



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 vorliegenden 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 Annahmen und Projektumfang . . . . .	2
2.2 Hardware . . . . .	3
2.2.1 Raspberry Pi 3 . . . . .	3
2.2.2 Luxonis OAK-D . . . . .	4
2.2.3 Intel Movidius Myriad X . . . . .	4
2.2.4 RC522 RFID-Modul . . . . .	5
2.2.5 Ultraschallsensor . . . . .	5
2.2.6 RFID . . . . .	6
2.3 Software . . . . .	8
2.3.1 Einrichtung . . . . .	8
2.3.2 Tunneling-Software . . . . .	8
2.3.3 SSH . . . . .	8
2.3.4 Programmiersprache . . . . .	9
2.3.5 Darknet und YOLO v3 . . . . .	9
2.3.6 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 . . . . .	12
3.5 Validierung . . . . .	13
3.6 Evaluation der Pipeline . . . . .	13
3.6.1 Cropping . . . . .	14
3.6.2 OCR . . . . .	14

3.6.3	Laufzeit . . . . .	16
3.6.4	Zwischenfazit . . . . .	16
<b>4</b>	<b>Verbindung</b>	<b>18</b>
4.1	Sprachassistenten-Steuerung . . . . .	18
4.2	Weboberfläche . . . . .	18
4.3	RFID . . . . .	18
4.4	Ultraschallsensor . . . . .	18
4.5	Piezo-Lautsprecher . . . . .	18
4.6	Steckplan . . . . .	18
4.7	Schaltlogik . . . . .	18
4.8	Weboberfläche . . . . .	18
<b>5</b>	<b>Risiken und Limitierungen</b>	<b>21</b>
<b>6</b>	<b>Wirtschaftliche Aspekte</b>	<b>22</b>
<b>7</b>	<b>Zusammenfassung</b>	<b>23</b>
7.1	Fazit . . . . .	23
7.2	Ausblick . . . . .	23
<b>8</b>	<b>Technischer Ausblick</b>	<b>26</b>
<b>Anhang</b>		
<b>A</b>	<b>Ablaufplan</b>	<b>27</b>
<b>Literaturverzeichnis</b>		<b>28</b>

# Abbildungsverzeichnis

3.1	Evaluation mit $\text{fuzzy}_{ratio}$ . . . . .	15
3.2	Evaluation mit $\text{partial}_{ratio}$ . . . . .	15
3.3	Evaluation mit eigener Funktion . . . . .	15
4.1	Steckplan . . . . .	19
4.2	Programmablauf . . . . .	20

# 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)

Der Projektbericht behandelt die Umsetzung des Projektes *SmartGarage* einer Gruppe von Wirtschaftsinformatikstudenten an der dualen Hochschule Baden-Württemberg - Mannheim. Inhalt des Projektes ist eine Smarte 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. 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 angewandt werden können. Die Kennzeichenerkennung selbst funktioniert zwar prinzipiell und in einzelnen Tests, jedoch ist die Gesamterkennungsrate für den Praktischen Betrieb zu gering. Dieses Problem ließe sich mit einiger Nacharbeit, einem größeren Datensatz oder besserer Hardware vermutlich lösen.

# 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 Annahmen und Projektumfang

Als Ausgangslage dient in diesem Projekt eine handelsübliches elektrisch öffnenbares 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. Die Ansteuerung und Regelung des Garagentor-Motors selbst sind also nicht Bestandteil dieser Projektarbeit.

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

- Amazon Alexa oder Google Home Integration, um das Garagentor auch als "Fußgängeründ 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 Features wurden 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.
- 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 mit 1 bzw. 4 oder 8 GB trotzdem über genug Arbeitsspeicher um einfache Bildverarbeitung durchführen zu können.

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 aufgelöteten 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 Zweite 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.

Die vierte Version (4 Mod. B) hat eine schnellere CPU mit vier ARM Cortex-A72-Kernen, 4GB 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 Stereosensoren 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 Myriad X-VPU verbunden. Stromversorgung und Datentransfer erfolgen über USB-C, wobei sowohl USB 2.0 als auch USB 3.0 unterstützt werden.

Die Kamera wird über die DepthAI-Plattform und deren Python-API gesteuert. Hierbei kann der Entwickler selbst entscheiden, welche Sensoren und Funktionen der VPU er in welchem Ausmaß nutzt.<sup>1</sup>

Als Vision Processing Unit verwendet die OAK-D den Myriad X der Intel-Tochterfirma Movidius ist eine Vision Processing Unit (VPU), die auf IoT-Anwendungen spezialisiert ist. Sie 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 neuronalen Netzen ermöglicht. Movidius gibt eine Performance von 1 TOPS (1 Billion Operationen pro Sekunde) an.

### 2.2.3 Intel Movidius Myriad X

Der Myriad X der Intel-Tochterfirma Movidius ist eine Vision Processing Unit (VPU), die auf IoT-Anwendungen spezialisiert ist. Sie 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 neuronalen Netzen ermöglicht. Movidius gibt eine Performance von 1 TOPS (1 Billion Operationen pro Sekunde) an.

---

<sup>1</sup>OpenCV AI Kit: OAK—D— OpenCV.AI o.D.

## 2.2.4 RC522 RFID-Modul

Das RC522-Modul von NXP Semiconductors 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.

## 2.2.5 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 Durchgangregelung, die Höhenmessung, die Lagerregelung, den Kollisionsschutz oder auch die Füllstandserfassung sowie die Objekterkennung und Objektzählung ist Ultraschall als Messtechnik in fast jeder Branche einsetzbar.<sup>2</sup> Ultraschall selbst liegt bei Frequenzen über dem Frequenzbereich, den ein Mensch hören kann. Also über 20 kHz bis zu 1 GHz.<sup>3</sup> Erzeugt werden kann Ultraschall über zwei verschiedene Methoden, pneumatisch oder elektrisch beziehungsweise piezoelektrisch oder magnetostraktiv.<sup>4</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 beispielsweise bei 331,6 m/s während die Schallgeschwindigkeit bei 20° Celsius bereits 343,8 m/s beträgt. Man kann hierbei also feststellen, dass sich der Schall bei einer wärmeren Temperatur schneller bewegt. Auch einen Einfluss auf die Messgenauigkeit kann neben der Temperatur auch der Luftdruck 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 akkurates Messergebnis zu erzielen.<sup>5</sup> Die meistverwendete Art von Ultraschallsensoren sind Ultraschall-Abstandssensoren,

---

<sup>2</sup>Vgl. Hering und Schönfelder 2018, S. 182.

<sup>3</sup>Vgl. Hering und Schönfelder 2018, S. 177.

<sup>4</sup>Vgl. Schnell und Stefan 2018, S. 70.

<sup>5</sup>Vgl. Schnell und Stefan 2018, S. 71.

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 Abstandsbemessung im Projekt SmartGarage wird der HC SR04 Ultraschallsensor implementiert. Der HC SR04 Ultraschallsensor besteht aus zwei Komponenten die als Trigger bzw. Sender und Echo bzw. Empfänger fungieren. Die Funktionsweise ist die Selbe, wie bereits allgemein zu Ultraschall-Abstandssensoren erläutert wurde. Angesteuert wird der Ultraschallsensor über die GPIO Ports des Raspberry Pis. Die Messung selbst wird mit Hilfe eines Python-Skripts zur Abstandsmessung gesteuert.

Eingesetzt wird die Abstandsbemessung mittels Ultraschallsensors im Projekt für die Messung, ob ein Objekt, also in diesem Anwendungsfall ein Automobil, in der Garage geparkt ist, oder nicht. Daraus lässt sich die Information ableiten, ob die Garage frei oder belegt ist. Diese Information wird über die Weboberfläche ausgegeben.

## 2.2.6 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 untereinander kommunizieren. RFID-Systeme gehören wie der Barcode 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 beachtet werden, dass die Komponenten richtig ausgerichtet sind. Dies ist bei dem Scan eines Barcodes notwendig. Mittels Radio-Frequenz-Identifikation können Objekte ohne spezielle Ausrichtung erkannt

werden. Ebenfalls ist die Erkennung mehrerer Objekte gleichzeitig möglich, auch wenn diese beispielsweise in einer Verpackung sind.<sup>6</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>7</sup> Ein bekanntes Beispiel aus der Geschäftswelt sind RFID-Chipkarten, mit denen man beispielsweise 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 auf den Firmenparkplatz mit seinem Auto fahren, muss dieser 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 Zugriff auf den Firmenparkplatz in Form von Öffnen der Schranke.

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 wie der Name verbirgt Radiosender sowie 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>8</sup>

Die Implementierung eines RFID-Systems in der SmartGarage bietet eine weitere Möglichkeit die Garage zu öffnen. Für das Projekt wird daher ein RFID-RC522 Modul an den Raspberry Pi angeschlossen. Dieses dient als Lesegerät, welches später den Transponder auslesen wird. 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.

Als Transponder kann im Projekt jede 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. Wenn diese an das Lesegerät gehalten wird, wird kein Zugriff gewährt, da der Code auf der Karte nicht mit den Codes, die in der

---

<sup>6</sup>Vgl. Helmus et al. 2011, S. 11.

<sup>7</sup>Vgl. Kern 2006, S. 33.

<sup>8</sup>Vgl. Kern 2006, S. 34.

Whitelist festgelegt sind, übereinstimmt. Als Beispiel für eine erfolgreiche Identifizierung wird im Projekt ein RFID-Chip mit dem richtigen Code beschriftet. Wird dieser an das Lesegerät gehalten, wird abgeglichen ob dieser in der Whitelist enthalten ist. Da dieser in der Whitelist enthalten ist, wird Zugriff auf die SmartGarage gewährt.

Bei tatsächlicher Implementierung besteht hierbei neben Chipkarten und RFID-Karten auch die Möglichkeit sich einen NFC-Chip beziehungsweise RFID-Chip mittels Implantat unter die Haut einsetzen zu lassen. Dies bietet den Vorteil, dass man immer Zugriff auf seine Garage hat, ohne einen externen Gegenstand, den man verlieren kann, bei sich haben zu müssen.

## 2.3 Software

### 2.3.1 Einrichtung

Für dieses Projekt wurde ein Raspberry Pi 3 verwendet. Als Betriebssystem 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 langsamen Internetanschlüsse der Teammitglieder als unpraktikabel erwies. Gelöst wurde das Problem durch die Verwendung von SSH. Damit ließen sich Befehle deutlich schneller und effizienter ausführen, weil eine grafische Aufbereitung und die Übertragung der Bilddaten entfielen. Allerdings machte diese Lösung wiederum die Installation einer Tunneling-Software wie *Ngrok* notwendig, die den Raspberry Pi öffentlich zugänglich macht.

### 2.3.2 Tunneling-Software

### 2.3.3 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.

### 2.3.4 Programmiersprache

Als Programmiersprache für die Schaltlogik wurde Python ausgewählt, da Python leistungsfähig, vielseitig einsetzbar, sehr leicht erlernbar und sich auf dem Gebiet Data Science zur verbreitetsten Programmiersprache entwickelt hat.<sup>9</sup>

### 2.3.5 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>10</sup>

Zusammen mit dem Darknet-Framework werden auch vorkonfigurierte und -trainierte Modelle bereitgestellt. Eines davon ist YOLO (You only look once). Es klassifiziert und lokalisiert 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>11</sup>

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

### 2.3.6 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 und lediglich ein mit dem Internet verbundener Browser benötigt wird. Des weiteren lassen sich in einer Weboberfläche auch Informationen, die für den Besitzer der SmartGarage relevant sind, ausgeben. Hierzu gehört beispielsweise die Information, ob sich aktuell ein Fahrzeug in der Garage befindet. Hierzu wird ein Ultraschallsensor implementiert, der diese Information auf der Weboberfläche bereitstellt. Hierzu wird eine einfache

---

<sup>9</sup>Vgl. Raschka und Mirjalili 2019, S. XIV.

<sup>10</sup>Redmon 2013–2016.

<sup>11</sup>yolov3.



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 Implementierung dieser Logs in einer Weboberfläche steht diese Information dem Besitzer auch unabhängig vom Standort zur Verfügung.

Für die Realisierung einer Weboberfläche gibt es viele Realisierungsansätze 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. Da die Implementierung im Rahmen des Projekts jedoch 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 ein Template für die Startseite erstellt sowie ein Template 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.

# 3 Kennzeichenerkennung

## 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 frei verfügbar war und für das Projekt in Frage käme. Es handelte sich um das *THI License Plate Dataset* der TH Ingolstadt. Eine Anfrage beim dort zuständigen Prof. TODO 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 in der gleichen Perspektive 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

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. Nachdem diese 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>1</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. Nach mehreren Tagen Aufwand wurde die Entscheidung getroffen, den ersten Schritt der geplanten Pipeline zu entfernen und eine geometrische Lösung zum ersten Schritt zu machen.

### 3.3 Geometrische Lösung

Nach mehren gescheiterten Anläufen der versuchten Implementierung des Darknet Fine-tuning galt es alternative Ansätze zu finden und umzusetzen. Der Mangel an Modellen hinsichtlich der im Paragraph (Datenbeschaffung) genannten Gründe des Datenschutzes und der Wirtschaftlichkeit, galt es auf ein alternativ Ansatz zurückzugreifen. Hierbei hat man auf eine allgemeine und Repetitive Eigenschaft eines Frontalen PKW Bildes gesetzt. Anderst als mittels eines trainierten Neuronalen Netzes auf die Erkennung von an Autos befestigter Nummer Schildern. Macht man sich deren Rechteckige Eigenschaften zu Gunsten und gestaltet eine Pipeline die auf die Erkennung von Rechtecken innerhalb eines Bildes.

### 3.4 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 gescannten Dokument in für Menschen verständlichen Text umzuwandeln. Für diesartige Anwendungen gibt es eine Vielzahl von Anwendungsfällen in der heutigen Zeit. Dazu gehört beispielsweise die Umwandlung Informationen, die ausschließlich in einem Print-Medium zu finden sind,

---

<sup>1</sup>Luxonis1.

in ein digitales Medium, die einfache Erkennung eines Coupons oder auch das Auslesen eines schriftlich ausgefüllten Formulars mit Hilfe des Computers.<sup>2</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 auch ohne einen Aufwand für die Kosten der Texterkennung diese durchzuführen.

## 3.5 Validierung

Nach erfolgreichem Proof-of-Concept musste unter den technisch möglichen Varianten der Verarbeitungspipelines die beste ausgewählt werden. Hierzu wurde ein Skript geschrieben, dass 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 konnten schlussendlich auch weiterführende Optimierungen vorgenommen und Fehlerquellen lokalisiert werden.

## 3.6 Evaluation der Pipeline

Die Kennzeichenerkennungspipeline besteht aus folgenden Schritten:

vor gespeicherten Datei  
 Umwandlung in Graustufen  
 Bilateraler Filter  
 Canny-Algorithmus zu Kantenfindung  
 Erkennung von Rechtecken  
 Cropping

---

<sup>2</sup>Vgl. Bekhit 2022, S. 81.

R mit vorgefertigter Lösung

Im ursprünglichen Plan gab es am Anfang dieser Pipeline ein neuronales Netz, mit dem eine erste Eingrenzung des Kennzeichens erfolgen sollte. Dieser Schritt wurde jedoch aufgrund von technischen Problemen entfernt.

Als OCR-Lösung war EasyOCR geplant, weil es trotz geringem Ressourcenaufwand gute Ergebnisse liefert. Dieser Ansatz scheiterte, weil EasyOCR von Torch abhängt, Torch ein 64bit-Betriebssystem voraussetzt und Pi OS für den Raspberry Pi 3 nur als 32bit-Version verfügbar ist. Die Alternative ist 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/tesseractidcard2007.pdf>

### 3.6.1 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.

### 3.6.2 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 einzuordnen.

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

Abbildung 3.1: 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 3.2: 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 `0G 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 3.3: 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.

### 3.6.3 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 easyOCR 2,27 Sekunden und für die Bilderkennung mit Tesseract 2,67 Sekunden benötigt. Das entspricht einer Gesamtlaufzeit 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.

### 3.6.4 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 benötigten. 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.



# 4 Verbindung

## 4.1 Sprachassistenten-Steuerung

## 4.2 Weboberfläche

## 4.3 RFID

## 4.4 Ultraschallsensor

## 4.5 Piezo-Lautsprecher

## 4.6 Steckplan

Auf ?? ist die schlussendlich verwendete Steckung abgebildet. Der dazugehörige Elektrische Schaltplan findet sich in Anlage X

## 4.7 Schaltlogik

## 4.8 Weboberfläche

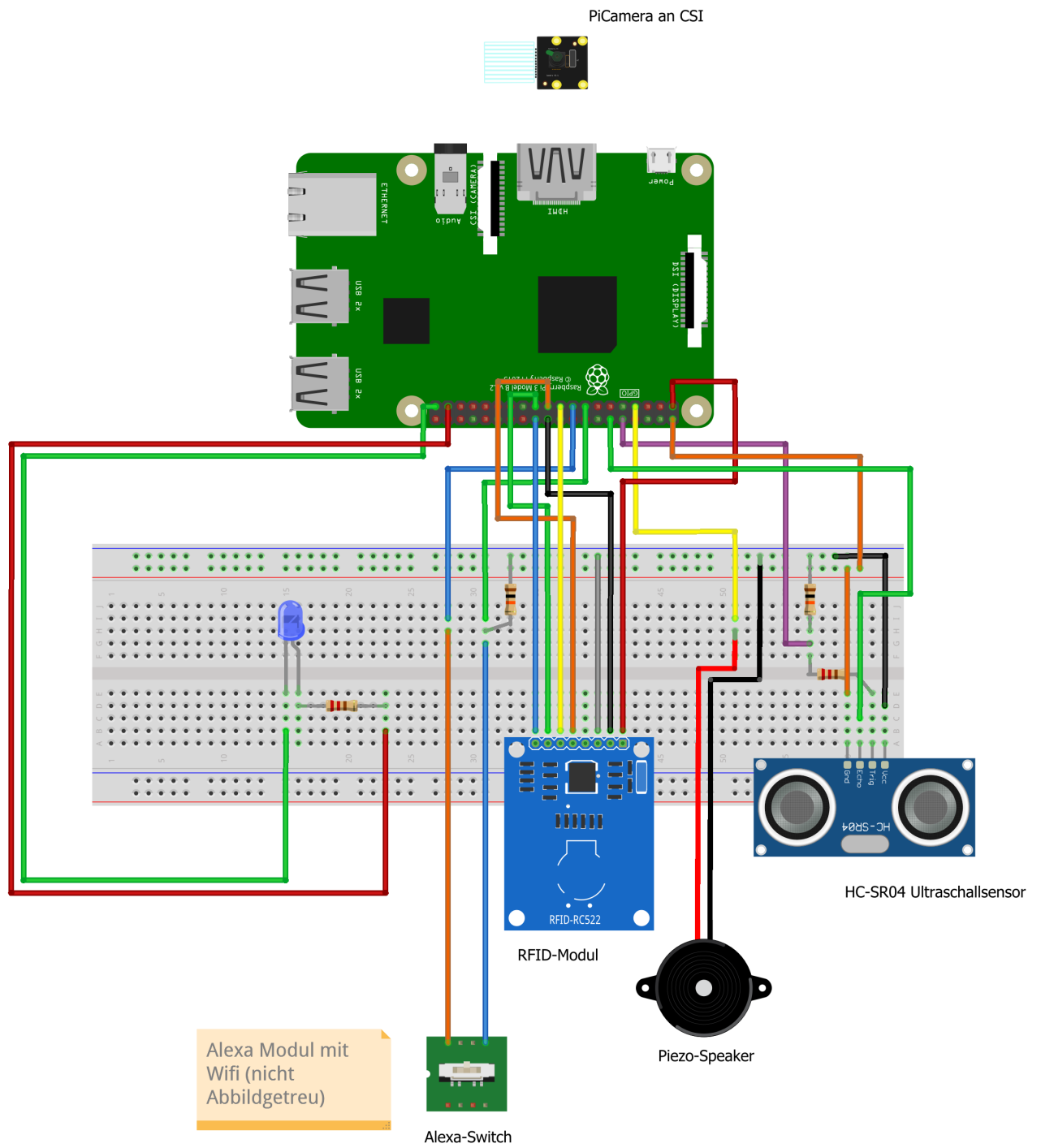


Abbildung 4.1: Steckplan der gesamten Hardware  
Quelle: Eigene Darstellung

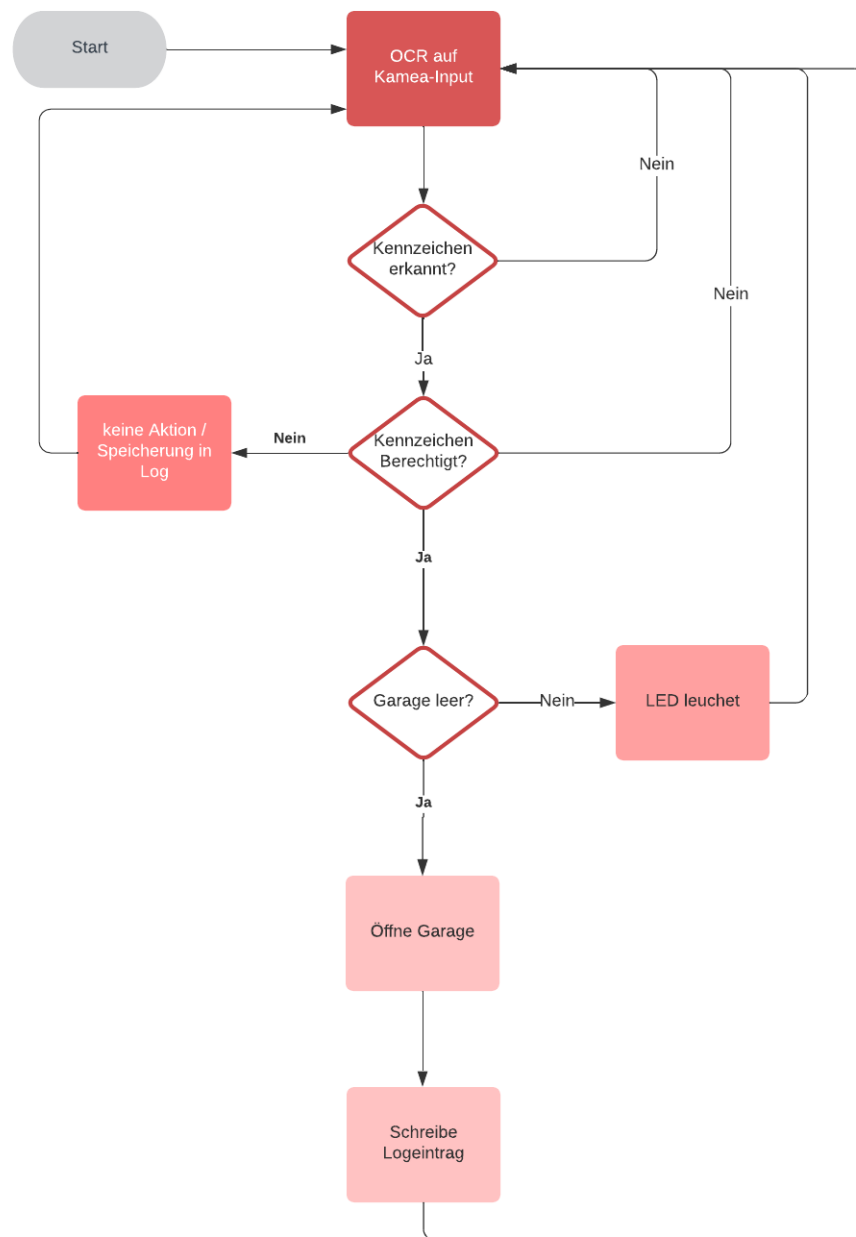
Programmablaufplan Garagentoröffner

Abbildung 4.2: Ablauf der Kennzeichenerkennungs-Schleife  
Quelle: Eigene Darstellung

# 5 Risiken und Limitierungen

Mehrere problematische Punkte konnten innerhalb der relativ kurzen Projektlaufzeit noch nicht ausreichend adressiert werden.

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 natürlich durch die optische Erkennung 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

## 6 Wirtschaftliche Aspekte

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

3x Workshops a 12 Stunden 1x 24 Stunden

### Materialkosten

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
<b>Summe</b>	<b>253,39</b>

Materialkosten je Einheit Raspberry Pi 3 39,95 RFID-Modul RC552 5,00 Ultraschallmodul HC-SR04 4,99

# 7 Zusammenfassung

Dieses Kapitel enthält die Zusammenfassung der Arbeit mit Fazit und Ausblick.

## 7.1 Fazit

Zusammenfassend lässt sich feststellen, dass die Implementierung der

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 und Darknet hier 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 und es für den Alltagsgebrauch einer Weiterentwicklung bedarf.

## 7.2 Ausblick

Das Projekt SmartGarage bietet in vielen Punkten Potential und Ausblick auf weitere Features. Besonders der Ausblick auf die kommerzielle Nutzung des Systems ist vielversprechend. Grund hierfür ist, dass es aktuell noch kein Gesamtpaket kommerziell zu kaufen gibt, das alle Features, die im Projekt implementiert sind, enthält. Auch die kommerzielle Nutzung

im Öffentlichen Raum ist möglich. So beispielsweise bei Tiefgaragen sowie Parkhäusern. Hierbei besteht die Möglichkeit das Projekt in einer abgewandten Form kommerziell nutzbar zu machen. Hierbei ist es beispielsweise denkbar, die Weboberfläche so umzugestalten, dass dort Parkplätze bereits vorab am Handy oder Tablet gebucht werden können und der Kunde sein Nummernschild angibt. Ist der Kunde dann vor Ort kann er aufgrund der Nummernschilderkennung einfach und ohne weiteren Aufwand in das Parkhaus oder die Tiefgarage fahren und hat bereits einen reservierten Parkplatz. Dies ermöglicht potentiellen Betreibern von diesem System, die Parkdauer des Kunden direkt zu berechnen, da das Loggen der Ein- und Ausfahrten in diesem Anwendungsbeispiel diese Information bereits mitliefert.

Einen weiteren Nutzen für die Öffentliche Sicherheit kann das Projekt mit einer Anbindung an die Polizei bieten. Durch die Nummernschilderkennung wird bei Einfahrt das Nummernschild erkannt und es ist im System ersichtlich, welches Auto in welcher Garage geparkt hat. Dies macht es wie im Anwendungsbeispiel der Tiefgaragen und Parkhäuser möglich, dass die Polizei direkt nachverfolgen kann, wo ein Nummernschild zuletzt geparkt hat. Wird so beispielsweise ein Verkehrsunfall mit Fahrerflucht gemeldet, bei dem der flüchtige Fahrer danach zu einem Kaufhaus fährt, welches über ein Parkhaus mit einem solchen System verfügt, kann die Polizei den flüchtigen Fahrer nach Einfahrt in das Parkhaus direkt lokalisieren. Dies ist auch im Falle von Privatgaragen möglich, sollten diese auch die Information über Ein- und Ausfahrt mit der Polizei teilen. Im Falle dessen bietet das System hier auch den Vorteil, dass die Zustellung von Strafzetteln vereinfacht wird, da hier die Strafzettel immer an die richtige Adresse gesendet werden.

Potential zur Verbesserung weist das Projekt im Punkt Sicherheit auf. Hierbei ist die Rede von sowohl Sicherheit im Sinne von Schutz vor einem Einbruch als auch Sicherheit im Sinne von Unfallvermeidung. Die Vermeidung von Unfällen spielt vor allem eine Rolle, falls es sich um ein Garagentor handelt, welches sich nach außen öffnet. Hierbei ist es sinnvoll, auch eine Abstandsmessung außerhalb durchzuführen, um das Öffnen der Garage zu blockieren, falls sich etwas vor dem Tor befindet. Hierbei kann es sonst zu Kollisionen zwischen dem Tor der Garage und beispielsweise einem Auto kommen. Kommt es jedoch vor, dass vor der Garage Passanten laufen könnten, könnte es hierbei auch zu schwereren Unfällen kommen, im Falle dass das Garagentor mit einem vorbeifahrenden Fahrrad kollidiert. Die Sicherheit vor einem Einbruch lässt sich auch durch verschiedene Methoden noch optimieren. Eine Möglichkeit ist hierbei beispielsweise die Implementierung einer Sperre, die das Garagentor mittels Nummernschilderkennung nicht öffnet, sollte sich ein Auto bereits in der Gara-

ge befinden. Eine derartige Implementierung schützt nicht nur vor eventuell gefälschten Nummernschildern, sondern auch vor False-Positives im System, wodurch die Garage fehlerhafterweise bei der Einfahrt eines Kennzeichens, welches nicht in der Whitelist steht, geöffnet werden kann.



## 8 Technischer Ausblick

Die Nutzung eines 64bit-Betriebssystems würde die Nutzung von easyOCR ermöglichen, was in der Evaluation zu deutlich besserer Performance geführt hat. Mit zusätzlichem Aufwand ist denkbar, die ursprünglich geplante Eingrenzung des Kennzeichens durch ein neuronales Netz, welches auf der Myriad X VPU läuft, nachträglich einzubauen und die Schwächen der aktuellen geometrischen Lösung zu umgehen. easyOCR basiert auf Torch und es ist laut Intel möglich, PyTorch-Modelle in die Intermediate Representation umzuwandeln. easyOCR könnte also auf die Myriad X VPU portiert werden, um noch bessere Performance zu erreichen.

# A Ablaufplan

# Literaturverzeichnis

- Bekhit, Ahmed Fathi (2022). *Computer Vision and Augmented Reality in iOS*. Springer Verlag. ISBN: 978-1-4842-7461-3.
- eWeLink DC5V (2022). URL: <https://www.kaufland.de/product/348563329/> (besucht am 15.02.2022).
- 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.
- Internet der Dinge - Weltweit (2022). URL: <https://de.statista.com/outlook/tmo/internet-der-dinge/weltweit> (besucht am 20.01.2022).
- Kern, Christian (2006). *Anwendung von RFID-Systemen*. Springer Verlag. ISBN: 978-3-540-27725-5.
- OpenCV AI Kit: OAK—D— OpenCV.AI (o.D.). <https://store.opencv.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/>.
- 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.
- Smart Home - Weltweit (2022). URL: <https://de.statista.com/outlook/dmo/smart-home/weltweit> (besucht am 20.01.2022).