzeichen heranfahrender Fahrzeuge erkennt, auswertet und gegebenenfalls das Garagentor öffnet. Zusätzlich werden weitere Öffnungsmöglichkeiten wie durch einen Sprachassistenten, RFID, und eine Weboberfläche implementiert sowie eine Logging-Funktion. Die technischen Grundlagen der verwendeten Bauteile und Software werden dargestellt sowie die Probleme und Lösungswege bei der Umsetzung erläutert und begründet.

Der Bericht kommt zu dem Schluss, dass bis auf die Kennzeichenerkennung sämtliche Öffnungsmöglichkeiten fehlerfrei und zuverlässig eingerichtet und angewandt werden können. Die Kennzeichenerkennung selbst funktioniert zwar prinzipiell und in einzelnen Tests, jedoch ist die Gesamterkennungsrate für den Praktischen Betrieb zu gering. Es gibt Anhaltspunkte dafür, dass sich das Problem mit einiger Nacharbeit oder besserer Hardware vermutlich lösen lässt.

## Abstract (english)

This project report contains the implementation of the project *SmartGarage* of a group of BIS-students of DHBW Mannheim. Goal of the project is to develop a smart garage door opener, that will recognize the licence place of any incoming car, evaluate the letters and opens the door for authorized vehicles. Additionally other useful opening options have been implemented, like opening with a digital virtual assistant, Website, and RFID-Sensor as well as a logging functionality. The fundamental operating principles of the used modules and software are explained and the encountered problems as well as solutions are explained and described. The report comes to the conclusion, that apart from the licence plate recognition, all parts could be implemented and worked well. The licence plate recognition itself works sporadically, but the error rate and runtime is very high. Hence the authors deem the product unfit for practical and commercial use. There are, however, indications that the problems might be solved with more softwareengineering work or better hardware.

# 1 Motivation

Im wachstumsstarken Markt des Internet of Things (IoT) war der Bereich der Smart Home Technologien in 2021 mit knapp 14 Mrd. Euro das größte Marktsegment.<sup>1</sup> Bis 2026 wird von einer jährlichen Wachstumsrate von 11,6% ausgegangen.<sup>2</sup> Neben vielen bereits gut am Markt bedienten Use-Cases wie die Lichtsteuerung, Bewässerungssteuerung für Zimmerpflanzen, schaltung von Steckdosen oder der Temperaturregelung gibt es allerdings einige noch nicht angebotene, die daher ein großes Potential haben. Unter Anderem zählen wir hierzu die Personen- und Objekterkennung, die mittlerweile auch mit kleinen und preiswerten IoT-Devices möglich ist. Beispielsweise könnten Licht und Musik in einem Raum durch die individuellen Präferenzen der sich darin aufhaltenden Personen gesteuert werden. Ebenfalls möglich wäre eine automatische Türöffnung durch Gesichtserkennung oder eine automatische Öffnung der Garage durch Erkennung des Autos.

Letzteres wurde als Use-Case für dieses Projekt ausgewählt, also eine in Netzwerk eingebundene automatisierte Garagentorsteuerung, die das Kennzeichen heranfahrender Fahrzeuge erkennt und, sofern sie zur Einfahrt berechtigt sind, das Garagentor öffnet.

---

<sup>1</sup>Internet der Dinge - Weltweit 2022.

<sup>2</sup>Smart Home - Weltweit 2022.

# 2 Grundlagen

## 2.1 Projektumfang

Als Ausgangslage dient in diesem Projekt eine handelsübliches elektrisch öffnbares Garagentor, das mit einer Fernbedienung gesteuert wird. Dabei wird davon ausgegangen, dass es sich um eine 1-Kanal Steuerung handelt, also dass ein einmaliges Drücken eines Knopfes der Fernbedienung das Garagentor öffnet und ein weiteres Betätigen desselben Knopfes das Garagentor wieder schließt. Weiterhin wird davon ausgegangen, dass das Garagentor beim Auftreffen auf ein Hindernis automatisch reversiert. Dies ist sogar, wie durch ein Gerichtsurteil des OLG Frankfurt von 2015 festgestellt, gesetzlich vorgeschrieben.<sup>1</sup> Die Ansteuerung und Regelungstechnik des Garagentor-Motors selbst sind also nicht Bestandteil dieser Projektarbeit. Beim Erkennen eines berechtigten Kennzeichens und der Öffnung des Garagentors soll auch ein Logeintrag erstellt werden, um Ein- und Ausfahrten nachvollziehen zu können. Der hierzu konzipierte Programmablaufplan befindet sich in Anhang A.1.

Neben der automatisierten Öffnung durch die Kennzeichenerkennung wurden folgende Features als Projektumfang festgelegt:

- Amazon Alexa oder Google Home Integration, um das Garagentor auch als „Fußgänger“ und ferngesteuert öffnen zu können (um Gartengeräte zu entnehmen oder den Postboten ein Paket abstellen zu lassen)
- Logging der Ein- und Ausfahrenden Fahrzeuge, um das Produkt eventuell später auch für kommerzielle Parkhäuser und Tiefgaragen nutzen zu können

Folgende Funktionalitäten wurden zunächst als optional festgehalten und deren Implementierung vom Projektverlauf abhängig gemacht:

- Lichtschranke zur Erkennung ob die Garage bereits belegt ist. In diesem Fall soll das Tor nicht geöffnet werden und es einem anderen Fahrzeug, das ebenfalls einfahrtsberechtigt ist, ermöglicht werden VOR der Garage zu parkieren ohne ständig durch das sichtbare Kennzeichen die automatische Öffnung auszulösen.

---

<sup>1</sup>Vgl. Urteil vom OLG Frankfurt 2015.

- Öffnung per Transponder, um das Tor vor Ort und ohne Smartphone öffnen zu können, falls der Akku leer ist oder man anderen Personengruppen (evtl. temporären) Zutritt erteilen möchte

## 2.2 Hardware

Aufgrund der vielseitigen Verwendbarkeit, des geringen Preises, der guten Konnektivität und Kompatibilität wurde sich für ein Raspberry Pi 3+ bzw. 4 entschieden. Diese Modelle sind hervorragend für IoT-Anwendungen geeignet und verfügen - anders als kleinere Plattformen wie Arduino - mit 1 bzw. 4 oder 8 GB trotzdem über genug Arbeitsspeicher um einfache Bildverarbeitung durchführen zu können.<sup>2</sup>

Als einfachste Schnittstelle zum Torantrieb wurde der Handsender identifiziert. Hier genügt es zwei Drähte am Ein- und Ausgang des Aktivierungsknopfes anzulöten und diese mit einem Relais zu verbinden. Dann kann durch die General Purpose Input/Output (GPIO)-Pins das Relais verbunden und geöffnet oder geschlossen werden und somit ein Betätigen der Fernbedienung simuliert werden.

### 2.2.1 Raspberry Pi 3

Der Raspberry Pi ist ein vollständiger Computer in der Größe einer Kreditkarte. Er wird von der nicht profitorientierten Raspberry Pi Foundation entwickelt und in Großbritannien hergestellt. Er besitzt keinen aufgelösten Speicher, sondern bootet direkt von einer MicroSD-Karte. Die Raspberry Pi Foundation entwickelt auch das offizielle Betriebssystem Pi OS auf Basis von Linux (Debian).

Die dritte Version des Raspberry Pi (3 Mod. B v1.2) hat 1GB RAM und als CPU vier ARM Cortex-A53-Kerne, welche die ARMv8-A-Mikroarchitektur implementieren. Für dieses Modell gibt es aktuell keine offizielle 64-bit-Version von Pi OS, was die Kompatibilität mit bestimmter Software einschränkt. Die Schnittstellen umfassen MicroUSB für die Stromversorgung, HDMI für die Bildausgabe, 3,5mm Klinke für Audio, RJ-45 (Ethernet), 4x USB 2.0, 26 GPIO-Pins und zwei Flachband-Header für den offiziellen Touchscreen und die offizielle Kamera.

---

<sup>2</sup>todo.

Die vierte Version (4 Mod. B) hat eine schnellere CPU mit vier ARM Cortex-A72-Kernen, 2GB bis 8GB RAM, fest verbautes Wi-fi und bluetooth sowie zwei Micro-HDMI-Buchsen. Die Form ist gleich geblieben.

### 2.2.2 Luxonis OAK-D

Die Luxonis OAK-D ist eine IoT-Kamera, die einen RGB-Sensor und ein Paar Stereo-sensoren hat. Der Stereosensor nutzt einen Sony IMX378-Sensor und kann Video in 4k aufnehmen. Das Stereopaar nutzt Omnivision OV9282-Sensoren mit einer Auflösung von 1280x800. Die Sensoren sind direkt mit der integrierten Vision Processing Unit (VPU) verbunden. Stromversorgung und Datentransfer erfolgen über USB-C, wobei sowohl USB 2.0 als auch USB 3.0 unterstützt werden.

Als Vision Processing Unit verwendet die OAK-D die Myriad X VPU der Intel-Tochterfirma Movidius. Diese ist auf IoT-Anwendungen spezialisiert und unterstützt bis zu 8 Kameras mit einer Auflösung von 4k. Sie kann Aufgaben wie Stereosicht und Bildverarbeitung sehr effizient ausführen und ist dabei nicht auf Datentransfers zu externem Speicher angewiesen.

Zusätzlich ist sie mit einer Neural Compute Engine ausgestattet, die ein schnelles und Energieeffizientes Ausführen von Inferenz auf neuralen Netzen ermöglicht. Movidius gibt eine Performance von 1 TOPS (1 Billion Operationen pro Sekunde) an.

Die Kamera wird über die DepthAI-Platform und deren Python-API gesteuert. Hierbei kann der Entwickler selbst entscheiden, welche Sensoren und Funktionen der VPU er in welchem Ausmaß nutzt.<sup>3</sup>

### 2.2.3 Ultraschallsensor

Ultraschallsensoren finden in der heutigen Zeit in vielen Anwendungsbereichen Verwendung. Beispiele hierfür ist beispielsweise die Werkstoffprüftechnik, medizinische Diagnostik oder auch Näherungsschalter. Ein weiteres bekanntes Beispiel aus dem Alltag ist die Verwendung in Fahrzeugen, bei denen Systeme, die beispielsweise beim Einparken helfen sollen, auf Ultraschallsensoren basieren. Durch Anwendungsfelder wie die Durchhangregelung, die Höhenmessung, die Lagerregelung, den Kollisionsschutz oder auch die Füllstandserfassung

---

<sup>3</sup>Vgl. *OpenCV AI Kit: OAK—D— OpenCV.AI o.D.*

sowie die Objekterkennung und Objektzählung ist Ultraschall als Messtechnik in fast jeder Branche einsetzbar.<sup>4</sup> Ultraschall selbst liegt über dem Frequenzbereich, den ein Mensch hören kann; also über 20 kHz bis zu 1 GHz.<sup>5</sup> Erzeugt werden kann Ultraschall über zwei verschiedene Methoden, pneumatisch oder elektrisch beziehungsweise piezoelektrisch oder magnetostriktiv.<sup>6</sup> Da sich Ultraschallwellen über die Luft bewegen, spielt die Temperatur bei der Messung eine Rolle und kann die Genauigkeit beeinflussen. Die Schallgeschwindigkeit bei einer Temperatur von 0° Celsius liegt so bei 331,6 m/s, während die Schallgeschwindigkeit bei 20° Celsius bereits 343,8 m/s beträgt. Man kann also festhalten, dass sich der Schall bei einer wärmeren Temperatur schneller bewegt. Neben der Temperatur kann auch der Luftdruck einen Einfluss auf die Messgenauigkeit haben. Bei ansteigendem Luftdruck nimmt die Schallgeschwindigkeit zu. Auch die Zusammensetzung der Luft spielt eine Rolle, also unter anderem der CO<sub>2</sub>-Gehalt der Luft sowie die relative Luftfeuchte. Diese Einflussfaktoren sind bei Entwurf von Ultraschallsensoren zu beachten, um später ein möglich akkurate Messergebnis zu erzielen.<sup>7</sup> Die meistverwendete Art von Ultraschallsensoren sind Ultraschall-Abstandssensoren, die aus einem Sender und Empfänger bestehen, die beide in demselben Gehäuse eingebaut sind. Bei dieser Art von Ultraschallsensoren wird der Sender periodisch angesteuert, sodass dieser einen Ultraschallimpuls von 100 Mikrosekunden bis zu 1 Millisekunde in einem Frequenzbereich von etwa 40 bis 400 kHz aussendet. Nachdem ein Signal ausgesendet wurde, versucht der Empfänger das Echo des Ultraschalls zu erfassen. Dies ist möglich, wenn sich innerhalb der Schallkeule, die vom Sender ausgesendet wird, ein Objekt befindet. Ist dies der Fall, wird der Ultraschallimpuls vom Objekt reflektiert, sodass dieser wieder zurück zum Sender gesendet wird, und dadurch vom Empfänger aufgenommen werden kann.

Für die Abstandsmessung im Projekt SmartGarage wird der HC SR04 Ultraschallsensor verwendet, da dieser einer der billigsten und weit verbreitetsten Sensoren ist. Angesteuert wird der Ultraschallsensor über die GPIO Ports des Raspberry Pis. Die Messung selbst wird mit Hilfe eines Python-Skripts gesteuert.

---

<sup>4</sup>Vgl. Hering und Schönfelder 2018, S. 182.

<sup>5</sup>Vgl. Hering und Schönfelder 2018, S. 177.

<sup>6</sup>Vgl. Schnell und Stefan 2018, S. 70.

<sup>7</sup>Vgl. Schnell und Stefan 2018, S. 71.

## 2.2.4 RFID

RFID, kurz für Radio-Frequenz-Identifikation, ist eine Technologie, die verwendet wird um Gegenständen oder auch Personen sowie Tieren eine Kennzeichnung zu vergeben. Zu einem solchen RFID-System gehören zwei grundlegende Komponenten, ein Transponder sowie ein Lesegerät, die mittels Radiowellen miteinander kommunizieren. RFID-Systeme gehören wie Barcodes zu sogenannten Auto-ID-Systemen, die in der Lage sind ein Objekt zu identifizieren. Der Barcode ist das bekannteste Auto-ID-System und lässt sich im Alltag beispielsweise in Supermärkten oder im Einzelhandel aufgedruckt auf Produkte auffinden. Die Radio-Frequenz-Identifikation hat gegenüber dem Barcode als Auto-ID-System jedoch einen Vorteil. Bei der Radio-Frequenz-Identifikation muss nicht dafür gesorgt werden, dass die Komponenten richtig ausgerichtet sind. Dies ist bei dem Scan eines Barcodes notwendig. Ebenfalls ist die Erkennung mehrerer Objekte gleichzeitig möglich, auch wenn diese in einer Verpackung sind.<sup>8</sup>

Zur Implementierung eines RFID-Systems ist ein Transponder und ein Lesegerät notwendig. Das Lesegerät ist meist stationär angebracht während der Transponder eine Karte, ein Chip oder auch ein Mikrochip sein kann.<sup>9</sup> Ein bekanntes Beispiel aus der Geschäftswelt sind RFID-Chipkarten, mit denen man Zugang auf ein Betriebsgelände, in ein Büro oder auch dem Firmenparkplatz erlangt. Eine gängige Implementierung ist somit, dass ein Mitarbeiter eine Chipkarte erhält, die den Transponder verkörpert. Will dieser Mitarbeiter beispielsweise mit seinem Auto auf einen Firmenparkplatz fahren, muss er vor einer Schranke halten, und seine Chipkarte an das Lesegerät, welches stationär vor dieser Schranke angebracht ist, halten. Das Lesegerät liest die Chipkarte mit Hilfe von Radiowellen aus und kann somit identifizieren, dass es sich um einen Mitarbeiter handelt und gewährt durch Öffnen der Schranke Zugang.

Wichtig zu beachten bei Radio-Frequenz-Identifikationssystemen ist Frequenz. Die Wahl dieser hat einen hohen Einfluss auf die Funktion und Sicherheit dieser Systeme. Grund hierfür ist, dass nicht nur RFID-Systeme die Radiofrequenz verwenden, sondern auch Radiosender und andere Funkanlagen. Die gängigen Frequenzen für RFID-Systeme sind daher 120 - 135 kHz für die Low Frequenz, 13,56 MHz für die High Frequenz sowie 868 MHz, 915 MHz, 2,45 GHz und 5,5 GHz für die Ultra-High Frequenzen. Der gängige Standard ist hierbei jedoch 13,56 MHz.<sup>10</sup>

---

<sup>8</sup>Vgl. Helmus et al. 2011, S. 11.

<sup>9</sup>Vgl. Kern 2006, S. 33.

<sup>10</sup>Vgl. Kern 2006, S. 34.

Aufgrund des geringen Preises und der großen Verbreitung wurde sich in diesem Projekt für das RC552-Modul von NXP Semiconductors entschieden. Es ist ein RFID-Transponder, der das 13.56Mhz ISM-Band nutzt und dadurch keine Probleme mit anderen Funkverbindungen verursacht. Es kann über UART, I<sup>2</sup>C oder SPI kommunizieren und wird mit 3.3V betrieben, passt also auf die GPIO-Pins des Raspberry Pi. Eine Besonderheit ist der Interrupt-Pin, mit dem ein externes Gerät aufgeweckt werden kann, sobald ein RFID-Tag erkannt wird. Es wird mit zwei RFID-Tags geliefert, die beliebig beschrieben und ausgelesen werden können.

## 2.3 Software

### 2.3.1 Programmiersprache

Als Programmiersprache für die Schaltlogik und die Auswertung der Sensordaten wurde Python ausgewählt, da Python leistungsfähig, vielseitig einsetzbar, sehr leicht erlernbar und sich auf dem Gebiet Data Science zur verbreitetsten Programmiersprache entwickelt hat.<sup>11</sup>

### 2.3.2 Darknet und YOLO v3

Darknet ist ein in C und CUDA geschriebenes Framework für neuronale Netze, das auf die Verarbeitung von Bilddaten spezialisiert ist. Es wird von Joseph Chet Redmon entwickelt.<sup>12</sup>

Zusammen mit dem Darknet-Framework werden auch vorkonfigurierte und -trainierte Modelle bereitgestellt. Eines davon ist YOLO (You only look once). Es klassifiziert und lokaliert Objekte in Bildern mit sehr hoher Performance. In der dritten Version wird ein mAP von 57.9 erreicht. Ein Modell besteht aus einer Konfigurationsdatei und Gewichten.<sup>13</sup>

Inferenz mit Darknet-Modellen kann auch direkt auf der Myriad X VPU ausgeführt werden, wenn sie in ein proprietäres Format konvertiert werden.

---

<sup>11</sup>Vgl. Raschka und Mirjalili 2019, S. XIV.

<sup>12</sup>Vgl. Redmon 2013–2016.

<sup>13</sup>Vgl. Redmon und Farhadi 2018.

### 2.3.3 OCR

OCR (Optical Character Recognition) ist das automatische Erkennen von Text innerhalb eines Bildes. Diese Technologie ermöglicht es, den Text in einem Bild oder einem gescannnten Dokument in für Maschinen verarbeitbaren Text umzuwandeln. Hierfür gibt es eine Vielzahl von Anwendungsfällen in der heutigen Zeit. Dazu gehört beispielsweise die Umwandlung von Informationen, die ausschließlich in einem Print-Medium zu finden sind, in ein digitales Medium, die einfache Erkennung eines Coupons oder auch das Auslesen eines schriftlich ausgefüllten Formulars mit Hilfe des Computers.<sup>14</sup>

Für die optische Texterkennung gibt es mittlerweile eine Vielzahl von angebotenen Algorithmen und Services. Darunter beispielsweise Textract, welches von Amazon bereitgestellt und verkauft wird. Allerdings gibt es auch Open-Source Algorithmen dafür wie EasyOCR und Tesseract. Diese ermöglichen es die Texterkennung auch ohne hohe Kosten effektiv durchzuführen.

### 2.3.4 SSH

Secure Shell ist ein Netzwerkprotokoll, mit dem verschlüsselt auf entfernte Rechner zugegriffen werden kann. Der entfernte Rechner muss dafür direkt erreichbar sein (über ein lokales Netzwerk oder eine Portweiterleitung). SSH ist ein Client-Server-Protokoll, wobei der entfernte Rechner der Server und der lokale Rechner der Client ist. Neben klassischen textbasierten Shells können mit SSH auch verschlüsselte Dateiübertragungen (per SFTP) und grafische Anwendungen (per X11-Forwarding) realisiert werden.<sup>15</sup>

### 2.3.5 Betriebssystem

Als Betriebssystem des Raspberry Pi wurde aufgrund der Spezialisierung auf die Hardware Pi OS (Build vom 30.10.2021) gewählt. Aufgrund der coronabedingt größtenteils Remote erfolgten Teamarbeit musste der Raspberry Pi von Teilen des Teams per Fernzugriff gesteuert werden. Hierfür wurde zuerst das vorinstallierte realVNC verwendet, das sich jedoch in der Praxis durch die langsamten Internetanschlüsse der Teammitglieder als unpraktikabel erwies. Gelöst wurde das Problem durch die Verwendung von SSH. Damit lassen sich

---

<sup>14</sup>Vgl. Bekhit 2022, S. 81.

<sup>15</sup>**bibid.**

Befehle deutlich schneller und effizienter ausführen, weil eine grafische Aufbereitung und die Übertragung der Bilddaten entfällt. Zusätzlich werden durch SSH mehrere parallele Anmeldungen möglich. Allerdings machte diese Lösung wiederum die Installation einer Tunneling-Software notwendig, die den Raspberry Pi im öffentlichen Netzwerk zugänglich macht.

### 2.3.6 Tunneling-Software

Tunneling-Software wird eingesetzt, um ein Endgerät durch eine Firewall und das Internet hinweg auf einem anderen Gerät zugreifbar zu machen, ohne entsprechende Portfreigaben und Weiterleitungen einrichten zu müssen. Die momentan am weitesten verbreiteten Services für Linux sind:<sup>16</sup>

- ngrok
- Localtunnel
- Serveo
- pagekite
- Teleconsole

Sie unterscheiden sich in ihrem Funktionsumfang, Art und Anzahl maximaler Verbindungen und sind teilweise kostenpflichtig. Als Lösung für die Konfiguration und Wartung innerhalb des Teams wurde *ngrok* verwendet. Der Service ist aber in der kostenlosen Version auf eine Verbindung begrenzt und erhält auch nach jedem Neustart eine neue Adresse, auf der der Rasberry Pi erreichbar ist. Für die Weboberfläche wurde daher eine kostenpflichtige Version von *pagekite* verwendet, hiermit kann eine für den Endanwender leichter zu merkende permanente URL festgelegt werden.

### 2.3.7 Weboberfläche

Mit Hilfe einer Weboberfläche lässt sich eine weitere Möglichkeit, die SmartGarage zu öffnen, implementieren. Der Vorteil dabei ist, dass diese unabhängig vom Endgerät aufgerufen werden kann; es werden lediglich eine Internetverbindung und ein Browser benötigt. Des Weiteren lassen sich in einer Weboberfläche auch Informationen, die für den Besitzer der

---

<sup>16</sup>Vgl. *Top 4 BEST Ngrok Alternatives In 2022: Review And Comparison* 2022.

SmartGarage relevant sind, wie beispielweise ob sich aktuell ein Fahrzeug in der Garage befindet, ausgegeben werden. Diese Information wird von einem Ultraschallsensor bereitgestellt. Hierzu wird eine einfache wenn-Funktion verwendet, die angibt, dass die Garage belegt ist, falls sich innerhalb eines gewissen Abstands ein Objekt befindet, und dass die Garage frei ist, wenn sich innerhalb dieses Abstands kein Objekt befindet. Ein weiterer wichtiger Nutzen der Weboberfläche für die Bereitstellung von Informationen ist das Anzeigen von Logs. Hierzu wird eine Log-Datei erstellt, in die alle Dienste, mit denen sich die SmartGarage öffnen lässt, einen Log-Eintrag schreiben, sobald sie diese öffnen. Damit wird dokumentiert, wann und wie die SmartGarage geöffnet wird. Durch die Sammlung und Darstellung dieser Logs in einer Weboberfläche steht diese Information dem Besitzer auch unabhängig vom Standort zur Verfügung.

Für die Realisierung einer Weboberfläche gibt es viele Ansätze und Möglichkeiten sowie Frameworks die zur Verwendung herangezogen werden können. So bieten sich beispielsweise beliebte Webframeworks wie React, Angular, NextJS, Nuxt, Vue oder viele weitere an.

# 3 Kennzeichenerkennung

Dieses Kapitel behandelt die Entwicklung der Kennzeichenerkennung und ihre anschließende Evaluation.

## 3.1 Datenbeschaffung

Die Güte eines Modells hängt stark von der Qualität und Quantität der beim Training verwendeten Datensätze ab. Jedoch sind Datensätze von deutschen Kennzeichen aufgrund der Datenschutz-Grundverordnung (DSGVO) sehr schwer zu beschaffen. Zudem sind Anwendungen im Bereich Kennzeichenerkennung von großem kommerziellen Nutzen und dementsprechend nicht frei verfügbar. Die Recherche ergab, dass nur ein einziger Datensatz verfügbar war und für das Projekt in Frage käme. Es handelte sich um das *THI License Plate Dataset* der TH Ingolstadt.<sup>1</sup> Eine Anfrage beim dort zuständigen Prof. Zimmer blieb aber leider unbeantwortet, sodass eine andere Lösung gefunden werden musste. Da das Modell die Kernfunktion des Projekts ist, entschied sich das Projektteam dazu, selbst einen kleinen Datensatz zu erstellen. Hierzu wurden 118 Bildaufnahmen von Fahrzeugen in verschiedenen Landkreisen angefertigt. Dabei wurde darauf geachtet, möglichst immer aus ähnlichen Perspektiven zu fotografieren. Da sich die Kamera zur Kennzeichenerkennung nicht mittig im Garagentor selbst befestigen lässt, sondern entweder darüber oder seitlich versetzt montiert wird, wurden alle Aufnahmen von einer leicht seitlich versetzten Position aus getätigt.

## 3.2 Modelltraining

Erster Ansätze bestanden dabei aus der dauerhaften Erkennung, ob sich ein Schriftzug innerhalb des Kamerasichtfelds befindet. Die Umsetzung erfolgte dabei mittels des Python Package EasyOCR, das in einem späteren Kapitel näher erläutert wird. Die Erprobung des Ansatzes stellte sich jedoch als ineffizient heraus, da, trotz reduzierter Bild Streaming Rate, eine konstante hohe Beanspruchung der begrenzten Rechenkapazitäten auftritt und die Laufzeit zu extrem lang wird. Eine Alternative, um Rechenleistung einzusparen, wäre

---

<sup>1</sup>Vgl. *THI License Plate Dataset* 2019.

die Lokalisierung der auszulesenden Nummernschilder innerhalb des Sichtfelds der Kamera. Durch die Angabe, ob und wo sich ein Kennzeichen im aufgenommenen Bild befindet, ist man in der Lage, nicht mehr auf die Überprüfung des Gesamten Bildes angewiesen zu sein und den Texterkennungsprozess dazu nur auszulösen, wenn ein Nummernschild erkannt wird.

Um in den aufgenommenen Bildern die Kennzeichen einzugrenzen, sollte zuerst YOLO v3 zum Einsatz kommen. Aufgrund der geringen Menge an selbst verfügbaren Daten und der hohen Ähnlichkeit zu anderen Aufgaben wurde hier ein Transfer-Learning-Ansatz angewandt. Grundlage war die yolov3-Konfiguration von Redmon und die auf dem COCO-Datensatz erstellten Gewichte.

Um das Transfer-Learning auszuführen, muss zuerst Darknet kompiliert werden. Das Training kann zwar auf einer CPU ausgeführt werden, das hätte jedoch in diesem Fall zu Laufzeiten von mehreren Tagen geführt. Eine GPU ist also für das Training unabdingbar. Hierbei gab es aufgrund der proprietären Nvidia-Grafiktreiber und den zusätzlichen Paketen CUDA und CUDNN große Schwierigkeiten, weil diese in den aktuellen Versionen sehr instabil sind. Nachdem diese Schwierigkeiten überwunden waren, konnte ein Modell in etwa zwei Stunden trainiert werden. Inferenz auf einem einzelnen Bild dauert im CPU-Modus auf einem i7 8850U mehr als 20 Sekunden. Für Inferenz mit dem Raspberry Pi muss also unbedingt die Myriad X VPU verwendet werden, die in der OAK-D Kamera verbaut ist. Luxonis erreicht mit dieser Hardware und tiny YOLO v4 stabile 30fps.<sup>2</sup> Um auf der OAK-D ausgeführt zu werden, muss das Darknet-Modell zuerst in das .pb-Format von Tensorflow umgewandelt werden. Von dort aus muss es in das Format „Intermediate Representation“ von OpenVino und über eine proprietäre API zu einem Blobfile konvertiert werden. Dieser Schritt war trotz mehrtägigem Aufwand nicht erfolgreich. Die API hat unerklärte Fehler geworfen. Aus Zeitgründen wurde schlussendlich die Entscheidung getroffen, die Erkennung des Kennzeichenbereiches auf dem Bild anderweitig zu lösen.

### 3.3 Geometrische Lösung

Hierbei wurde auf einen nicht Deep-Learning-basierten Ansatz zurückgegriffen. Anders als bei der Verwendung eines trainierten Neuronalen Netzes, was auf die Erkennung von an Autos befestigten Nummernschildern zielt, gilt es bei der verwendeten Methode, das beste

---

<sup>2</sup>Vgl. Luxonis o.D.

Rechteck innerhalb einer des Bildes zu detektieren. Die Kamera kann dann später so ausgerichtet werden, dass sie direkt auf Kennzeichenhöhe Bilder aufnimmt und somit andere Rechtecke vermeidet. Kernkonzept der Detektion eines Rechtecks innerhalb des zu verarbeitenden Bildes ist der Ramer–Douglas–Peucker Algorithmus. Ursprüngliches Ziel des Algorithmus ist die Kurvenglättung. Dabei hat man eine Kette an Knoten und Vektoren der Länge  $n$ . Der Algorithmus schreitet und durchläuft einen Graphen dabei Schrittweise. Als Approximation der Strecke  $P_1$  und  $P_n$  wird deren Vektor genommen und überprüft, welcher Knoten des Graphen die größte Distanz zu diesem Vektor hat. Liegt dieser Knoten außerhalb eines zuvor definierten Toleranzwertes  $\epsilon$ . Der dabei neu ausgemachte Punkt ( $P_x$ ) dient als neuer Punkt innerhalb des Graphen, alle anderen Punkte, die sich zwischen  $P_1$  und  $P_x$  befinden werden, weg approximiert. Der Prozess wiederholt sich immer wieder bis keine Punkte mehr innerhalb der einzelnen Schritte außerhalb des Toleranzwertes  $\epsilon$ .<sup>3</sup> Unter der Voraussetzung, dass man nach einem geschlossenen Graphen, der aus vier Vektoren besteht, sucht, ist man somit in der Lage, Ausreißer durch Kurvenglättung auszuschließen und alle Rechtecke auszumachen, die sich innerhalb eines Bildes befinden. Die vier Knotenpunkte des synthetisierten Rechtecks dienen dabei als Koordinate für die Bildextraktion des Nummernschildes. Die Pipeline zur Nummernschilderkennung besteht aus einem Pre-processing Part und der Detektion die mittels des Ramer–Douglas–Peucker Algorithmus als Kernmodul der Erkennung fungiert. Umgesetzt wird der Prozess mittels der Python Bibliothek *opencv*. Zuerst durchlaufen die von der Kamera gestreamten Bilder mehrere Bildverarbeitungsschritte. Dabei wird das Bild in Graustufen umgewandelt, Noise reduziert und mit gegebenen Kernels Kanten detektiert. Die Kanten werden dabei in Vektoren erfasst. Durch die Beschaffenheit der Vektoren können mittels des Peucker Algorithmus jene Vektoren gefunden werden, die in der Summe 0 ergeben und somit einen Geschlossenen Graphen bilden. Durch die Detektion vier solcher Vektoren ist man in der Lage, innerhalb des Bildes Rechtecke zu finden. Bei gegebener Detektion eines Rechtecks wird diese extrahiert und erst dann mittels der Texterkennungsprozesse ausgelesen, jene werden im nächsten Kapitel erklärt.

## 3.4 OCR

Nach der Erkennung des Nummernschildes wird das dabei extrahierte Bild im nächsten Pipeline Schritt durch das ein OCR-Package ausgelesen und der Text als String generiert.

---

<sup>3</sup>Vgl. Hershberger und Snoeyink 1992.

Als OCR-Lösung war EasyOCR geplant, weil es trotz geringem Ressourcenaufwand gute Ergebnisse liefert. Dessen Funktionsweise wird im folgenden Abschnitt kurz behandelt.

Das Modell besteht aus zwei Schritten, zuerst die Detektion der einzelnen Buchstaben, danach deren Erkennung. Für erstere Aufgabe wird sich des CRAFT (Character Region Awareness for Text Detection) Algorithmus bedient.<sup>4</sup> Durch die Ausmachung der einzelnen Charaktere weist das Modell besonders bei verzerrten oder nicht linearen Texten, im Gegensatz zur Detektion von Wortblöcken, bessere Ergebnisse auf. Das Modell basiert dabei auf einem Convolutional Neuronal Network mit skip-connections und Charakteristiken eines U-Netzes. Es wird mit Texten jeglicher Form und Druckschrift trainiert. Nach erfolgreichem Training ist es in der Lage, für die Einzelne Buchstaben einen "region score" und einen Äffinitäts score auszugeben. Ersterer bezieht sich dabei, wie der Name schon ausdrückt, auf die Koordinaten, die die Buchstabe, lokalisieren, zweiteres auf die Wahrscheinlichkeit des Zusammenhangs einzelner Buchstaben in einem Wort. Durch das Anpassen des Modells auf auch nicht lineare Texte gilt es als vielversprechend bei der Umsetzung des *SmartGarage* Projekts, da die Kamera in diesem Fall nicht Frontal auf das Nummernschild zeigen wird. Es entsteht verzerrte Schrift, die das OCR-Modell erkennen muss.

Im Folgenden Bild sind die einzelnen Zusammenhänge der Detektion eines Wortes dargestellt:

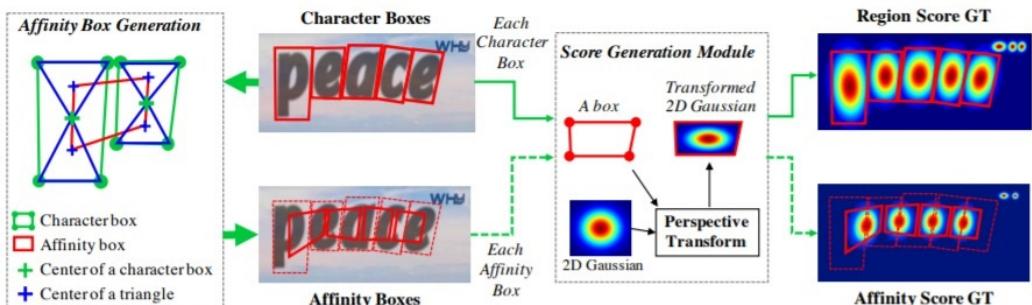


Abbildung 3.1: Wördetektion

Nach der Lokalisierung eines Kennzeichen und der daraus gewonnenen Koordinaten, werden die gewonnenen Informationen in EasyOCR weiterverarbeitet und mittels eines CRNN (Convolutional Recurrent Neural Network) Netzes bestehend aus drei Komponenten ausgelesen.

---

<sup>4</sup>Vgl. Baek et al. 2019.

Es handelt sich dabei um eine Feature Extraction, des Gesuchten Inputs mittels eines Res-Nets, um ein LSTM (Long-short term memory) Netz für das sequenzielle Labeling von Charakteren innerhalb eines Wortes. LSTM Netze eignen sich dabei besonders für die Verarbeitung sequenzieller Daten. Die dritte Komponente besteht aus einem CTC (connection temporal classification) Netz und ist für das Decoding der Outputs innerhalb des LSTM Netzes zuständig.<sup>5</sup>

Dieser Ansatz scheiterte jedoch, weil EasyOCR von Torch abhängt, Torch ein 64bit-Betriebssystem voraussetzt und Pi OS für den Raspberry Pi 3 zum Zeitpunkt der Projektdurchführung nur als 32bit-Version verfügbar war. Die Alternative Lösung war Tesseract, eine OCR-Engine, die ursprünglich von Hewlett-Packard entwickelt wurde. Seit 2005 ist es Open-Source und von 2006 bis 2018 wurde es von Google weiterentwickelt. <https://tesseract-ocr.github.io/docs/tesseracticdar2007.pdf>

Tesseract bedient sich eines LSTM Netzes, benötigt jedoch kein 64Bit System. Bei der Umsetzung wird auf *Pytesseract* und ein vortrainiertes englisches Modell der Texterkennung zurückgegriffen.<sup>6</sup> Dabei wird auch wieder auf ein Preprocessing gesetzt, dass das Bild in Graustufen konvertiert. Des Weiteren wird versucht mittels des Gauß-Verfahrens und gegebenen Filtern, Noise herauszufiltern. Dritter Schritt des Preprocessings ist das hervorhebenden der Kanten mittels der Konvertierung des Bildes in sogenannte *Blobs*, dabei handelt es sich um die Umwandlung eines Graubildes mittels eines festgelegten Schwellenwerts in Binäre Werte.<sup>7</sup> Das Modell versucht, mittels Vektoren Linien auszumachen, auf denen sich die Schriftzüge befinden. Danach geht Tesseract dazu über, zeilenweise Wörter mittels der Bemessung von Abständen auszumachen und Wörter in Bounding-Boxes zu fassen. Die Erkennung der Texte geschieht über das vortrainierte englische LSTM Netz. Dieses weist bei der Umsetzung jedoch deutlich schlechtere Resultate auf als das zuvor verwendete CRNN (EasyOCR). Die Funktionalität und Genauigkeit beider Modelle wird in einem separaten Kapitel ausgewertet.

---

<sup>5</sup>Vgl. *JaidedAI/EasyOCR: Ready-to-use OCR with 80+ supported languages and all popular writing scripts including Latin, Chinese, Arabic, Devanagari, Cyrillic and etc.* o.D.

<sup>6</sup>Vgl. *Tesseract User Manual / tessdoc* o.D.

<sup>7</sup>Vgl. Singh und Bhushan 2019.

## 3.5 Evaluation der Pipeline

Nach erfolgreichem Proof-of-Concept musste unter den technisch möglichen Varianten der Verarbeitungspipelines die beste ausgewählt werden. Hierzu wurde ein Notebook geschrieben, das die Pipeline in leicht abgewandelter Form für jedes Bild des gesammelten Datensatzes durchführt und das Ergebnis oder eventuelle Fehler in den einzelnen Bearbeitungsschritten festhält. Damit konnten verschiedene OCR-Verfahren untereinander verglichen werden. Ebenso war es möglich die bereits festgestellten Unterschiede in der Bearbeitungsgeschwindigkeit und Qualität von der Berechnung am Laptop gegenüber der Bearbeitung auf dem RasPi zu quantifizieren. Anhand der festgestellten Metriken können schlussendlich auch weiterführende Optimierungen vorgenommen und Fehlerquellen lokalisiert werden.

Die Kennzeichenerkennungspipeline besteht aus folgenden Schritten:

- Laden der kurz zuvor gespeicherten Datei
- Preprocessing
  - Umwandlung zu Graustufen
  - Bilateraler Filter
  - Canny-Algorithmus zu Kantenfindung
  - Douglas-Peucker-Algorithmus zur Erkennung von Rechtecken
  - Cropping
- OCR mit vorgefertigter Lösung

### 3.5.1 Cropping

Bei 85 von 118 Bildern wurde eine Region als Kennzeichen identifiziert und freigestellt. Diese ist nicht immer korrekt, es wurden neben Kennzeichen auch Fenster, Ziegelsteine und Straßenschilder freigestellt. Das ist ein Problem, das durch die Vorschaltung von yolov3 verhindert werden könnte. Eine genauere Evaluation dieses Schrittes ist, ohne die Ergebnisse der nachgeschalteten OCR-Lösung zu berücksichtigen, wenig sinnvoll, weil der Bildausschnitt unvorhersehbare Folgen für das OCR-Ergebnis hat.

### 3.5.2 OCR

Beide OCR-Lösungen liefern ungenaue Ergebnisse. Die Plaketten zwischen den Blöcken der Kennzeichen werden oft als Zeichen interpretiert. Genauso wird zwischen den Blöcken manchmal ein Leerzeichen erkannt und manchmal nicht. Bei einem genauen Vergleich dieser Ergebnisse mit einer Liste von erlaubten Kennzeichen würde daher sehr selten ein Treffer auftreten.

Aus diesem Grund wird stattdessen das Python-Paket `thefuzz` verwendet, welches sich die Levenshtein-Distanz zu Nutze macht, um die Ähnlichkeit mehrerer Strings zu quantifizieren. Um die Levenshtein-Distanz zwischen zwei Strings zu bestimmen, wird ein String zum Anderen umgeformt. Erlaubte Operationen sind dabei das Einfügen eines neuen Zeichens und das Entfernen eines bestehenden Zeichens. Die minimal benötigte Anzahl dieser Operationen ist die Levenshtein-Distanz.

Hieraus wird ein prozentualer Wert abgeleitet, der die Genauigkeit der Erkennung widerspiegelt. Für die finale Evaluation mit dieser Methode wurde ein Threshold von 45% gewählt, um das gelesene Kennzeichen als richtig einzurordnen.

Mit easyOCR wurden 28 von 85 Kennzeichen richtig gelesen, mit Tesseract 17.

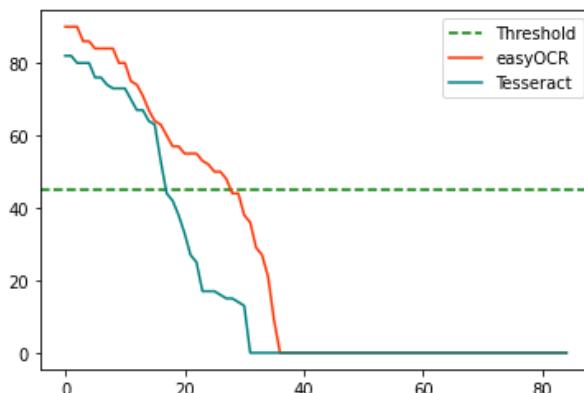


Abbildung 3.2: Evaluation mit fuzzy ratio

Neben dem vollständigen Fuzzy Matching können mit der Funktion `fuzz.partial_ratio()` auch Substrings berücksichtigt werden. Das ist sinnvoll, um beispielsweise die Namen von Autohäusern aus den OCR-Ergebnissen herauszufiltern. Mit dieser Methode (Threshold 45%) liest easyOCR 31 von 85 Kennzeichen richtig, Tesseract 22. Dies entspricht 26% bzw. 19% der Ausgangsmenge von 118 Kennzeichen.

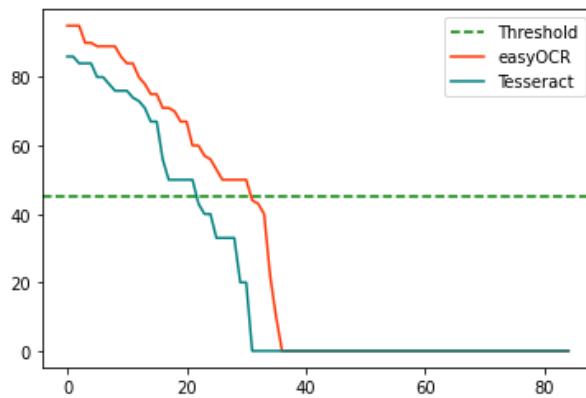


Abbildung 3.3: Evaluation mit partial ratio

Abgesehen von den bestehenden Methoden zum Vergleich mehrerer Strings wurde auch ein spezialisierter Ansatz entwickelt. Sowohl easyOCR als auch Tesseract haben die Schwäche, dass sie manchmal die Blöcke der Kennzeichen vertauschen. Um dieses Problem zu umgehen, wurde folgende, auf das vorliegende Problem spezialisierte, Funktion zur Evaluation entwickelt:

```

1 def custom_match(read, label):
2     for x in label:
3         if x not in read:
4             return 0
5     return 100

```

Die Werte 0 und 100 dienen der einheitlichen Visualisierung. `read` ist der Text, den die OCR-Lösung erkannt hat. `label` ist das Label des jeweiligen Bildes in der Form OG AE 1337. Die leerzeichengetrennten Teile des Labels werden durch Python in jeweils einzelnen Schleifendurchläufen behandelt. Mit dieser Methode erzielt easyOCR 21 richtige Ergebnisse, Tesseract 5.

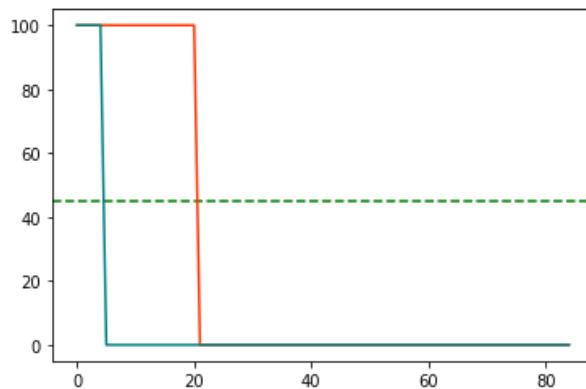


Abbildung 3.4: Evaluation mit eigener Funktion

Es ist zu beachten, dass nicht alle im Vorigen Schritt freigestellten Bildsegmente auch wirklich Kennzeichen sind. Ein Mensch könnte an dieser Stelle also auch keine perfekte Leistung erreichen.

Methode	easyOCR	Tesseract
fuzzy Ratio	28	17
partial Ratio	31	22
eigene Funktion	21	5

Tabelle 3.1: Performance von easyOCR und Tesseract

### 3.5.3 Laufzeit

Für eine praktische Anwendung darf die Laufzeit der gesamten Pipeline nicht zu lang sein.

Erstens sollte die Zeit zwischen dem Auftauchen des Autos vor der Kamera und dem ersten Scan konsistent sein, zweitens kann die Pipeline im gleichen Zeitraum mehrmals ausgeführt werden und so trotz ungenauen Modellen gute Ergebnisse erzielen. Die Laufzeit hängt direkt von der verwendeten Hardware ab.

Zur Evaluation wird die Dauer der einzelnen Schritte (Cropping und OCR) gemessen und zu Durchschnitten aggregiert.

Auf dem selbst erstellten Datensatz wurde mit einem Lenovo Thinkpad T580 (Intel i7-8850U) für das Cropping durchschnittlich 1,37 Sekunden, für die Texterkennung mit ea-

syOCR 2,27 Sekunden und für die Bilderkennung mit Tesseract 2,67 Sekunden benötigt. Das entspricht einer Gesamtaufzeit von rund 4 Sekunden, also etwa 15 Bildern pro Minute.

Wird jedoch stattdessen ein Raspberry Pi 3 verwendet, steigt die Laufzeit auf durchschnittlich 14,87 Sekunden für das Cropping und 17,62 Sekunden für die Texterkennung mit Tesseract, also insgesamt 32,49 Sekunden. EasyOCR wurde nicht evaluiert, weil es nicht mit der verwendeten Version von Pi OS kompatibel ist.

Beachtet man, dass das System im schlimmsten Fall kurz vor Ankunft des Fahrzeugs ein Bild aufnahm, so vergehen selbst bei einer direkten Erkennung etwa 60 Sekunden bis zur nächsten Aufnahme und Auswertung.

Schritt	Thinkpad T580	Raspberry Pi 3
Cropping	1.37s	14.87s
easyOCR	2.27s	-
Tesseract	2.67s	17.62s

Tabelle 3.2: Laufzeit von Cropping, easyOCR und Tesseract

### 3.5.4 Zwischenfazit

Geht man von der besten erreichten Erkennungsrate von 26% aus, so beträgt der Erwartungswert der benötigten Erkennungsdurchläufe  $100/26 = 3,84$ , welche wiederum 125 Sekunden benötigen. Dazu kommen durchschnittlich  $0.5 * 32$  Sekunden zur Beendigung der beim Eintreffen ablaufenden Schleife. Die Durchschnittliche Wartedauer beträgt also 141 Sekunden oder 2 Minuten und 21 Sekunden.

Natürlich ist hier zu beachten, dass beim Warten und der erneuten Bildverarbeitung das Bild ggf. nicht so stark vom vorherigen abweicht wie beim Datensatz und die gleichen Problembereiche hat. Die Erfahrungen im Praxistest auf der Straße lassen aber eher auf eine große Varianz der erkannten Flächen und Texte schließen, weshalb der Wert sich durchaus als Schätzung verwenden lässt.

Die durchschnittliche Wartedauer  $W$  bis zum Start der Öffnung lässt sich somit allgemein mit Formel 3.1 berechnen, wobei  $p$  die Erkennungswahrscheinlichkeit und  $t$  die Ausführungsduer des Skripts darstellt.

$$W = \frac{1}{p} * t + \frac{t}{2} \quad (3.1)$$

# 4 Alternative Öffnungsmöglichkeiten

Dieses Kapitel behandelt die Einbindung des Sprachassistenten, der Weboberfläche, des RFID-Moduls, des Ultraschallsensors und den praktischen Aufbau und Verkabelung aller Module.

## 4.1 Sprachassistenten-Steuerung

Zunächst wurde versucht den Sprachassisten Alexa von Amazon an den Raspberry Pi anzubinden. Die entsprechenden Bibliotheken konnten zwar installiert werden, auf der Developer-Oberfläche von Amazon erschienen allgemeine Fehlermeldungen die keinem konkreten Problem zuzuordnen waren. Die Verbindung zur Software auf dem Raspberry Pi konnte aber nachgewiesen werden, sodass wir von einer fehlerhaften Syntax der übertragenen Inhalte ausgehen, vermutlich durch Versionskonflikte. Eine Alternative Lösung mit Google Home schien zunächst sehr vielversprechend, da Goolge in der Dokumentation ebenfalls einen Raspberry Pi als Einrichtungsbeispiel verwendet. Es stellte sich jedoch heraus, dass auch hier trotz orndungsgemäßem Befolgen der Anleitung keine Verbindung zustande kam. Die erhaltenen Fehlermeldungen ließen auf ein Paketproblem in der verwendeten Betriebssystemversion zu, die wird schlussendlich nicht lösen konnten. Als Workaround wurde ein Alexa WiFi-Relais beschafft, das sich mit der App eines Drittanbieters (eWeLink) verbinden und dann in die Alexa-Umgebung integrieren lässt. Dieses Modul wurde nun so angeschlossen, dass es bei Aktivierung einen GPIO-PIN mit 3.3V Spannung versorgt. Dieser Vorgang kann wiederum mit einer einfachen Schleife in einem Python Skript auf dem RasPi abgefragt werden und weitere Aktionen wie logging etc. ausgeführt werden. Prinzipiell wäre auch ein einfacherer, direkter Anschluss des Relais an den Garagentoröffner in Parallelschaltung möglich gewesen. Auf diesen wurde aber aus Kontroll- und Sicherheitsgründen verzichtet.

## 4.2 Weboberfläche

Da die Implementierung im Rahmen des Projekts auf einem Raspberry Pi stattfindet und Python bereits für viele Anwendungen innerhalb des Projekts verwendet wird, wird sich für das in Python geschriebene Webframework Flask entschieden. Dies hat den Vorteil, dass andere Komponenten, die bereits in Python geschrieben sind, direkt ohne Mehraufwand in die Weboberfläche integriert werden können.

Daher wird im Rahmen des Projekts für die Realisierung einer Weboberfläche eine einfache FlaskApp erstellt, die den üblichen Aufbau hat mit einer flaskapp.py, die die Weboberfläche steuert, und auf einzelne Templates, die wiederum in HTML geschrieben werden, zugreift. Hierbei wird in der flaskapp.py-Datei die Initialisierung der Weboberfläche sowie das Routing und dem Datentransfer zwischen den Seiten festgelegt sowie auch die Funktion, die die Garage öffnet. Als Templates die als Seite geladen werden, wird eines für die Startseite erstellt und eines für die Seite, auf der die Logs angezeigt werden sollen. Die Hauptseite, die bei Aufruf der Seite geladen wird, enthält einen Button, der mittels der in der flaskapp.py dafür festgelegten Funktion die SmartGarage öffnet und einen Eintrag in der Logs-Datei erstellt. Des Weiteren enthält diese Seite die Information ob in der Garage ein Fahrzeug steht, oder nicht. Dies geschieht über die Ultraschallmessung, wie bereits zuvor erläutert. Ebenfalls wird von dieser Seite aus auf die Logs-Seite verlinkt. Die Logs-Seite bezieht die Daten aus der Logs-Datei und gibt diese auf der Weboberfläche aus, wodurch der Besitzer der Garage die geloggten Öffnungen einsehen kann.

## 4.3 RFID

Die Implementierung eines RFID-Systems in der SmartGarage bietet eine weitere Möglichkeit die Garage zu öffnen. Für das Projekt wurde daher ein RFID-RC522 Modul an den Raspberry Pi angeschlossen, welches als Lesegerät Transponder auslesen kann. Angesteuert vom Raspberry Pi wird das Modul mittels der GPIO-Ports ähnlich wie beispielsweise bereits dem Ultraschallsensors. Das Auslesen wird dann mit Hilfe eines Python-Skripts gesteuert und mit einer Whitelist abgeglichen. Trotz der komplexen Technik gelang der Anschluss dank eines online verfügbaren Tutorials problemlos.<sup>1</sup> Als Transponder kann im Projekt jede

---

<sup>1</sup>Vgl. *How to setup a Raspberry Pi RFID RC522 Chip - Pi My Life Up 2022.*

beschreibbare RFID-Karte oder RFID-Chipkarte verwendet werden, solange diese die vorausgesetzte Frequenz von 13,56MHz unterstützt. Im Projekt wird hierfür eine RFID-Karte, die nicht mit dem richtigen Code beschriftet ist, als Beispiel für eine fehlerhafte Identifikation verwendet. Die großen Vorteile der RFID-Öffnungsmöglichkeit gegenüber z.B. einem Schlüssel sind die Protokollierbarkeit und die einfache und kostengünstige Sperrung falls ein Transponder geklaut oder verloren wird. In diesem Fall genügt es, den entsprechenden Eintrag aus der Whitelist zu entfernen.

## 4.4 Ultraschallsensor

TODO voherigen Teil aufsplitten, Quelle

## 4.5 Öffnungssimulation

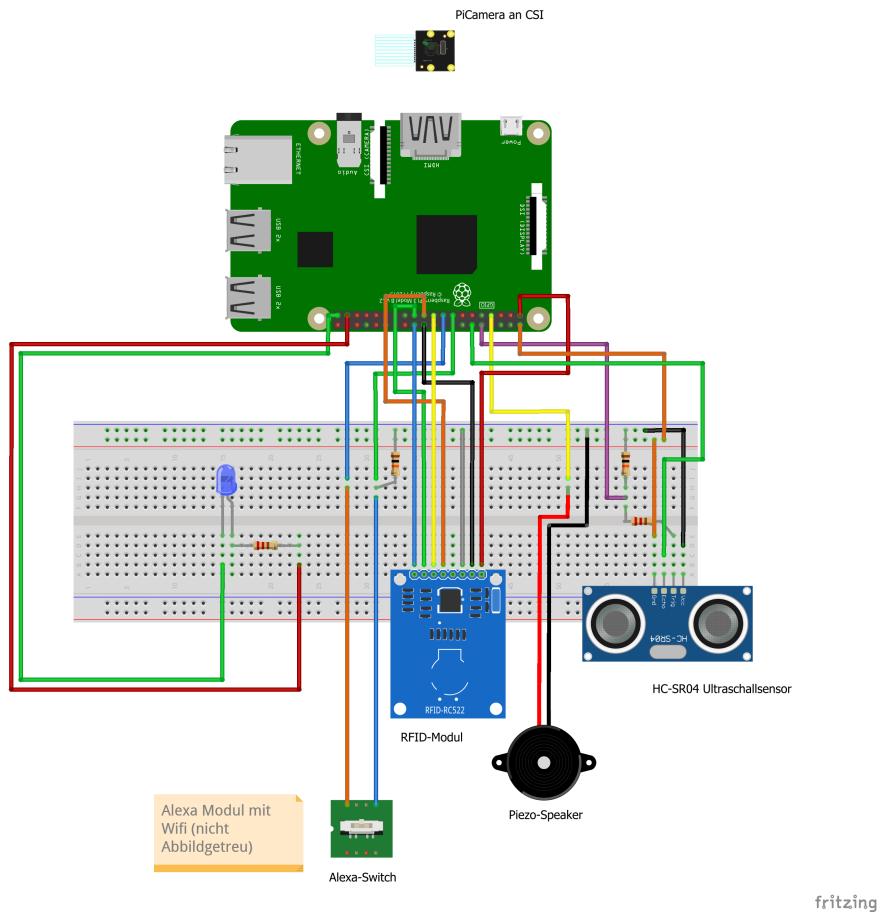
Um die Betätigung des Handsenders bei der Entwicklung simulieren und Testen zu können, wurde statt dem Relais zunächst eine LED an die GPIO-Pins des RasPi angeschlossen. Da die Lichtverhältnisse manchmal schwierig waren, wurde zusätzlich ein sogenannter aktiver Summer eingesetzt. Dieser erklingt dann zum Beispiel, wenn der richtige RFID Chip an das Lesegerät gehalten wird und somit die Garage geöffnet oder geschlossen wird. Dabei handelt es sich um einen Oszillator der, wird er einer gewissen Spannung unterstellt, anfängt zu Vibrieren und damit ein Ton erzeugt. Ausgeführt und koordiniert wird der Buzzer über ein Simples Python Skript das mittels der Bibliothek GPIO (General-Purpose Input/Output) in der Lage ist den Buzzer an oder auszuschalten. Dabei wird durch einen definierten Port, in diesem Fall „Port 4“, mittels einer *for* Schleife die Spannung je nach *positivem* oder *negativem* Signal Ton koordiniert und eine andere Tonsequenz abgespielt.

Zu Präsentationszwecken wurde der Code zum Aktivieren der LED bzw. des Buzzers dann durch einen anderen ersetzt, der per SSH auf einen zweiten Raspberry Pi zugreift und dort ein Video eines öffnenden und schließenden Garagentors abspielt. Dabei wird lokal eine Binärvariable geändert um den Zustand des Garagentors (offen oder geschlossen) speichert und das richtige Video auswählt. Jedoch funktioniert diese Lösung nicht ganz zuverlässig, da nach jedem Zugriff das Video des vorherigen Versuchs trotz geschlossener SSH-Verbindung offen bleibt.

## 4.6 Steckplan

Auf Abbildung 4.1 ist die schlussendlich verwendete Steckung abgebildet. Hierbei war zu beachten, dass die im Internet verfügbaren Anleitungen und Tutorials nicht unbedingt miteinander kompatibel sind und einige GPIO-Pins von mehreren Vorlagen verwendet werden. Hier muss dementsprechend ein anderer freier Pin mit gleicher I/O-Funktion gefunden werden und der Code angepasst werden. Außerdem ist die Spannung zwingend zu beachten. Manche Module, wie der Ultraschallsensor, benötigen eine Spannung von 5V, andere nur 3.3V. Die GPIO-Pins des Raspberry Pi, sind auf 3.3V ausgelegt und nicht vor Überspannung geschützt. Ein versehentliches Verbinden der beiden Spannungsnetze kann zu irreperablen Hardwareschäden führen. Deshalb wurde beispielsweise am Ultraschallsensor ein Spannungssteiler eingebaut.

Der zur Steckung gehörige elektrische Schaltplan findet sich in Anhang B.1. Auf Abbild 4.2 ist der Aufbau in der Praxis abgebildet. Das Objekt rechts auf dem Bild ist ein 4G-Router, der aufgrund der schlechten Internetverbindung bei Gruppenarbeiten vor Ort benötigt wurde. Weitere Bilder des Versuchsaufbaus in Anhang D.2



fritzing

Abbildung 4.1: Steckplan der gesamten Hardware  
Quelle: Eigene Darstellung

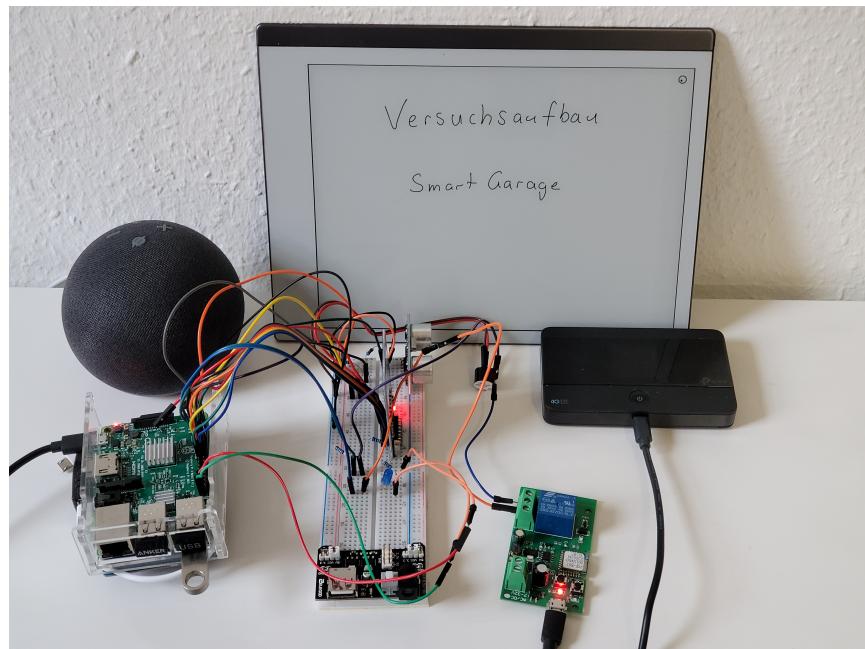


Abbildung 4.2: Fertiger Aufbau aller Module  
Quelle: Eigene Darstellung

# 5 Risiken und Limitierungen

Mehrere problematische Punkte konnten innerhalb der relativ kurzen Projektlaufzeit noch nicht ausreichend adressiert werden. Sie wurden aber erfasst und in diesem kurzen Abschnitt notiert.

Zum einen hat die verwendete Software noch Schwierigkeiten mit Störungen durch Fahrzeug- und Kennzeichenhalterbeschriftungen und deutsche Umlaute, die in manchen Kennzeichen vorhanden sind, können noch nicht erkannt werden. Dies war auf den erfassten Testdaten in Mannheim/Heidelberg nicht nicht problematisch, wird es aber für andere Regionen sein. Zum anderen ergeben sich durch die optische Erkennung naturgemäß etliche Probleme bei erschwerten Sichtverhältnissen wie Schnee, Regen, Dunkelheit oder schlicht verschmutzten Kennzeichen.

Ein weiterer bedeutender Punkt ist die mangelnde Sicherheit des Systems. Da die Öffnung des Garagentors nur durch das Kennzeichen geschieht, ist das Missbrauchs- bzw. Einbruchspotenzial groß. Im schlimmsten Fall reicht ein bedrucktes DIN-A4 Blatt mit dem richtigen Kennzeichen um die Garage zu öffnen. Auch Hackerangriffe sind prinzipiell möglich.

Ebenfalls sollte bei der Installation der Datenschutz beachtet werden. So ist es nicht erlaubt öffentliche Bereiche zu filmen.

Eine Messung des Stromverbrauchs hat nicht stattgefunden, dementsprechend kann auch keine Aussage über die laufenden Betriebskosten des RasPi mit allen Modulen und der Luxonis-Kamera getätigt werden.

# 6 Wirtschaftliche Aspekte

Auch wenn die technische Konzeption und Umsetzung im Mittelpunkt dieser Arbeit stehen, soll aufgrund der Ausrichtung des Studiengangs hier auch kurz auf einige betriebswirtschaftliche Aspekte des Projekts eingegangen werden.

## Entwicklungskosten Zeitaufwand pro Teammitglied

- 2x Projektkonzeption/Kick-Off online 2 Stunden
- 3x Workshops a 12 Stunden
- 2x Workshop a 8 Stunden
- 1x Praxistest 4 Stunden
- 1x Abschluss-Workshop a 24 Stunden
- 8 Stunden Ausarbeitung Projektbericht / Dokumentation

Bei einem angenommenen Stundenlohn von 15€, der für Werkstudenten mit vergleichbaren Kenntnissen angemessen ist, ergeben sich bei den aufsummierten 368 Mannstunden geschätzte kalkulatorische Entwicklungskosten von 5520€

## Materialkosten in Euro, inkl. Versand

Raspberry Pi 3	39,95 <sup>1</sup>
RFID-Modul RC552	5,00 <sup>2</sup>
Ultraschallmodul	8,94 <sup>3</sup>
Amazon Alexa Wifi Modul	13,99 <sup>4</sup>
OAK-D Kamera	185,51 <sup>5</sup>
<b>Summe</b>	<b>253,39</b>

Die geschätzte Installationszeit pro Einheit für Verkabelung des Kameramoduls und des RasPi, Befestigung des RFID-Sensors, Ausrichtung der Kamera und Tests beträgt für einen geübten Elektriker ca. 4 Stunden. Bei einem angenommenen Kosten von 60€/Stunde wären dies 240€ (ohne Anfahrt). Der Verkaufspreis inklusive Installation müsste also mindestens bei knapp 500€ liegen, um einen Deckungskostenbeitrag zu erzielen und die Entwicklungskosten von 5520€ zu decken.

# 7 Fazit

Dieses Kapitel enthält die Zusammenfassung des Projektberichts mit einem kurzen Fazit und Ausblick, der weitere Verbesserungen vorschlägt.

## 7.1 Zusammenfassung

Zusammenfassend lässt sich feststellen, dass die Implementierung der Module gut funktioniert hat und diese zuverlässig funktionieren.

Dies ist nicht zuletzt der weiten Verbreitung des Raspberry Pi und der verwendeten Bauteile unter Hobbybastlern geschuldet, die in der Szene zahlreiche und umfangreiche Tutorials veröffentlichten.

Als weitaus problematischer als gedacht erwies sich jedoch die Umsetzung der Kennzeichenerkennung sowohl durch die Luxonis-Kamera als auch die Alternativlösung. Bei Ersterer scheinen die Schnittstellen und Softwareversionen noch nicht restlos aufeinander abgestimmt zu sein und es finden sich wenige Artikel und Hilfestellungen im Internet, was beides auch an dem erst kürzlich erfolgten Markteintritt liegen kann.

Die schlussendlich verwendete geometrische Lösung funktioniert prinzipiell, jedoch war die Erkennungsrate mit bestenfalls 26% und die Programmablaufdauer knapp 30 Sekunden unter bzw. über den Erwartungen.

Da die Kennzeichenerkennung aber ein elementarer Kernbestandteil des Projekts ist, kommen wir zu dem Schluss, dass die technische Umsetzung des Projekts prinzipiell machbar ist, die Reaktionszeit und die Fehlerquote jedoch zu hoch ist für einen sinnvollen Betrieb. Bei über Zwei Minuten Wartezeit haben die meisten Personen wohl ihr Garagentor schneller manuell Geöffnet. Daher bedarf es für den Alltagseinsatz einer Weiterentwicklung.

## 7.2 Ausblick

Die Nutzung eines 64bit-Betriebssystems würde die Nutzung von easyOCR ermöglichen, was in der Evaluation zu deutlich besserer Performance führte. Mit zusätzlichem Aufwand

ist denkbar, die ursprünglich geplante Eingrenzung des Kennzeichens durch ein neuronales Netz, welches auf der Myriad X VPU läuft, nachträglich zu verwenden und die Schwächen der aktuellen geometrischen Lösung zu umgehen. Alternativ könnte auch kommerzielle Software zur Kennzeichenerkennung eingekauft werden.

Wäre das Performance-Problem gelöst, beständen vielfältige Möglichkeiten das System zu erweitern oder in anderen Use-Cases zu verwenden. Auch eine kommerzielle Nutzung bei Tiefgaragen sowie Parkhäusern oder Mautsystemen wäre möglich. Dies ermöglicht potentiellen Betreibern von diesem System, die Parkdauer oder Fahrtstrecke des Kundens direkt zu berechnen, da das Loggen der Ein- und Ausfahrten in diesem Anwendungsbeispiel diese Information bereits mitliefert.

Eine Weiterentwicklung des RFID-Zugangs und Kombination mit einer Zeitsteuerung wäre ebenfalls möglich, so könnte der Postbote mit seinem Chip zwischen 8 und 12 Uhr Zutritt erhalten oder der Gärtner zwischen 9 und 17 Uhr. Auch wäre es möglich weitere IoT-Geräte anzuschließen und beispielsweise bei der Einfahrt schonmal die Lichter im Haus anzuschalten, einen Kaffee zu kochen oder das Badezimmer anzuheizen oder bei der Ausfahrt alle Lichter auszuschalten und die Alarmanlage scharf zu schalten. Der Integration ins Smart Home sind keine Grenzen gesetzt. SmartGarage kann also nicht nur das Garagentor, sondern auch die Tür zu vielen spannenden und nützlichen Anwendungsmöglichkeiten öffnen.



# A Ablaufplan

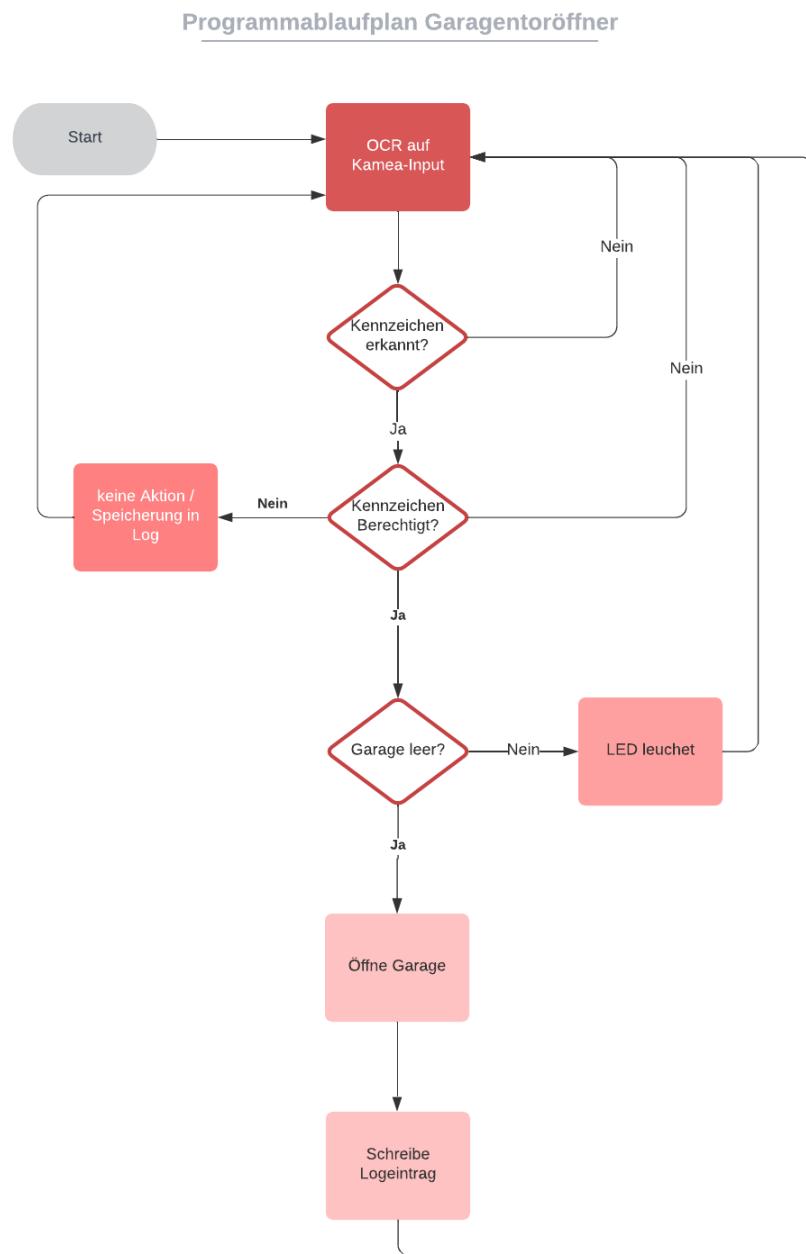
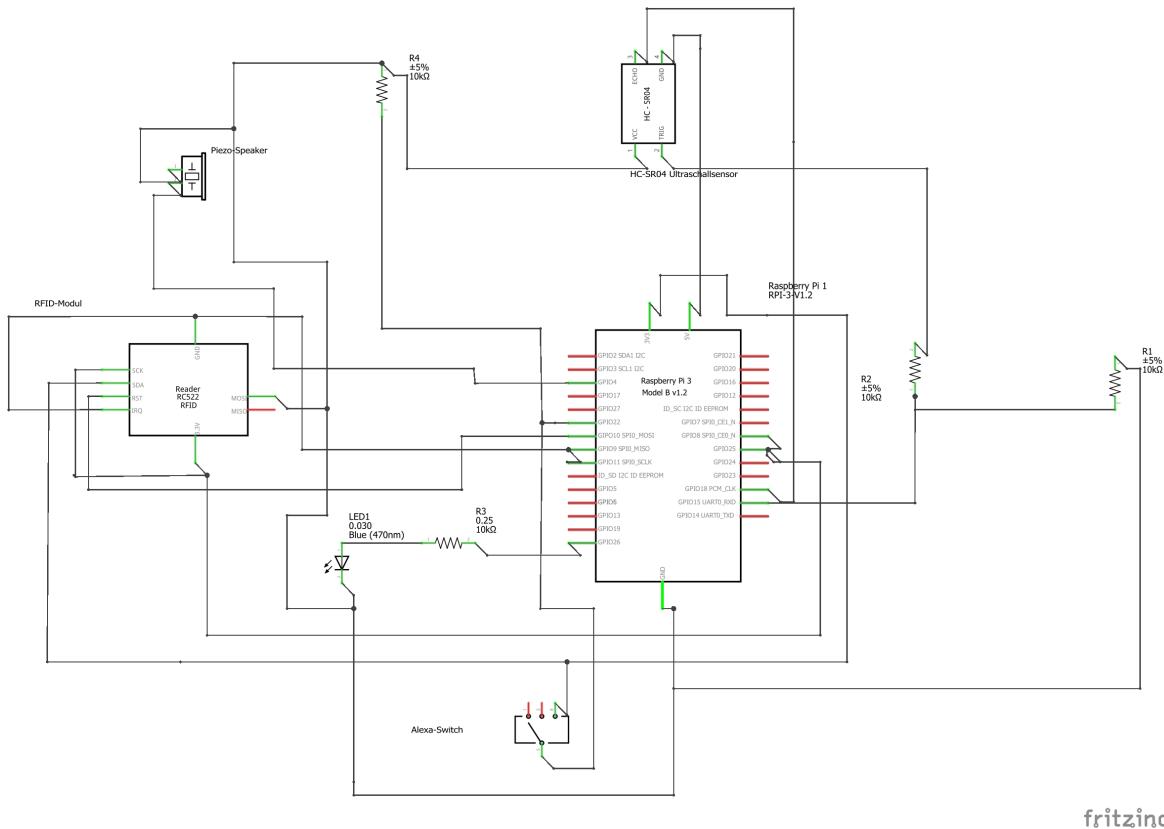


Abbildung A.1: Ablauf der Kennzeichenerkennungs-Schleife  
Quelle: Eigene Darstellung

# B Schaltplan



fritzing

Abbildung B.1: Finaler Schaltplan aller Bauteile  
Quelle: Eigene Darstellung



# C Steckplan

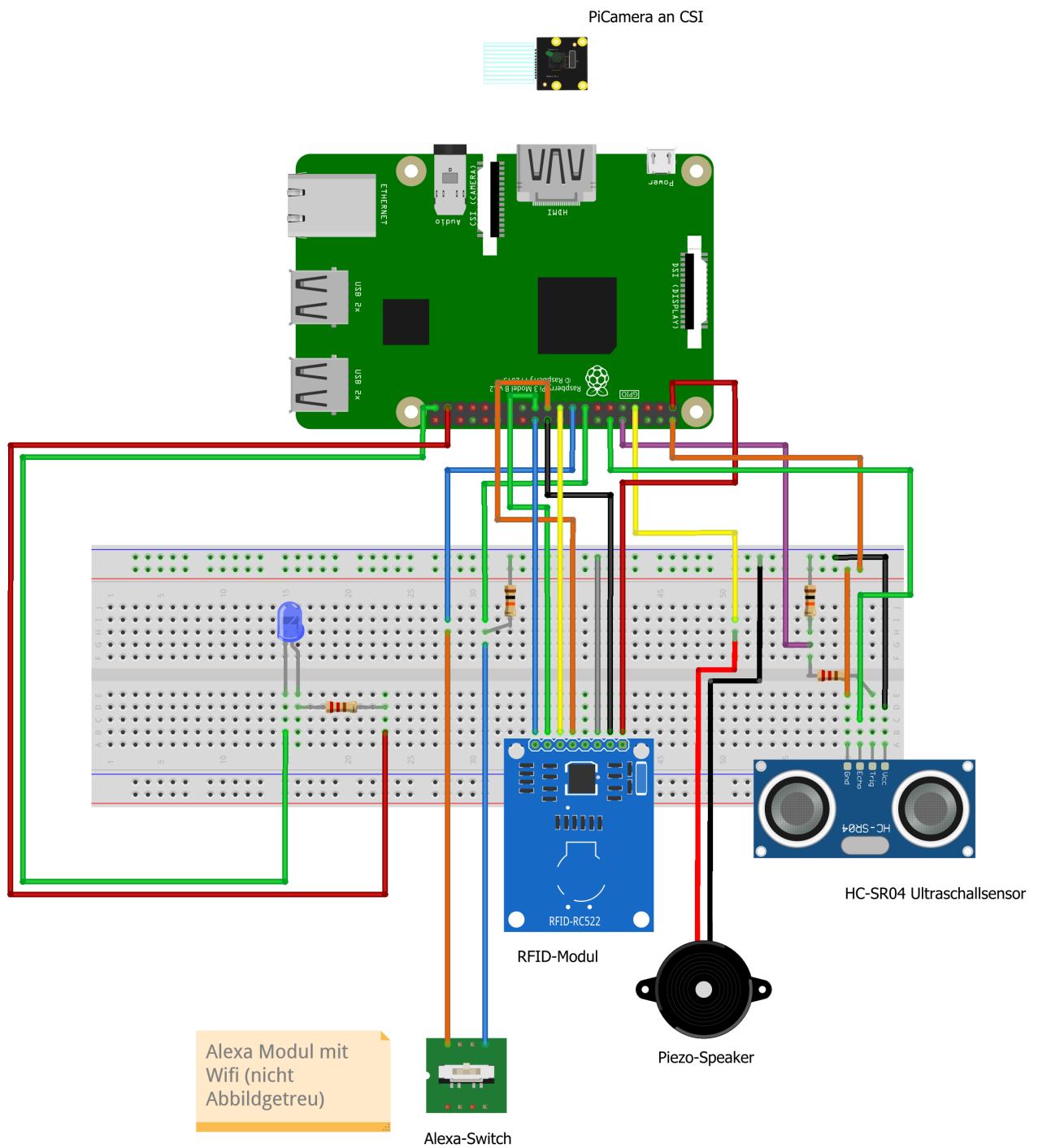


Abbildung C.1: Steckplan der gesamten Hardware  
Quelle: Eigene Darstellung

## D Versuchsaufbau

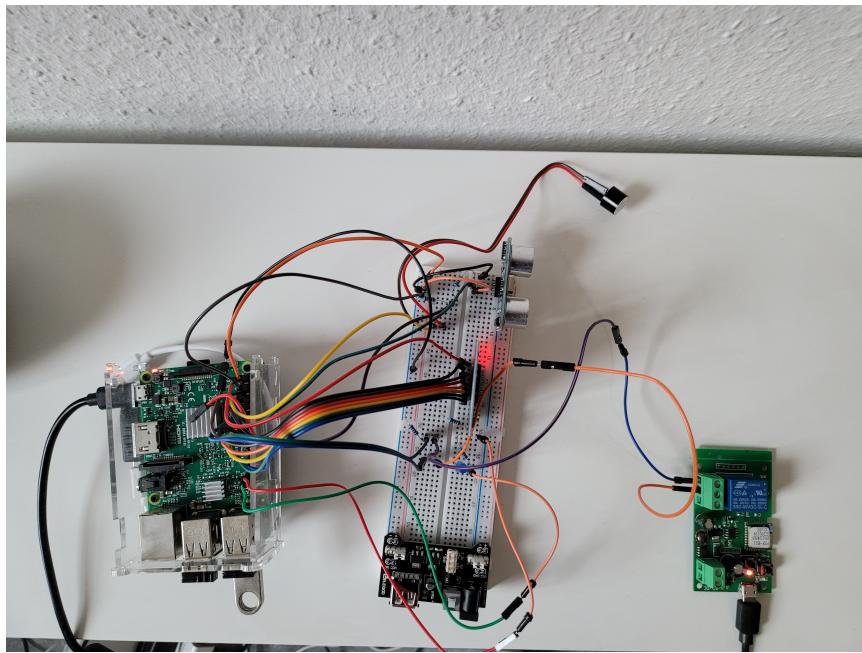


Abbildung D.1: Raspberry Pi, Breadboard und Alexa-Modul  
Quelle: Eigene Darstellung



Abbildung D.3: Notlösung mit Raspi-Kamera nach Defekt  
Quelle: Eigene Darstellung

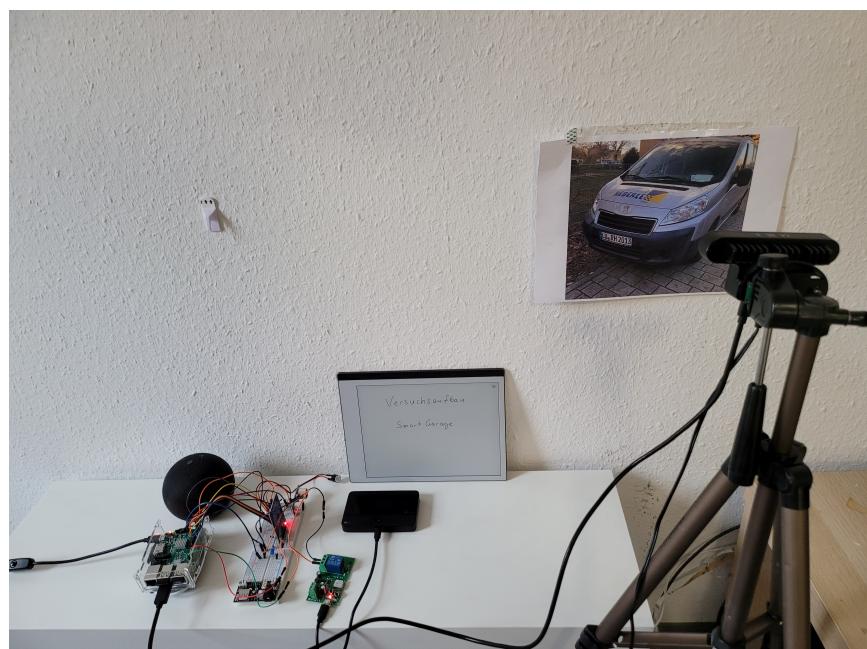
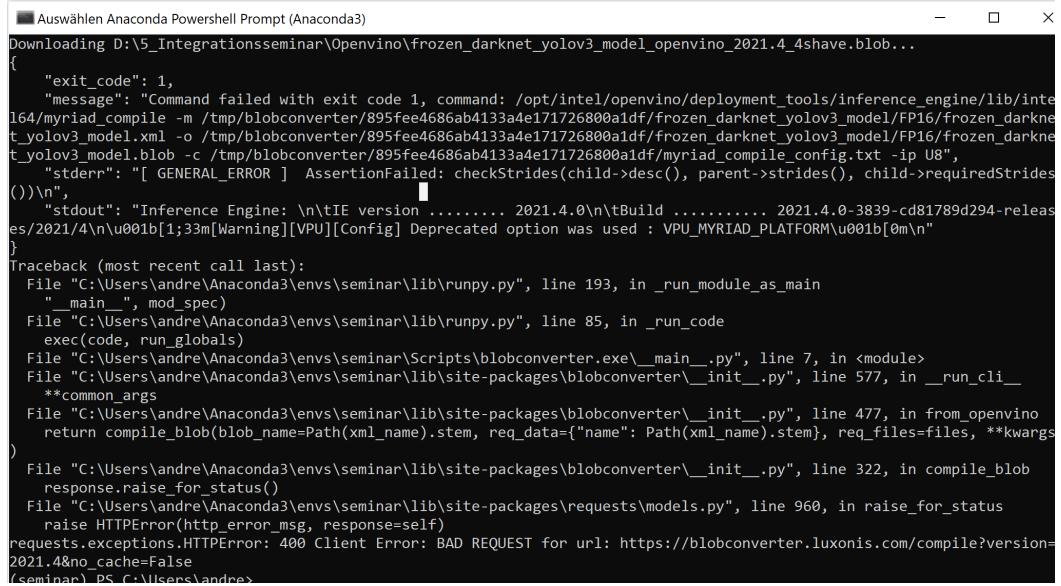


Abbildung D.2: Aufbau mit Kamera  
Quelle: Eigene Darstellung

# E Fehlermeldungen



A screenshot of a Windows PowerShell window titled "Auswählen Anaconda Powershell Prompt (Anaconda3)". The window displays an error message from the OpenVINO blob converter. The message indicates that a command failed with exit code 1, specifically related to the 'myriad\_compile' tool. It shows the command used: 'myriad\_compile -m /tmp/blobconverter/895fee4686ab4133a4e171726800a1df/frozen\_darknet\_yolov3\_model/FP16/frozen\_darknet\_yolov3\_model.xml -o /tmp/blobconverter/895fee4686ab4133a4e171726800a1df/frozen\_darknet\_yolov3\_model/FP16/frozen\_darknet\_yolov3\_model.blob -c /tmp/blobconverter/895fee4686ab4133a4e171726800a1df/myriad\_compile\_config.txt -ip U8'. The stderr output shows an assertion failure related to strides. The stdout output provides version information for the inference engine and blob converter. The traceback shows the call stack from runpy.py to blobconverter.exe.

```
Auswählen Anaconda Powershell Prompt (Anaconda3)
Downloading D:\5_Integrationsseminar\Openvino\frozen_darknet_yolov3_model_openvino_2021.4_4share.blob...
{
    "exit_code": 1,
    "message": "Command failed with exit code 1, command: /opt/intel/openvino/deployment_tools/inference_engine/lib/intel64/myriad_compile -m /tmp/blobconverter/895fee4686ab4133a4e171726800a1df/frozen_darknet_yolov3_model/FP16/frozen_darknet_yolov3_model.xml -o /tmp/blobconverter/895fee4686ab4133a4e171726800a1df/frozen_darknet_yolov3_model/FP16/frozen_darknet_yolov3_model.blob -c /tmp/blobconverter/895fee4686ab4133a4e171726800a1df/myriad_compile_config.txt -ip U8",
    "stderr": "[ GENERAL_ERROR ] AssertionFailed: checkStrides(child->desc(), parent->strides(), child->requiredStrides())\n",
    "stdout": "Inference Engine: \n\tIE version ..... 2021.4.0\n\tBuild ..... 2021.4.0-3839-cd81789d294-releases/2021/4\n\tu0001b[1;33m[Warning][VPU][Config] Deprecated option was used : VPU_MYRIAD_PLATFORM\u0001b[0m\n"
}
Traceback (most recent call last):
  File "C:\Users\andre\Anaconda3\envs\seminar\lib\runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "C:\Users\andre\Anaconda3\envs\seminar\lib\runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "C:\Users\andre\Anaconda3\envs\seminar\Scripts\blobconverter.exe\__main__.py", line 7, in <module>
  File "C:\Users\andre\Anaconda3\envs\seminar\lib\site-packages\blobconverter\__init__.py", line 577, in __run_cli__
    **common_args
  File "C:\Users\andre\Anaconda3\envs\seminar\lib\site-packages\blobconverter\__init__.py", line 477, in from_openvino
    return compile_blob(blob_name=Path(xml_name).stem, req_data={"name": Path(xml_name).stem}, req_files=files, **kwargs)
  File "C:\Users\andre\Anaconda3\envs\seminar\lib\site-packages\blobconverter\__init__.py", line 322, in compile_blob
    response.raise_for_status()
  File "C:\Users\andre\Anaconda3\envs\seminar\lib\site-packages\requests\models.py", line 960, in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 400 Client Error: BAD REQUEST for url: https://blobconverter.luxonis.com/compile?version=2021.4&no_cache=False
(seminar) PS C:\Users\andre>
```

Abbildung E.1: Fehlermeldung beim Konvertieren des Darknet-Blobfiles

# Literaturverzeichnis

- Baek, Youngmin et al. (2019). „Character region awareness for text detection“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, S. 9365–9374.
- Bekhit, Ahmed Fathi (2022). *Computer Vision and Augmented Reality in iOS*. Springer Verlag. ISBN: 978-1-4842-7461-3.
- Converting a PyTorch\* Model — OpenVINO™ documentation — Version(latest)* (o.D.). [https://docs.openvino.ai/latest/openvino\\_docs\\_MO\\_DG\\_prepare\\_model\\_convert\\_model\\_Convert\\_Model\\_From\\_PyTorch.html](https://docs.openvino.ai/latest/openvino_docs_MO_DG_prepare_model_convert_model_Convert_Model_From_PyTorch.html). (Accessed on 02/23/2022).
- eWeLink DC5V (2022). URL: <https://www.kaufland.de/product/348563329/> (besucht am 15.02.2022).
- Foundation, Raspberry Pi (o.D.[a]). *Buy a Raspberry Pi 3 Model B*. <https://www.raspberrypi.com/products/raspberry-pi-3-model-b/>. (Accessed on 02/22/2022).
- (o.D.[b]). *Buy a Raspberry Pi 4 Model B*. <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>. (Accessed on 02/22/2022).
- HC-SR04 Ultraschallsensor (2022). URL: <https://www.conrad.de/de/p/hc-sr04-abstandsmessung-ultraschall-ultrasonic-sensor-module-802235231.html> (besucht am 15.02.2022).
- Helmus, Manfred et al. (2011). *RFID-Baulogistikleitstand*. Springer Verlag. ISBN: 978-3-8348-1577-4.
- Hering, Ekbert und Gert Schönfelder (2018). *Sensoren in Wissenschaft und Technik*. Springer Verlag. ISBN: 978-3-658-12561-5.
- Hershberger, John Edward und Jack Snoeyink (1992). „Speeding up the Douglas-Peucker line-simplification algorithm“. In:
- How to setup a Raspberry Pi RFID RC522 Chip - Pi My Life Up* (2022). <https://pimylifeup.com/raspberry-pi-rfid-rc522/>. (Accessed on 02/25/2022).
- Internet der Dinge - Weltweit* (2022). URL: <https://de.statista.com/outlook/tmo/internet-der-dinge/weltweit> (besucht am 20.01.2022).
- JaidedAI/EasyOCR: Ready-to-use OCR with 80+ supported languages and all popular writing scripts including Latin, Chinese, Arabic, Devanagari, Cyrillic and etc.* (o.D.). <https://github.com/JaidedAI/EasyOCR>. (Accessed on 02/19/2022).
- Kern, Christian (2006). *Anwendung von RFID-Systemen*. Springer Verlag. ISBN: 978-3-540-27725-5.

- Luxonis (o.D.). *RGB & Tiny YOLO — DepthAI documentation*. [https://docs.luxonis.com/projects/api/en/latest/samples/Yolo/tiny\\_yolo/](https://docs.luxonis.com/projects/api/en/latest/samples/Yolo/tiny_yolo/). (Accessed on 02/19/2022).
- OpenCV AI Kit: OAK—D— OpenCV.AI* (o.D.). <https://storeopencv.ai/products/oak-d>. (Accessed on 02/19/2022).
- Raschka, Sebastian und Vahid Mirjalili (2019). *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. Third edition. ISBN: 9781789955750.
- Raspberry Pi 3b* (2022). URL: <https://www.berrybase.de/raspberry-pi/raspberry-pi-computer/boards/raspberry-pi-3-modell-b> (besucht am 15.02.2022).
- Redmon, Joseph (2013–2016). *Darknet: Open Source Neural Networks in C*. <http://pjreddie.com/darknet/>.
- Redmon, Joseph und Ali Farhadi (2018). „YOLOv3: An Incremental Improvement“. In: *arXiv*.
- RFID-Modul RC552* (2022). URL: <https://www.az-delivery.de/products/rfid-set> (besucht am 15.02.2022).
- Schnell Gerhard, Hesse und Stefan (2018). *Sensoren für die Prozess- und Fabrikautomation*. Springer Verlag. ISBN: 978-3-658-21172-1.
- Schnirch, Andreas, Nadine Ridinger und Felix Weschenfelder (2020). *Raspberry Pi im Informatik- und Technikunterricht*. Springer Verlag. ISBN: 978-3-658-28792-4.
- Singh, Jaskirat und Bharat Bhushan (2019). „Real Time Indian License Plate Detection using Deep Neural Networks and Optical Character Recognition using LSTM Tesseract“. In: *2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, S. 347–352. DOI: 10.1109/ICCCIS48478.2019.8974469.
- Smart Home - Weltweit* (2022). URL: <https://de.statista.com/outlook/dmo/smart-home/weltweit> (besucht am 20.01.2022).
- Smith, R. (2007). „An Overview of the Tesseract OCR Engine“. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Bd. 2, S. 629–633. DOI: 10.1109/ICDAR.2007.4376991.
- Tesseract User Manual / tessdoc* (o.D.). (Accessed on 01/29/2022). URL: <https://tesseract-ocr.github.io/tessdoc/#Tesseract-with-lstm>.
- THI License Plate Dataset* (2019). <https://www.thi.de/forschung/carissma/c-isafe/thi-license-plate-dataset/>. (Accessed on 02/25/2022).
- Top 4 BEST Ngrok Alternatives In 2022: Review And Comparison* (Feb. 2022). <https://www.softwaretestinghelp.com/ngrok-alternatives/>. (Accessed on 02/25/2022).
- Urteil vom OLG Frankfurt* (2015). @<https://openjur.de/u/775737.html>. (Accessed on 01/29/2022).