

# DIPLOMARBEIT

**Advanced Parking Monitoring (APM)**



---

**Durchgeführt 2020/21 von**

Philipp Kraft

Dennis Köb

Samuel Bleiner

**Betreuer**

---

Dipl.-Ing. Christoph Stüttler

---

**Abgabevermerk**

Original, 20.04.2021

Dipl.-Ing. Christoph Stüttler

Digital, 20.04.2021

AV Dipl.-Ing. Leopold Moosbrugger

## Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Rankweil, 20. April 2021

---

Philipp Kraft

---

Dennis Köb

---

Samuel Bleiner

## Kurzfassung

In der nachfolgenden Arbeit wird ein System vorgestellt, welches die Parkplatzverwaltung und -überwachung vereinfachen soll. Um zu erreichen, dass diese Lösung modular und individuell auf jedem Parkplatz eingesetzt werden kann, werden zwei eigenständige Module, eines für die Kennzeichenerkennung und eines für die Fahrzeugerkennung entwickelt. Diese Module sollen ihre Daten auf eine Datenbank hochladen können, welche dann über eine intuitive Web-Applikation verwaltet werden kann. Mithilfe der Kennzeichenerkennung sollen die unterschiedlichen Fahrzeuge eindeutig identifiziert werden und damit auch der Status des Fahrzeuges auf dem Parkplatz überwacht werden können. Die Fahrzeugerkennung detektiert den Status einer einzelnen Parklücke und dadurch kann überwacht werden, welche Parklücke besetzt ist und welche nicht. Diese Daten werden in der Web-Applikation anschaulich dargestellt, welche neben umfangreichen Darstellungsmöglichkeiten auch eine eigene Benutzeroberfläche für Parkplatzbetreiber und Nutzer bietet. Dieses System ist durch seinen modularen Aufbau in der Lage, sowohl Firmenparkplätze als auch öffentliche Parkplätze jeglicher Größe zu bedienen.

## Abstract

In the following paper, a system is presented, which shall improve parking lot surveillance and management. To make sure that this solution can be used individually on every kind of parking lot, there will be developed two independent modules, one for license plate detection and one for vehicle detection. These modules shall be able to send their data to a database, which can be accessed and controlled via an intuitive web application. With the license plate detection, it shall be possible to identify each vehicle and watch their status on the parking lot. The vehicle detection detects the status of each parking spot and therefore it can be monitored which parking spot is currently taken and which is free to be used. These data are visualized in the web application, which offers extensive display options as well as different user interfaces for parking lot owners and customers. Through its modular structure, the system can provide great service for company parking lots and public parking lots of any size.

## Vorwort

In den Jahren unserer Ausbildungszeit ist uns immer wieder aufgefallen, dass viele öffentliche gebührenpflichtige Parkplätze nicht mit neuen Innovationen betrieben werden. Daraufhin haben wir uns durch zusätzliche Anregung von unserem Betreuungslehrer Dipl.-Ing. Christoph Stüttler dazu entschieden, diese Thematik für unsere Diplomarbeit heranzuziehen.

Diese Diplomarbeit soll aufzeigen, wie eine modernes System zur Parkplatzverwaltung aussehen könnte. Es werden in dieser Arbeit verschiedene Themenbereiche wie Webinterfaces, künstliche Intelligenz und Mikrocontroller behandelt.

## Danksagung

Mit dieser Seite wollen wir uns bei allen Personen bedanken die uns zum Erfolg unserer Diplomarbeit geführt haben. Wir bedanken uns ausdrücklich bei unserem Betreuungslehrer Dipl.-Ing. Christoph Stüttler für seine fachliche Expertise sowie für alle kreativen Ratschläge während des gesamten Betreuungszeitraumes.

Unser hauptsächlicher Dank gilt der Firma Omicron für die finanzielle Unterstützung ohne welche die Diplomarbeit in dieser Form nicht existieren würde.

Ein herzliches Dankeschön geht an unsere Eltern für ihre Hilfe und für ihren Beistand während unserer gesamten Ausbildungszeit.

# Inhaltsverzeichnis

<b>Eidesstattliche Erklärung</b>	i
<b>Kurzfassung</b>	ii
<b>Abstract</b>	iii
<b>Vorwort</b>	iv
<b>Danksagung</b>	v
<b>1 Projektteam</b>	10
<b>2 Projektbetreuer</b>	11
<b>3 Projektsponsor</b>	11
<b>4 Projektplanung</b>	12
4.1 OpenProject . . . . .	12
4.2 Phasen . . . . .	12
4.3 Arbeitspakete . . . . .	12
4.4 Gantt-Diagramm . . . . .	14
4.5 Zeitaufwendungen . . . . .	17
<b>5 Einleitung</b>	18
<b>6 Kennzeichenerkennung</b>	19
6.1 Einleitung . . . . .	19
6.2 Anforderungen . . . . .	19
6.3 Bildverarbeitung . . . . .	20
6.3.1 Einleitung . . . . .	20
6.3.2 Bilaterale Filterung . . . . .	21
6.3.3 Thresholding . . . . .	22
6.3.4 Erosion . . . . .	23
6.3.5 Farbraum . . . . .	23

6.3.5.1	RGB . . . . .	24
6.3.5.2	Graustufen . . . . .	24
6.3.5.3	BGR . . . . .	24
6.3.6	Konturerkennung . . . . .	24
6.4	Kennzeichenerkennungsprogramm . . . . .	25
6.4.1	Einleitung . . . . .	25
6.4.2	Programmiersprache . . . . .	25
6.4.3	Konzept . . . . .	26
6.4.3.1	Bildaufnahme . . . . .	26
6.4.3.2	Kennzeichenerfassung . . . . .	26
6.4.3.3	Kennzeichensegmentierung . . . . .	26
6.4.3.4	Zeichenerkennung . . . . .	27
6.4.3.5	Anbindung an Datenbank . . . . .	27
6.4.4	Ablauf . . . . .	27
6.4.5	Verwendete Libraries . . . . .	28
6.4.5.1	Versionsübersicht der verwendeten Libraries . . . . .	28
6.4.5.2	Jupyter Notebook . . . . .	29
6.4.5.3	Numpy . . . . .	30
6.4.5.4	OpenCV . . . . .	31
6.4.5.5	Matplotlib . . . . .	33
6.4.5.6	Keras . . . . .	34
6.4.5.7	Tensorflow . . . . .	34
6.4.5.8	local_utils . . . . .	37
6.4.5.9	Scikit-learn . . . . .	37
6.4.5.10	Requests . . . . .	37
6.4.5.11	Gpiozero . . . . .	38
6.4.5.12	Picamera . . . . .	39
6.4.6	Wichtige Programmteile . . . . .	40
6.4.6.1	WPOD-NET-Modell laden . . . . .	40
6.4.6.2	Kennzeichen lokalisieren . . . . .	41
6.4.6.3	Bild aufnehmen . . . . .	42

6.4.6.4	Zeichen mittels Konturerkennung finden . . . . .	44
6.4.6.5	Bildverarbeitung . . . . .	45
6.4.6.6	Visualisierung mittels Matplotlib . . . . .	46
6.4.6.7	Modell für Zeichenerkennung laden und anwenden . . . . .	48
6.4.6.8	Resultat mittels API an Datenbank senden . . . . .	50
6.5	Raspberry Pi . . . . .	51
6.5.1	Einleitung . . . . .	51
6.5.2	Wahl des Raspberry Pi . . . . .	51
6.5.3	Kamera . . . . .	52
6.6	Maschinelles Lernen . . . . .	53
6.6.1	Einleitung . . . . .	53
6.6.2	Definition . . . . .	54
6.6.3	Vergleich Maschinelles Lernen / Bildverarbeitung . . . . .	54
6.6.4	Verwendete Modelle für Maschinelles Lernen . . . . .	56
6.6.4.1	WPOD-NET . . . . .	56
6.6.4.2	MobileNetV2 . . . . .	57
6.7	Modell Training . . . . .	58
6.8	Gehäuse . . . . .	61
6.8.1	Anforderungen . . . . .	61
6.8.2	Umsetzung . . . . .	62
6.9	Test . . . . .	65
6.9.1	Einleitung . . . . .	65
6.9.2	Erster Test . . . . .	65
6.9.3	Zweiter Test . . . . .	66
6.9.4	Fazit . . . . .	71
<b>7</b>	<b>Fahrzeugerkennung</b>	<b>72</b>
7.1	Anforderungen . . . . .	72
7.2	Vorstudie . . . . .	72
7.3	Erkennung von Metallen über Spulen . . . . .	72
7.3.1	Messung ferromagnetischer Metalle . . . . .	75

7.3.1.1	RL-Oszillator mit Timer Baustein . . . . .	76
7.3.2	Messung paramagnetischer Metalle . . . . .	81
7.3.2.1	LC Oszillatoren . . . . .	82
7.3.2.2	Einfluss von Induktivitätsänderungen auf LC-Oszillatoren . . . . .	83
7.3.2.3	Messung der Frequenz eines Colpitts-Oszillator . . . . .	85
7.4	RS485 Bussystem . . . . .	88
7.4.1	Benutzte Standards . . . . .	88
7.4.1.1	ASCII . . . . .	88
7.4.1.2	USB . . . . .	88
7.4.1.3	UART . . . . .	88
7.4.1.4	RS485 . . . . .	89
7.4.1.5	RJ45 . . . . .	91
7.4.2	Aufbau . . . . .	92
7.4.2.1	Funktion der Adresslogikleitung . . . . .	94
7.4.2.2	Versorgungsleitungen VCC und GND . . . . .	96
7.4.3	Implementation eines eigenen Protokolls . . . . .	97
7.4.3.1	Aufbau von Datenframes . . . . .	98
7.4.3.2	Steuerabläufe . . . . .	99
7.5	Mikrocontroller Slave-Gerät . . . . .	104
7.5.1	Überblick . . . . .	104
7.5.2	Mikrocontroller . . . . .	106
7.5.3	Lineare Spannungsregler . . . . .	107
7.5.4	RS485 Pegelwandler . . . . .	108
7.5.5	Oszillator mit LM393D Komparator . . . . .	108
7.5.6	Digitale Ein- und Ausgänge IO0 /IO1 . . . . .	110
7.5.7	weitere Anschlüsse . . . . .	111
7.5.7.1	Separater VCC Eingang . . . . .	111
7.5.7.2	RJ45 Anschlüsse . . . . .	111
7.5.7.3	SPI Programmierananschluss . . . . .	111
7.5.8	Atmega328PB . . . . .	112
7.5.8.1	Zuweisung der Pins . . . . .	112

7.5.8.2	Fuse-Bits und Quartz . . . . .	113
7.5.8.3	SPI-Programmierung . . . . .	116
7.5.9	Software des Slaves . . . . .	119
7.5.9.1	Konfiguration der Register . . . . .	119
7.5.9.2	UART Betrieb . . . . .	121
7.5.9.3	Auswertung von Datenframes . . . . .	122
7.5.9.4	Taktmessung . . . . .	124
7.5.10	Layout des Slave-Gerätes . . . . .	127
7.5.11	Gehäuse . . . . .	129
7.6	USB-Master . . . . .	129
7.6.1	USB-Bussadapter Gerät . . . . .	130
7.6.1.1	FT232RL . . . . .	130
7.6.1.2	USB-C Anschluss . . . . .	132
7.6.1.3	Layout des Master-Geräts . . . . .	133
7.6.1.4	Gehäuse . . . . .	134
7.6.2	Master Programm . . . . .	134
7.6.2.1	Erkennung des USB-Geräts . . . . .	134
7.6.2.2	Frequenzauslesung . . . . .	135
7.6.2.3	Auswertung . . . . .	136
7.6.2.4	API-Post . . . . .	137
7.6.3	RaspberryPi als Mastergerät . . . . .	138
7.6.3.1	SSH Remote Zugriff . . . . .	138
7.6.3.2	Code Deployment . . . . .	139
<b>8</b>	<b>Webinterface</b> . . . . .	<b>141</b>
8.1	Einleitung . . . . .	141
8.2	Verwendete Technologien . . . . .	142
8.2.1	HTML . . . . .	142
8.2.1.1	Beispielhafte HTML Seite . . . . .	142
8.2.2	CSS . . . . .	144
8.2.3	JavaScript . . . . .	146

8.2.4	PHP . . . . .	147
8.2.5	TailwindCSS . . . . .	147
8.2.6	Laravel . . . . .	147
8.3	Lokale Entwicklungsumgebung mit Laragon . . . . .	149
8.3.1	Einleitung . . . . .	149
8.3.2	Benötigte Software . . . . .	149
8.3.3	Konfiguration von PHP . . . . .	150
8.3.4	Installation von phpMyAdmin . . . . .	152
8.4	Lokale Entwicklungsumgebung mit WSL und Docker . . . . .	153
8.4.1	Einleitung . . . . .	153
8.4.2	Benötigte Software . . . . .	153
8.4.3	Installation von WSL . . . . .	153
8.4.3.1	1. Schritt: WSL aktivieren . . . . .	153
8.4.3.2	2. Schritt: Virtual Machine aktivieren . . . . .	154
8.4.3.3	3. Schritt: Linux Kernel Update . . . . .	154
8.4.3.4	4. Schritt: WSL 2 . . . . .	154
8.4.3.5	5. Schritt: Linux-Distribution herunterladen . . . . .	154
8.4.4	Installation von Docker . . . . .	155
8.4.5	Laravel Sail . . . . .	156
8.5	Production Server . . . . .	156
8.5.1	Einleitung . . . . .	156
8.5.2	Benötigte Software . . . . .	156
8.5.3	Installation des LAMP Stacks . . . . .	157
8.5.3.1	Apache . . . . .	157
8.5.3.2	PHP . . . . .	158
8.5.3.3	MariaDB . . . . .	158
8.5.3.4	phpMyAdmin . . . . .	159
8.5.3.5	Webinterface Virtual Host . . . . .	162
8.5.3.6	Installation von Composer . . . . .	162
8.5.4	Deployment mit Github Actions . . . . .	163
8.5.4.1	Git Setup . . . . .	163

8.5.4.2	Deploy Script . . . . .	164
8.5.4.3	Server Deploy Script . . . . .	165
8.5.4.4	Github Actions . . . . .	166
8.6	Backend . . . . .	168
8.6.1	Einleitung . . . . .	168
8.6.2	Routing . . . . .	169
8.6.2.1	Routing in Laravel . . . . .	169
8.6.2.2	Verfügbare Web Routen . . . . .	169
8.6.3	Controller . . . . .	171
8.6.4	Artisan CLI . . . . .	172
8.6.5	Datenbank . . . . .	173
8.6.5.1	Seeder . . . . .	173
8.6.5.2	Migrations . . . . .	174
8.6.6	Eloquent ORM . . . . .	176
8.6.7	Laravel Sanctum . . . . .	177
8.6.8	Sessions . . . . .	177
8.6.9	Middleware . . . . .	178
8.6.10	Service Provider . . . . .	178
8.6.11	Form Validation . . . . .	178
8.7	Frontend . . . . .	179
8.7.1	Einleitung . . . . .	179
8.7.2	Blade Templates . . . . .	179
8.7.3	Components . . . . .	180
8.7.3.1	Anonymous Components . . . . .	180
8.7.3.2	Components erstellen . . . . .	180
8.7.3.3	Components verwenden . . . . .	181
8.7.3.4	Attribute übergeben . . . . .	181
8.7.4	Layouts . . . . .	182
8.7.5	Navbar . . . . .	185
8.7.6	Content Header . . . . .	187
8.7.7	Session Alerts . . . . .	188

8.7.8	Sidebar . . . . .	190
8.7.9	Footer . . . . .	193
8.8	Funktionen . . . . .	193
8.8.1	Einleitung . . . . .	193
8.8.2	Dashboard . . . . .	193
8.8.3	Benutzerverwaltung . . . . .	195
8.8.3.1	Liste aller Benutzer . . . . .	196
8.8.3.2	Benutzer erstellen/editieren . . . . .	197
8.8.3.3	Profilbilder . . . . .	198
8.8.4	Rollen- und Rechteverwaltung . . . . .	200
8.8.4.1	Liste der verfügbaren Rechte . . . . .	200
8.8.4.2	Vergabe von Rechten und Rollen mit Slugs . . . . .	202
8.8.4.3	Authentisierung . . . . .	203
8.8.4.4	Rolle erstellen . . . . .	204
8.8.5	Neuigkeiten . . . . .	205
8.8.6	Parkplatzverwaltung . . . . .	207
8.8.6.1	Parklücken . . . . .	208
8.8.7	Kennzeichenverwaltung . . . . .	209
8.8.8	Erkennungsverlauf . . . . .	212
8.8.9	Seiten Einstellungen . . . . .	213
8.8.9.1	Funktion der Einstellungen . . . . .	214
8.8.10	Profil . . . . .	215
8.8.11	Lokalisierung . . . . .	216
8.8.11.1	Funktion der Sprachauswahl . . . . .	216
8.9	API . . . . .	218
8.9.1	Einleitung . . . . .	218
8.9.2	Autorisierung mit Bearer Token . . . . .	218
8.9.3	API Tokens erstellen . . . . .	219
8.9.4	Erkennungen . . . . .	220
8.9.4.1	GET /api/v1/detections . . . . .	220
8.9.4.2	GET /api/v1/detections/{id} . . . . .	220

8.9.4.3	POST /api/v1/detections . . . . .	223
8.9.5	Kennzeichen . . . . .	223
8.9.5.1	GET /api/v1/plates . . . . .	223
8.9.5.2	GET /api/v1/plates/{id} . . . . .	224
8.9.6	Parkplätze . . . . .	225
8.9.6.1	GET /api/v1/lots . . . . .	225
8.9.6.2	GET /api/v1/lots/{id} . . . . .	226
8.9.7	Parklücken . . . . .	228
8.9.7.1	GET /api/v1/spots . . . . .	228
8.9.7.2	GET /api/v1/spots/{id} . . . . .	229
8.9.7.3	POST /api/v1/lots/{id}/spots . . . . .	231
8.9.8	Einstellungen . . . . .	233
8.9.8.1	GET /api/v1/settings . . . . .	233
8.9.9	Benutzer . . . . .	234
8.9.9.1	GET /api/v1/users . . . . .	234
8.9.9.2	GET /api/v1/users/{id} . . . . .	235
<b>9</b>	<b>Zusammenfassung und Ausblick</b>	<b>237</b>
<b>10</b>	<b>Anhang</b>	<b>238</b>
<b>Abbildungsverzeichnis</b>		<b>239</b>
<b>Tabellenverzeichnis</b>		<b>244</b>
<b>Codeverzeichnis</b>		<b>245</b>
<b>Abkürzungsverzeichnis</b>		<b>249</b>
<b>Literaturverzeichnis</b>		<b>251</b>

## 1 Projektteam



**Philipp Kraft**

E-Mail: Mail@Philipp-Kraft.com



**Dennis Köb**

E-Mail: Dennis.Koeb@gmail.com



**Samuel Bleiner**

E-Mail: Bleiner.Samuel@gmail.com

## 2 Projektbetreuer



**Dipl.-Ing. Christoph Stüttler**

E-Mail: [Christoph.Stuettler@ht-rankweil.at](mailto:Christoph.Stuettler@ht-rankweil.at)

## 3 Projektsponsor



**OMICRON electronics GmbH**

Oberes Ried 1, 6833 Klaus

Telefon: +43 59495

## 4 Projektplanung

Die Projektplanung ist der wichtigsten Teil des Projektmanagements. Die Rahmenbedingungen für das Projekt wurden bereits beim Antrag der Diplomarbeit festgelegt.

### 4.1 OpenProject

Für das Projektmanagement wird die Softwareanwendung OpenProject<sup>1</sup> verwendet. Der Vorteil dieser Software ist, dass diese nicht wie eine normale Desktop-Anwendung lokal auf dem Rechner des Benutzers läuft, sondern auf einem Server. Ein weiterer Vorteil davon ist, dass das kollaborative arbeiten stark vereinfacht wird. OpenProject bietet die Möglichkeit die Anwendung in der eigenen Infrastruktur zu installieren und somit eine vollständige Kontrolle über die eigenen Daten.

### 4.2 Phasen

Die gesamte Projektplanung ist in vier Phasen aufgeteilt:

- **Themenfindungsphase**

In der Themenfindungsphase geht es um das Finden des expliziten Themas, dabei werden die Rahmenbedingungen mit dem Betreuer festgelegt.

- **Planungsphase**

In der Planungsphase werden Arbeitspakete erstellt und den einzelnen Mitgliedern des Projektes zugewiesen.

- **Entwicklungsphase** Die Entwicklungsphase ist der längste und wichtigste Teil des Projekts, dabei werden alle Arbeitspakete so gut wie möglich abgearbeitet.

- **Abschlussphase** In der Abschlussphase geht es um das Schreiben der Diplomarbeit so wie letzte technische Feinschliffe in der Software und Hardware.

### 4.3 Arbeitspakete

Grundsätzlich sind die Arbeitspakete in drei Themenbereiche aufgeteilt:

---

<sup>1</sup><https://www.openproject.org>

- Kennzeichenerkennung (Samuel)
- Fahrzeugerkennung (Dennis)
- Webinterface (Philipp)

TYPE	ID ↑	SUBJECT
PHASE	42	Themenfindungsphase
PHASE	43	▼ Planungsphase
TASK	62	Logo
TASK	63	Projektname
TASK	64	Aufgabenverteilung
TASK	65	Individuelle Recherche
MILESTONE	44	Abgabe Antrag Schule
MILESTONE	54	Hochladen des Antrags
PHASE	55	▼ Entwicklungsphase
TASK	49	▼ Kennzeichenerkennung
TASK	72	OpenCV Kennzeichenerkennung
TASK	75	Python Kennzeichenerkennung mit Machine Learning
TASK	77	RaspberryPi Kamera
TASK	78	Kennzeichenerkennungssoftware auf RaspberryPi
TASK	79	Kamerabilder als Input
TASK	80	API Anwendung
TASK	51	▼ Webinterface
TASK	58	Rollenverwaltung
TASK	59	Kennzeichen
TASK	70	Benutzerverwaltung
TASK	71	Rechtesystem
TASK	73	Erkennungen
TASK	74	Parkplätze
TASK	76	API

Abbildung 1: Arbeitspakete Teil 1

<b>TASK</b>	84	▼ Fahrzeugerkennung
<b>TASK</b>	85	Simulation der Schaltungen
<b>TASK</b>	86	Schaltungsanalyse
<b>TASK</b>	87	Konzeptprototyp
<b>TASK</b>	88	Slave Schaltungsdesign
<b>TASK</b>	89	PCB-Design
<b>TASK</b>	90	Aufbau & Messung des Slave Prototyp
<b>TASK</b>	91	Gehäuse CAD zeichnen und 3D drucken
<b>TASK</b>	92	Slave Firmware schreiben
<b>TASK</b>	93	Master Anwendung schreiben
<b>TASK</b>	94	Raspberry Pi Umgebung einrichten / Remote Zugriff
<b>PHASE</b>	57	Abschlussphase
<b>MILESTONE</b>	60	Zwischenpräsentation
<b>MILESTONE</b>	66	Vorabgabe
<b>MILESTONE</b>	67	Elektronik Forum
<b>MILESTONE</b>	68	Hochladen der Arbeit
<b>MILESTONE</b>	69	Schriftliche Abgabe

Abbildung 2: Arbeitspakete Teil 2

#### 4.4 Gantt-Diagramm

Gantt-Diagramme oder Balkenplan sind spezielle Balkendiagramme um verschiedene Arbeitspakete oder Aktivitäten auf einer Zeitachse darzustellen.

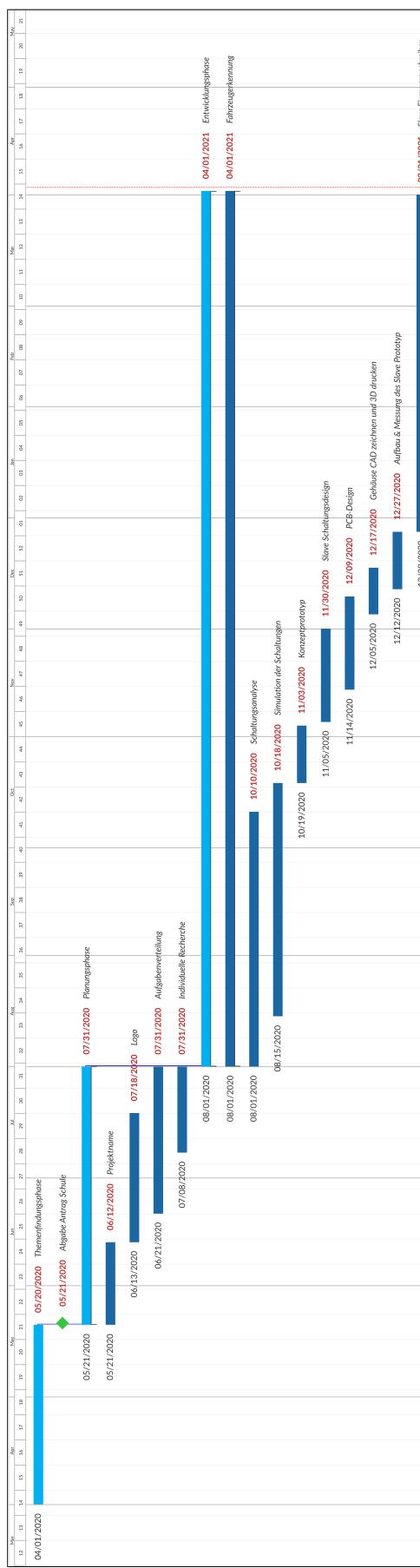


Abbildung 3: Gantt-Chart Teil 1

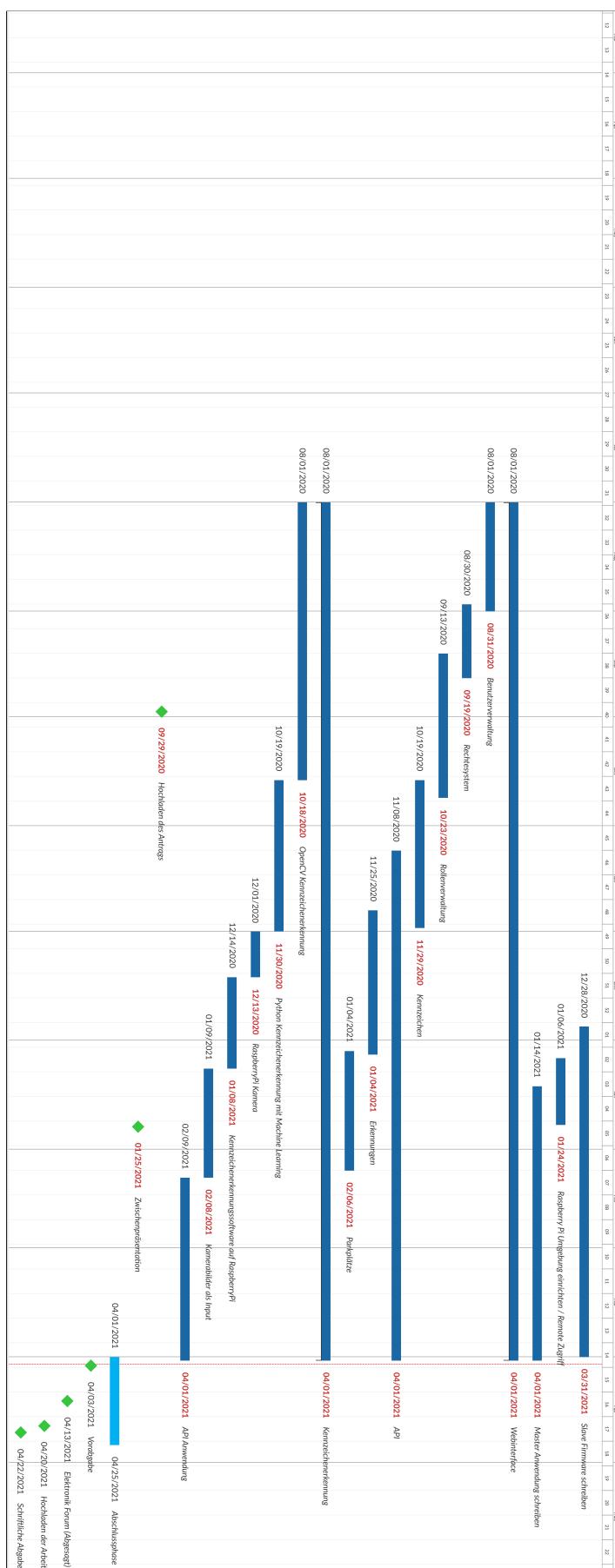


Abbildung 4: Gantt-Chart Teil 2

## 4.5 Zeitaufwendungen

Die Dokumentation der Zeitaufwendungen erfolgt über ein Tabellenkalkulationsprogramm, dabei wird folgendes dokumentiert:

- Datum
- Typ
- Beschreibung
- Aufwand in Stunden

Die komplette Übersicht der aufgewendeten Stunden ist im Anhang ersichtlich.

<b>Zeitaufwendungen</b>	
Philipp Kraft	181.5 h
Dennis Köb	177 h
Samuel Bleiner	131.5 h
Gesamtaufwand: 490 h	

Tabelle 1: Zeitaufwendungen Übersicht

## 5 Einleitung

Das Ziel dieser Arbeit ist es, eine Vereinfachung von Parkplatzüberwachungen zu erstellen, welche möglichst überall eingesetzt werden kann. Dazu wird die Arbeit in drei individuelle Teile aufgeteilt, welche einfach miteinander verbunden werden können, um so einen modularen Aufbau bereitzustellen. Im ersten Teil geht es um die Erkennung der Kennzeichen von Fahrzeugen. Dies wird beim Betreten und Verlassen des Parkplatzes eingesetzt, um, festzustellen welches Fahrzeug sich im Moment auf dem Parkplatz befindet. Zudem können dadurch unerlaubte Fahrzeuge entdeckt werden und anschließend mögliche Schritte eingeleitet werden. Im zweiten Teil geht es um die Erkennung von Fahrzeugen in einer Parklücke. Dazu wird eine Spule unter jeder Parklücke verwendet, mit welcher festgestellt werden kann, ob sich im Moment dort ein Fahrzeug befindet. Dadurch kann festgestellt werden, welche Parkplätze besetzt sind und im Falle von mehrstöckigen Parkhäusern kann die Auslastung der einzelnen Etagen überwacht werden. Im dritten Teil geht es dann noch um die Darstellung und Verwaltung dieser Daten. Dazu wird eine Web-Applikation bereitgestellt, welche alle Daten übersichtlich darstellt und für den Parkplatzbetreiber und den Kunden unterschiedliche Benutzeroberflächen bietet. Zudem kann die Web-Applikation mit wenig Aufwand von jedem Parkplatzbetreiber personalisiert werden, was vor allem für Firmen interessant ist, wenn sie diese in ihr Intranet einbauen möchten. Die Verbindung dieser drei Teile erfolgt über eine eigens dafür geschriebenen API, wodurch die Daten über das Internet übertragen werden können.

## 6 Kennzeichenerkennung

### 6.1 Einleitung

Der Teil der Kennzeichenerkennung beschäftigt sich damit, aus aufgenommenen Fotos der einfahrenden und ausfahrenden Fahrzeuge, das Kennzeichen auszulesen und dieses an die Datenbank zu senden. Dies wird benötigt um damit es möglich ist zu bestimmen welches Fahrzeug sich im Moment auf dem Parkplatz befindet. Zudem kann dadurch jedes Fahrzeug eindeutig identifiziert werden, da das Kennzeichen einzigartig sein muss. In Zusammenhang mit der Datenbankverwaltung können so auch bestimmte Kennzeichen definiert werden, welche anders behandelt werden als andere Kennzeichen. Ein Beispiel dafür wäre kostenloses Parken für den Parkplatzbesitzer oder auch gesperrte Kennzeichen, bei welchen der Parkplatzbetreiber eine Benachrichtigung bekommt, wenn sich diese auf dem Parkplatz befinden. Die Verbindung zur Datenbank erfolgt über eine eigene API, mit welcher die Daten schnell und einfach übertragen werden können. Um das Programm für die Kennzeichenerkennung zu realisieren werden sowohl bewährte Bildverarbeitungsalgorithmen eingesetzt als auch Neuronale Netze, welche sich in den Bereich des Maschinellen Lernens einordnen lassen, wodurch die Genauigkeit gesteigert werden kann. Zusätzlich wird für das komplette Modul ein Gehäuse gefertigt in welchem die Elektronik fest verbaut werden kann und trotzdem die Möglichkeit zu weiteren Verbesserungen ermöglicht.

### 6.2 Anforderungen

Das Modul für die Kennzeichenerkennung muss einige Anforderungen erfüllen, um für seinen Einsatzzweck bestmöglich geeignet zu sein.

Als erstes muss die Software einige Anforderungen erfüllen. Sie muss eine hohe Genauigkeit erreichen, um in möglichst jedem einzelnen Fall ein korrektes Ergebnis zu liefern. Um diese Genauigkeit zu erreichen werden mehrere Modell für Maschinelles Lernen verwendet, mit welchen eine bessere Genauigkeit als mit anderen Methoden erreicht werden kann. Die Geschwindigkeit ist hingegen ein nicht ganz so wichtiger Punkt, da Fahrzeuge für eine erfolgreiche Erkennung nicht zu schnell fahren dürfen, um eine passende Fotoaufnahme zu ermöglichen. Dadurch wird der Zeitintervall zwischen den Fahrzeugen erhöht, weswegen der Fokus bei der Software nahezu vollständig auf die Genau-

igkeit gelegt werden kann und die Geschwindigkeit bis zu einem gewissen Punkt vernachlässigt werden kann. Das bedeutet, dass die Software für einen Durchlauf mehrere Sekunden benötigen darf, ohne dadurch auch nur annähernd die Anwendung zu behindern. Außerdem muss die Leistung eines Raspberry Pi 4 2GB für die Software ausreichen, da dieser für das Modul verwendet wird.

Als nächstes kommt der Auslöser für die Fotoaufnahme. Dieser muss so auslösen, dass das Fahrzeug sich in einer Position vor der Kamera befindet und muss unabhängig von der Art des Fahrzeuges, also egal ob es ein kleines Auto oder ein großer SUV ist, ungefähr an der gleichen Position auslösen. In dieser Applikation wird dafür ein manueller Button verwendet, aber in einer möglichen Verbesserung wäre es sinnvoll diesen durch eine Lichtschranke oder ähnliches zu ersetzen.

Der letzte Punkt ist dann noch das Gehäuse. Dieses muss stabil genug sein, um fest verbaut zu werden, Zugang zu den wichtigsten Anschlüssen bieten und eine Halterung für die Kamera sollte auch integriert sein. Um für die Entwicklung ein geeignetes Gehäuse zu entwickeln wurde hier ein Gehäuse mit einem 3D-Drucker erstellt. Das Gehäuse wird von oben und unten verschraubt und bietet die Möglichkeit die Kamera mit Schrauben zu sichern. Die Anschlüsse für die Stromversorgung, die USB-Ports, die HDMI-Ports, den Ethernet-Anschluss und die GPIO-Pins sind zugänglich damit es auch weiterhin möglich ist damit zu arbeiten, auch wenn kleine Änderungen durchgeführt werden. Zudem ist es dadurch auch möglich zu Test- und Entwicklungszwecken einen Bildschirm anzuschließen.

## 6.3 Bildverarbeitung

### 6.3.1 Einleitung

Die Bildverarbeitung ist ein zentrales Thema in dieser Applikation für Kennzeichenerkennung. Sie wird für die Zeichensegmentierung verwendet, sowie für die Vorbereitung von Bildern für andere Algorithmen. Im Folgenden werden die verwendeten Bildverarbeitungsfunktionen aufgelistet und deren Funktionsweise erläutert.

### 6.3.2 Bilaterale Filterung

Bilaterale Filterung ist eine Methode für eine kantenerhaltende Weichzeichnung eines Bildes.

Bei der Berechnung für den Farbwert des Ausgabepixels werden die benachbarten Pixel nicht nur mit ihrer Entfernung gewichtet, sondern auch mit ihrem eigenen Farbwert. Dadurch können einzelne farbliche Ausreißer herausgefiltert werden. Dies ist vor allem in der Bildverarbeitung wichtig, da dadurch die wichtigen Eigenschaften eines Bildes, wie zum Beispiel Kanten, erhalten bleiben und verarbeitet werden können, aber einzelne abweichende Pixel herausgefiltert werden wodurch unnötige Informationen entfernt werden.



Abbildung 5: Vor Bilateraler Filterung



Abbildung 6: Nach Bilateraler Filterung

In Abbildung 5 kann man ein Bild von verschiedenen Lebensmitteln sehen. Wenn man genau hinsieht erkennt man vor allem bei den Blättern im Hintergrund und beim Brot viele detailreiche Texturen. Diese Texturen haben keine wichtige Texturen und sind deswegen unnötig. Um die Bildverarbeitung zu vereinfachen wendet man deswegen die bilaterale Filterung auf dieses Bild an, um diese detailreichen Texturen zu vereinfachen. In Abbildung 6 sieht man das Bild nach der bilateralen Filterung. Wenn man hier dann wieder genauer auf die Blätter und das Brot sieht, erkennt man, dass die detailreichen Texturen weichgezeichnet wurden, aber die Kanten sind genauso gut erkennbar wie vor der Filterung.

### 6.3.3 Thresholding

Das Thresholding oder auch Schwellenwertverfahren wird in der Bildverarbeitung verwendet, um Bilder zu segmentieren. Aus einem Graubild kann dadurch ein Binäres Bild erzeugt werden.

Bei diesem Verfahren wird ein bestimmter Schwellwert (En.: Threshold) definiert, welcher mit den Grauwerten der einzelnen Pixel des Bildes verglichen wird. Wenn der Grauwert den Schwellwert überschreitet, wird dieser durch einen weißen Pixel ersetzt und wenn der Grauwert kleiner als der Schwellwert ist, wird dieser durch einen schwarzen Pixel ersetzt. Dadurch erhält man ein Bild welches nur noch zwei Farben hat, Schwarz und Weiß. Dies wird deswegen eingesetzt, da dadurch viele Bildverarbeitungsalgorithmen schneller arbeiten und die Effizienz gesteigert wird.



Abbildung 7: Graustufenbild



Abbildung 8: Binäres Bild nach Thresholding

In Abbildung 7 sieht man ein solches Graubild welches nur verschiedene Graustufen aufweist. In Abbildung 8 sieht man das Bild nach dem Thresholding. Hier kann man nur noch das Boot mit den Menschen erkennen. Dies ist nicht nur für schnellere Bildverarbeitungsalgorithmen wichtig, sondern wird auch zur Objekterkennung in Bildern verwendet.

Um den Schwellwert zu bestimmen kann man diesen entweder variieren bis das gewünschte Ergebnis erscheint oder man verwendet Methoden, welche den Schwellwert automatisch bestimmen. Eine der bekanntesten Methoden zur Schwellwertbestimmung ist die Methode von Otsu<sup>2</sup>, welche mit dem Schwellenwert die Pixel in Vordergrund und Hintergrund unterteilt.

<sup>2</sup>Benannt nach Nobuyuki Otsu

### 6.3.4 Erosion

Erosion ist eine Funktion der Bildverarbeitung und ist in die morphologische Bildverarbeitung einzutragen. Diese beschäftigt sich primär mit der Verarbeitung von binären Bildern, welche man nach Thresholding erhält.

Erosion benötigt zwei Eingaben, das binäre Bild und einen Kernel. Der Kernel ist dabei die Angabe, nach welcher die Erosion durchgeführt wird. Der Kernel ist auch eine binäre Struktur, welche über jeden einzelnen Pixel des binären Bildes geschoben wird. Wenn der Kernel komplett mit dem binären Bild übereinstimmt, behält dieser Pixel seinen Wert und ansonsten wird er invertiert. Dabei muss jedoch darauf geachtet werden, dass die Polarität des binären Bildes und des Kernels übereinstimmt, da sonst die Erosion nicht richtig funktioniert. Als Resultat erhält man danach ein deutlicheres Bild bei welchem einzelne Pixelfehler herausgefiltert wurden und die Konturen besser erkennbar sind.



Abbildung 9: Binäres Bild nach Thresholding



Abbildung 10: Nach Erosion

In Abbildung 9 und 10 sieht man die Anwendung der Erosion. Die Konturen der einzelnen Zeichen im Kennzeichen sind in Abbildung 10 nach der Erosion deutlicher erkennbar als davor.

### 6.3.5 Farbraum

Der Farbraum eines Bildes enthält alle möglichen Farben eines Farbmodells. Das Farbmodell beschreibt dabei die Parameter, aus welchen die einzelnen Farben gebildet werden. Dies ist in der Bildverarbeitung relevant, da verschiedene Funktionen der Bildverarbeitung, unterschiedliche Farbräume verwenden und dieser deswegen korrekt eingestellt werden muss.

In dieser spezifischen Applikation werden die folgenden Farbräume verwendet:

### 6.3.5.1 RGB

RGB ist einer der häufigsten und bekanntesten Farbräume. Er basiert auf den drei Grundfarben Rot, Grün und Blau und wird vor allem bei Bildschirmen und in der Fotografie genutzt. Die Farben setzen sich in diesem Modell aus dem jeweiligen Rot-, Grün- und Blauanteil der einzelnen Pixel zusammen.

### 6.3.5.2 Graustufen

Bei einem Graustufen-Bild, zu sehen in Abbildung 3, hat jeder Pixel einen Wert von 0 bis 255. Diese Werte erstrecken sich also von Schwarz bis Weiß und dazwischen liegen verschiedene Grautöne. Dieser Farbraum wird in der Bildverarbeitung häufig verwendet, da Konturen einfacher erkennbar sind und es nur einen Parameter gibt, welcher verarbeitet werden muss, wodurch die Effizienz diverser Algorithmen gesteigert werden kann. Zudem wird dieser Farbraum auch oft in Verbindung mit Thresholding verwendet.

### 6.3.5.3 BGR

Der BGR ist ein relativ unbekannter und wenig verwendeter Farbraum, da er sehr ähnlich zum RGB-Farbraum ist. Der einzige Unterschied zwischen diesen beiden liegt in der Anordnung der Parameter. Bei BGR sind die Parameter spiegelverkehrt zu RGB, das heißt es kommt zuerst der Blauanteil, dann der Grünanteil und zum Schluss der Rotanteil. Insgesamt ergibt dies für die einzelnen Pixel zwar die gleichen Farben, aber die Funktionen der Bildverarbeitung müssen trotzdem das Bild im passenden Farbraum erhalten. So verwendet zum Beispiel die Funktion „imread“ von OpenCV den BGR-Farbraum und die Funktion „im Show“ von Matplotlib verwendet den RGB-Farbraum. Wenn man diese Funktionen also nacheinander anwendet, muss dazwischen der Farbraum umgewandelt werden.

### 6.3.6 Konturerkennung

Die Konturerkennung ist eine wichtige Funktion in der Bildverarbeitung mit welcher Objekte in einem Bild gefunden werden können. In dieser Applikation wird sie für die Zeichensegmentierung eingesetzt.

Die Konturerkennung wird hauptsächlich bei binären Bildern verwendet. Eine Kontur kann dabei

wie im Folgenden definiert werden. Man überprüft jeden einzelnen Pixel und sieht nach, ob ein benachbarter Pixel einen anderen Farbwert aufweist. Falls dies zutrifft muss der zu prüfende Pixel zu einer Kontur gehören. Wenn dies auf mehrere zusammenhängende Pixel zutrifft, bedeutet das, dass diese zusammen eine Kontur bilden.

Die Funktion „findcontours“ von OpenCV, welche in dieser Applikation verwendet wird, ist eine Funktion für Konturerkennung und kann weiße Objekte auf einem schwarzen Hintergrund erkennen. Sie basiert auf dem Algorithmus von Suzuki von 1985<sup>3</sup> und liefert eine Liste mit allen Konturen. Die Konturen werden in der Liste als ein Array von Koordinaten abgespeichert.

## 6.4 Kennzeichenerkennungsprogramm

### 6.4.1 Einleitung

Die Software ist der wichtigste und größte Teil der Kennzeichenerkennung. Sie erhält ein Bild, in welchem ein Auto mit einem Kennzeichen enthalten ist und liefert am Ende dieses Kennzeichens und sendet dieses dann automatisch an die Datenbank. Die Software kann entweder über Bildverarbeitung oder mit Modellen für Maschinelles Lernen realisiert werden. Der erste Ansatz bei dieser Anwendung war mit klassischer Bildverarbeitung, welche aber nicht die gewünschte Genauigkeit erreicht hat, weswegen dann auf Maschinelles Lernen gewechselt wurde.

### 6.4.2 Programmiersprache

Die verwendete Programmiersprache für die Kennzeichenerkennung ist Python. Python ist eine höhere Programmiersprache, welche übersichtlich und leicht lesbar ist. Sie ist vor allem für Bildverarbeitung und Anwendungen mit Maschinellem Lernen gut geeignet, da es dafür hoch optimierte und effiziente Bibliotheken gibt wie zum Beispiel OpenCV, Numpy und Tensorflow. Dadurch ist Python für diese Anwendung besser geeignet als zum Beispiel C++. Dieses wäre zwar normalerweise effizienter, bietet aber weniger optimierte Bibliotheken in diesem Bereich, wodurch es hier weniger gut geeignet ist.

<sup>3</sup>Topological structural analysis of digitized binary images by border following

### 6.4.3 Konzept

Das Programm für die Kennzeichenerkennung basiert auf fünf Stufen. Die erste Stufe ist die Bildaufnahme, die zweite ist die Kennzeichenerfassung mittels Maschinellem Lernen, die dritte ist die Kennzeichensegmentierung mithilfe von Bildverarbeitung, die vierte ist die Zeichenerkennung mittels Maschinellem Lernen und die fünfte ist die Anbindung an die Datenbank.

#### 6.4.3.1 Bilddurchsuche

Um ein Bild verarbeiten zu können und aus diesem ein Kennzeichen auslesen zu können, muss zuerst ein Bild vorliegen. Dieses wird über den RaspberryPi mit der RaspberryPi-Kamera aufgenommen. Um das Bild aufzunehmen, muss einfach ein Auslöser aktiviert werden und dann wird das Bild aufgenommen und im richtigen Ordner abgespeichert. Zuvor wird noch überprüft ob sich in diesem Ordner bereits ein Bild befindet und falls eines vorhanden ist wird es gelöscht. Dadurch werden mögliche Fehler durch mehrere Bilder verhindert.

#### 6.4.3.2 Kennzeichenerfassung

Die Kennzeichenerfassung hat die Aufgabe, das Kennzeichen im Eingabebild zu lokalisieren. Dies geschieht mittels Maschinellem Lernen mit dem Modul WPOD-NET von Sérgio Montazolli Silva und Cláudio Rosita Jung<sup>4</sup>. Dieses verwendet zuerst das Modul YOLOv2 welches zur Echtzeitobjekterkennung verwendet werden kann und in dieser Anwendung zur Erkennung von Fahrzeugen verwendet wird. Danach werden die Koordinaten des Kennzeichens ermittelt und dieses aus dem Bild ausgeschnitten und abgespeichert.

#### 6.4.3.3 Kennzeichensegmentierung

Die Kennzeichensegmentierung hat das Ziel die einzelnen Zeichen im Kennzeichen zu separieren und so zu vorbereiten, dass die darauffolgende Zeichenerkennung damit arbeiten kann. Dazu wird das Bild mit dem Kennzeichen zuerst in Graustufen konvertiert, dann mit einem bilateralen Filter gefiltert, mit Thresholding in ein binäres Bild umgewandelt und anschließend mittels Erosion besser erkennbar gemacht. Danach werden im verarbeiteten Bild die Konturen gesucht, sortiert und anhand dieser die einzelnen Zeichen herausgefiltert.

<sup>4</sup>License Plate Detection and Recognition in Unconstrained Scenarios

#### 6.4.3.4 Zeichenerkennung

Die letzte Stufe der Kennzeichenerkennung ist die Zeichenerkennung. In dieser werden die einzelnen Zeichen erkannt und als Text abgespeichert. Dies funktioniert über eine eigenes Neuronales Netz basierend auf MobileNetV2<sup>5</sup>, welches mit einem Datensatz von über 35 000 Bildern auf die Erkennung von Zeichen aus Bildern trainiert wurde.

#### 6.4.3.5 Anbindung an Datenbank

Nachdem das Kennzeichen als Text abgespeichert wurde, muss diese Information in die Datenbank übergeben werden. Dazu wird die eigene API angewandt, welcher man diese Informationen übergeben muss und als Rückgabe die Information bekommt, ob sich das Fahrzeug nun innerhalb oder außerhalb des Parkplatzes befindet.

#### 6.4.4 Ablauf

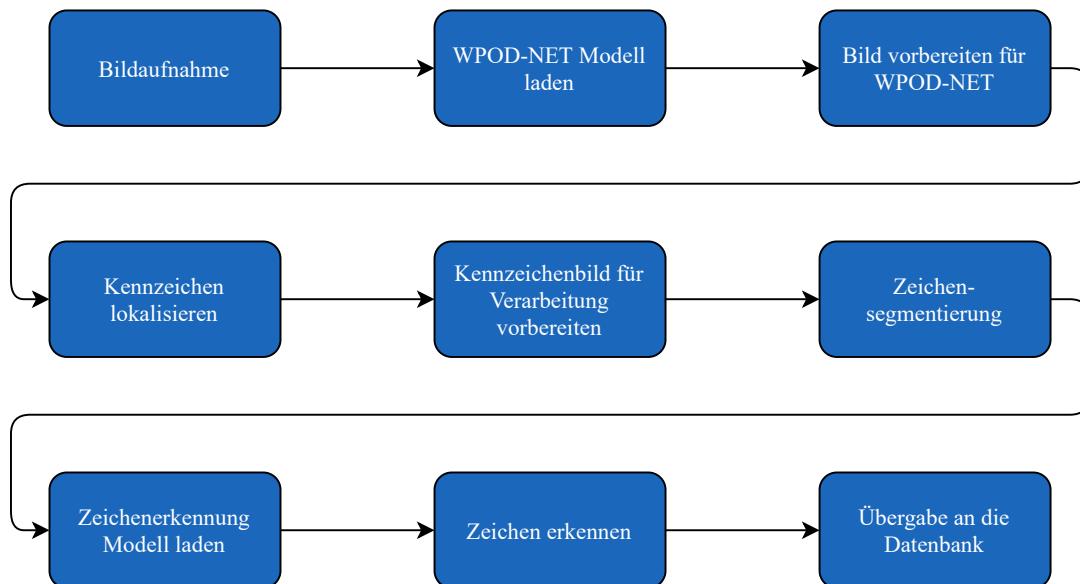


Abbildung 11: Ablaufdiagramm der Kennzeichenerkennung

<sup>5</sup>Neuronales Netz für Computer Vision

Im oberen Diagramm ist der Ablauf des Programms angegeben. Zuerst wird mit einem Button der Auslöser betätigt und damit das Foto aufgenommen. Dann wird das erste Modell für Maschinelles Lernen für die Kennzeichenerkennung „WPOD-NET“<sup>6</sup> geladen. Bevor das Bild diesem Modell übergeben werden kann, muss es noch angepasst werden damit das Modell damit arbeiten kann. Danach kann damit das Kennzeichen im Bild lokalisiert werden. Im Anschluss wird dieses Kennzeichenbild mit mehreren Bildverarbeitungsalgorithmen verarbeitet, um dann die einzelnen Zeichen zu segmentieren. Danach kann dann das Modell für die Zeichenerkennung geladen werden und dieses dann auch angewendet werden, um das Ergebnis zu erhalten. Dieses Ergebnis wird dann noch mit einer API an die Datenbank übergeben.

## 6.4.5 Verwendete Libraries

### 6.4.5.1 Versionsübersicht der verwendeten Libraries

Im Folgenden werden die verwendeten Versionen der benötigten Libraries aufgelistet, um das Programm zu starten. Dies wird noch einmal unterteilt in die Versionen, welche auf Windows 10 benötigt werden und jene auf Raspbian<sup>7</sup>, da auf Raspbian manche Versionen noch nicht verfügbar sind, neuere und verbesserte Versionen verfügbar sind oder manche Versionen nur auf komplizierten Umwegen installierbar sind.

Windows:

- h5py = 2.10.0
- imutils = 0.5.3
- Keras = 2.4.3
- matplotlib = 3.3.2
- notebook = 6.1.5
- numpy = 1.18.5
- opencv-python = 4.4.0.44
- scikit-learn = 0.23.2

<sup>6</sup>Modell für Maschinelles Lernen für Kennzeichenerfassung

<sup>7</sup>Raspbian ist ein Unix-basiertes Betriebssystem für den Raspberry Pi

- tensorflow = 2.3.1
- requests = 2.24.0

Raspbian:

- h5py = 2.10.0
- imutils = 0.5.3
- Keras = 2.4.3
- matplotlib = 3.3.3
- notebook = 6.1.5
- numpy = 1.19.4
- opencv-python = 4.4.0.46
- scikit-learn = 0.24.0
- tensorflow = 2.4.0
- requests = 2.21.0

#### 6.4.5.2 Jupyter Notebook

Jupyter Notebook ist eine nützliche Erweiterung für Python, wenn es um den Bereich der Daten-Visualisierung und Maschinelles Lernen geht. Sie ist eine Unteranwendung des Open-Source Projektes „Project Jupyter“, welches von großen Partnern wie Microsoft und Google unterstützt und verwendet wird. Mit Jupyter Notebook ist es möglich in einer Web-Anwendung ein Live-Script abzuarbeiten und in Kombination mit Matplotlib die Daten visuell darzustellen, abzuspeichern und zu überwachen. Im Gegensatz zur direkten Darstellung in der Entwicklungsumgebung, wird der letzte Durchgang des Scripts, inklusive aller Bilder und Grafiken gespeichert. Zudem ist es auch hervorragend für das Training und die dazugehörende Überwachung von Modulen für Maschinelles Lernen geeignet. Außerdem bietet Jupyter Notebook noch die Möglichkeit mehrere Scripts parallel abzuarbeiten und diese individuell zu überwachen. In dieser Applikation wird es für ein solches

Training und für die anschauliche Visualisierung von Daten genutzt.

Die Installation von Jupyter Notebook ist simpel, da man es einfach über pip<sup>8</sup> mit dem folgenden Befehl installieren kann:

```
1 pip install notebook
```

Code 1: PIP Installation von Jupyter Notebook

Um das Jupyter Notebook aufzurufen muss im Terminal „jupyter notebook“ eingegeben werden und dann öffnet sich automatisch die Web-Applikation. Danach hat man Zugriff auf komplette Dateistruktur des eigenen Projektes und kann die einzelnen Python-Scripts starten. Dabei ist zu beachten, dass man keine gewöhnliche .py-Datei benötigt, sondern eine .ipynb-Datei benötigt, wozu man einfach eine Kopie des normalen Scripts erstellt und die Dateiendung ändert.

#### 6.4.5.3 Numpy

Numpy ist eine weit verbreitete Library für Python, welche sich mit der Berechnung und Verarbeitung von mehrdimensionalen Arrays beschäftigt. Damit können komplexe Berechnungen und Algorithmen oft effizienter verarbeitet werden, wobei aber beachtet werden muss, dass der Code an die mehrdimensionalen Arrays angepasst werden muss. Numpy selbst ist auch eine Voraussetzung für OpenCV, welches Numpy-Arrays verwendet, um Bilder zu verarbeiten. Dazu werden 3-dimensionale Arrays verwendet, wodurch die Verarbeitung dieser Bilder und die Bildverarbeitungsalgorithmen schneller sind. In dieser Applikation wird Numpy in Zusammenarbeit mit OpenCV verwendet und auch für die weitere Verarbeitung der Daten von OpenCV, um zum Beispiel die Koordinaten des Kennzeichens zu speichern.

Um Numpy zu installieren muss folgender Befehl in der Konsole eingegeben werden:

```
1 pip install numpy
```

Code 2: PIP Installation von Numpy

<sup>8</sup>Paketverwaltungsprogramm für Python

#### 6.4.5.4 OpenCV

OpenCV ist die wichtigste und mächtigste Library die in dieser Applikation verwendet wird. Sie wurde ursprünglich in C++ geschrieben, aber es gibt auch eine Version in Python, welche in dieser Applikation verwendet wird. OpenCV ist eine freie Library für Bildverarbeitung, Computer Vision<sup>9</sup> und Maschinelles Lernen, welche von Intel gestartet wurde und mittlerweile die wichtigste und am weitesten verbreitete Library in diesem Bereich ist. Sie verwendet Numpy-Arrays, um für eine effiziente Verarbeitung von Bildern zu sorgen. Einige der wichtigsten Funktionen stellen die klassischen Bildverarbeitungsalgorithmen wie zum Beispiel Filterung und Farbraumanpassungen, die Verarbeitung und Auswertung von Kamerabildern und auch die Kompatibilität mit Deep-Learning<sup>10</sup> dar. In dieser Applikation wird OpenCV für diverse Bildverarbeitungsalgorithmen, die Zusammenarbeit mit Modellen für Maschinelles Lernen, die Verarbeitung eines Kamerabildes und das allgemeine Arbeiten mit Bildern verwendet.

Die Installation von OpenCV ist dabei etwas komplizierter als bei anderen Librarys.

Windows:

Unter Windows ist die Installation vergleichbar mit anderen Libraries, es muss einfach der folgende Befehl in der Konsole eingegeben werden:

```
1 pip install opencv-python
```

Code 3: PIP Installation von OpenCV

Raspbian:

Auf dem Raspberry Pi gestaltet sich die Installation von OpenCV um einiges schwieriger, da die Installation nicht über pip gemacht werden kann, sondern es manuell kompiliert werden muss.

<sup>9</sup>Die Computergestützte Auswertung und Verarbeitung von Kamerabildern

<sup>10</sup>Methode für Maschinelles Lernen welche Neuronale Netze nutzt

Als erstes werden alle Tools und Bibliotheken installiert, welche für OpenCV benötigt werden. Dazu verwendet man folgenden Befehl im Terminal:

```
1 $ sudo apt-get install build-essential git cmake pkg-config libjpeg8-dev
  ↳ libtiff4-dev libjasper-dev libpng12-dev libavcodec-dev
  ↳ libavformat-dev libswscale-dev libv4l-dev libgtk2.0-dev
  ↳ libatlas-base-dev gfortran
```

Code 4: Installation benötigter Tools und Bibliotheken für OpenCV

Danach kann man mit dem nächsten Befehl OpenCV von einem GitHub-Repository klonen:

```
1 $ sudo apt-get install build-essential git cmake pkg-config libjpeg8-dev
  ↳ libtiff4-dev libjasper-dev libpng12-dev libavcodec-dev
  ↳ libavformat-dev libswscale-dev libv4l-dev libgtk2.0-dev
  ↳ libatlas-base-dev gfortran
```

Code 5: Klonen von OpenCV von GitHub

Im nächsten Schritt wird OpenCV mit den folgenden Befehlen kompiliert:

```
1 cd ~/opencv && mkdir build && cd build
2
3 cmake -D CMAKE_BUILD_TYPE=RELEASE \
4 -D CMAKE_INSTALL_PREFIX=/usr/local \
5 -D INSTALL_PYTHON_EXAMPLES=ON \
6 -D INSTALL_C_EXAMPLES=ON \
7 -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
8 -D BUILD_EXAMPLES=ON ..
9 make -j4
```

Code 6: Kompilieren von OpenCV

Wenn das Kompilieren erfolgreich beendet wurde, kann OpenCV abschließend installiert werden.

```
1 $ sudo make install && sudo ldconfig
```

Code 7: Abschließende Installation von OpenCV

Danach sollte OpenCV fertig installiert und eingerichtet sein und es kann in einem Python Projekt verwendet werden.

#### 6.4.5.5 Matplotlib

Matplotlib ist eine Python-Library mit welcher Daten und Berechnungen visuell dargestellt werden können. Damit können statische, animierte und interaktive Diagramme erstellt werden, was die Datenauswertung um einiges erleichtert. Die Syntax ähnelt sehr stark jener von MATLAB, wodurch die Bedienung sehr einfach ist, wenn man schon Erfahrung mit MATLAB hat. Wie auch in MATLAB kann man die Diagramme beliebig anordnen und dadurch das Layout der Darstellung selbst festlegen. Es funktioniert auch hervorragend in Zusammenarbeit mit Jupyter Notebook, wodurch man die Daten in einer Web-Applikation visualisieren kann. Die visuelle Darstellung von Daten ist vor allem in der Entwicklung und beim Arbeiten mit visuellen Ergebnissen wie Bildern von Vorteil. In dieser Applikation wird Matplotlib für die Darstellung der Ergebnisse von Bildverarbeitungsalgorithmen, die Ausgabe von Daten und Resultaten und auch für Test- und Entwicklungszwecke verwendet. Im unteren Bild kann man ein Beispiel sehen bei welchem Matplotlib verwendet wird, um einige Bilder mit einem Titel anzuzeigen.

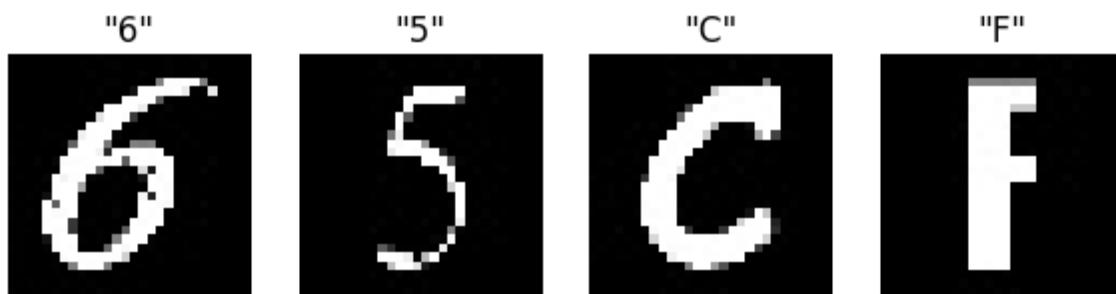


Abbildung 12: Beispiel einer Anwendung von Matplotlib

Um Matplotlib zu installieren muss das Folgende in der Konsole eingegeben werden:

```
1 pip install matplotlib
```

Code 8: PIP Installation von Matplotlib

#### 6.4.5.6 Keras

Keras ist eine Deep-Learning Schnittstelle für diverse Machine Learning Frameworks wie zum Beispiel Tensorflow oder Theanos. Damit wird die Bedienung und Anwendung dieser Frameworks vereinfacht und es bietet auch diverse Funktionen, um Inputs mit den Modellen für Maschinelles Lernen kompatibel zu machen. Keras ist ein Teil von Tensorflow, wird aber eigenständig weiterentwickelt, um die Kompatibilität mit anderen Machine Learning Frameworks aufrechtzuerhalten. In dieser Applikation wird es in Zusammenarbeit mit Tensorflow verwendet, um mit den verwendeten Modellen für Maschinelles Lernen zu arbeiten.

Um Keras zu installieren führt man folgenden Befehl in der Konsole aus:

```
1 pip install keras
```

Code 9: PIP Installation von Keras

#### 6.4.5.7 Tensorflow

Tensorflow ist eines der weltweit beliebtesten Frameworks für Maschinelles Lernen und wird von weltweit erfolgreichen Firmen wie zum Beispiel Google, AMD oder auch Intel verwendet. Es bietet eine umfassende Plattformen für jegliche Anwendungen für Maschinelles Lernen und ist in Zusammenarbeit mit APIs wie zum Beispiel Keras leicht zu verwenden. In dieser Applikation wird damit ein neuronales Netz trainiert und mehrere Modelle für Maschinelles Lernen verwaltet und verwendet.

Bei der Installation von Tensorflow muss beachtet werden, dass sich diese von Windows zu Raspbian stark unterscheidet. Raspbian unterstützt offiziell die benötigte Version von Tensorflow noch nicht und deswegen muss dieses über ein paar Umwege installiert werden.

Windows:

Die Installation von Tensorflow unter Windows ist einfach, da man es wie andere Libraries einfach über pip installieren kann.

```
1 pip install tensorflow
```

Code 10: PIP Installation von Tensorflow

Raspbian:

Bei Raspbian muss Tensorflow manuell kompiliert werden, da die aktuelle Version noch nicht offiziell unterstützt wird. Diese funktioniert aber trotz dieses Umweges einwandfrei und wird mit den folgenden Befehlen in der Konsole installiert.

Mit dem ersten Befehl werden die benötigten Tools installiert:

```
1 $ sudo apt-get install cmake curl
```

Code 11: Benötigte Tools für Tensorflow

Danach kann man die neuste Tensorflow Version von GitHub herunterladen:

```
1 $ wget -O tensorflow.zip  
→ https://github.com/tensorflow/tensorflow/archive/v2.4.0.zip
```

Code 12: Tensorflow von GitHub herunterladen

Anschließend muss Tensorflow entpackt werden:

```
1 $ unzip tensorflow.zip  
2 $ mv tensorflow-2.4.0 tensorflow  
3 $ cd tensorflow
```

Code 13: Entpacken von Tensorflow

Im nächsten Schritt müssen noch zusätzlich erforderliche Libraries installiert werden:

```
1 $ ./tensorflow/lite/tools/make/download_dependencies.sh
```

Code 14: Zusätzlich erforderliche Libraries

Danach kann die Installation kompiliert werden:

```
1 $ ./tensorflow/lite/tools/make/build_aarch32_lib.sh
```

Code 15: Kompilieren von Tensorflow

Danach muss man die Installation mit den folgenden Befehlen abschließen:

```
1 $ cd ~/tensorflow/tensorflow/lite/tools/make/downloads/flatbuffers  
2 $ mkdir build  
3 $ cd build  
4 $ cmake ..  
5 $ make -j4  
6 $ sudo make install  
7 $ sudo ldconfig
```

Code 16: Abschließen der Installation von Tensorflow

Nach diesem Schritt sollte Tensorflow Installation funktionieren und man kann es wie jede andere Library in Python einbinden.

#### 6.4.5.8 local\_utils

local\_utils ist ein einfaches Python-Script, welches die Arbeit mit WPOD-NET im Bereich der Kennzeichenerkennung vereinfacht und in dieser Applikation für die einfachere Nutzung von WPOD-NET verwendet wird. Es beinhaltet die Funktion „detect\_lp“ mit welcher ein Bild des Kennzeichens und die Koordinaten des Kennzeichens ermittelt werden können. Dieses Script kann über den folgenden Link von GitHub heruntergeladen werden: [https://github.com/quangnhat185/Plate-detect\\_and\\_recognize/blob/master/local\\_utils.py](https://github.com/quangnhat185/Plate-detect_and_recognize/blob/master/local_utils.py)

#### 6.4.5.9 Scikit-learn

Scikit-learn ist eine freie Library für Maschinelles Lernen und basiert auf Numpy und Matplotlib. Es wird vor allem verwendet, um mit großen visuellen Datensätzen neuronale Netze zu trainieren. Es bietet Funktionen, um Modelle anzupassen, Daten vorzubereiten, Modelle zu trainieren und ähnliches. In dieser Applikation wird es für die Normalisierung von Labels verwendet, um diese für Modelle für Maschinelles Lernen anzupassen und für das zufällige Aufteilen von Arrays in Test und Trainingsteile, welche für das Training eines Neuronalen Netzes benötigt werden.

Für die Installation von Scikit-learn muss die folgende Zeile in der Konsole ausgeführt werden:

```
1 pip install scikit-learn
```

Code 17: PIP Installation von Scikit-learn

#### 6.4.5.10 Requests

Requests ist eine Library mit welcher HTTP-Anfragen in Python vereinfacht werden. Sie wird häufig verwendet, um mit APIs zu kommunizieren und ist eine der beliebtesten Python Librarys, da sie für API-Anwendungen so gut wie notwendig ist. In dieser Applikation wird sie für die Kommunikation zu einer eigenen API für den Datenbankzugriff verwendet. Dadurch ist es ohne größere Schwierigkeiten möglich, das Bild des Kennzeichens, die ID für den jeweiligen Parkplatz und das Resultat des Kennzeichens an die eigene Datenbank zu senden und auch eine Antwort zu erhalten, ob der Zugriff erfolgreich war. Um die Daten mit dieser Library zu übertragen müssen diese einfach nur in einem Dictionary eingetragen werden und dann mit dem korrekten Schlüsselwort über die API

gesendet werden.

Die Installation von Requests erfolgt mit folgendem Befehl in der Konsole:

```
1 pip install requests
```

Code 18: PIP Installation von Requests

#### 6.4.5.11 Gpiozero

Gpiozero ist eine Python-Library für den Raspberry Pi. Mit ihr ist es möglich auf die GPIO-Pins des Raspberry zuzugreifen, Signale von diesen auszulesen und Signale an diesen auszugeben. Dies wird oft verwendet, um Sensoren auszuwerten oder um Aktoren zu steuern, da diese Pins frei programmierbar sind, wodurch man jede beliebige Anwendung realisieren kann. Der Raspberry Pi kann zwar nur Aktoren in seiner Leistungskategorie ansteuern, dies kann aber zum Beispiel mit einem Relais umgangen werden, wodurch der Raspberry Pi zu einem mächtigen Werkzeug für die Sensorik und Aktorik wird. Ein Beispiel für die möglichen Anwendungen ist ein Button oder eine Lampe. In dieser Applikation wird mit dieser Library ein gedrückter Button detektiert, welcher als Auslöser für die Kamera fungiert.

In der folgenden Abbildung sieht man ein Bild der GPIO-Pins des Raspberry Pi, welche mit dieser Library angesteuert werden können:

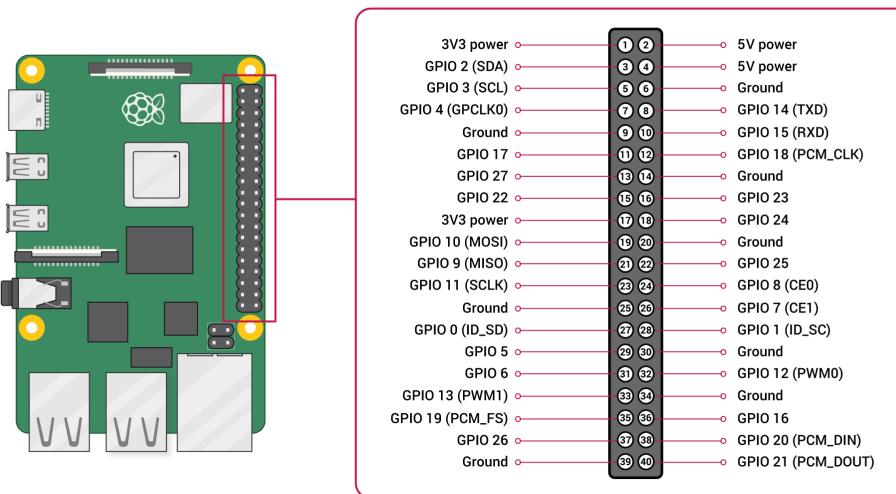


Abbildung 13: GPIO-Pins des Raspberry Pi

Diese Library kann nur auf dem Raspberry Pi installiert werden. Um die Library Gpiozero zu installieren kann pip verwendet werden:

```
1 pip install gpiozero
```

Code 19: PIP Installation von Gpiozero

#### 6.4.5.12 Picamera

Die Library picamera wird verwendet, um auf das Kameramodul des Raspberry Pi zuzugreifen. Die Library bietet etliche Funktionen an, mit welchen die Kamera gesteuert werden kann, wie zum Beispiel Fotos aufzunehmen, Videoaufnahmen zu starten und zu beenden, Fotoeinstellungen zu ändern und ähnliches. Die Library ist notwendig, wenn man über Python das Kameramodul bedienen will. In dieser Applikation wird sie verwendet, um ein Vorschaubild des aufgenommenen Bildes auf dem Bildschirm darzustellen, falls ein Bildschirm angeschlossen wird und um im Anschluss das Bild aufzunehmen und korrekt zu skalieren.

Die Installation von picamera ist nur auf dem Raspberry Pi möglich, da nur dieser damit arbeiten kann. Für die Installation von picamera muss folgender Befehl in der Konsole eingegeben werden:

```
1 pip install picamera
```

Code 20: PIP Installation von Picamera

## 6.4.6 Wichtige Programmteile

Im folgenden Abschnitt werden die wichtigsten Teile des Kennzeichenerkennungsprogrammes genauer betrachtet und erklärt.

### 6.4.6.1 WPOD-NET-Modell laden

```
1 def load_model(path):
2     try:
3         path = splitext(path)[0]
4         with open ('%s.json' % path, 'r') as json_file:
5             model_json = json_file.read()
6             model = model_from_json(model_json, custom_objects={})
7             model.load_weights('%s.h5' % path)
8             print("-> Model loading finished")
9     return model
10 except Exception as e:
11     print(e)
```

Code 21: WPOD-NET Modell laden

Für das Lokalisieren des Kennzeichens wird das Modell für Maschinelles Lernen WPOD-NET verwendet. Dieses basiert auf YOLOv2, mit welchem eine Echtzeitobjekterkennung möglich ist und wurde für diese Anwendung dazu angepasst, dass es möglich ist Kennzeichen in einem Bild zu lokalisieren. WPOD-NET versucht zuerst auf Grundlage von YOLOv2 in dem übergebenen Bild ein Fahrzeug zu erkennen, erstellt dann auf dieser Grundlage eine Region, in welcher sich mit hoher Wahrscheinlichkeit das Kennzeichen befindet und liefert dann die Koordinaten des Kennzeichens

im Bild. Im oberen Code sieht man die Funktion „load\_model“ mit welcher das WPOD-Modell mit den gewichteten Werten geladen wird. Dazu muss dieser Funktion nur der Pfad für das WPOD-NET File übergeben werden. Die Funktion versucht dann ein .json-File zu öffnen und erstellt aus diesem mit der Funktion „model\_from\_json“ ein Model mit dem WPOD-NET arbeiten kann. Danach wird im selben Ordner in dem auch das .json-File gefunden wurde nach einem .h5-File gesucht. Dieses enthält die gewichteten Werte für das Modell und wird dann geladen. Wenn alles funktioniert hat, erhält man als Rückgabe das Modell mit den geladenen Werten. Im unteren Code sieht man die Anwendung dieser Funktion. Zuerst wird dabei der Pfad für das WPOD-NET File definiert und dieser Pfad wird dann der Funktion „load\_model“ übergeben und das Modell als Rückgabewert in der Variable „wpod\_net“ abgespeichert.

```

1   wpod_net_path = "wpod-net.json"
2   wpod_net = load_model(wpod_net_path)

```

Code 22: Anwendung des WPOD-NET Modells

#### 6.4.6.2 Kennzeichen lokalisieren

```

1   def get_plate(image_path, Dmax=608, Dmin=256):
2       vehicle = preprocess_raw_image(image_path)
3       ratio = float(max(vehicle.shape[:2])) / min(vehicle.shape[:2]))
4       side = int(ratio * Dmin)
5       bound_dim = min(side, Dmax)
6       _, LpImg, _, cor = detect_lp(wpod_net, vehicle, bound_dim,
7                                     lp_threshold=0.5)
8       return LpImg, cor

```

Code 23: get\_plate

Um das Kennzeichen zu lokalisieren wird die Funktion „get\_plate“ angewendet. Dieser Funktion muss man den Pfad für das aufgenommene Bild übergeben. Danach wird auf dieses Bild die

Funktion „`preprocess_raw_image`“ angewandt, welches im folgenden Code ersichtlich ist:

```
1 def preprocess_raw_image(image_path):  
2     img = cv2.imread(image_path)  
3     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  
4     img = img / 255  
5     img = cv2.resize(img, (224,224))  
6     return img
```

Code 24: Bild vorbereiten für WPOD-NET

Die Funktion „`preprocess_raw_image`“ liest das Bild mittels OpenCV ein, ändert dann den Farbraum von BGR zu RGB da OpenCV das Bild mit BGR einliest, WPOD-NET aber RGB benötigt und anschließend wird noch die Größe des Bildes angepasst.

Danach geht es wieder in der Funktion „`get_plate`“ weiter. Dort wird danach in den nächsten Codezeilen das Seitenverhältnis des Bildes definiert und in einer Variable abgespeichert. Dies ist für die nächste Funktion „`detect_lp`“ notwendig. Diese Funktion ist eine Hilfsfunktion für den Umgang mit WPOD-NET, welcher man das aufgenommene und vorbereitete Bild übergeben muss und als Rückgabe ein Bild des Kennzeichens und die Koordinaten des Kennzeichens zurückgibt. Im unteren Codeteil befindet sich die Anwendung der Funktion „`get_plate`“ wobei bei erfolgreicher Anwendung der Funktion noch zusätzlich die Koordinaten des Kennzeichens im Terminal ausgegeben werden.

```
1 input_image = image_path[0]  
2 LpImg,cor = get_plate(input_image)  
3 print("-> License Plate found in:", splitext(basename(input_image))[0])  
4 print("-> Coordinates of License Plate: \n", cor)
```

Code 25: Kennzeichen lokalisieren

#### 6.4.6.3 Bild aufnehmen

```
1 #Kamerabildvorschau
2 camera.start_preview(alpha=220)
3
4 #Warten bis Button gedrückt wird
5 button.wait_for_press()
6
7 #Altes Bild löschen
8 try:
9     os.remove('/home/pi/Documents/Kennzeichenerkennung/Plate_examples'
10           ↪ '/image.jpg')
11 except OSError:
12     pass
13
14 #Bild schießen und abspeichern
15 camera.capture('/home/pi/Documents/Kennzeichenerkennung/Plate_examples'
16           ↪ '/image.jpg')
17 camera.stop_preview()
```

#### Code 26: Bild aufnehmen

In diesem Codeteil befindet sich die Aufnahme des Fahrzeugsbildes. Im ersten Schritt wird dabei die Kameravorschau geöffnet, damit es bei einem angeschlossenen Bildschirm einfacher ist das Foto aufzunehmen. Diese Vorschau ist leicht durchsichtig auf dem Bildschirm damit man trotzdem alles andere bedienen kann. Danach wird gewartet bis der Button gedrückt wird. Auf diese Weise dient der Button als Auslöser für die Kamera. Wenn der Button gedrückt wurde, wird zuerst versucht im gewünschten Zielordner das vorhergehende Bild zu löschen damit es zwischen den Bildern zu keinem Konflikt kommt. Dieses vorhergehende Bild wird auch nicht wieder benötigt, da das Bild des Kennzeichens im vorhergehenden Programmdurchlauf schon an die Datenbank übergeben wurde. Erst danach wird das eigentliche Foto für diesen Programmdurchlauf aufgenommen und im Zielordner abgespeichert. Danach wird die Kameravorschau wieder beendet.

#### 6.4.6.4 Zeichen mittels Konturerkennung finden

```
1   cnt, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL,  
2   ↳ cv2.CHAIN_APPROX_SIMPLE)
```

Code 27: Konturen finden

Der nächste wichtige Abschnitt des Programmes beschäftigt sich damit, aus dem Kennzeichenbild die einzelnen Zeichen herauszusuchen und einzeln darzustellen, damit diese dann von einem weiteren Modell für Maschinelles Lernen erkannt werden können. Dazu wird als erstes die Funktion „findContours“ von OpenCV angewendet, welcher man ein binäres Bild des Kennzeichens übergeben muss und als Rückgabe ein Array von Koordinaten aller gefundenen Konturen ausgibt.

```
1   def sort_contours(contours):  
2       i = 0  
3       boundingBoxes = [cv2.boundingRect(c) for c in contours]  
4       (contours, boundingBoxes) = zip(*sorted(zip(contours, boundingBoxes),  
5       ↳ key=lambda b: b[1][i], reverse = False))  
6       return contours
```

Code 28: Konturen sortieren

Im nächsten Schritt wird die Funktion „sort\_contours“ benötigt, welche die Konturen aus dem vorherigen Array sortiert. Sortieren bedeutet dabei, dass zuerst Rechtecke bei den Konturen aufgespannt werden und mit diesen die Konturen von links nach rechts geordnet werden, damit sich am Ende beim Resultat des Kennzeichens die korrekte Reihenfolge ergibt.

```
1  for c in sort_contours(cnt):
2      (x, y, w, h) = cv2.boundingRect(c)
3      if h / lp_img.shape[0] >= 0.5 and h / lp_img.shape[0] <= 0.95:
4          seperated = erosion[y:y+h,x:x+w]
5          seperated = cv2.resize(seperated, dsize=(30, 60))
6          hierarchy, seperated = cv2.threshold(seperated, 220, 255,
7              cv2.THRESH_BINARY + cv2.THRESH_OTSU)
8          found_characters.append(seperated)
9
10 print("-> Detected {} characters".format(len(found_characters)))
```

Code 29: Filterung der korrekten Konturen

#### 6.4.6.5 Bildverarbeitung

Im gesamten Programm wird ab und zu ein Bildverarbeitungsalgorithmus verwendet, aber zwischen der Kennzeichenlokalisierung und der Zeichenerfassung befindet sich der Hauptteil davon. Um die einzelnen Zeichen zu lokalisieren müssen zuvor einige Bildverarbeitungsalgorithmen und Funktionen angewandt werden und an anderen Stellen werden meistens die gleichen Funktionen angewandt wie dort. Im folgenden Code sieht man diese Bildverarbeitung:

```
1 lp_img = cv2.convertScaleAbs(LpImg[0], alpha=(255))
2 api_img = cv2.cvtColor(lp_img, cv2.COLOR_BGR2RGB)
3 gray = cv2.cvtColor(lp_img, cv2.COLOR_BGR2GRAY)
4 blur = cv2.bilateralFilter(gray, 11, 15, 15)
5 thresh = cv2.threshold(blur, 127, 255, cv2.THRESH_BINARY_INV +
→ cv2.THRESH_OTSU)[1]
6 kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
7 erosion = cv2.erode(thresh, kernel, iterations=1)
```

Code 30: Bildverarbeitungsalgorithmen

In der ersten Zeile wird auf das Kennzeichenbild die Funktion „convertScaleAbs“ angewandt welche jeden Array-Eintrag zu einem Farbwert konvertiert mit dem OpenCV arbeiten kann. Danach wird in einem eigenen Schritt der Farbraum von BGR zu RGB gewechselt, da die API für die Übertragung zur Datenbank ein RGB-Bild des Kennzeichens benötigt. Für die eigentliche Weiterverarbeitung wird der Farbraum in der nächsten Zeile von BGR zu einem Graustufenbild geändert. Anschließend wird darauf ein bilaterales Filter angewandt, welches unnötiges Bildrauschen entfernt, um die weiteren Schritte zu vereinfachen. Danach wird auf das dadurch entstandene Bild Thresholding angewandt, um es in ein binäres Bild zu konvertieren, welches für die Konturerkennung notwendig ist. Um die Qualität dieses binären Bildes noch etwas zu verbessern wird danach mit einem 3x3 Kernel Erosion verwendet. Danach ist das Bild fertig vorbereitet für die Konturerkennung.

#### 6.4.6.6 Visualisierung mittels Matplotlib

Um die einzelnen Bildverarbeitungsmechanismen zu überprüfen, die Werte anzupassen oder relevante Daten darzustellen, ist es hilfreich mit Hilfe der Library Matplotlib eine Visualisierung durchzuführen. Zusätzlich wird auch noch Jupyter Notebook verwendet, wodurch die Visualisierung nur in der Web-Applikation sichtbar ist, und damit werden nicht unzählige Tabs geöffnet. Im unteren Code sieht man eine solche Anwendung von Matplotlib.

```

1  plt.figure(figsize=(10,5))
2  plt.subplot(2,2,1)
3  plt.axis(False)
4  plt.imshow(gray, cmap='gray', vmin = 0, vmax = 255)
5  plt.subplot(2,2,2)
6  plt.axis(False)
7  plt.imshow(blur, cmap='gray', vmin = 0, vmax = 255)
8  plt.subplot(2,2,3)
9  plt.axis(False)
10 plt.imshow(thresh, cmap='gray', vmin = 0, vmax = 255)
11 plt.subplot(2,2,4)
12 plt.axis(False)
13 plt.imshow(erosion, cmap='gray', vmin = 0, vmax = 255)

```

Code 31: Anwendung einer Visualisierung mit Matplotlib

Prinzipiell funktioniert die Library Matplotlib gleich wie eine figure in MATLAB. Im oberen Codeteil werden damit die einzelnen Bildverarbeitungsalgorithmen dargestellt. Zuerst muss eine figure erstellt werden, in welcher die einzelnen Subplots platziert werden. Damit kann man die zusammengehörenden Visualisierungen gruppieren. Danach muss für jedes Bild ein eigener Subplot erzeugt werden, welchen man mit den mitgegebenen Parametern in der figure platzieren kann. Danach wird bei allen Bildern die Achsenbeschriftung deaktiviert, da diese hier nicht benötigt wird. Danach wird das gewünschte Bild ausgewählt und da alle Bilder in Graustufen sind, wird diese angepasst und der Farbwertebereich auf 0 bis 255 gesetzt. Im unteren Bild sieht man die Ausgabe dieser Visualisierung.

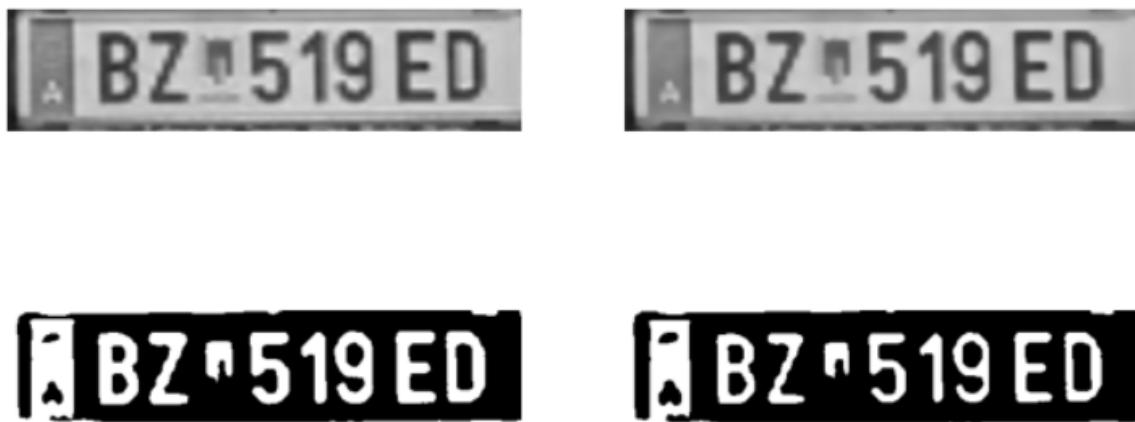


Abbildung 14: Visualisierung mit Matplotlib

#### 6.4.6.7 Modell für Zeichenerkennung laden und anwenden

Für die Zeichenerkennung wird ein eigenes Neuronales Netz verwendet, welches auf MobileNetV2 basiert. Dafür wird primär die Funktion „predict\_from\_model“ verwendet, welche im unteren Code ersichtlich ist.

```

1 def predict_from_model(image, model, labels):
2     image = cv2.resize(image,(80, 80))
3     image = npy.stack((image,),*3, axis = -1)
4     prediction = labels.inverse_transform([npy.argmax
5         → (model.predict(image[npy.newaxis,:,:]))])
6     return prediction

```

Code 32: predict\_from\_model

Vor der Anwendung dieser Funktion muss wieder, wie beim ersten Modell für Maschinelles Lernen, das Modell zuerst geladen werden. Danach kann diese Funktion verwendet werden, welcher man das Bild einer Zeichens und das Model übergeben muss. Die Funktion „predict\_from\_model“ wendet dann nur dieses Modell an und übergibt das vermutete Ergebnis in ein Array.

```

1  fig = plt.figure(figsize = (15,3))
2  cols = len(found_characters)
3  grid = gridspec.GridSpec(ncols= cols, nrows= 1, figure = fig)
4
5  result = ''
6  for i, character in enumerate(found_characters):
7      fig.add_subplot(grid[i])
8      title = npy.array2string(predict_from_model(character, model,
9          labels))
10     plt.title('{}'.format(title.strip("[]")), fontsize=20)
11     result += title.strip("[]")
12     plt.axis(False)
13
14  print("-> License Plate:", result)

```

Code 33: Anwendung der Zeichenerkennung

Im oberen Bild sieht man die Anwendung der vorherigen Funktion. Prinzipiell wird für jedes vorher gefundene Zeichen das Modell angewendet und das Resultat von einem Array zu einem String konvertiert. Dieser String wird dann zum Ergebnis-String hinzugefügt, wodurch man am Ende das komplette Kennzeichen in einem String hat. Zusätzlich wird das komplette Kennzeichen mit den vermuteten Ergebnissen mit Matplotlib visualisiert. Dies ist im unteren Bild ersichtlich.



Abbildung 15: Visualisiertes Ergebnis der Kennzeichenerkennung

#### 6.4.6.8 Resultat mittels API an Datenbank senden

Wenn das Programm zu einem Resultat gekommen ist, muss dieses noch an die Datenbank übergeben werden. Der dafür zuständige Code ist im nächsten Codeteil ersichtlich.

```
1 #API-Definitionen
2
3 url = "http://dev.philipp-kraft.com/api/v1/detections"
4
5 lp_result = {}
6
7 lp_files = {}
8
9 lp_result["name"] = result
10 lp_result["parking_lot_id"] = "2"      #Parkplatz-ID
11 lp_files=[('image', ('kennzeichen.png', open('lp_image/lp.png', 'rb'), 'image/png'))]
12
13 header = {"Authorization": "Bearer"
14     ↪ 5|4NcJgCrR9FkeTFeTvRGHLQqoJBeY8IDqxborpuuS"}
15
16
17 #Senden der Daten an die Datenbank und Ausgabe der Response
18
19 resp = requests.post(url, data=lp_result, files=lp_files, headers=header)
20 print("-> API-Response:", resp.text)
```

Code 34: Senden der Ergebnisse an die Datenbank

Im ersten Schritt werden die URL für die API und für die Ergebnisse zwei Dictionaries erstellt. In das erste Dictionary wird der String für das Kennzeichen und die ID-Nummer des Parkplatzes eingetragen. In das zweite Dictionary kommt das Bild des Kennzeichens. Im nächsten Schritt muss noch ein Bearer Token definiert werden, welcher auf der APM-Website generiert werden kann. Dieser wird zur Autorisierung benötigt. Danach können diese Daten über einen Post-Befehl an die API gesendet werden. Um den Status dieser Kommunikation zu überprüfen wird im Anschluss noch die Antwort der API im Terminal ausgegeben. Bei erfolgreicher Kommunikation antwortet diese mit dem aktuellen Status des Fahrzeuges auf dem Parkplatz.

## 6.5 Raspberry Pi

### 6.5.1 Einleitung

Damit die Software der Kennzeichenerkennung arbeiten kann, benötigt es eine geeignete Hardware. Diese muss dabei die folgenden Eigenschaften aufweisen, um für die Kennzeichenerkennung geeignet zu sein:

- Möglichst klein
- Nicht zu teuer
- Möglichkeit eine Kamera anzuschließen
- Schnell
- Internetanbindung

### 6.5.2 Wahl des Raspberry Pi

In dieser Applikation wird ein RaspberryPi 4B 2GB verwendet, da er alle zuvor genannten Eigenschaften am besten erfüllt.

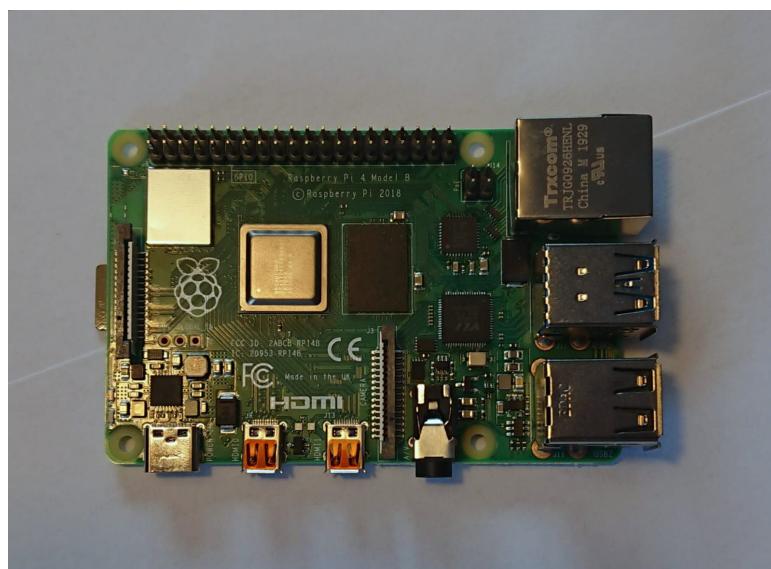


Abbildung 16: Raspberry Pi

Er hat die Größe einer Scheckkarte, wodurch es möglich ist ein kompaktes Gehäuse zu bauen, welches man einfach montieren kann. Er ist mit 35€ pro Stück nicht zu teuer. Er hat einen integrierten Kameraanschluss und es gibt unzählige Kameras, welche damit kompatibel sind. Er hat für seine Größe eine sehr gute Rechenkraft und ist damit in der Lage das komplette Programm für die Kennzeichenerkennung in einer akzeptablen Zeit abzuarbeiten. Er besitzt außerdem einen Ethernet-Anschluss und ein WLAN-Modul, wodurch er sehr einfach mit dem Internet verbunden werden kann.

### 6.5.3 Kamera

Für den Raspberry Pi muss zudem noch die passende Kamera ausgewählt werden, um die Bilder von den Kennzeichen aufzunehmen. Dafür gibt es Unmengen an kompatiblen Kameras, aber hier wird das Originalzubehör von RaspberryPi verwendet. Dies hat den Grund, dass diese Kamera leicht zu verwenden, sehr klein, mit Schrauben einfach zu befestigen und günstig ist. Zudem liefert sie ein qualitativ hochwertiges Bild, welches für die Software gut verarbeitbar ist.



Abbildung 17: Raspberry Pi mit Kamera

## 6.6 Maschinelles Lernen

### 6.6.1 Einleitung

Die Kennzeichenerkennung ist eine sehr komplexe Anwendung, da es unzählige Kennzeichen gibt und jedes einzelne davon erkannt werden sollte. Dabei ist nicht nur auf verschiedene Länderkennzeichen zu achten, welche sich stark voneinander unterscheiden, sondern auch auf Sonderkennzeichen, welche sich sogar auf nationaler Ebene von den normalen Kennzeichen unterscheiden. Zudem besteht noch die Problematik, dass sich die Kennzeichen oft in unterschiedlichen Zuständen befinden, wie zum Beispiel mit Dreck beschmutzt oder ähnliches. Das bedeutet, dass eine in der Praxis anwendbare Kennzeichenerkennung mit all diesen Gegebenheiten zureckkommen muss, damit das Kennzeichen in nahezu jeder möglichen Situation erkannt werden kann. Um dies zu erreichen wird das Programm für die Kennzeichenerkennung in drei größere Teile aufgeteilt. Die Kennzeichenlokalisierung, die Zeichenerfassung und die Zeichenerkennung. Die Zeichenerfassung gestaltet sich relativ einfach, da diese bereits ein Bild des Kennzeichens erhält, in dem ansonsten keine größeren Störfaktoren beinhaltet sind. Diese muss also nur die Konturen innerhalb dieses Bildes finden und diese so sortieren, dass nur noch die einzelnen Zeichen übrig bleiben. Bei der Kennzeichenlokalisierung und der Zeichenerfassung gestaltet sich die Lage schon etwas schwieriger. Die Kennzeichenlokalisierung steht vor dem Problem, dass wenn man klassisch mit der Bildverarbeitung nach einer passenden Kontur für das Kennzeichen sucht, kann dieser Algorithmus durch Störfaktoren wie ein zufällig ähnlich großes Orts- oder Straßenschild getäuscht werden, wodurch es zu einem Fehler kommt. Auch Fenster oder ähnlich proportionierte Gegenstände können zu solch einem Fehler führen. Die Lösung dafür befindet sich im Bereich des Maschinellen Lernens. Anstatt nur nach einer Kontur zu suchen, wird mit solch einem Modell gezielt nach einem Kennzeichen gesucht. Dies ist möglich, da so ein Modell mit vielen möglichen Eingaben und den passenden Ausgaben dazu trainiert wird ein Muster zwischen den Ein- und Ausgaben zu finden, mit welchem es dann ähnliche Probleme selbstständig lösen kann. Auch bei der Zeichenerkennung bietet sich solch ein Modell an, da es darauf trainiert werden kann, einzelne Zeichen mit einer hohen Genauigkeit zu erkennen. Hier wäre es zwar ohne Maschinelles Lernen auch möglich ein relativ gutes Ergebnis zu erzielen, indem man mittels Korrelation von Vergleichsbildern das Zeichen findet, welches am wahrscheinlichsten passen würde, aber in dieser Applikation wird auch hierfür ein Modell für Maschinelles Lernen verwendet.

### 6.6.2 Definition

Maschinelles Lernen beschreibt ein Modell, welches zuerst in einer Trainingsphase trainiert werden muss, um danach ähnliche Probleme näherungsweise lösen zu können. Dazu wird in der Trainingsphase ein Datensatz bereitgestellt welcher mögliche Eingaben und meistens auch die passenden Ausgaben enthält. Durch die vielen verschiedenen Eingaben und Ausgaben ist es für das Modell möglich ein Muster zwischen diesen zu erkennen, welches durch dieses Training verbessert werden kann. Es gilt dabei, je länger die Trainingsphase dauert desto genauer wird das Modell. Der Aufwand besteht dabei darin, die gigantischen Datensätze zur Verfügung zu stellen, da diese oft zuvor erstellt werden müssen. Zudem ist das Training ein sehr rechenintensiver Vorgang, welcher für ein gutes Ergebnis oft mehrere Stunden benötigt. In den letzten Jahren hat sich im Bereich des Modell Trainings eine neue Entwicklung namens „Deep Learning“ verbreitet, mit welcher eine höhere Genauigkeit als mit bisherigen Modellen erreicht werden kann. Dabei werden künstliche Neuronale Netze erstellt, welche der Vernetzung von Neuronen in Lebewesen nachempfunden sind. Diese künstlichen Neuronen sind Verknüpfungspunkte zwischen Eingang und Ausgang des Modells und sind je nach Komplexität in verschiedene Schichten aufgeteilt. Um diese künstlichen Neuronen zu trainieren, werden deren Gewichtungswerte angepasst. Deswegen muss bei solchen Modellen auch oft eine Datei mit den gewichteten Werten eingebunden werden.

### 6.6.3 Vergleich Maschinelles Lernen / Bildverarbeitung

Zum Beginn dieser Applikation wurde ein Ansatz ohne den Einsatz von Maschinellem Lernen verfolgt. Dazu wurde ein Testprogramm geschrieben, welches das Kennzeichen über eine Konturerkennung lokalisieren sollte. Dies führte aber zu häufig auftretenden Fehlern. Einer dieser Fehler ist in der unteren Abbildung ersichtlich.



Abbildung 18: Fehler bei Konturerkennung

Wie in der oberen Abbildung erkennbar ist, hat das Programm nicht das Kennzeichen erkannt, sondern ein Fenster im Hintergrund, welches eine ähnliche Kontur wie das Fenster aufweist. Dies ist deswegen aufgetreten, da das Kennzeichen über seine Kontur gefunden werden sollte und dazu alle Konturen im Bild aufgelistet werden und aus dieser Liste dann das Kennzeichen herausgesucht werden sollte. Dies führte in dieser Testversion aber nahezu nie zu einem Ergebnis, welches in der Praxis anwendbar gewesen wäre. Durch eine Anpassung der Software wäre es zwar möglich gewesen die Genauigkeit zu verbessern und dafür zu sorgen, dass mit einer größeren Wahrscheinlichkeit die richtige Kontur ausgewählt wird, aber bei einer Kontur mit den gleichen Proportionen wie ein Kennzeichen, wäre die Konturerkennung wieder schnell an ihre Grenzen gestoßen. Deswegen wurde der Ansatz komplett verändert und für die Kennzeichenlokalisierung ein Modell für Maschinelles Lernen verwendet, welches auf deutlich elegantere und effizientere Weise ein besseres Resultat erzielen kann.



Abbildung 19: Resultat mit maschinellem Lernen

Zum Vergleich mit dem Testprogramm sieht man im oberen Bild das Ergebnis der Variante mit Maschinellem Lernen mit demselben Eingangsbild. Dabei wurde ohne Probleme das Kennzeichen perfekt erkannt und im weiteren Programmablauf aus dem Eingangsbild ausgeschnitten.

#### 6.6.4 Verwendete Modelle für Maschinelles Lernen

##### 6.6.4.1 WPOD-NET

Das „Warped Planar Object Detection Network“ (WPOD-NET) ist ein künstliches Neuronales Netz, welches von Sérgio Montazolli Silva und Cláudio Rosita Jung<sup>11</sup> entwickelt wurde. Es wurde entwickelt, um aus Bildern die Kennzeichen von Fahrzeugen herauszusuchen und bietet im Gegensatz zu anderen Modellen den entscheidenden Vorteil, dass es bei diesem Modell irrelevant ist aus welchem Winkel das Bild vom Kennzeichen aufgenommen wird. Andere Modelle funktionieren oft nur wenn das Foto vom Kennzeichen direkt von vorne aufgenommen wurde, da ansonsten das Bild verzerrt wird und dadurch das Kennzeichen nicht mehr ausgelesen werden kann. Dieses Modell schafft es auch aus einem von der Seite aufgenommen Bild das Kennzeichen in den meisten Fällen gut genug herauszulesen, dass dieses Bild danach weiterverarbeitet werden kann. Dies bietet den Vorteil, dass die Kamera für das Gerät nicht fix verbaut werden muss, sondern theoretisch auch in einem mobilen Gerät installiert werden kann.

Die Kennzeichenerkennung dieses Modells basiert auf zwei Schritten. Zuerst wird im Bild nach einem Fahrzeug gesucht und dieses ausgeschnitten, um danach mit WPOD-NET das Kennzeichen aus diesem Bild herauszusuchen. Die Fahrzeugarkennung basiert dabei auf dem YOLOv2<sup>12</sup> Netzwerk, welches für Echtzeitobjekterkennung verwendet wird und neben vielen anderen Gegenständen auch in der Lage ist Fahrzeuge zu erkennen. Nach dieser Fahrzeugarkennung wird das Bild in das WPOD-NET eingegeben.

<sup>11</sup>Sérgio Montazzolli und Claudio Jung. »License Plate Detection and Recognition in Unconstrained Scenarios«. In: Sep. 2018. URL: [https://www.researchgate.net/figure/Detailed-WPOD-NET-architecture\\_fig3\\_327861610](https://www.researchgate.net/figure/Detailed-WPOD-NET-architecture_fig3_327861610)

<sup>12</sup>Joseph Redmon und Ali Farhadi. »YOLO9000: Better, Faster, Stronger«. In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: <http://arxiv.org/abs/1612.08242>

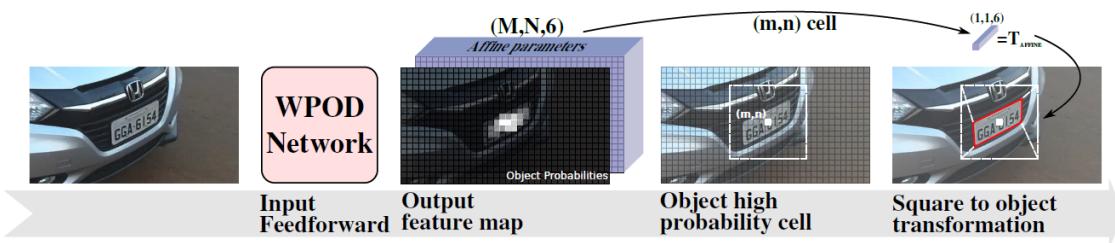


Abbildung 20: Ablauf von WPOD-NET

Im oberen Bild ist der prinzipielle Ablauf der Kennzeichenerkennung ersichtlich. Zu Beginn wird das Bild mit dem Kennzeichen in das WPOD-NET eingegeben und dieses gibt dann eine Matrix mit gewichteten Werten zurück, welche für jede einzelne Zelle eine Wahrscheinlichkeit angibt, dass sich dort ein Kennzeichen befindet. Um die Zelle, die am wahrscheinlichsten ist, wird dann ein Rechteck gespannt in welchem sich mit sehr hoher Wahrscheinlichkeit das Kennzeichen befindet. Danach wird auf die Wahrscheinlichkeitswerte für das gesamte Rechteck Thresholding angewandt, um zu bestimmen in welchen Zellen sich das Kennzeichen befindet. Dadurch hat man dann diejenige Region gefunden, in der sich das Kennzeichen befindet und kann dieses, wenn nötig in ein horizontal und vertikal ausgerichtet Rechteck umwandeln. Dadurch erhält man dann ein Bild des Kennzeichens.

#### 6.6.4.2 MobileNetV2

MobileNetV2<sup>13</sup> ist die Grundlage für das eigene trainierte Neuronale Netz. MobileNetV2 ist eine Grundlage für diverse Neuronale Netze und speziell für Anwendungen im Bereich der Computer Vision geeignet. Einige Beispiel dafür sind die Erkennung von Gesichtsmerkmalen, Echtzeitobjekterkennung, Unterscheidung diverser Hunderassen oder ähnliches. Dabei ist das Modell je nach Bedürfnissen einstellbar, womit für eine schnellere und bessere Leistung weniger Schichten für das Neuronale Netz verwendet werden, oder aber für eine sehr genaue und präzise Funktion mehr Schichten verwendet werden. Dies ist dadurch möglich, dass das Netzwerk auf einer variablen Schichtanzahl aufgebaut ist. In dieser Applikation wird es deswegen verwendet, da damit in diesem Bereich eine hohe Präzision erzielbar ist, welche für die Kennzeichenerkennung benötigt wird.

<sup>13</sup>Mark Sandler u. a. »Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation«. In: *CoRR* abs/1801.04381 (2018). arXiv: 1801.04381. URL: <http://arxiv.org/abs/1801.04381>

## 6.7 Modell Training

Um das Modell für die Zeichenerkennung zu trainieren wurde ein eigenes Programm geschrieben, welches das Modell mittels Tensorflow trainiert. Dazu wird ein Datensatz von 35 000 Bildern von Zeichen verwendet, bei welchen die richtige Ausgabe aus dem Dateiname auslesbar ist. Im folgenden Bild sieht man ein Beispiel für so ein Bild aus dem Datensatz.

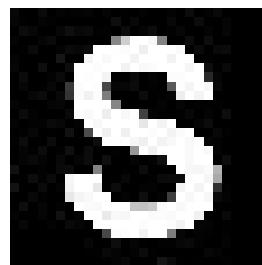


Abbildung 21: Beispiel eines Zeichens aus dem Datensatz

Das Modell wird durch die verschiedenen Schriftarten und die variierende Qualität der Bilder nicht direkt auf die Erkennung von Zeichen aus einem Kennzeichen trainiert, sondern auf die Erkennung von Zeichen in jeglicher Situation. Dies benötigt zwar länger, um das Modell zu trainieren, aber es ist dadurch genauer und kann auch in anderen Anwendungen ohne größere Umstellungen verwendet werden.

In den nächsten Codeteilen sieht man ein paar wichtige Ausschnitte aus dem Code für das Modell Training.

```
1 def create_model(lr=1e-4, decay=1e-4/30,
2     ↳ training=False, output_shape=y.shape[1]):
3     baseModel = MobileNetV2(weights="imagenet", include_top=False,
4         ↳ input_tensor=Input(shape=(80, 80, 3)))
5
6     headModel = baseModel.output
7     headModel = AveragePooling2D(pool_size=(3, 3))(headModel)
8     headModel = Flatten(name="flatten")(headModel)
9     headModel = Dense(128, activation="relu")(headModel)
10    headModel = Dropout(0.5)(headModel)
```

```

9   headModel = Dense(output_shape, activation="softmax")(headModel)
10
11 model = Model(inputs=baseModel.input, outputs=headModel)
12
13 if training:
14     for layer in baseModel.layers:
15         layer.trainable = True
16     optimizer = Adam(lr=lr, decay = decay)
17     model.compile(loss="categorical_crossentropy",
18                   optimizer=optimizer, metrics=["accuracy"])
19
20 return model

```

Code 35: Erstellen des Modells basierend auf MobileNetV2

Im ersten Teil sieht man das grundlegende Erstellen des Modells auf der Grundlage von MobileNetV2. Zusätzlich werden noch die Einstellungen für Tensorflow getroffen wie zum Beispiel die Größe der Eingabe und weitere Parameter, welche von Tensorflow für die Erstellung des Modells benötigt wird. Im unteren Teil wird noch definiert welcher Optimizer verwendet werden soll und dass während des Trainings die Genauigkeit des Modells aufgenommen werden soll.

```

1 INIT_LR = 1e-4
2 EPOCHS = 30
3
4 model = create_model(lr=INIT_LR, decay=INIT_LR/EPOCHS, training=True)
5
6 BATCH_SIZE = 64
7
8 my_checkpointer = [EarlyStopping(monitor='val_loss', patience=5,
9                             verbose=0),
10                        ModelCheckpoint(filepath="License_character_recognition.h5",
11                                      verbose=1, save_weights_only=True)]

```

```
9   result = model.fit(image_gen.flow(trainX, trainY, batch_size=BATCH_SIZE),  
10      steps_per_epoch=len(trainX) // BATCH_SIZE,  
11      validation_data=(testX, testY),  
12      validation_steps=len(testX) // BATCH_SIZE,  
13      epochs=EP0CHS, callbacks=my_checkpointer)
```

Code 36: Trainieren des Modells und Einstellungen anpassen

Im zweiten Teil wird zuerst die Standard Lernrate des Modells und die Anzahl der Epochen definiert. Die Epochen sind eine Art Meilenstein, nach welchem die Genauigkeit dargestellt und beurteilt wird, um den Trainingserfolg zu beurteilen. Außerdem werden nach jeder Epoche die gewichteten Werte angepasst und abgespeichert. Danach wird noch eingestellt nach welcher Anzahl an Epochen ohne signifikante Verbesserung das Training frühzeitig beendet werden soll, und wohin die gewichteten Werte abgespeichert werden sollen. Danach werden Tensorflow die Daten übergeben, um das Training zu beginnen. Die Dauer dieses Vorgangs unterscheidet sich von Computer zu Computer, und hat bei dieser Applikation in etwa 10 Stunden gedauert.

Im nächsten Bild sieht man den Trainingsverlauf des Modells über 30 Epochen.

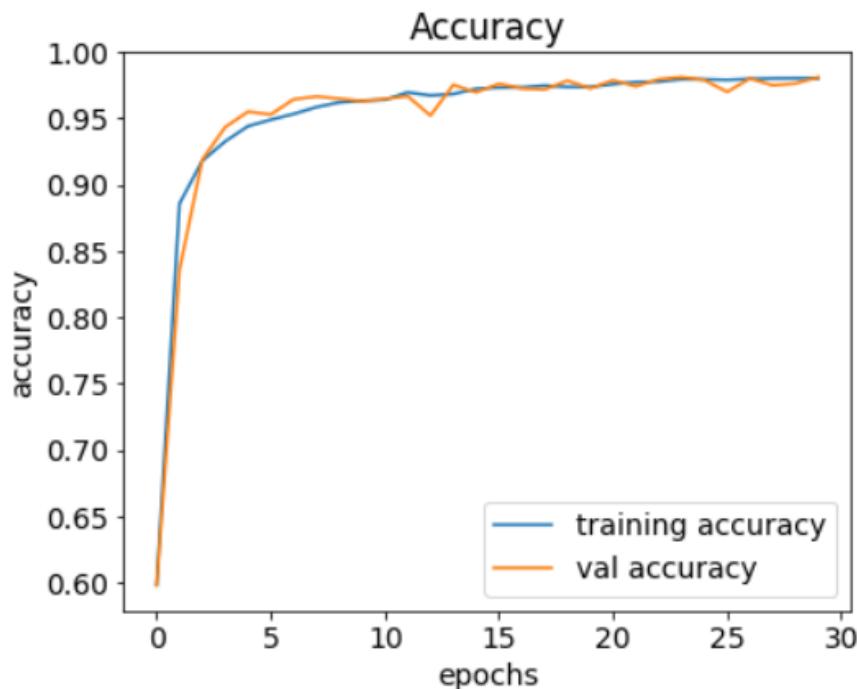


Abbildung 22: Verlauf des Modell Trainings

Im oberen Bild kann man erkennen, dass die Genauigkeit des Modells bereits nach wenigen Epochen einen Wert von über 90% erreicht hat und nach den 30 Epochen auf einen Endwert von 98,02% erreicht hat. Das bedeutet, dass bei einem Kennzeichen mit durchschnittlich 7 Zeichen eine insgesamte Genauigkeit von 86,93% erreicht wird. Diese Genauigkeit könnte mit einem längeren Training theoretisch auf einen Wert von nahezu 100% gesteigert werden, aber dazu benötigt es einen stärkeren Computer, welcher zudem noch eine ganze Weile nicht benutzt werden kann.

## 6.8 Gehäuse

### 6.8.1 Anforderungen

Die Anforderungen an das Gehäuse sind recht simpel. Es soll das Modul vor äußeren Einwirkungen schützen, eine Möglichkeit bieten, um die Kamera des Raspberry Pi zu befestigen und für weitere Test- und Entwicklungszwecke soll es die wichtigsten Anschlüsse des Raspberry Pi zugänglich machen, damit diese verwendet werden können. Zudem sollen die Entwicklungs- und Produktionskosten eher gering sein, weswegen sich ein Gehäuse aus dem 3D-Drucker am besten eignet.

### 6.8.2 Umsetzung

Das 3D-Modelling des Gehäuses erfolgte mit dem Programm „Autodesk Fusion 360“/(Fußnote), mit welchem 3D-Modelle erstellt und konvertiert werden können, damit man sie mit dem 3D-Drucker ausdrucken kann. In den folgenden Bildern sieht man die Renderansichten der einzelnen Teile.

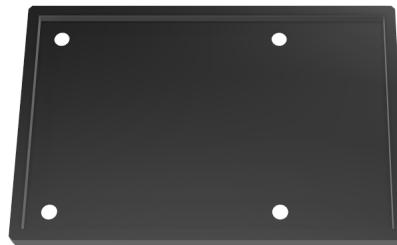


Abbildung 23: Boden des Gehäuses

Im ersten Bild sieht man die Renderansicht des Bodens. Auf diesen wird der Raspberry Pi gelegt und mit den 4 Löchern an den mittleren Teil geschraubt. Dadurch ist der Raspberry Pi sicher befestigt und kann im Gehäuse nicht verrutschen.

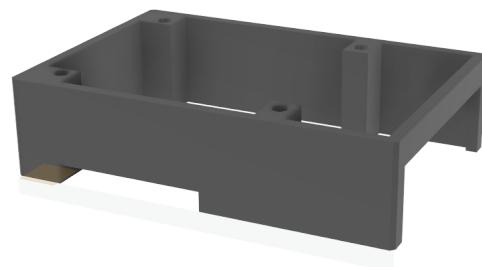


Abbildung 24: Mittelteil des Gehäuses

Der mittlere Teil besitzt 4 Verstärkungen auf der inneren Seite, in welche man Gewinde einfügen kann. Dadurch können der Deckel und der Boden angeschraubt werden. Zudem befinden sich auf

zwei Seiten Ausschnitte durch welche die Stromversorgung, die HDMI-Ports, die USB-Ports und der Ethernet-Anschluss des Raspberry Pi zugänglich sind. Dadurch kann der Raspberry Pi nahezu ohne Einschränkungen mit den wichtigsten Funktionen betrieben werden.

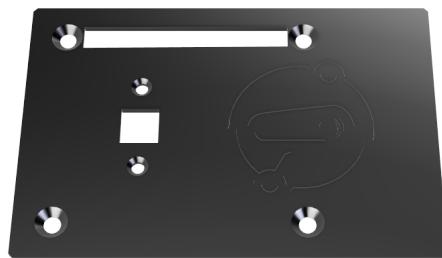


Abbildung 25: Deckel des Gehäuses

Das letzte Einzelteil ist der Deckel, welcher wieder die gleichen 4 Löcher wie der Boden aufweist, um mit dem Mittelteil verschraubt zu werden. Zusätzlich befinden sich im Deckel noch zwei Löcher, um die Kamera festzuschrauben und ein rechteckiges Loch, durch welches die Kamera gesteckt werden kann. Außerdem befindet sich auf dem Deckel noch das eingravierte Logo von APM. In der nächsten Abbildung sieht man dann noch eine Gesamtansicht des Gehäuses.

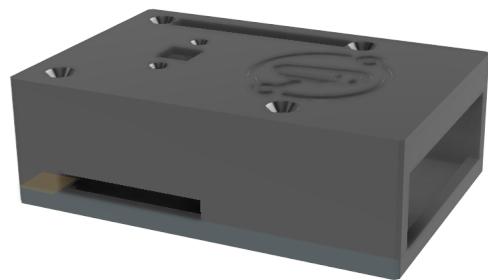


Abbildung 26: Gesamtesansicht des Gehäuses

Diese Einzelteile wurden dann mit einem 3D-Drucker gefertigt. In den nächsten Bildern sieht man

die einzelnen gefertigten Teile und eine Gesamtansicht.



Abbildung 27: Gedruckte Einzelteile des Gehäuses



Abbildung 28: Gesamtesansicht des gedruckten Gehäuses

Das fertige Gehäuse erfüllt alle zu Beginn definierten Anforderungen und ist somit bestens für diese Anwendung geeignet.

## 6.9 Test

### 6.9.1 Einleitung

Um die Kennzeichenerkennung zu testen und herauszufinden wie genau sie in der Praxis ist, wurden mehrere Tests durchgeführt, welche aufschlussreiche Ergebnisse geliefert haben. Für den Test wird als Stromversorgung eine Powerbank verwendet und ein Laptop wird mit dem Raspberry Pi über VNC<sup>14</sup> verbunden um die Ergebnisse auszuwerten.

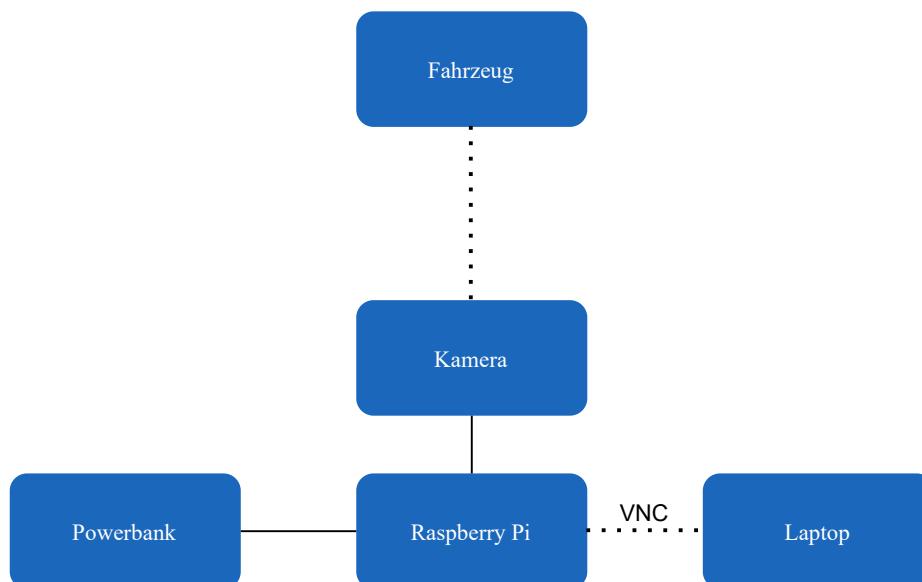


Abbildung 29: Testaufbau

### 6.9.2 Erster Test

Im ersten Test wurde die Kennzeichenerkennung zum ersten Mal in der Praxis ausprobiert. Dies geschah in einer sehr schwierigen Umgebung mit schlechter Beleuchtung und vielen Lichtreflexionen durch Schnee. Im nachfolgenden Bild sieht man ein Bild dieses Tests.

<sup>14</sup>Virtual Network Computing



Abbildung 30: Erster Test

Dieses Bild war eines der wenigen Bilder welches nicht durch Lichtreflexionen überbelichtet wurde, daran konnte man bemerken, dass die Kamera des Raspberry Pi eindeutig nicht für solche Verhältnisse gemacht wurde, aber trotzdem keine allzu schlechten Ergebnisse liefert. Die Kennzeichenerkennung lieferte für dieses Bild das Ergebnis „82538BB“. Dies war für kein schlechter Erster Test, aber er zeigte, dass die Kennzeichenerkennung noch Schwierigkeiten mit der Unterscheidung von ähnlichen Zeichen wie zum Beispiel „8“ und „B“ oder auch „2“ und „Z“ hatte. Um dieses Problem zu lösen, wurde das Modell Training noch einmal überarbeitet, um die Genauigkeit der Zeichenerkennung zu verbessern. Dies führte zu einer signifikanten Verbesserung, welche im zweiten Test deutlich zu erkennen ist.

### 6.9.3 Zweiter Test

Der zweite Test fand unter besseren Licht- und Umgebungsbedingungen statt und zusätzlich wurde auch die Genauigkeit der Zeichenerkennung verbessert. Während des zweiten Tests wurde mit über 10 verschiedenen Fahrzeugen die Kennzeichenerkennung ausgetestet. Im Folgenden werden einige der unterschiedlichen Testbilder und deren Resultate gezeigt und analysiert.



Abbildung 31: Zweiter Test | Bild 1

**Resultat:** FK773FG

In diesem Bild wurde das Kennzeichen nahezu perfekt erkannt. Der einzige Unterschied liegt in der dritten „7“, welche vom Programm als „3“ erkannt wurde.



Abbildung 32: Zweiter Test | Bild 2

**Resultat:** Nicht erkannt

Dieses Bild führte zu einem Fehler, weil in dem Bild kein Kennzeichen erkannt wurde. Dies kann zwei mögliche Ursachen haben. Entweder wurde von YOLOv2 im Bild kein Fahrzeug erkannt, oder es wurde von WPOD-NET kein Kennzeichen erkannt. Dies sind mögliche Fälle, da beide Modelle keine Genauigkeit von 100% aufweisen.



Abbildung 33: Zweiter Test | Bild 3

**Resultat:** FK 596 GP

In diesem Bild wurde das Kennzeichen ohne jeglichen Fehler ausgelesen und die Kennzeichenerkennung hat damit einwandfrei funktioniert.



Abbildung 34: Zweiter Test | Bild 4

**Resultat:** FK0K

Dieses Bild lieferte ein sehr interessantes Resultat. Wie im Bild eindeutig erkennbar ist, hat die Kamera das komplette Bild massiv überbelichtet , wodurch einiges sehr schlecht erkennbar ist. Trotz dieser erschwerten Bedingungen hat es die Kennzeichenerkennung aber geschafft einige der Zeichen korrekt auszulesen. Die anderen Zeichen wurden gar nicht erst erkannt, was auf die Zeichensegmentierung zurückzuführen ist.



Abbildung 35: Zweiter Test | Bild 5

**Resultat:** B668IR

Auch in diesem Bild wurde das Kennzeichen ohne jeglichen Fehler ausgelesen.

#### 6.9.4 Fazit

In den Tests ist klar erkennbar, dass die Kennzeichenerkennung selbst gute Resultate liefert. Vor allem die Steigerung der Genauigkeit vom ersten zum zweiten Test ist dabei hervorzuheben. Die berechnete Genauigkeit von etwa 87% für eine fehlerfreie Kennzeichenerkennung konnte mit dem zweiten Test durchaus bestätigt werden. Die größte Schwachstelle der Kennzeichenerkennung ist jedoch die Kamera, welche sehr sensibel auf die unterschiedlichen Lichtverhältnisse reagiert. Ansonsten hat die Kennzeichenerkennung nahezu alle Kennzeichen fehlerfrei ausgelesen und nur ein einziges Kennzeichen wurde nicht erkannt.

## 7 Fahrzeugerkennung

### 7.1 Anforderungen

Das Ziel der Fahrzeugerkennung ist es Fahrzeuge auf mehrere Parklücken eines Parkplatzes zu erkennen. Die daraus gewonnenen Zustände sollen an das Webinterface übermittelt und an den jeweilige Parklücken über LEDs ausgegeben werden.

### 7.2 Vorstudie

#### 7.3 Erkennung von Metallen über Spulen

Grundsätzlich werden in der Realität häufig Spulen verwendet, welche unter dem Asphalt verbaut sind, um darüberliegende Fahrzeuge zu detektieren. Als Messprinzip wird die Änderung des magnetischen Widerstands  $R_m$  bei konstanter magnetischer Spannung  $U_m$  und daraus resultierende magnetischen Fluss  $\phi$ . Es gelten für diese Größen der folgende Zusammenhänge:

$$\Phi = \frac{U_m}{R_m} \quad (1)$$

Wobei

$\Phi$  = magnetischer Fluss

$R_m$  = magnetischer Widerstand

$U_m$  = magnetische Spannung

Der magnetische Widerstand lässt sich wiederum durch die Eigenschaften der Spule bestimmen.

Die Formel hierfür lautet:

$$R_m = \frac{N \cdot (2a + 2b)}{\mu_0 \cdot \mu_r \cdot A} \quad (2)$$

Wobei

$$A = a \cdot b \quad (3)$$

$N$  = Anzahl der Windungen der Spule

$a$  = Breite der Spule in m

$b$  = Länge der Spule in m

$\mu_0 = 1,2566 \cdot 10^{-6} \text{ N/A}^2$  = magnetische Feldkonstante

$\mu_r$  = relative Permeabilität

$A$  = Fläche der Spule

Bis auf die relative Permeabilität sind alle anderen Variablen konstant. Daraus lässt sich schlussfolgern, dass der magnetische Fluss  $\Phi$  anhand der Gleichung 1 und 2 proportional zur relativen Permeabilität ist.

$$\Phi \propto \mu_r \quad (4)$$

Der Fluss  $\Phi$  hängt somit auch von den Materialien ab durch die er fließt. In der nächsten Abbildung kann man erkennen, wie ein Fahrzeug über eine im Boden installierte Spule den magnetischen Fluss  $\Phi$  und somit die magnetische Flussdichte  $B$  beeinflussen kann.

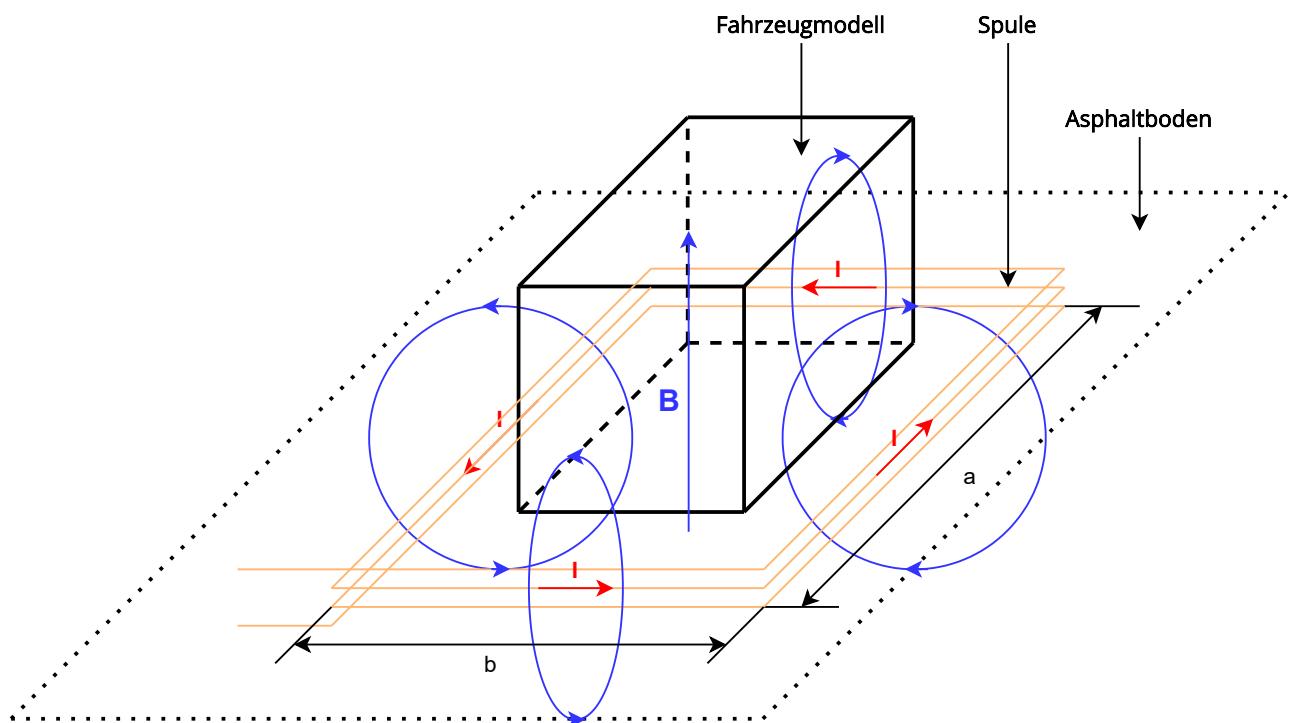


Abbildung 36: Installation der Spule

Um diese Größen auslesbar zu machen, müssen diese magnetischen bei einer direkten Messung in elektrische Größen umgewandelt werden. Hierfür gelten folgende Zusammenhänge:

$$L = \frac{N \cdot \Phi}{I} = \frac{\Psi}{I} \quad (5)$$

Wobei

$L$  = Induktivität der Spule

$I$  = Strom der durch die Spule fließt

$N$  = Anzahl der Windungen der Spule

$\Psi$  = Verkettete Fluss

$$u(t) = L \cdot \frac{di(t)}{dt} \quad (6)$$

Wobei

$L$  = Induktivität der Spule

$i(t)$  = Strom der durch die Spule fließt zum Zeitpunkt t

$u(t)$  = Spannung die an der Spule anliegt zum Zeitpunkt t

$t$  = Zeit in s

So lässt sich bei Bekanntheit von Strom und Spannung auf die Induktivität und mit der Gleichung 5 und mit den Zusammenhang 4 auf die magnetischen Eigenschaften des Materials rückschließen. Diese Art der Detektion bietet viele praktische Vorteile.

- **Größerer Messbereich**

Im Vergleich zu anderen Detektionsmethoden wie einer Leichtschranke kann man einen größeren Bereich durch die Wirkfläche der Spule abdecken. So lassen sich auch kleinere Kraftfahrzeuge wie Motorräder oder Mopeds besser erkennen.

- **Schutz vor Umweltfaktoren**

Durch den Verbau im Boden ist die Messeinrichtung vor Umwelteinflüssen wie Regen, Frost, hohen beziehungsweise niedrigen Temperaturen und Korrosion besser geschützt. Dies verringert auch den Einfluss dieser Störfaktoren auf die Eigenschaften Spule und somit auf die daraus resultierenden Messergebnisse.

- **Ausschließung von Materialien**

Alle nicht metallische Stoffe werden von diesem Messprinzip nicht wahrgenommen. So können Verschmutzungen wie Blätter und Staub, welche visuelle Sensoren stören können, die Detektion nicht behindern.

### 7.3.1 Messung ferromagnetischer Metalle

Um diese Detektionsverfahren zu verstehen muss zuerst der Zusammenhang zwischen Induktivität und relativer Permeabilität verstanden werden. Aus den Gleichungen 2, 1 und 5 kann die folgende Proportionalität ermittelt werden:

$$L \propto \mu_r \quad (7)$$

Bei ferromagnetischen Stoffen wie Eisen ist die relative Permeabilität sehr viel größer als 1. Wenn nun ein Auto oder eine anderes Vehikel sehr viel Eisen beinhaltet wirkt sich dies steigernd auf die Induktivität der Spule aus. Die folgenden Verfahren nutzen dieses Prinzip um die Belegung einer Parklücke zu bestimmen.

### 7.3.1.1 RL-Oszillator mit Timer Baustein

Bei diesem Messverfahren wird die Änderung der Induktivität  $L$  über die Änderung einer Stromladekurve über einen Widerstand  $R$  ermittelt. Eine Simulation in LTSPice mit folgendem Schaltbild kann die Auswirkung auf eine Ladekurve bei gleicher Spannung und gleichem Widerstand gut darstellen.

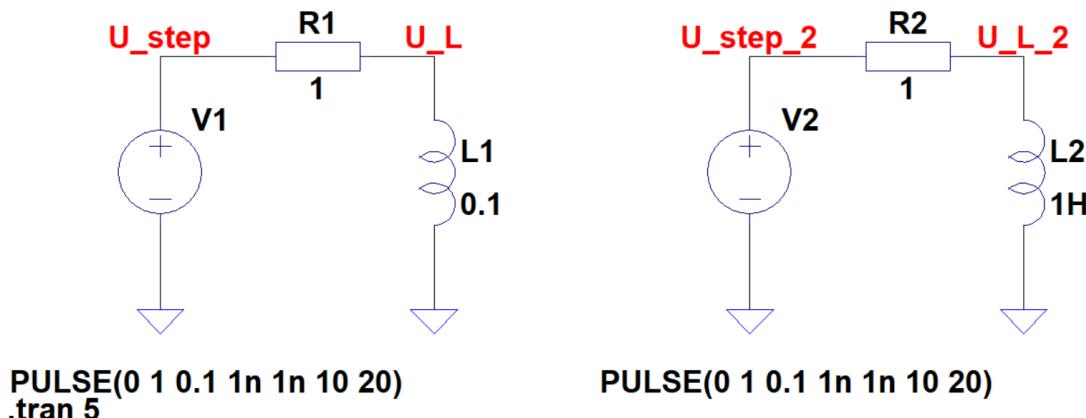


Abbildung 37: LTSPice Blockbild zweier RL-Glieder

Die daraus resultierende Simulation im Zeitbereich ergibt folgendes Ergebnis: hmm

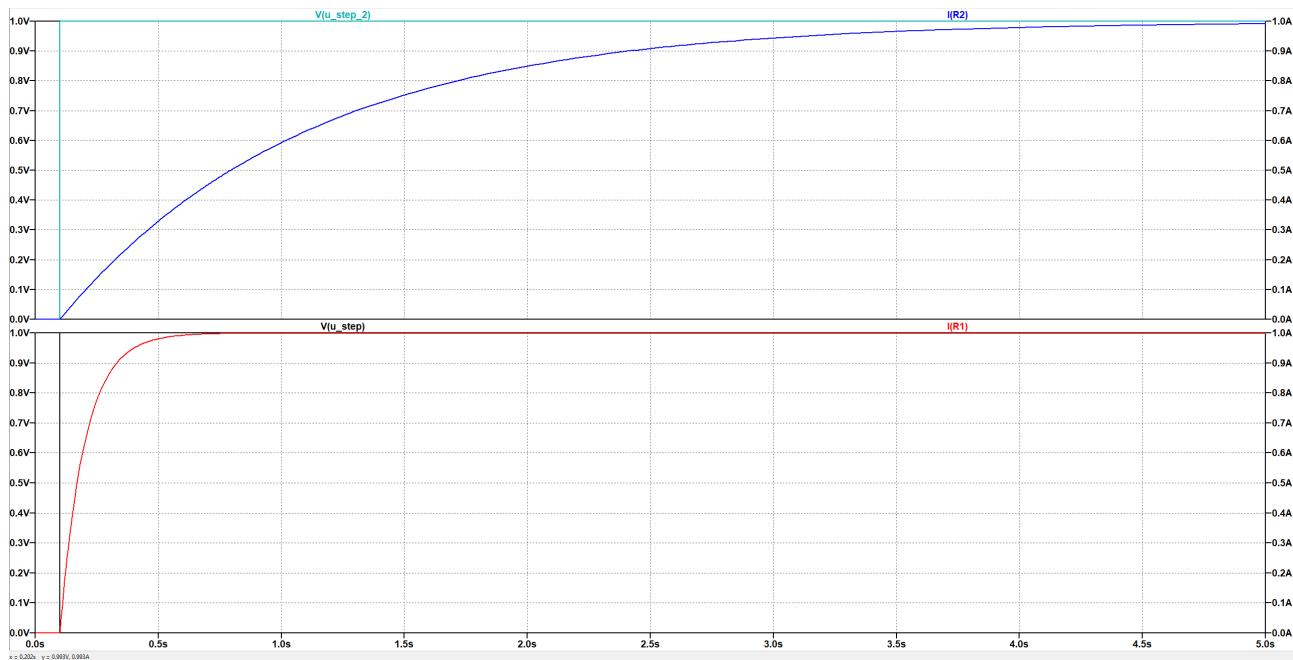


Abbildung 38: Stromkurven zweier RL-Glieder

Es ist aus der letzten Abbildung erkennbar, dass sich eine größere Induktivität verlangsamt auf die Zunahme des Stromes auswirkt. Für Einschaltvorgänge kann diese Stromkurve für RL-Glieder folgendermaßen beschrieben werden.

$$i(t) = \frac{U_0}{R} \cdot (1 - e^{-\frac{t}{\tau}}) \quad (8)$$

$$\tau = \frac{L}{R} \quad (9)$$

Wobei

$i(t)$  = Strom der durch die Spule fließt zum Zeitpunkt  $t$

$L$  = Induktivität der Spule in H

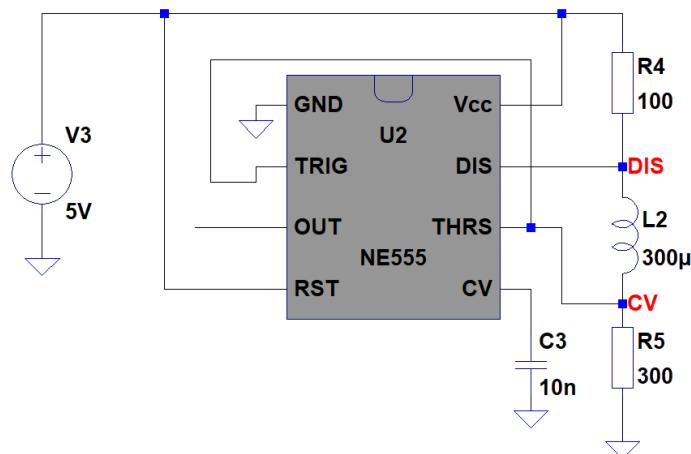
$R$  = Widerstand in  $\Omega$

$U_0$  = Spannung die nach dem einschalten anliegt in V

$t$  = Zeit in s

$\tau$  = Zeitkonstante in  $\frac{1}{s}$

Timer Bausteine wie der NE555 nutzen diese solche Verlaufskurven wenn sie als astabile Kippstufe konfiguriert sind. Es gibt einen Eingang dieses Baustein meist CV für Control Voltage der die Spannung überwacht. Überschreitet diese Spannung zweit drittel der Betriebsspannung schaltet er einen weiteren Pin, meist DIS für Discharge, auf 0V beziehungsweise auf Masse. Unterschreitet jedoch die Spannung am CV Pin ein drittel der Betriebsspannung so wird der DIS Pin auf Betriebsspannung geschalten. Diese beiden Pins sind über RC- oder RL- Glieder verbunden. Es entsteht somit ein Oszillator, welcher die Spannung am DIS Pin anhebt und absenkt. Der NE555 kann dieses Signal über einen internen Komparator in ein Rechtecksignal umwandeln, welches besser von Mikrocontrollern ausgelesen werden kann. Der Mikrocontroller kann so die Anzahl der einkommenden Takte über einen gewissen Zeitraum zählen und daraus eine Oszillatofrequenz errechnen. Die Zeitsignale lassen sich in LTSPice mit dem NE555 als Timer-Baustein simulieren.



.tran .01m

Abbildung 39: RL-Oszillator mit NE555

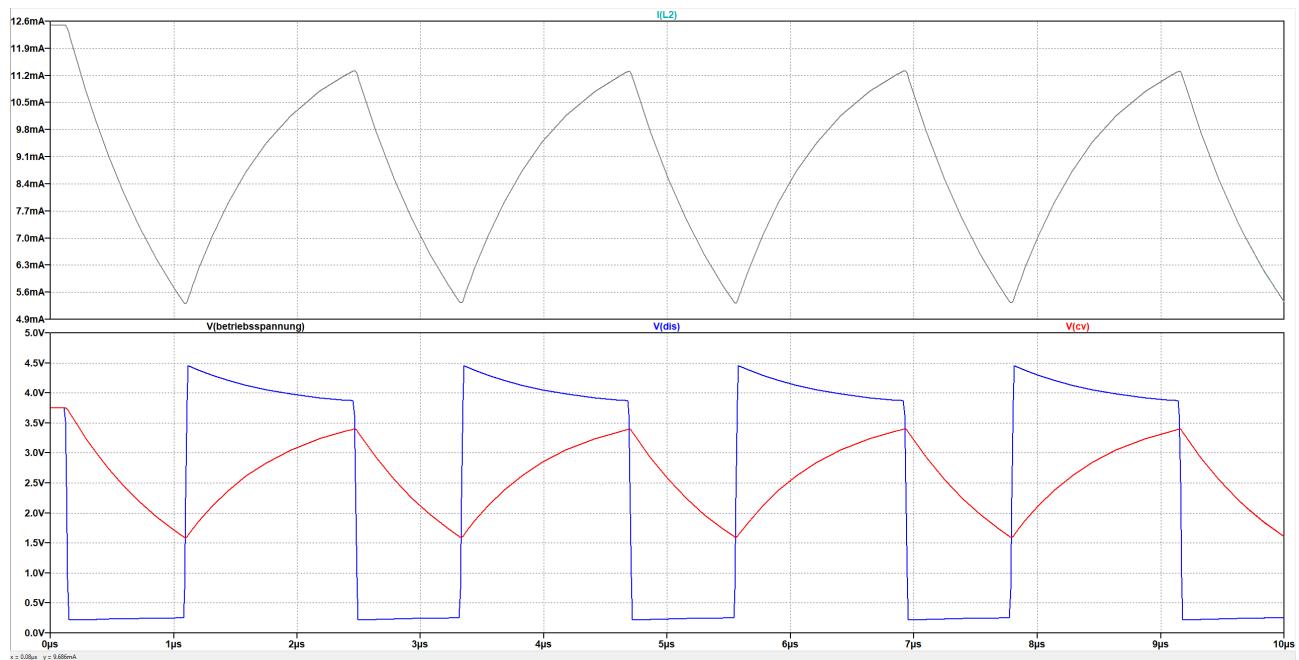


Abbildung 40: RL-Oszillator mit NE555 Zeitsignale

Im oberen Diagramm ist der Verlauf des Stromes durch die Spule zu erkennen der wie erwartet zu- und abnimmt. Im unteren Diagramm ist in schwarz die Betriebsspannung als Referenz in rot der CV Pin und in blau der DIS Pin des NE555. Das rote Signal wechselt zwischen zwei dritteln der Betriebsspannung und einem drittel der Betriebsspannung mit einer Frequenz von  $f = 438 \text{ kHz}$  bei einer Induktivität von  $L = 300 \mu\text{H}$ , einem Widerstand  $R4 = 100\Omega$  und einem Widerstand  $R5 = 300\Omega$ . Die Simulation entspricht hier der Theorie, jedoch bricht das blaue Signal am PIN DIS in der Einschaltphase ein. Der Grund dafür ist der interne Widerstand des NE555, der ab einem gewissen Stromverbrauch für einen Spannungsabfall sorgt. Da es meist besser ist mit niedrigen Frequenzen zu arbeiten ist es nach den Gleichungen 8 und 9 entweder nötig die Induktivität zu erhöhen oder die Widerstände zu verkleinern. Die Induktivität lässt sich aber entweder durch erhöhen der Windungen oder durch Vergrößerung der Spule erreichen, was in vielen Fällen nicht möglich ist oder zunehmend kostspielig ist. Die Reduktion der Widerstände führt zu einer Zunahme des Stromes und der Verlustleistung, was auch unerwünscht ist. Aus diesen Gründen ist der RL-Oszillator nur für hohe Frequenzen geeignet. In der nächsten Abbildung sieht man die Schaltung bei verschiedenen Induktivitäten nämlich bei  $L2 = 100 \mu\text{H}$  und bei  $L2 = 1 \text{ mH}$ .

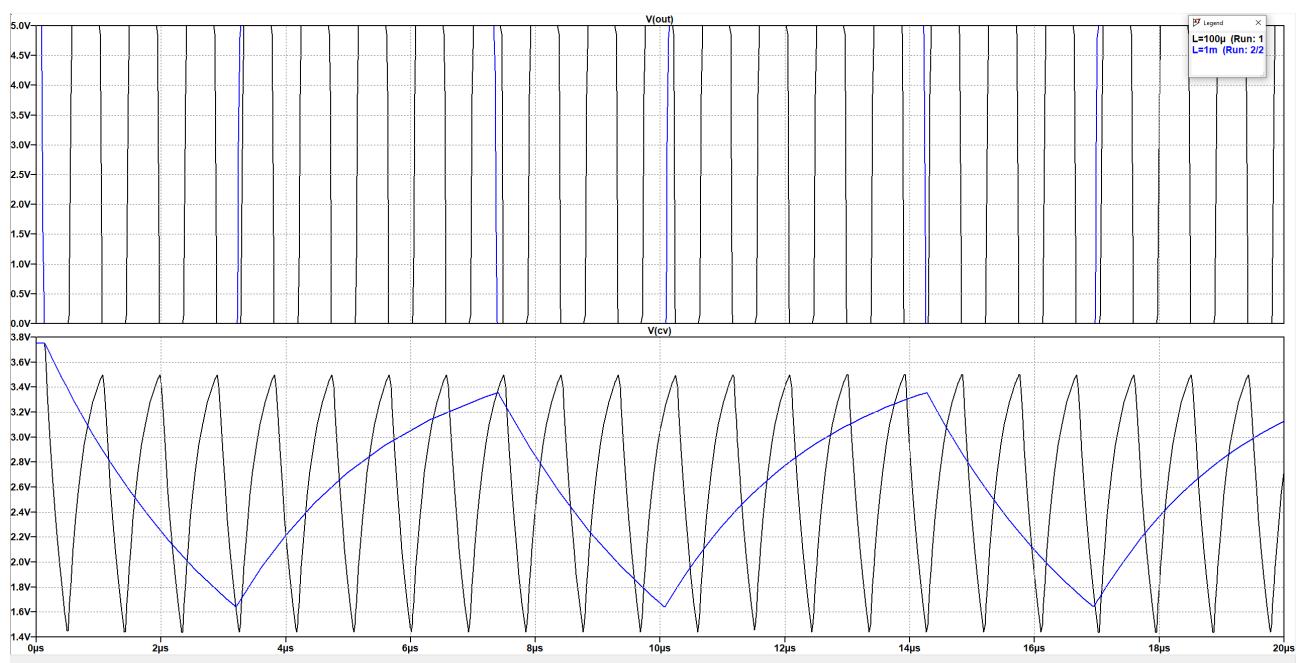


Abbildung 41: Vergleich des NE555-Oszillators bei unterschiedlichen L

Eine Verzehnfachung der Induktivität führt zu einer Reduktion der Oszillatorkreisfrequenz um den Faktor 10.

$$f \propto \frac{1}{L} \quad (10)$$

### 7.3.2 Messung paramagnetischer Metalle

Da nicht alle Metalle ferromagnetische Eigenschaften haben werden viele Stoffe wie Aluminium, Kupfer oder diverse Legierungen von einer niederfrequenten Messung der Induktivität nicht wahrgenommen. Ein Effekt, welcher bei elektrischen Leitern abhilfe schaffen kann, sind die Eddy Currents. Wenn in einem Leiter ein magnetisches Feld einwirkt erzeugt dieses im Objekt Ströme, die wiederum ihr eigenes magnetische Feld erzeugen. Die fließenden Ströme erzeugen im Objekt Ohmsche Verluste, welche normalerweise unerwünscht sind. Sie wirken auch einer Änderung der magnetischen Feldes entgegen, da sie selbst Energie brauchen um ihre Richtung zu ändern. Je größer die Frequenz der eingespeisten Felddichte  $B$  desto stärker wirkt die Flussdichte  $B_{eddy}$  senkend auf die Gesamtflussdichte und so auf den magnetischen Fluss  $\Phi$ . Nach der Gleichung 5 reduziert sich die Induktivität mit zunehmender Frequenz.

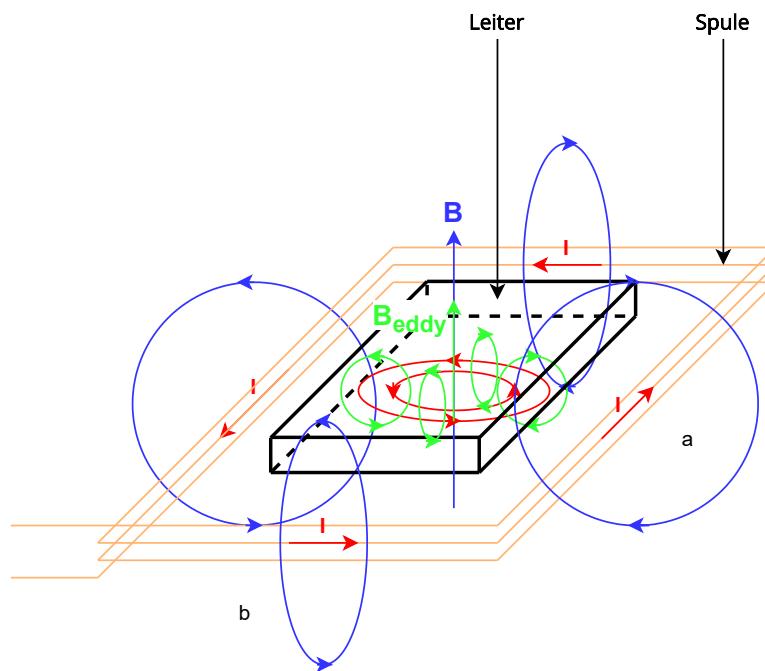


Abbildung 42: Eddy Currents in einem Leiter

Bei sehr hohen Frequenzen ist jedoch zu beachten, dass der Skin-Effekt die Querschnittsfläche, durch die der Wechselstrom fließen kann, reduziert. Somit nimmt der Effekt der Eddy-Currents ab. Diese Phänomene werden jedoch nicht in dieser Arbeit behandelt, da sie außerhalb des Rahmens der Fahrzeugerkennung liegen.

### 7.3.2.1 LC Oszillatoren

Im Vergleich zum RL-Oszillator, der über eine astabile Kippstufe lauft, kann ein LC Oszillator seine Resonanzfrequenz nur durch das verstellen der Induktivität und der Kapazität eines Kondensators verändern. Zudem gibt es in der Theorie keine Wirkverluste, da die Energie abwechselnd zwischen magnetischer und elektroscher Energie umgewandelt wird. In der Realität gibt es jedoch ohmsche Verluste, die zu einem Abklingen der Schwingung führen. Es braucht daher ein aktives Glied wie ein Transistor oder ein Operationsverstärker, der den Oszillator mit Energie versorgt.

Der zeitliche Verlauf einer ungedämpften Schwingung kann in LTSPice simuliert werden indem wir eine Sprungfunktion auf ein LC Glied geben und einen Widerstand in Serie zum Kondensator oder zur Spule geben.

**OSC**

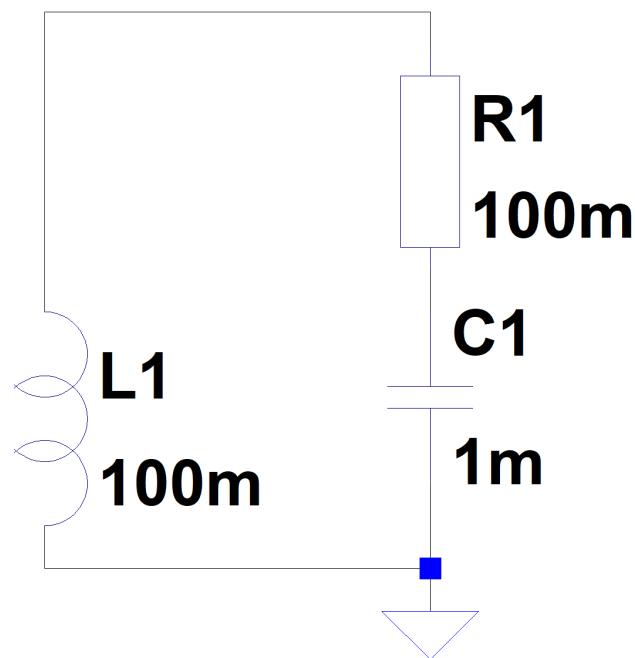


Abbildung 43: Parallelschwingkreis in LTSpice

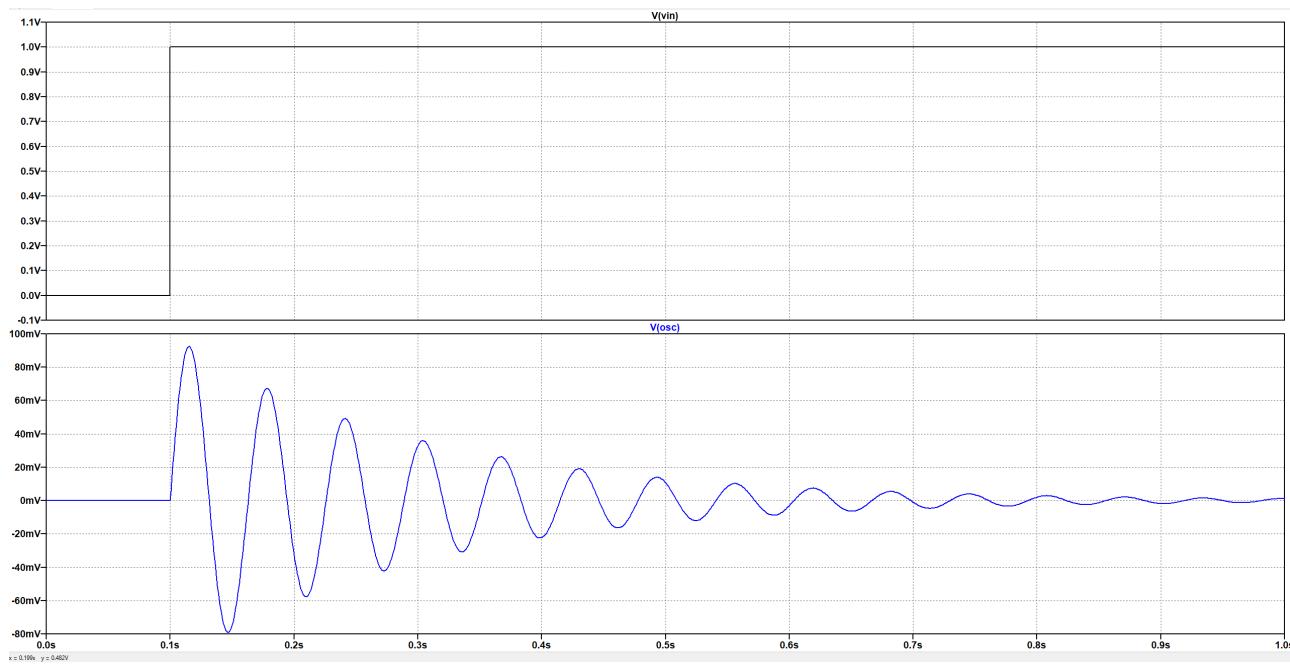


Abbildung 44: Gedämpfte Schwingung

Aus der Simulation ergibt sich über eine Cursormessung für die angegebenen Werte  $L = 100 \text{ mH}$  und  $C = 100 \text{ mF}$  eine Resonanzfrequenz einen Wert von  $f = 15.85 \text{ Hz}$ . Die Resonanzfrequenz für einfache LC-Schwingkreise lässt sich mit der Thomsonschen Schwingungsgleichung errechnen:

$$f = \frac{1}{2 \cdot \pi \sqrt{L \cdot C}} \quad (11)$$

Wobei

$f$  = Resonanzfrequenz des LC-Gliedes

$L$  = Induktivität der Spule

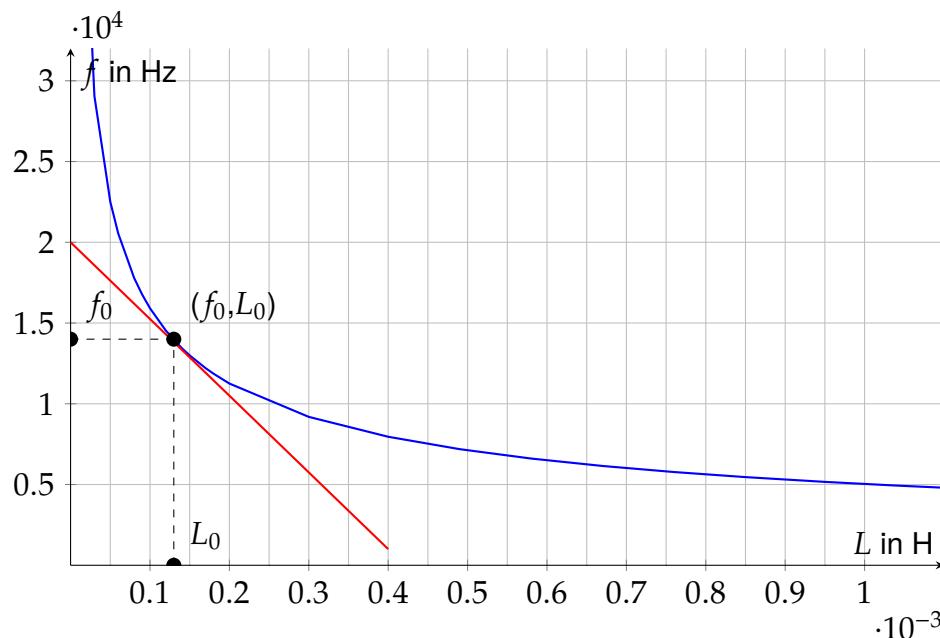
$C$  = Kapazität des Kondensators

Wenn man die Werte  $L = 100 \text{ mH}$  und  $C = 100 \text{ mF}$  in die Gleichung 11 einsetzt erhält man eine Resonanzfrequenz von  $f = 15.915 \text{ Hz}$ . Allgemein tritt in LC-Glieder bei einer Phasenlage von  $\varphi_{LC} = 180^\circ$  und bei der Frequenz nach der Thomsonschen Gleichung 11 Resonanz auf.

### 7.3.2.2 Einfluss von Induktivitätsänderungen auf LC-Oszillatoren

Für die Fahrzeugerkennung ist es von Relevanz zu wissen wie die Frequenz dieser Oszillatoren auf die Änderung der Induktivität der im Boden verbauten Spule reagiert. Wenn eine Parklücke

frei ist besitzt die Spule eine Induktivität von  $L_0$  und hat nach der Thomsonschen Gleichung 11 eine Resonanzfrequenz  $f_0$ . Für kleine Änderungen  $\Delta L$  kann eine Näherung der Frequenzänderung gemacht werden indem man in den Punkt  $(f_0, L_0)$  in den Graphen der Thomsonschen Gleichung aufgetragen über die Induktivität  $L$  eine Tangente in diesen Punkt legt. Bei einer Kapazität von  $C = 1 \mu\text{F}$  erhält man folgenden Graphen:



Mit dieser Näherung kann man mit des Differentials der Thomsonschen Gleichung 11 die Änderung des Funktionswertes  $f(L \pm \Delta L) = \Delta f \approx df$

$$\begin{aligned} df &= \frac{\partial f}{\partial L} \Big|_{L_0} \cdot dL \\ df &= \frac{\partial}{\partial L} \left( \frac{1}{2\pi\sqrt{L_0 C}} \right) \Delta dL \\ df &= -\frac{1}{4\pi} (L_0 C)^{-\frac{3}{2}} \cdot \Delta L \end{aligned}$$

Für relative Änderungen der Induktivität  $\Delta L$  um den Wert  $L_0$  gilt:

$$\frac{df}{f_0} = -\frac{1}{2C} \cdot \frac{\Delta L}{L_0} \quad (12)$$

Das bedeutet, dass eine kleine relative Änderung der Induktivität  $\frac{\Delta L}{L_0}$  bei gleicher Kapazität  $C$  für eine halb so große relative Änderung in der Frequenz führt. Dieser Ansatz gilt nur bei kleinen  $\Delta L$ .

### 7.3.2.3 Messung der Frequenz eines Colpitts-Oszillatorm

Der Colpitts-Oszillatorm ist eine Variante des LC-Oszillatorm und wird in der Praxis oft mit einem aktiven Element betrieben. Im Vergleich zu einem einfachen LC-Glied besitzt dieser Oszillatorm zwei Kondensatoren. Das Schaltbild sieht so aus:

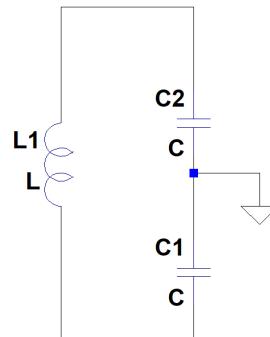


Abbildung 45: Colpitts LC-Glied

Dieses LC-Glied wird als Rückkoppelglied eingesetzt und sorgt für die nötige Phasendrehung von  $\varphi_{LC} = 180^\circ$ . Wie man in der obigen Abbildung sehen kann liegen die Kondensatoren  $C_1$  und  $C_2$  in Serie. Es ergibt sich daher eine Gesamtkapazität von  $C_{ges}$ .

$$C_{ges} = \frac{C_1 \cdot C_2}{C_1 + C_2} \quad (13)$$

Durch einsetzen in die Thomsonsche Gleichung erhält man für die Resonanzfrequenz eines Colpitts-Oszillatorm:

$$f = \frac{1}{2 \cdot \pi \sqrt{L \cdot \left( \frac{C_1 \cdot C_2}{C_1 + C_2} \right)}} \quad (14)$$

Wobei

$f$  = Resonanzfrequenz des Colpitts-Oszillatorm

$L$  = Induktivität der Spule

$C_1$  = Kapazität des Kondensators  $C_1$

$C_2$  = Kapazität des Kondensators  $C_2$

Das aktive Element wird an die Anschlüsse der Spule angeschlossen muss selbst eine Phasendrehung von  $180^\circ$  liefern. Zusätzlich muss die Verstärkung des Gliedes größer als 1 damit eine dauerhafte Schwingung entstehen kann. Mit einem Operationsverstärker muss der nicht invertierende Eingang auf eine Bezugsspannung gelegt werden. Diese Bezugsspannung wird oft in die Mitte des Versorgungsspannungsbereich gelegt. Bei einem Spannungsbereich von 0 V bis 5 V ist eine Bezugsspannung von 2,5 V sinnvoll. Der invertierende Eingang wird an das Colpitts LC-Glied angeschlossen. Der Ausgang des Operationsverstärker wird über einen Widerstand an das LC-Glied geschalten, damit das Rückkoppelglied besser vor Übersteuerungen am Ausgang des Operationsverstärkers geschützt ist.

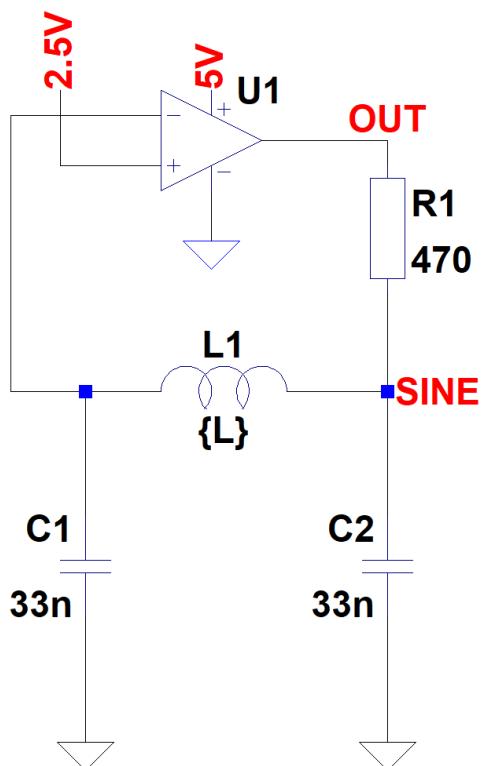


Abbildung 46: Aktiver Colpitts-Oszillator

Wenn man nun verschiedene Werte für Induktivität einsetzt ergibt sich durch die Simulation dieses Schaltbildes folgende Zeitsignale für den Ausgang des Operationsverstärker OUT und der Spannung SINE:

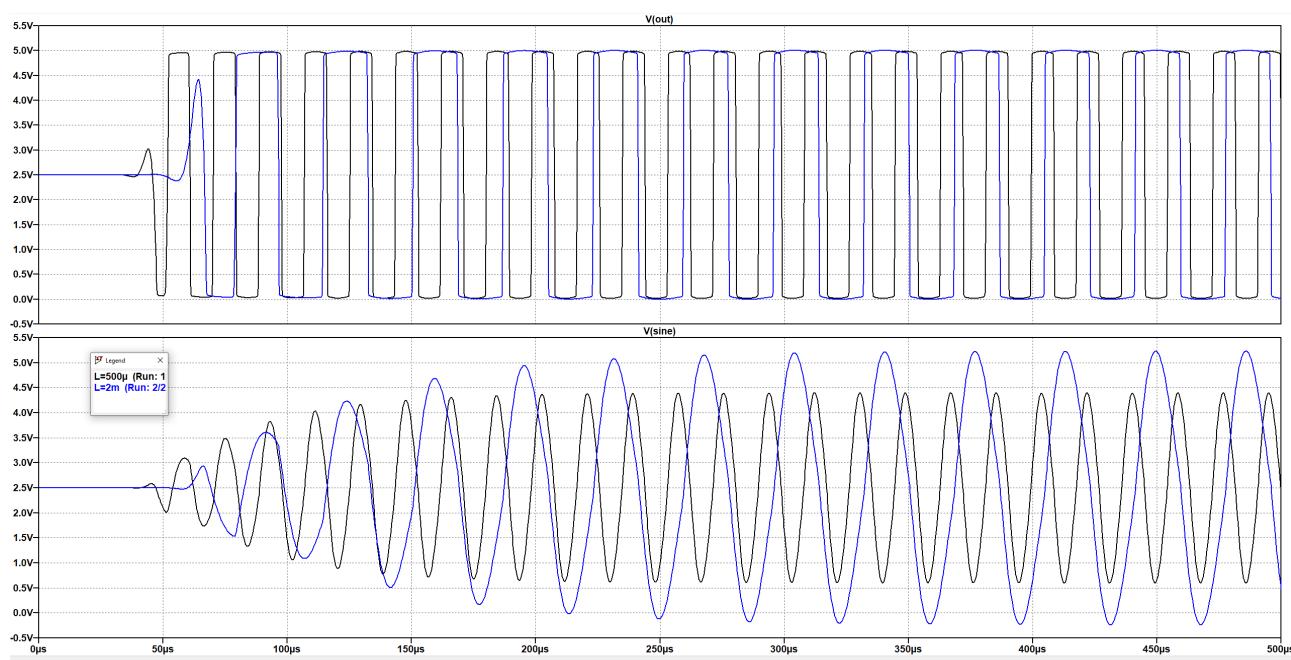


Abbildung 47: Zeitsignale Colpitts-Oszillator

Es ist ersichtlich, dass eine Zunahme der Induktivität zu einer Abnahme in der Frequenz der beiden Signale führt.

## 7.4 RS485 Bussystem

### 7.4.1 Benutzte Standards

#### 7.4.1.1 ASCII

ASCII ist eine amerikanischer Standard zur Kodierung einer 7 Bit langen folge. Dieser Standard ermöglicht es Zeichen wie 'A' oder 'b' sowie auch Zahlen und gewissen Sonderzeichen in Bits umzusetzen.

#### 7.4.1.2 USB

Unter USB versteht man eine serielle Bussystem, welches dazu dient Peripherie Geräte an einen Computer zu verbinden. USB ermöglicht es Geräte während der Laufzeit des Betriebssystems anzustecken und wieder zu entfernen. Je nach Version des USB Standards ist es möglich unterschiedliche Datenraten und Leistungen zu verwenden, wobei neuere Standards Rückkompatibel zu alten Versionen sind. Für die Ansteuerung des Bussystems ist es notwendig einen Computer anzuschließen. Der USB Standard bietet sich hier an, da er weit verbreitet ist und daher mit vielen Computern kompatibel ist.

#### 7.4.1.3 UART

Unter UART versteht man eine asynchrone serielle Schnittstelle. Die elektrische Schnittstelle braucht daher auch keine Takteleitung die dem Empfänger der Daten mitteilt wann er zu lesen hat. Stattdessen gibt es zwei Datenleitungen nämlich eine für das Senden TX und eine für das Lesen RX. Auf diesen Datenleitungen befinden sich nur die Bitströme des Empfängers und des Senders. Sie besitzen einen Ruhepegel von logisch 1. UART synchronisiert sich idem es beim Start der Daten ein Startbit schickt, welche Sender und Empfänger synchronisiert. Diese beiden Geräte brauchen daher auch genaue interne Uhren um sich über den Zeitraum des Datenaustausches synchron zu halten. Die Dauer dieses Datenaustausches hängt vor allem von der Bitrate auch genannt Baudrate ab aber auch die Konfiguration der Anzahl von Datenbits und Paritätsbits haben Einfluss auf diese Größe.

In der folgenden Abbildung ist zeitliche Verlauf einer Datenleitung dargestellt es wird das Zeichen 'A' in ASCII codiert übertragen bei einer Baudrate von 57600 Zeichen pro Sekunde.

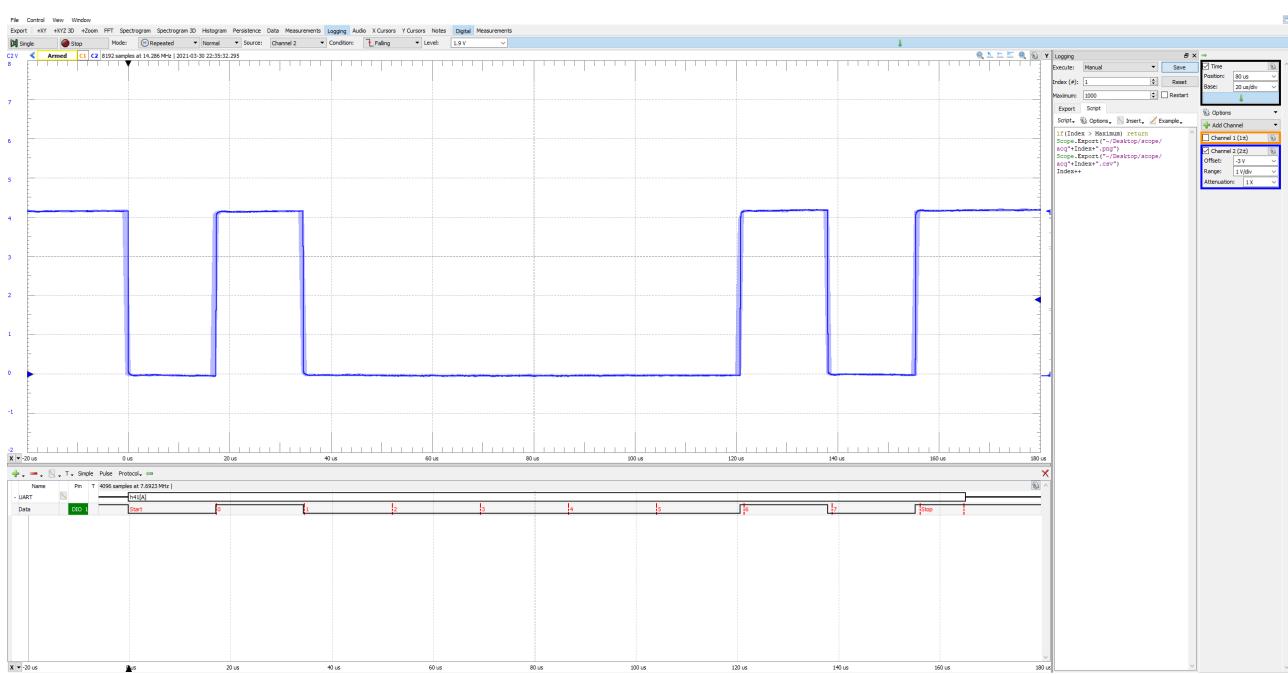


Abbildung 48: Beispiel einer UART Zeichenübertragung

#### 7.4.1.4 RS485

RS485 ist ein Standard für eine physikalische Schnittstelle für Datenübertragung von UART Daten. Er besitzt zwei symmetrische Leitung oft mit A und B bezeichnet, dessen Differenzspannung für die Auswertung des Datenstromes herangezogen wird. Er ist dafür ausgelegt mehrere Geräte an die gleiche Leitungen anzuschließen. Die maximale Anzahl an Geräten ist mit 32 fix vorgegeben. Es ist zusätzlich notwendig die Spannungen  $U_A$  und  $U_B$  im Ruhezustand über einen IC oder einen Spannungsteiler auf fixe Potentiale zu legen. Der Spannungsteiler muss wie folgt aussehen um dem Standard zu entsprechen.

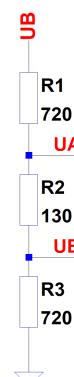


Abbildung 49: RS485 Widerstandsnetzwerk

Um diese physikalische Umwandlung zu ermöglichen ist eine Umwandlungs IC erforderlich. In dieser Arbeit wurde ausschließlich der MAX485CSA für Wandlung zwischen den Standardpegeln der UART und die der RS485 Schnittstelle verwendet.

Dieser IC ist Bussfähig und kann daher seine Ausgänge hochohmig machen. Er besitzt folgende Eingänge und Ausgänge:

- **!RE: Read enable**

Wenn dieser Eingang auf logisch 0 gesetzt ist wertet der IC des Differenzsignal  $U_{AB}$  aus und liefert das Ergebnis an den Ausgang RO. Ansonsten setzt er den Ausgang RO in den Tristate beziehungsweise auf hochohmig.

- **DE: Data enable**

Wenn dieser Eingang auf logisch 1 gesetzt ist wird die Spannung am Eingang DI in die zugehörigen Pegel  $U_A$  und  $U_B$  umgewandelt. Ansonsten setzt er den Eingang DI in den Tristate beziehungsweise auf hochohmig.

- **RO: Read out**

Ist die eingelesene Spannung die sich logisch aus der Differenzspannung  $U_{AB}$  ergibt wenn !RE auf logisch 0 ist.

- **DI: Data In**

Ist die eingehende Spannung die auf die RS485 Schnittstelle geschrieben werden soll, wenn der Eingang DE auf logisch 1 ist.

Die Auswertung der Differenzspannung  $U_{AB}$  kann auch in der folgenden Tabelle aus dem Datenblatt des MAX485CSA entnommen werden.

**Table 2. Receiving**

INPUTS			OUTPUT
$\overline{RE}$	DE	A-B	RO
0	0	$\geq +0.2V$	1
0	0	$\leq -0.2V$	0
0	0	Inputs open	1
1	0	X	High-Z*

X = Don't care

High-Z = High impedance

\*Shutdown mode for MAX481/MAX483/MAX487

Abbildung 50: Logische Tabelle für  $U_{AB}$

In der folgenden Abbildung wird wieder an 'A' ASCII kodiert versendet nur dieses mal wird eine RS485 Schnittstelle mit den entsprechenden Pegeln verwendet verwendet. In blau ist die Spannung  $U_A$  und in orange die Spannung  $U_B$  zu sehen. Darunter befindet sich das logische UART Signal am Ausgang RO.

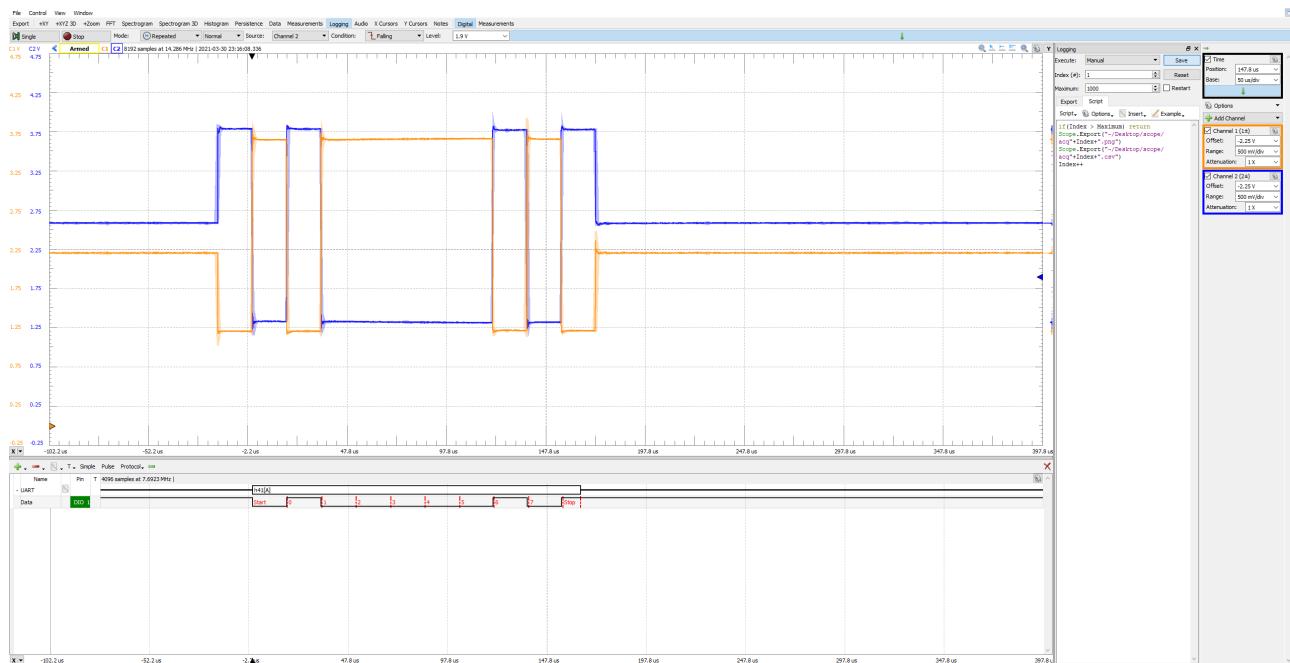


Abbildung 51: Beispiel einer RS485 Zeichenübertragung

#### 7.4.1.5 RJ45

Um die Anzahl der Kabel zu minimieren und um das Anschließen von Geräten an den Bus einfach zu halten ist ein mehradriger Stecker mit zugehörigem Kabel nötig.

Der RJ-45 Stecker ist eine genormter Stecker und besitzt insgesamt 8 Adern und ist für Kommunikationsanschlüsse gedacht. Er kann daher größere Datenraten zulassen und viele auf dem Markt vorhandene Kabel haben zusätzliche Abschirmungen, die die Datenleitungen vor elektrischen Störungen schützen.



Abbildung 52: RJ45 Steckerbuchse und Kabel

#### 7.4.2 Aufbau

Das Bussystem wir über eine Master Slave Struktur verwaltet was heißt das nur das ein einzelnes Master-Gerät als erstes andere Teilnehmer am Bus ansprechen kann. Slave-Geräte können nur nach Anfrage des Masters auf den Bus schreiben. Neben der Steuerung der Kommunikation sollte

das Master-Gerät aus praktischen Gründen ebenfalls die Slave-Geräte mit Energie versorgen und ihnen eindeutige Adressen zuweisen können.

In den unteren Abbildungen sind der logische Aufbau und der physikalische Aufbau zu sehen. Auf die einzelnen Funktionen diverser Leitungen wird in den folgenden Paragraphen eingegangen.

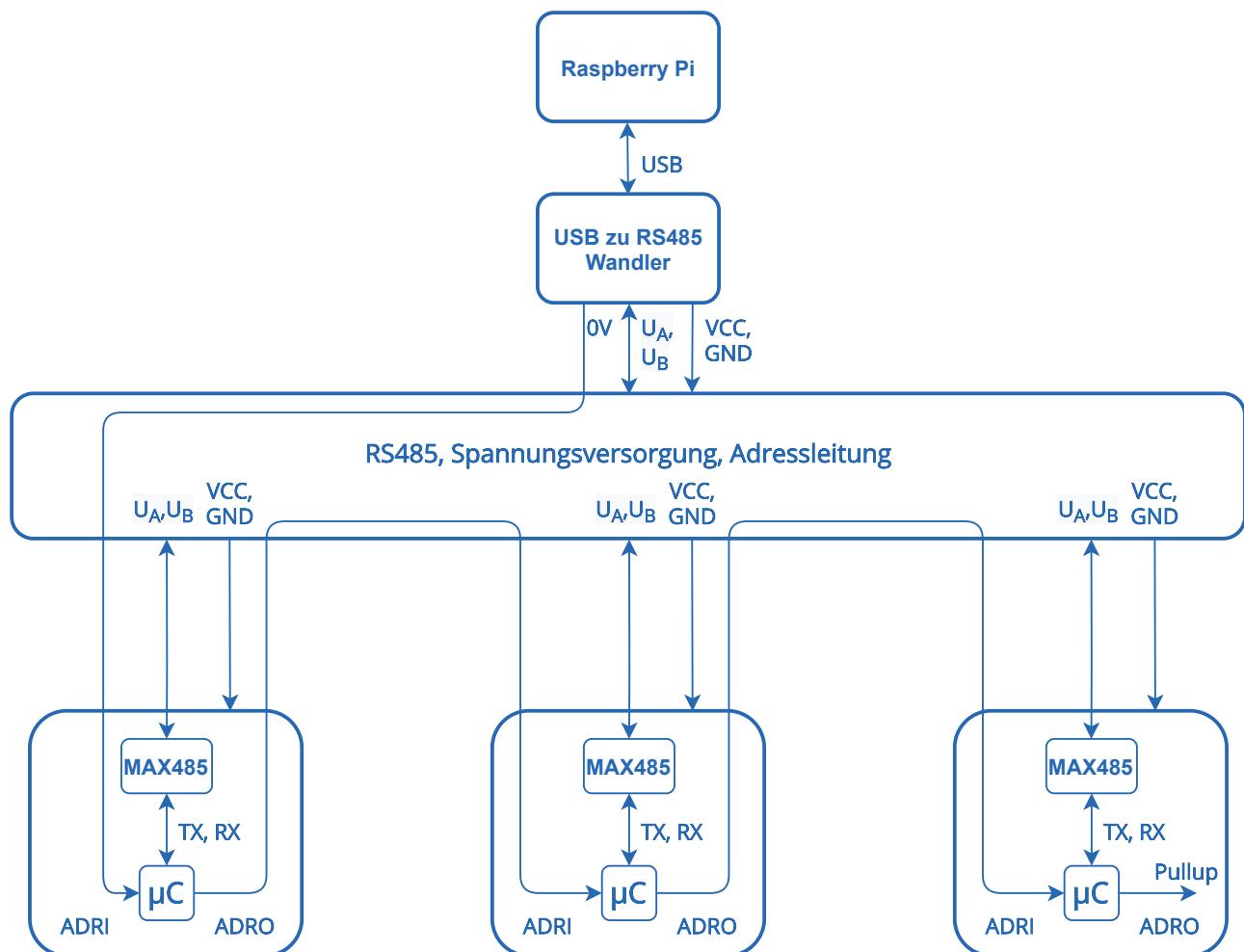


Abbildung 53: Logische Übersicht des Bussystems



Abbildung 54: Physische Übersicht des Bussystems

Ganze rechts oben in schwarz ist der RaspberryPi der hier als Computer dient es kann aber auch ein anderer Computer verwendet werden. Unterhalb des RaspberryPi ist der USB-RS485 Wandler der mit einem USB-Kabel am RaspberryPi angeschlossen ist und von einer Versorgungsspannung hängt. Der USB-Rs485 Wandler ist wiederrum mit einem RJ45-Kabel an einen Slave verbunden. In diesem Bild sind zwei Slaves an das Bussystem geschlossen worden. Es können aber bis zu 31 sein. Ganz rechts ist der Leitungsabschluss der in Abbildung 49 zu sehen ist.

Die elektrische Belegung der Leitungen ist wie in der folgenden Tabelle festgelegt worden. Mit dieser Belegung ist es möglich die Slave-Geräte an den RS485 Datenbus zu legen, sie mit Spannung zu versorgen und die Adressvergabe mit einer zusätzlichen Logikleitung zu regeln.

#### 7.4.2.1 Funktion der Adresslogikleitung

Da alle Teilnehmer des RS485-Busses parallel an den Datenleitung A und B liegen kann der Master sie voneinander nicht unterscheiden. Dieses Problem beseitigt die Adressleitung, welche die einzelnen Geräte logisch aneinanderreihrt.

Pinnummer	Farbe	Netz	Beschreibung
1	Orange / weiß	GND	Masse
2	Orange	+5V	5 Volt Versorgungsspannung
3	Grün / weiß		unbelegt
4	Blau	A	RS485 positives Datensignal
5	Blau / weiß	B	RS485 negatives Datensignal
6	Grün	VCC	Positive Versorgungsspannung +9V bis 30V
7	Braun / weiß	GND	Masse
8	Braun	ADR	Logiksignal zur Vergabe der Slave Adressen

Tabelle 2: RJ45 allgemeine Pinbelegung

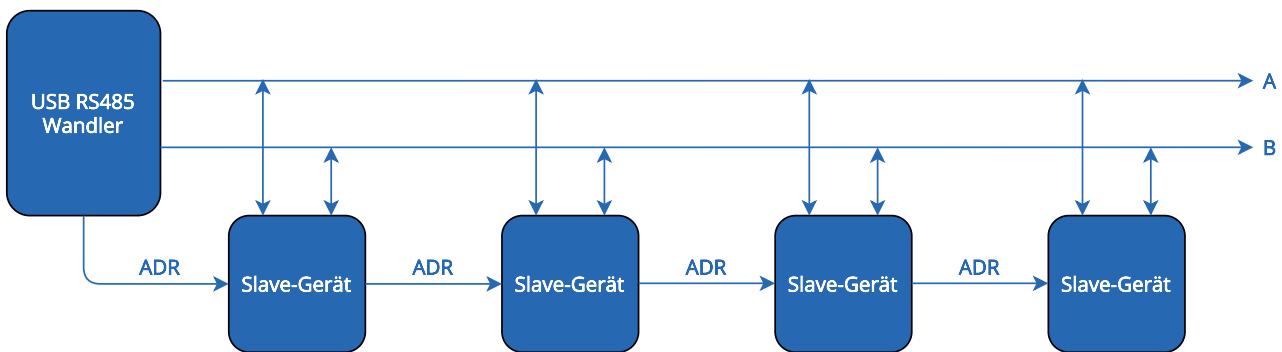


Abbildung 55: Anordnung der Slave-Geräte mit Adresslogikleitung

Jedes Slave-Gerät setzt seine Adresslogikleitung über einen Pullup-Widerstand auf logisch 1 wenn es keine Adresse zugeordnet bekommen hat. Als erstes in der Reihe ist der Master, der seine Adresslogikleitung auf logisch 0 setzt und so dem ersten Gerät in der Kette bekannt gibt, dass es eine Adresse zugewiesen bekommt. Wenn es diese Adresse bekommen hat legt das Slave-Gerät seine ausgehende Adresslogikleitung ADRO auf logisch 0. So kann das nächste Gerät in der Kette erkennen, dass es eine Adresse zugewiesen kriegt, indem es auf die eingehende Adressleitung ADRI achtet.

In den folgenden Abbildungen sieht man wie die Geräte über den RJ45 Stecker elektrisch am Bus hängen. Der Master hat nur einen Stecker und hat seine Adresslogikleitung fix auf logisch 0 gelegt. Die Slaves hingegen brauchen zwei Stecker um die Adresskette zu bilden und unterscheiden zwischen eingehender Adresslogikleitung ADRI und ausgehender Adresslogikleitung ADRO.

Wenn jedoch die Slaves falsche herum angeschlossen werden und ADRI und ADRO vertauscht sind kann dieser Fehler vom Mikrocontroller erkannt und behoben werden. Die genauen Details dieser Funktion werden in späteren Abschnitten erklärt.

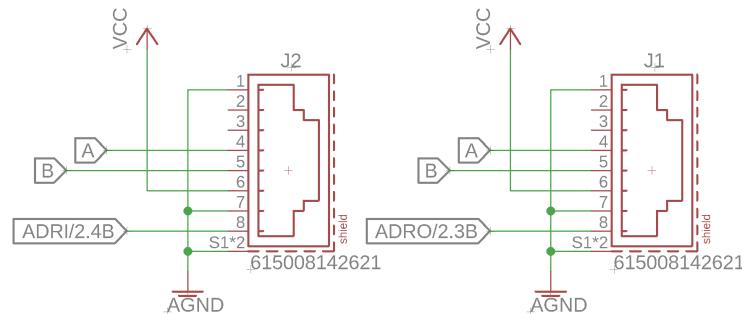


Abbildung 56: RJ45 Pinbelegung des Slave-Geräts

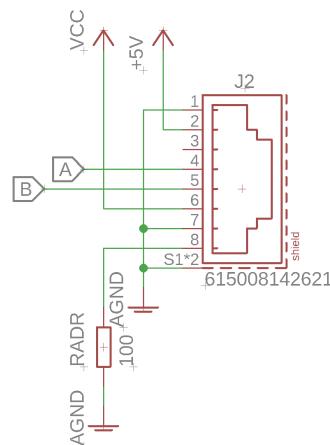


Abbildung 57: RJ45 Pinbelegung des Master-USB-Geräts

#### 7.4.2.2 Versorgungsleitungen VCC und GND

Da es für jeden Parkplatz ein Slave-Gerät gibt ist die Länge der Busleitung nicht unbeträchtlich. Es kann so entlang der Leitung zu abfallen in der Versorgungsspannung aufgrund des Widerstands der langen Leitung. Um dem entgegenzuwirken muss die Versorgungsspannung VCC bezogen auf Masse größer als die benötigte Spannung an den Geräten sein. Die Slave-Geräte besitzen alle einen Linearregler der für einen konstante Spannung von 5V sorgt. Die Eingangsspannung des Spannungsreglers muss je nach Ausführung um eine gewisse Differenzspannung größer sein als die am Ausgang benötigte Spannung sein. Dieser Wert wird als *Dropout Voltage* bezeichnet. Für den IC L78L05ACD lässt sich aus dem Datenblatt folgende Spannung nachlesen: 2 V

**Table 4. Electrical characteristics of L78L05C - Refer to the test circuits,  $T_J = 0$  to  $125^\circ\text{C}$ ,  $V_I = 10\text{ V}$ ,  $I_O = 40\text{ mA}$ ,  $C_I = 0.33\text{ }\mu\text{F}$ ,  $C_O = 0.1\text{ }\mu\text{F}$  unless otherwise specified**

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
$V_O$	Output voltage	$T_J = 25^\circ\text{C}$	4.6	5	5.4	V
$V_O$	Output voltage	$I_O = 1$ to $40\text{ mA}$ , $V_I = 7$ to $20\text{ V}$	4.5		5.5	V
		$I_O = 1$ to $70\text{ mA}$ , $V_I = 10\text{ V}$	4.5		5.5	
$\Delta V_O$	Line regulation	$V_I = 8.5$ to $20\text{ V}$ , $T_J = 25^\circ\text{C}$			200	mV
		$V_I = 9$ to $20\text{ V}$ , $T_J = 25^\circ\text{C}$			150	
$\Delta V_O$	Load regulation	$I_O = 1$ to $100\text{ mA}$ , $T_J = 25^\circ\text{C}$			60	mV
		$I_O = 1$ to $40\text{ mA}$ , $T_J = 25^\circ\text{C}$			30	
$I_d$	Quiescent current	$T_J = 25^\circ\text{C}$			6	mA
		$T_J = 125^\circ\text{C}$			5.5	mA
$\Delta I_d$	Quiescent current change	$I_O = 1$ to $40\text{ mA}$			0.2	mA
		$V_I = 8$ to $20\text{ V}$			1.5	
$eN$	Output noise voltage	$B = 10\text{ Hz}$ to $100\text{ kHz}$ , $T_J = 25^\circ\text{C}$		40		$\mu\text{V}$
$SVR$	Supply voltage rejection	$V_I = 9$ to $20\text{ V}$ , $f = 120\text{ Hz}$ $I_O = 40\text{ mA}$ , $T_J = 25^\circ\text{C}$	40	49		dB
$V_d$	Dropout voltage			2		V

Abbildung 58: Elektrische Eigenschaften des L78L05ACD

Daraus lässt sich schließen dass der IC mindestens eine Spannung von 7 V braucht um zu funktionieren. Als Eingangsspannung für alle Messung wurde 9 V verwendet. Nach Datenblatt ist die Eingangsspannung auf einen Maximalwert von 30 V beschränkt. Es ist zu beachten, dass eine Erhöhung der Eingangsspannung zu mehr Verlusten am Linearregler führt und somit für eine schlechte Effizienz sorgt.

#### 7.4.3 Implementation eines eigenen Protokolls

RS485 spezifiziert keine Protokoll unter welchem Busteilnehmer untereinander kommunizieren. Es gibt bereits viele fertige Lösungen wie zum Beispiel Modbus, welches auf einer Master/Slave-Architektur basiert. Es ist aber auch möglich ein eigenes Protokoll zu erstellen. In den folgenden Absätzen wird der Aufbau und die Funktion eines eigenen implementierten Protokolls für die Fahrzeugerkennung, erklärt.

#### 7.4.3.1 Aufbau von Datenframes

Mit UART ist es möglich, unter der Konfiguration von 8 Datenbits, eine Byte pro Datenaustausch zu senden. Diese Bytes bilden die Grundbausteine unseres sogenannten Datenframes, welcher eine Aneinanderreihung von Bytes ist. Jedes Byte kann eine unterschiedliche Funktion haben. Es kann zu einem einfache Daten beinhalten oder dem anderen Teilnehmer bestimmte Steueranfragen mitteilen. Die Datenframes sind in folgende Grundstruktur aufgeteilt.

Datenframe																
Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Name	START	ADR	CTRL	ARG	DATA_0 bis DATA_7								STOP	LF		
Hexwert	0x02													0x03	0x11	

Tabelle 3: allgemeiner Aufbau eines Datenframes

- **START, STOP und LF**

Diese Bytes helfen den Geräte den Anfang und des Ende eines Frames zu erkennen. Das START-Byte signalisiert den Anfang des Datenframes und hat einen fixen von 0x02 während das Byte STOP mit dem fixen Hexwert von 0x03 das Ende das Ende signalisiert. Zusätzlich kommt am Ende noch das LF-Byte mit einem Hexwert von 0x11. In ASCII kodiert ist das das Zeichen einer neue Zeile in einem Text und ist vor allem nützlich, da viele Bibliotheken dieses Zeichen als Endzeichen zu Datenauslesung aus einem seriellen Buffer verwenden.

- **ADR**

Jedes Gerät am Bus ist unter seiner eigenen Adresse erreichbar welche aus zwei Bytes besteht. Die Adressen werden über den Master verteilt, der eine fixe Adresse von 0x01 besitzt. Zusätzlich sind alle Slaves unter einer Broadcastadresse von 0x00 erreichbar. Es ergibt sich mit zwei Bytes eine Adressraum der Größe  $2^{16} = 65536$  inklusive Master und Broadcast.

- **CTRL**

Das Kontrollbyte CTRL signalisiert einen bestimmten Steuerbefehl. Mit ihm kann man eine Frequenzmessung der Spule an einem Slave auslösen oder einzelne LEDs ein- und ausschalten.

- **ARG**

Die Argumentbytes ARG1 und ARG2 geben dem Steuerbefehl zusätzliche Daten mit. So kann man zum Beispiel zwischen LED1 und LED0 und einschalten und ausschalten unterscheiden.

- **DATA**

Hier versteht man 8 Datenbytes, die für den Austausch von größeren Werten, wie einer Frequenz von nutzen sind. Diese Datenbytes sind in ASCII kodiert um Konflikte mit den START, STOP und LF Bytes zu vermeiden. Jedes Datenbyte kann daher einen Hexwert von **0x0** bis **0xF**. Der größtmögliche Wert, der sich verschicken lässt ist daher **0xFFFFFFFF** was in Dezimal einem Wert von  $2^{32} - 1 = 4\,294\,967\,294$  entspricht.

#### 7.4.3.2 Steuerabläufe

Um eine Aktion auszuführen schreibt der Master auf den Bus ein Datenframe und wartet auf die Antwort des angesprochenen Slaves.

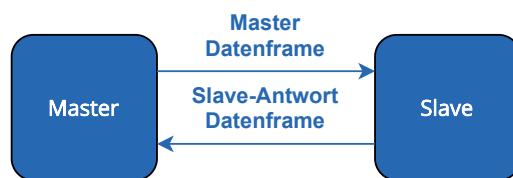


Abbildung 59: Ablauf von Steueranfragen

Die Art des Steuerbefehls wird durch das CTRL-Byte angegeben und definiert so die Inhalte der Bytes. Es muss aber zwischen einem Master Datenframe und einem Slave-Response Datenframe unterschieden werden. In den nächsten Abbildungen werden die möglichen Steuerbefehle für Master und Slave in Tabellenform angeführt.

MASTER	DATA_0													
	ADR_MSB	ADR_LSB	CTRL	ARG_1			MEASURE_TIME			MEASURE_TIME				
DEC	HEX	DEC	HEX	ASCII	DEC	HEX	NAME	DEC	HEX	NAME	DEC	HEX	NAME	
0	0x00	0	0x00	A	65	0x41	ADR	G	71	0x47	GIVE	0	0x00	-
0	0x00	0	0x00				R	82	0x52	REMOVE	0	0x00	-	
var	var	S	83	0x53	GET_STATE		0	0x00	-	0	0x00	-	-	
var	var	F	70	0x46	GET_FRO		0	0x00	-	0	0x00	-	MEASURE_TIME_MSB	
var	var	L	76	0x4C	GET_L		0	0x00	-	0	0x00	-	MEASURE_TIME_LSB	
var	var	0	48	0x30	100	D	68	0x44	SET_IN_OUT	var	var	IN_OUT	-	
var	var					B	66	0x42	READ	0	0x00	-	-	
var	var					I	73	0x49	SET	var	var	ON_OFF	-	
var	var	1	49	0x31	101	D	68	0x44	SET_IN_OUT	var	var	IN_OUT	-	
var	var					B	66	0x42	READ	0	0x00	-	-	
var	var					I	73	0x49	SET	0	0x00	ON_OFF	-	
var	var	C	67	0x43	CALIBRATE		-	-	-	0	0x00	-	FRO_MSB	
var	var						-	-	-	0	0x00	-	FRO_LSB	
var	var	P	80	0x50	PING		-	-	-	0	0x00	-	-	

Abbildung 60: Master Datenframe

SLAVE RESPONSE										DATA_0	
ADR_MSB		ADR_LSB		CTRL		ARG_1					
DEC	HEX	DEC	HEX	ASCII	DEC	HEX	Function	ASCII	DEC	HEX	NAME
0	0x00	1	0x01	A	65	0x41	ADR	G	71	0x47	GIVE
0	0x00	0	0x00	R	82	0x52	REMOVE	0	0x00	-	-
0	0x00	1	0x01	S	83	0x53	GET_STATE	0	0x00	-	100_IN_OUT
0	0x00	1	0x01	F	70	0x46	GET_FRQ	0	0x00	-	100_STATE
0	0x00	1	0x01	L	76	0x4C	GET_L	0	0x00	-	IO1_IN_OUT
0	0x00	1	0x01	O	48	0x30	IO0	D	68	0x44	SET_IN_OUT
0	0x00	1	0x01			0	B	66	0x42	READ	0
0	0x00	1	0x01			0	I	73	0x49	SET	var
0	0x00	1	0x01	1	49	0x31	IO1	D	68	0x44	SET_IN_OUT
0	0x00	1	0x01			0	B	66	0x42	READ	0
0	0x00	1	0x01			0	I	73	0x49	SET	0
0	0x00	1	0x01	C	67	0x43	CALIBRATE	-	-	0	0x00
0	0x00	1	0x01			0x00	0	-	-	0	0x00
0	0x00	1	0x01	P	80	0x50	PING	-	-	0	0x00
0	0x00	1	0x01						ADR_MSB	ADR_LSB	LOCAL_ADR_MSB
									LSB		CTRL_ARG_1
											ARG_2 CROSSOVER

Abbildung 61: Slave Response Datenframe

Um den Ablauf einer deutlicher darzustellen wird der Steuerbefehl IO1 mit Datenframe Antwort des Slaves gemessen. Wir erhalten folgendes Bild am Oszilloskop.

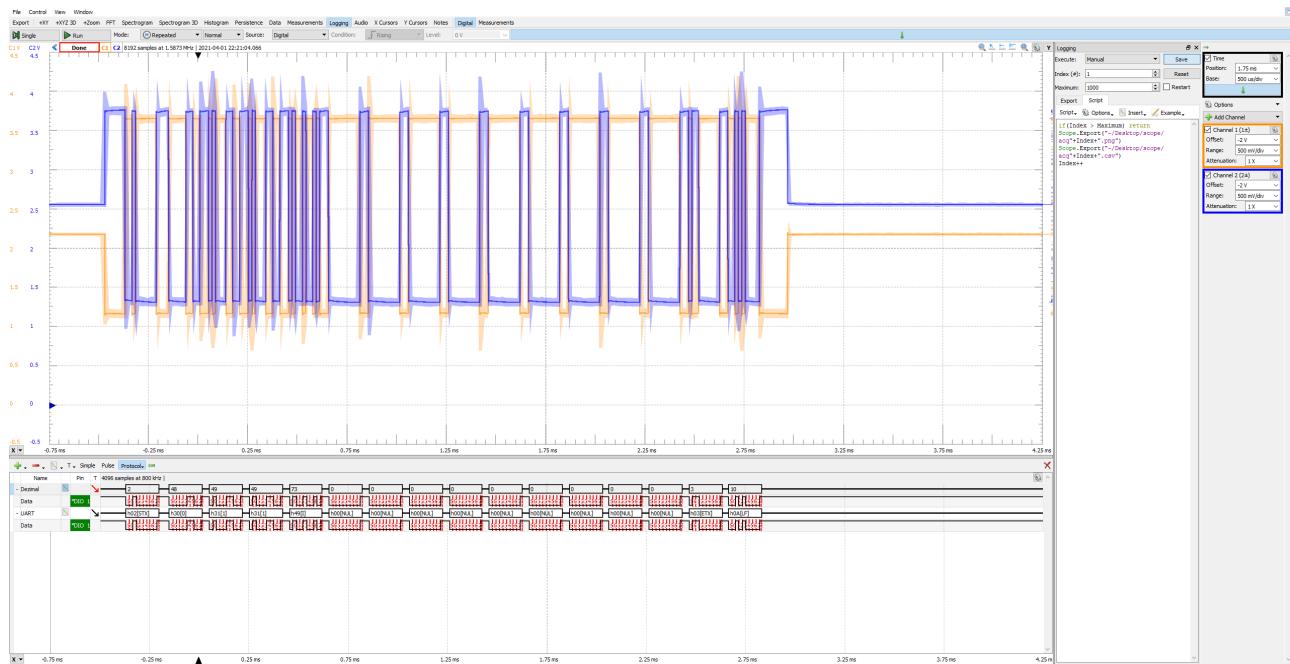


Abbildung 62: gemessene Slave Response bei Steuerbefehl IO1

Datenframe										
Byte	0	1	2	3	4	5	6	7-13	14	15
Name	START	ADR		CTRL	ARG		DATA		STOP	LF
Hex	0x02	0x30	0x31	0x31	0x49	0x00	0x30	0x00	0x03	0x11
ASCII	STX	0	1	1	I	NUL	0	NUL	ETX	LF

Tabelle 4: ausgelesene Bytes aus einem gemessenen Datenframe

Wir sehen dass die Bytes START, ETX und LF die dix zugewiesenen Werte `0x02`, `0x03` und `0x11` haben. Die ADR Bytes haben ASCII dekodiert die Zeichen '0' und '1' welche zusammengezählt die Adresse 1 ergeben. Dies ist die fixe Adresse des Masters und stimmt auch mit unserem Master/Slave System überein, wo die Slaves Datenframes nur an den Master zurück schicken. Das Kontrollbyte CTRL hat den Hexadezimalwert `0x31` und entspricht ASCII dekodiert dem Zeichen '1'. Sieht man nun in der Tabelle mit den Slave-Response Datenframes nach erhalten wir den Befehl IO1. Mit diesem Befehl wird ein digitaler Ein- und Ausgang des Mikrocontroller gesteuert. Das erste

Argumentbyte ergibt ASCII dekodiert das Zeichen 'I' was laut gleicher Tabelle SET bedeutet. Das zweite Argumentbyte hat den Hexadezimalwert 0x30 welches dekodiert das ASCII Zeichen '0' ist. Dieses Zeichen gibt an, dass der Ausgang IO0 auf logisch 0 zu schalten ist. Die Datenbytes für die den Befehl IO1 sind bis auf das erste alle von sich aus leer. Das erste Datenbyte soll bestätigen dass der Ausgang auf logisch 0 geschalten worden ist und tatsächlich steht diesem Byte das Zeichen '0'.

Der interne C-Code des Slave generiert für den IO1 Befehl die Antwort auf folgende Weise.

```

1  unsigned char response[8] = {0, 0, 0, 0, 0, 0, 0, 0};
2  IO1_OUT;
3  if(argument_2 == OFF)
4  {
5      IO1_OFF;
6      response[0] = 0x30;
7  }
8  else if(argument_2 == ON)
9  {
10     IO1_ON;
11     response[0] = 0x30;
12 }
13 rs485_write_frame(MASTER_MSB, MASTER_LSB, IO1, SET, NONE, response);

```

Code 37: Response C-Code des Slave für IO1

Das Slave-Gerät setzt IO1 als Ausgang und überprüft das zweite Argumentbyte auf dessen Inhalt und setzt den Ausgang je nachdem auf logisch 1 oder 0.

## 7.5 Mikrocontroller Slave-Gerät



Abbildung 63: Slave-Geräte mit Gehäuse



Abbildung 64: Slave-Geräte mit Gehäuse von vorne

### 7.5.1 Überblick

In der folgenden Abbildung sind alle wichtigen ICs, Ein- und Ausgänge und deren logische Verknüpfungen dargestellt.

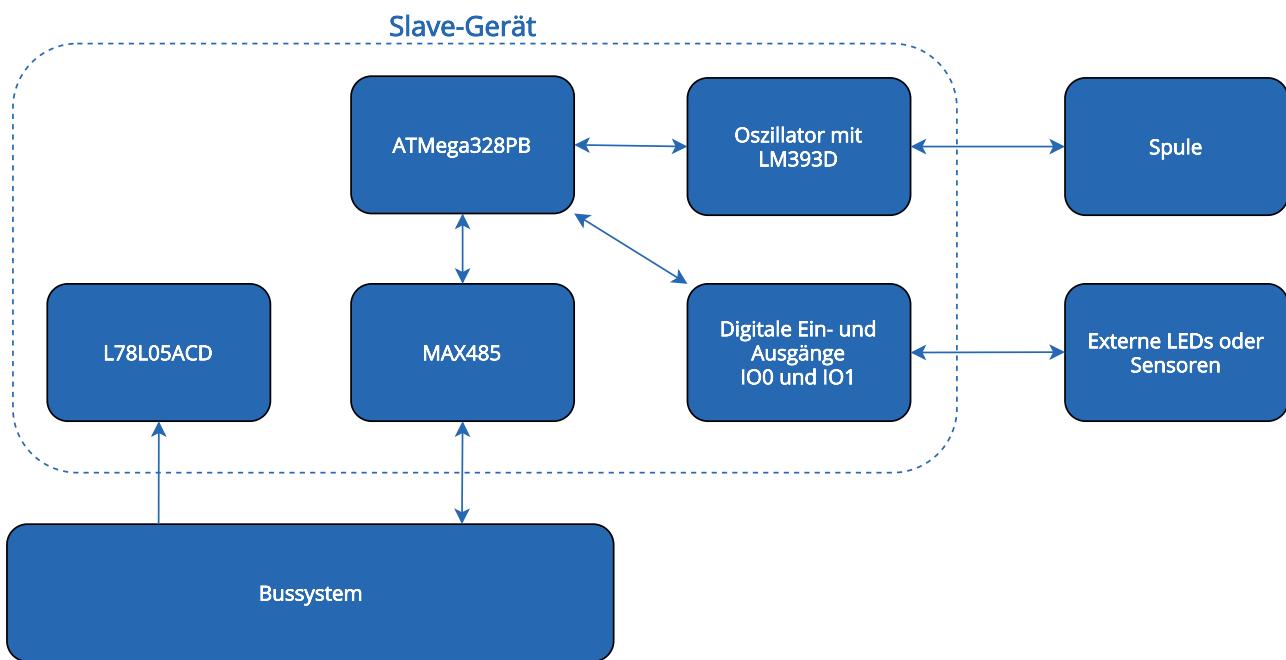


Abbildung 65: Logischer Aufbau des Slave-Gerät

- **Atmega328PB**

Ist der Mikrocontroller des Slave-Geräts. Er liest die Frequenz des Oszillators, kontrolliert die Ein- und Ausgänge IO0 und IO1 und ist für die serielle Kommunikation mit dem Bus verantwortlich.

- **L78L05ACD**

Ist ein Spannungsregler, der die erhöhte Spannung aus dem Bussystem absenkt und dem Slave zur Verfügung stellt.

- **MAX485**

Ist ein RS485 Pegelwandler, der die UART Signale des Mikrocontroller in die entsprechenden Pegel umwandelt

- **Oszillator**

An ihn wird die Spule angeschlossen. Er erzeugt mit einer Colpitts-Schaltung eine Resonanzfrequenz mit die vom Mikrocontroller ausgelesen wird.

- **IO0 und IO1**

Es handelt sich um digitale Eingänge und Ausgänge des Mikrocontroller, welche je nach

Anwendung unterschiedliche Funktionen haben können. An ihnen hängen zwei LEDs nämlich LED0 und LED1.

In den folgenden Absätzen wird genauer auf diese angeführten Blöcke eingegangen.

### 7.5.2 Mikrocontroller

Mikrocontroller (auch  $\mu$ C) sind Halbleiterchips die einen integrierten Prozessor, einen Arbeitsspeicher, einen Programmspeicher und eine Peripheriefunktionalität aufweisen. Viele dieser Mikrocontroller besitzen Schnittstellen wie SPI, IIC oder UART und können analoge Spannung einlesen sowie PWM Signale erzeugen. Ihre vielfältigen Eigenschaften machen diese ICs vielseitig einsetzbar. Sie werden oft in einfachen elektronischen Alltagsgegenständen wie Digitaluhren, Fernbedienungen oder Tastaturen gefunden.



Abbildung 66: Atmega328PU in DIP28

Für die Slave-Geräte ist ein Mikrocontroller sehr gut geeignet, da er keine komplexen Rechnungen durchführen muss. Er soll mit einer seriellen Schnittstelle arbeiten, Rechtecksignale im 40 kHz bis 200 kHz Bereich genau bestimmen und digitale Eingänge und Ausgänge aufweisen können. Auch Interrupts sind vorhanden, welche es möglich machen bei internen oder externen Ereignissen gewissen Programmteile auszuführen.

Der Atmega328PB ist ein Mikrocontroller der Microchip AVR Familie<sup>15</sup> und besitzt eine Maximale Taktfrequenz von 20 MHz und kann mit einer Betriebsspannung von 5 V arbeiten. Sein interner programmierbarer Flash-Speicher ist 32KB groß, was für diese Anwendung genügend ist.

### 7.5.3 Lineare Spannungsregler

Spannungsregler haben die Aufgabe eine Gleichspannung abzusenken und zu stabilisieren. Sie werden oft bei Schaltungen eingesetzt, bei denen entweder kleine Ströme fließen oder die Effizienz der Schaltung nicht im Vordergrund steht. Lineare Spannungsregler verwenden Leistungstransistoren um die Spannung abzusenken, was zu Verlustleistungen führt. Die Größe der Ausgangsspannung kann entweder über externe Beschaltung vorgegeben werden oder ist fix vom Hersteller vorgegeben. Spannungsregler stellen immer automatisch die richtige Ausgangsspannung ein selbst wenn die Eingangsspannung in einem vorgegebenen Bereich variiert.

Daher der Mikrocontroller eine Betriebsspannung von 5 V benötigt ist ein Spannungsregler mit dieser Ausgangsspannung erforderlich. Der L78L05ACD erfüllt diese Eigenschaften und kann eine Ausgangsspannung von 5 V bei einer Eingangsspannung von 7 V bis 7 V, bei einem maximalen Strom von 100 mA.

Symbol	Parameter		Value	Unit
$V_I$	DC Input voltage	$V_O = 3.3 \text{ to } 9 \text{ V}$	30	V
		$V_O = 12 \text{ to } 15 \text{ V}$	35	
		$V_O = 18 \text{ to } 24 \text{ V}$	40	
$I_O$	Output current		100	mA
$P_D$	Power dissipation		Internally limited <a href="#">(1)</a>	mW
$T_{STG}$	Storage temperature range		-65 to 150	°C
$T_{OP}$	Operating junction temperature range	for L78LxxAC / L78LxxC	0 to 125	°C
		for L78LxxAB	-40 to 125	

Abbildung 67: absolute Grenzwerte des L78L05ACD

<sup>15</sup><https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus>

### 7.5.4 RS485 Pegelwandler

Die Aufgabe des RS485-Pegelwandlers ist die Umsetzung der UART-Spannungspegel in RS485-Spannungspegel. Die Einzelheiten dieses Standards und die wichtigen Funktionen des MAX485CSA wurden im Absatz 7.4.1.4 beschrieben.

### 7.5.5 Oszillator mit LM393D Komparator

Der Oszillator dient dazu die Induktivität der Spule in eine Frequenz umzusetzen (Vergleiche Absatz 7.3.2.3). Dafür ist ein aktives Element nötig, das für eine Phasendrehung von  $180^\circ$  sorgen kann. Es kann dabei ein Operationsverstärker in Frage kommen aber auch ein einfacher Komparator ist hier ausreichend. Um das Sinussignal des Oszillators in ein Rechtecksignal für den Mikrocontroller umzuwandeln ist ein zusätzlicher Schmitttrigger nötig. Der Grund dafür ist eine Funktionalität des Mikrocontrollers. Er besitzt nämlich eine Interrupt Funktionalität, die auf Flanken von Rechtecksignalen reagiert. Über die Zählung der Flanken über einen Zeitraum lässt sich die Frequenz ermitteln. Das Schmitttrigger ist deshalb ausgewählt worden, weil das Eingangssignal einen gewissen Betrag überschreiten beziehungsweise unterschreiten muss bevor das Ausgangssignal sich ändert. Diesen Schwellwert ist bei einer Betriebsspannung von 5 V auf 250 mV festgelegt. Mit der folgenden Formel für den Schwellwert des Schmitttrigger lässt sich das Widerstandsverhältnis  $\frac{1}{10}$  errechnen.

$$U_{Schmitt} = U_B \cdot \frac{R_1}{R_2} \quad (15)$$

Wobei

$U_{Schmitt}$  = Schwellwert des Schmitttriggers

$I_B$  = Betriebsspannungsbereich

$R_1$  = Widerstand vom Eingang zum nicht invertierenden Eingang

$R_2$  = Widerstand im Rückkoppelzweig

Es wurden die Widerstandswerte  $R_1 = 10\text{k}\Omega$  und  $R_2 = 100\text{k}\Omega$  ausgewählt. Für einen allgemeinen Oszillator gilt folgendes Schaltbild wobei  $R_2$  der Widerstand vom Eingang zum nicht invertierenden Eingang und  $R_3$  der Widerstand im Rückkoppelzweig ist.

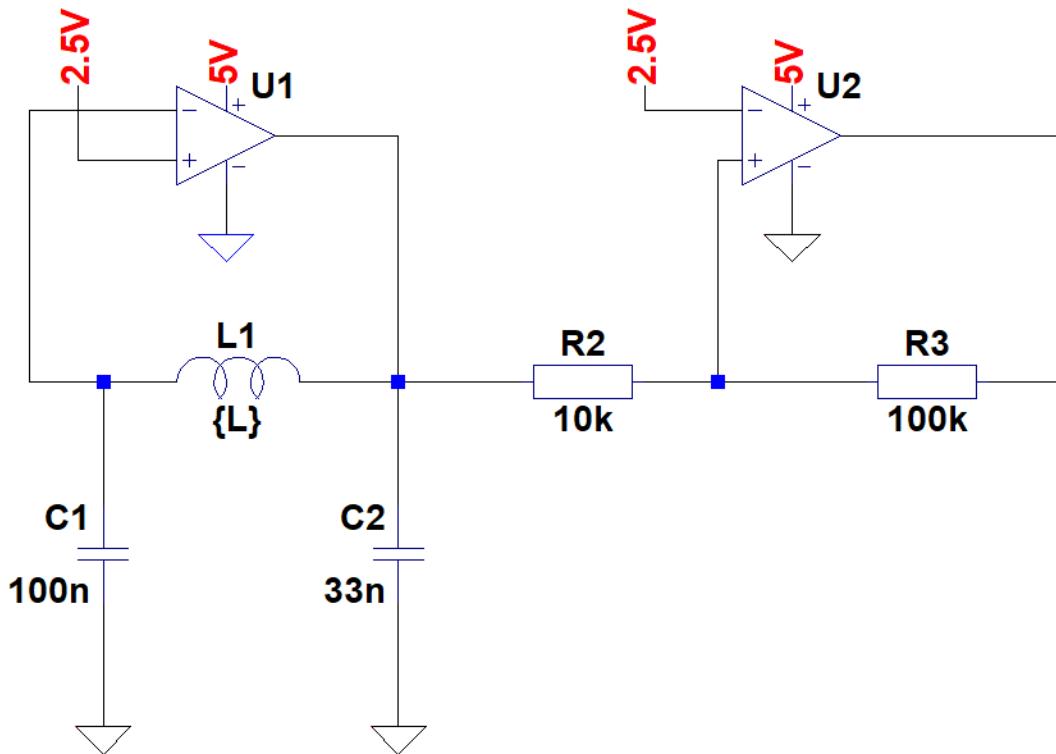


Abbildung 68: Colpitts Oszillatör mit Schmitttrigger

Als Komparator haben wir den LM393D IC ausgewählt, da er sehr günstig ist und weil er in SMD Form zwei Komparatoren in einem Package hat. Er ist jedoch ein sogenannter Open-Collector IC. Das heißt er kann seinen Ausgang nur auf Masse ziehen und er braucht daher einen Pullup-Widerstand auf eine positive Spannung um das gewünschte Rechtecksignal zu erzeugen. Diese Widerstände sind in der folgenden Abbildung mit *RCP1* und *RCP2* gekennzeichnet

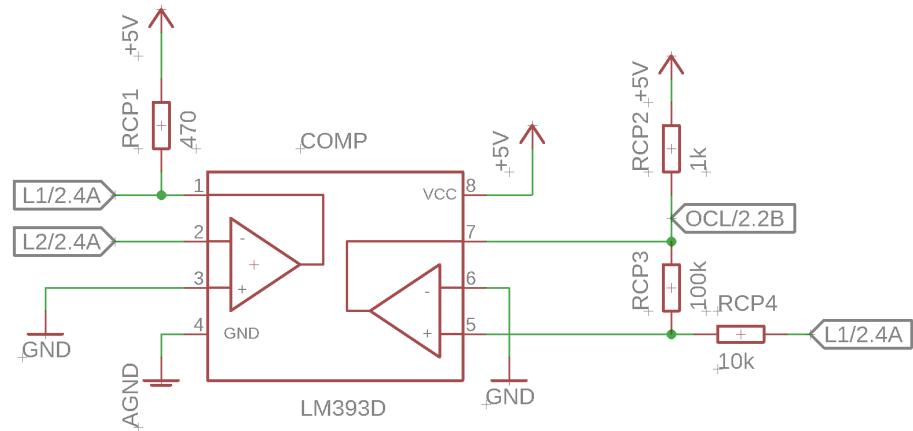


Abbildung 69: LM393D im Slave Schaltplan

Als Kondensatoren des Colpitts-Oszillators wurden die Werte 100 nF und 33 nF verwendet. Dies ist wichtig um die Resonanzfrequenz des Oszillators zu bestimmen. Für eine Spule mit einer Induktivität von 50 µH würde sich eine Frequenz nach der Gleichung 14 von 148,165 kHz ergeben.

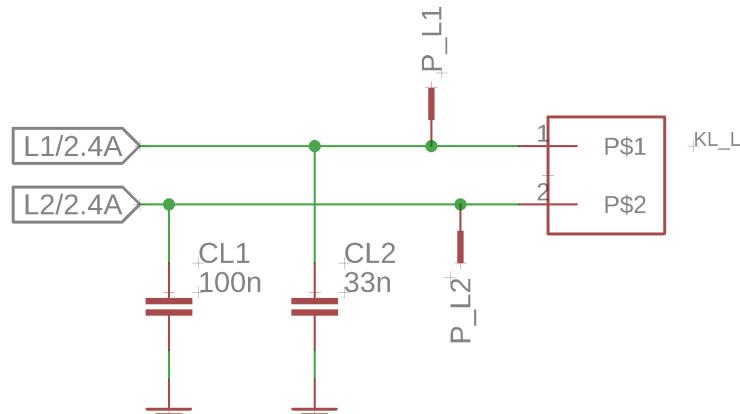


Abbildung 70: Colpitts Kondensatoren im Slave Schaltplan

Die Spule wird an den Anschluss der mit 'L' beschriftet ist angeschlossen (Siehe Abbildung 64)

### 7.5.6 Digitale Ein- und Ausgänge IO0 /IO1

Sie haben generell verschiedene Anwendungsmöglichkeiten und sind daher je nach Kundenwunsch anpassbar. Derzeit werden sie als visuelle Indikatoren verwendet um direkt am Parkplatz anzuzeigen, ob er besetzt ist. Auf der Platine sind zwei LEDs verbaut, die immer den Zustand des IO-Pins

anzeigen. Für den Parkplatz ist jedoch nötig die LEDs nach außen zu leiten. Dafür gibt es die Anschlüsse '0' und '1' die für IO0 und IO1 stehen sowie die Anschlüsse '+' und '-' die für 5 V und Masse stehen (Siehe Abbildung 64). Die Spannungsausgänge sind auch nützlich für externe Sensoren und Aktoren die nicht mehr wie 100 mA brauchen.

### 7.5.7 weitere Anschlüsse

#### 7.5.7.1 Separater VCC Eingang

In der Entwicklungsphase hat es sich praktisch erwiesen einen eigenen Spannungsanschluss am Gerät anzubringen. Dieser ist an den Spannungsregler angeschlossen und muss mit 9 V bis 30 V vorsorgt werden. Für den normalen Betrieb ist jedoch keinen direkte Versorgung des Slaves mit Spannung nötig, da er seine Spannung über den RJ45 Stecker bezieht.



Abbildung 71: Slave-Gerät mit Gehäuse von links

#### 7.5.7.2 RJ45 Anschlüsse

Wie im Absatz 7.4.2.1 erklärt müssen die Slaves in einer Kette verbunden werden um sie eindeutig zu identifizieren. Aus diesem Grund gibt es zwei RJ45-Stecker die die Slaves miteinander verbinden (Siehe Abbildung 54).

#### 7.5.7.3 SPI Programmieranschluss

Um den Mikrocontroller zu Programmieren braucht es einen Anschluss für einen externen Programmer, der den Maschinencode auf den hochlädt. Der Atmega328PB lässt eben über seine SPI Anschlüsse programmieren welche im folgenden Bild an den in rot gekennzeichneten Stecker geführt werden.

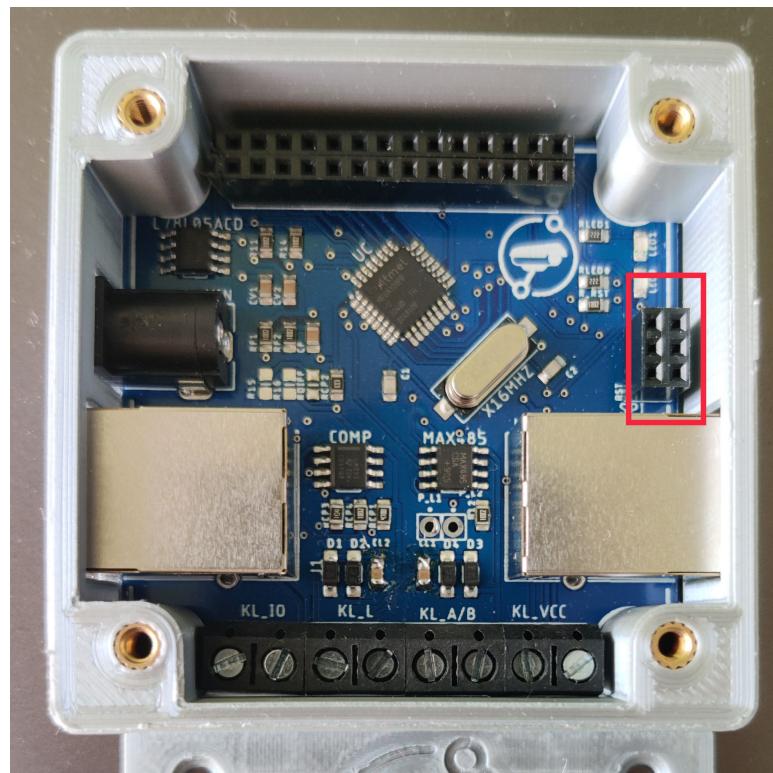


Abbildung 72: SPI Anschluss

In gleicher Orientation wie in Abbildung 72 gilt die folgende Netzbeschriftung.

MISO	+5V
SCK	MOSI
RST	GND

Tabelle 5: Netzbelegung des SPI Anschluss

### 7.5.8 Atmega328PB

#### 7.5.8.1 Zuweisung der Pins

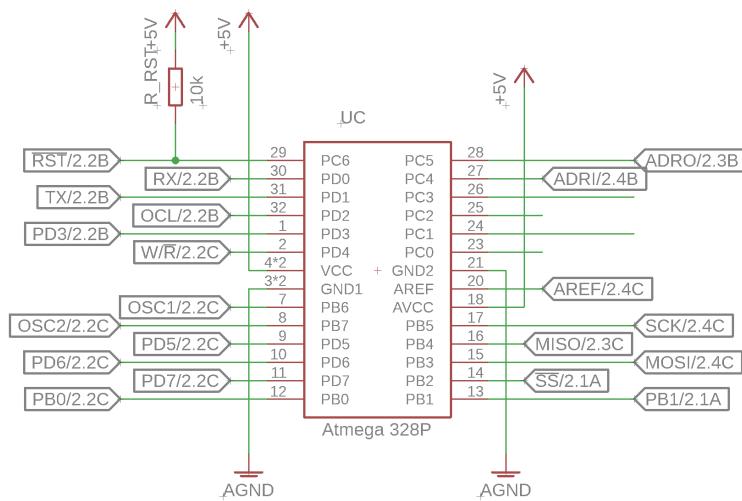


Abbildung 73: Pin-Netz Zuordnung des Atmega328PB im Slave Schaltplan

#### 7.5.8.2 Fuse-Bits und Quartz

Unter Fuse-Bits versteht man Bits die nicht durch die Software des Mikrocontroller verändert werden können und zur Konfiguration des Mikrocontrollers dienen. Sie können nur durch einen externen Programmer verändert werden, wenn sie nicht schon vom Hersteller gesperrt worden sind. Es gibt insgesamt drei Fuse-Bytes, die für die Einstellungen des Atmega328PB zuständig sind. Sie regeln den internen Taktvorteiler, geben die Größe des Bootsektors und regeln neben anderen Sachen die Programmierung über SPI.

Fuse-Byte	Low Fuse	Extended Fuse	High Fuse
Hex	0xFF	0xFD	0xDA
Bin	1111 1111	1111 1101	1101 1010

Tabelle 6: Konfiguration der Fuse Bits

**Table 32-7. Fuse Low Byte**

Low Fuse Byte	Bit No.	Description	Default Value
CKDIV8 <sup>(4)</sup>	7	Divide clock by 8	0 (programmed)
CKOUT <sup>(3)</sup>	6	Clock output	1 (unprogrammed)
SUT1	5	Select start-up time	1 (unprogrammed) <sup>(1)</sup>
SUT0	4	Select start-up time	0 (programmed) <sup>(1)</sup>
CKSEL3	3	Select Clock source	0 (programmed) <sup>(2)</sup>
CKSEL2	2	Select Clock source	0 (programmed) <sup>(2)</sup>
CKSEL1	1	Select Clock source	1 (unprogrammed) <sup>(2)</sup>
CKSEL0	0	Select Clock source	0 (programmed) <sup>(2)</sup>

Abbildung 74: Low Fuse-Byte

Für das Low-Byte gibt die Takteinstellungén vor. Der Slave ist auf einen externen 16 MHz Quartz angewiesen und er soll auch mit dieser Frequenz arbeiten. Der Taktteiler ist daher 1 welcher mit den Bit CKDIV = 1 eingestellt wurde. Die Taktquelle ist mit den Bits CKSEL0-3 auf einen externen Quartz umgestellt worden.

**Table 11-1. Device Clocking Options Select**

Device Clocking Option	CKSEL[3:0]
Low-Power Crystal Oscillator	1111 - 1000
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128 kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

**Note:** For all fuses, '1' means unprogrammed while '0' means programmed.

Abbildung 75: Einstelleoptionen für den internen Takt des Atmega328PB

Wenn man weiter im Datenblatt liest erhält man die Werte für die Trimmkondensatoren nämlich jeweils 22 nF.

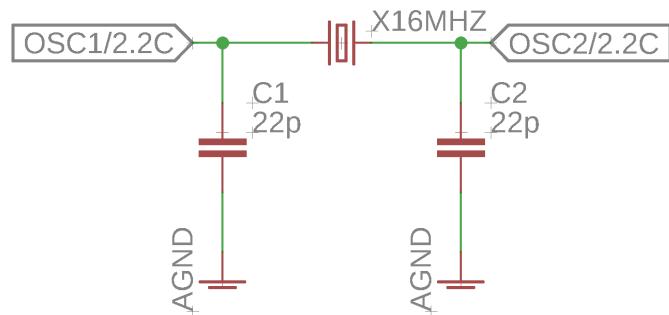


Abbildung 76: Quartz im Slave Schaltplan

**Table 32-6. Fuse High Byte**

High Fuse Byte	Bit No.	Description	Default Value
RSTDISBL <sup>(1)</sup>	7	External Reset Disable	1 (unprogrammed)
DWEN	6	debugWIRE Enable	1 (unprogrammed)
SPIEN <sup>(2)</sup>	5	Enable Serial Program and Data Downloading	0 (programmed, SPI programming enabled)
WDTON <sup>(3)</sup>	4	Watchdog Timer Always On	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed), EEPROM not preserved
BOOTSZ1	2	Select Boot Size (see Boot Loader Parameters)	0 (programmed) <sup>(4)</sup>
BOOTSZ0	1	Select Boot Size (see Boot Loader Parameters)	0 (programmed) <sup>(4)</sup>
BOOTRST	0	Select Reset Vector	1 (unprogrammed)

**Note:**

1. Refer to *Alternate Functions of Port C* in I/O-Ports chapter for description of RSTDISBL Fuse.
2. The SPIEN Fuse is not accessible in serial programming mode.
3. Refer to *WDTCSR – Watchdog Timer Control Register* for details.
4. The default value of BOOTSZ[1:0] results in maximum Boot Size. See table Boot Size Configuration in subsection Boot Loader Parameters in the previous chapter for details.

Abbildung 77: High Fuse-Byte

Hier ist zu beachten, dass das Bit SPIEN auf 0 gesetzt sein muss damit ein externer SPI-Programmer verwendet werden kann. Wird dieses Bit auf 1 gesetzt kann das nur schwer rückgängig gemacht werden.

**Table 32-5. Extended Fuse Byte for ATmega328PB**

Extended Fuse Byte	Bit No.	Description	Default Value
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
CFD	3	Disable Clock Failure Detection	0 (programmed, CFD disable)
BODLEVEL2 <sup>(1)</sup>	2	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL1 <sup>(1)</sup>	1	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL0 <sup>(1)</sup>	0	Brown-out Detector trigger level	1 (unprogrammed)

Abbildung 78: Extended Fuse-Byte

Mit diesem Byte wird der Brownout Reset Spannungspegel festgelegt. Wenn die Versorgungsspannung des Atmega328PB unter diesen Wert fällt bricht der Mikrocontroller seinen Operation aus und setzt sich zurück.

#### 7.5.8.3 SPI-Programmierung

Es gibt viele Möglichkeiten den einen Mikrocontroller mit unterschiedlichen Werkzeugen zu programmieren. Einen Variante ist die mit den unten Verwendeten Werkzeugen.

- **AVRDUEDE**

AVRDUEDE<sup>16</sup> ist eine Programmiersoftware für Mikrocontroller aus der Microchip-AVR Familie. Sie hat keine Benutzeroberfläche sondern hat nur eine Terminalausgabe.

- **PlatformIO**

PlatformIO ist Erweiterung für Visual Studio Code, die es ermöglicht Mikrokontroller Projekte zu entwickeln. Sie bietet auch die Möglichkeit an bereits existierende Compiler oder Programmiersoftware wie AVRDUEDE zu verwenden.

- **Arduino UNO als Programmer**

Der Arduino Uno ist ein Mikrocontroller basiertes Board, welches weit verbreitet ist und für das es große Softwareunterstützung gibt. Eine mögliche Anwendung dieses Boards ist die eines

<sup>16</sup><https://www.mikrocontroller.net/articles/AVRDUEDE>

SPI-Programmer. Dazu muss aber zuerst die richtige Software auf den Arduino hochgeladen werden. Dafür braucht man die zum Arduino zugehörige Arduino Entwicklungsumgebung.

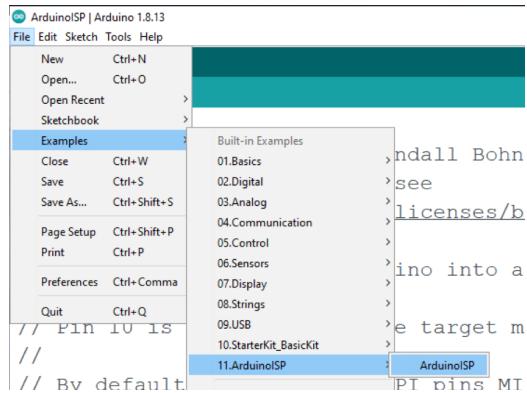


Abbildung 79: ArduinoISP Sketch auswählen

Der ArduinoISP Sketch macht den Arduino UNO zu einem AVR Programmer. Dieser Sketch muss dann kompiliert und hochgeladen werden. Dafür sind die in der folgenden Abbildung Einstellungen zu beachten. Der COM-Anschluss kann bei anderen Computern anders sein und sollte immer selbst bestimmt werden.

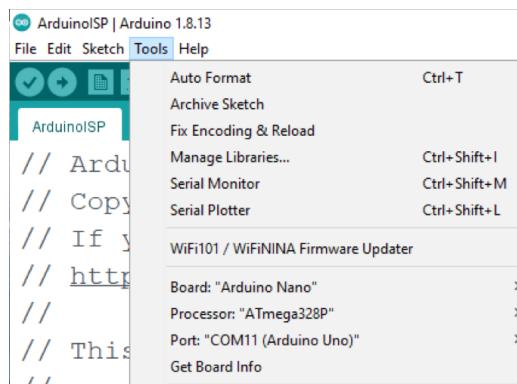


Abbildung 80: Arduino Upload Einstellungen für ArduinoISP Sketch

Wenn der Arduino als SPI-Programmer kann er wir im unteren Bild an den Slave angeschlossen werden.

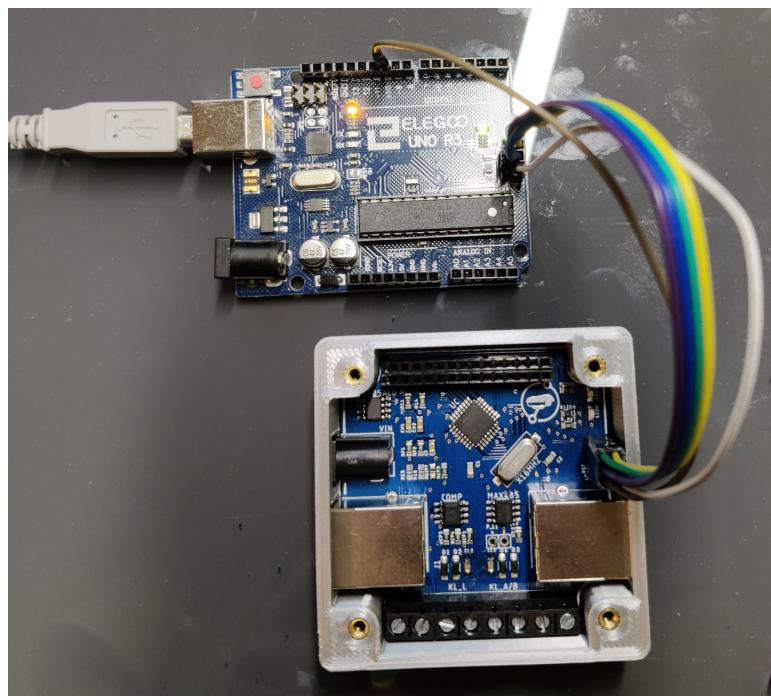


Abbildung 81: Anschluss eines externen SPI Programmer

Um die Fuse-Bits zu setzen ist es nötig AVRDUDE einmal mit den folgenden Parametern zu starten.  
Es muss wieder der zuerst der COM-Anschluss des Arduino am Computer ermittelt werden.

```
1 -U efuse:w:0xfd:m -U lfuse:w:0xff:m -U hfuse:w:0xda:m -e -v -p m328pb -c
  ↳ arduino -PCOM11 -b19200 -D -C"C:\Program Files
  ↳ (x86)\Arduino\hardware\tools\avr\etc\avrdude.conf"
```

Code 38: AVRDUDE Argumente für die Fuse-Bits

Damit man in Visual Studio Code über PlatformIO kompilieren und hochladen kann sind die folgenden Einstellung in der platformio.ini Datei zu treffen.

```
1 [env:ATmega328PB]
2
3 platform = atmelavr
4
5 board = ATmega328PB
6
7 framework = arduino
8
9 upload_port = COM11
10
11 upload_protocol = stk500v1
12
13 upload_speed = 19200
14
15 board_build.F_cpu = 16000000L
16
17 board_fuses.lfuse = 0xFF
18
19 board_fuses.hfuse = 0xDA
20
21 board_fuses.efuse = 0xFD
22
23 upload_flags =
24
25 -PCOM11
26
27 -b$UPLOAD_SPEED
28
29 -e
```

Code 39: PlatformIO Einstellungen

### 7.5.9 Software des Slaves

#### 7.5.9.1 Konfiguration der Register

Die Funktionen der Registerwerte wurden in den Code-Komentaren hinerlegt.

```

1 void init()
2 {
3     DDRC &= ~(1 << DDC4) & ~(1 << DDC5);      //ADRO und ADRI als Eingang
4     DDRB |= (1 << PORTB1) | (1 << PORTB2);    //I00 und I01 als Ausgang
5
6     PORTB &= ~(1<< PORTB2) & ~(1 << PORTB1);  //I00 und I01 auf logisch 0
7     ↳ legen.
8     PORTC |= (1 << PORTC4) | (1 << PORTC5);   //Pullupwiderstände von
9     ↳ ADRO und ADRI einschalten
10
11    rs485_init(56700, P_NONE, S_1, F_CPU);
12    reset();
13 }
```

Code 40: allgemeine Registereinstellungen

```

1 TCCR1A = 0;                                // OC1A
2   ↳ Abschalten
3 TCCR1B |= (1 << CS10) | (1 << CS11) | (1 << WGM12); // f-count =
4   ↳ 250kHz, CTC
5 OCR1A = 25000;                             // f-ISR = 10Hz
6 TIMSK1 |= (1 << OCIE1A);                  // Compare A
7   ↳ interrupt
```

Code 41: Timer Registereinstellungen

```

1  uint16_t ubrr = ((internal_clk/8UL/baudrate) - 1);
   ↳ //UBRR0L berechnen
2  parity = ((parity & 0x02) << UPM01) | ((parity & 0x01) << UPM00);
   ↳ //Parität für UCSR0C berücksichtigen
3  stopbit = ((stopbit & 0x01) << USBS0);
   ↳ //Stopbit für UCSR0C berücksichtigen
4  UBRR0L = (char)ubrr;
5  UBRR0H = (ubrr >> 8) & 0xFF;
6  UCSR0A |= (1 << U2X0);
7  UCSR0C |= (0 << UMSEL01) | (0 << UMSEL00) | parity | stopbit | (1 <<
   ↳ UCSZ01) | (1 << UCSZ00);           //Asynchron, Parität, Stopbits, 8
   ↳ Datenbits
8  UCSR0B |= (1 << RXCIE0) | (1 << RXEN0) | (1 << TXEN0);
   ↳ //Empfangsinterrupt, Empfang und Übertragung einschalten.

```

Code 42: UART Registereinstellungen

### 7.5.9.2 UART Betrieb

Daher der Atmega328PB nur ein UART-Buffergröße mir einer Größe von einem Byte besitzt. Muss zusätzlich per Software ein Ringbuffer mit Lesepointer pr und einem Schreibpointer pw erstellt werden. Die Struktur des Buffers sieht so wie folgend aus.

	BUF_SIZE-1	rs485_buf
	...	...
pw	3	Byte 3
	2	Byte 2
pr	1	Byte 1
	0	Byte 0

Abbildung 82: Slave UART-Buffer Struktur

Wenn ein neues Byte auf den Bus geschrieben werden lesen alle Slaves diese mit dem UART-Empfangsinterrupt in den Buffer an der stelle des Schreibpointers. Dieser wird danach um eins erhöht.

```
1 ISR(USART0_RX_vect)
2 {
3     rs485_buf[pw] = UDR0;
4     pw = (pw + 1) % BUF_SIZE;
5 }
```

Code 43: Speichern eines Bytes auf den Buffer

### 7.5.9.3 Auswertung von Datenframes

Um die erhaltenen Datenframes überhaupt auslesen zu können müssen sie erst im Buffer gefunden werden. Die Bytes START, STOP und LF helfen dabei den Anfang und das Ende des Datenframes zu finden. Zudem ist bekannt das ein Datenframe genau 16 Byte groß ist. Wenn der Lesepointer Schritt für Schritt um 1 erhöht wird kann der Buffer solang durchgegangen werden bis ein Datenframe gefunden wird. Diese Logik wird im folgenden Flussdiagramm verdeutlicht.

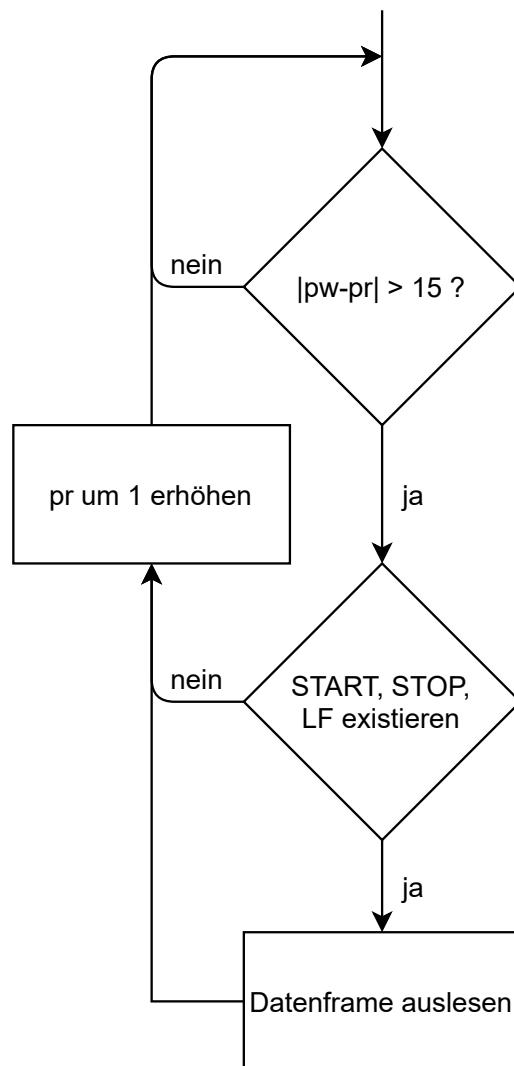


Abbildung 83: Flussdiagramm der Datenframeerkennung

Die ausgelesenen Daten werden in globale Variablen geschrieben zusammen mit einem Flag, dass bekannt gibt dass ein neuer Dateframe angekommen ist. In Folge wird in der Hauptschleife überprüft ob die Adresse mit der eigen übereinstimmt. Wenn dies der Fall ist wird der Steuerbefehl ausgeführt. Unten ist ein Beispielcode der bei der Änderung der eigenen Adresse ausgeführt wird.

```

1  else if(ADR_LSB == LOCAL_ADR_LSB && ADR_MSB == LOCAL_ADR_MSB)
2  {
3      if(control_byte == ADR)
4      {
5          if(argument_1 == GIVE)
6          {
7              LOCAL_ADR_MSB = DATA[0];
8              LOCAL_ADR_LSB = DATA[1];
9              unsigned char response[8] = {LOCAL_ADR_MSB, LOCAL_ADR_LSB, 0, 0,
10                             ↪ 0, 0, 0, 0};
11             rs485_write_frame(MASTER_MSB, MASTER_LSB, ADR, NONE,
12                             ↪ NONE, response);
13         }
14     }

```

Code 44: Auswertung der Daten eines Datenframes

#### 7.5.9.4 Taktmessung

Der Master gibt bei der Taktmessung die Größe des Messzeitraumes an in 10 ms Auflösung an. Die Aufgabe des Slaves ist es dann über diesen Zeitraum die Takte des Rechtecksignals vom Colpitts-Oszillator zu messen. Dafür werden sowohl Timer als auch Eingangsflankeninterrupts benötigt. Wenn der Slave die Anweisung zur Messung bekommt wird der folgenden Programmteil aktiv.

```
1  else if(controll_byte == GET_FRQ)
2  {
3      measure_ms = argument_1;
4      count_ticks = 0;
5      count_ms_ticks = 0;
6      MEASURE_ON;
7      while(INT0_EN) {}
8
9      unsigned char buf[] = "00000000";
10     sprintf(buf, "%08lx", (unsigned long) count_ticks);
11     rs485_write_frame(MASTER_MSB, MASTER_LSB, GET_FRQ, argument_1,
12                         → NONE, buf);
13 }
```

Code 45: Slave C-Code zur Taktmessung

Zuerst werden alle globalen Zählvariablen auf 0 zurückgesetzt und die Timer und Flankeninterrupts eingeschalten. Das Programm wartet dann solange bis die Messung abgeschlossen ist und sendet die Anzahl der Takte in ASCII kodiert an den Master zurück.

```

1  else if(controll_byte == GET_FRQ)
2  {
3      measure_ms = argument_1;
4      count_ticks = 0;
5      count_ms_ticks = 0;
6      MEASURE_ON;
7      while(INT0_EN) {}
8
9      unsigned char buf[] = "00000000";
10     sprintf(buf, "%08lx", (unsigned long) count_ticks);
11     rs485_write_frame(MASTER_MSB, MASTER_LSB, GET_FRQ, argument_1,
12                         → NONE, buf);
13 }
```

Code 46: Slave C-Code zur Taktmessung

Die Zählung der Takte Funktioniert wie folgt.

```

1  ISR(INT0_vect)
2  {
3      count_ticks++;
4 }
```

Code 47: Slave C-Code zur Taktmessung

In der Flankeninterrupt Routine werden die Takte gezählt und in eine globale Variable geschrieben.

```
1 ISR(TIMER1_COMPA_vect)
2 {
3     count_ms_ticks++;
4     if(count_ms_ticks >= measure_ms)
5     {
6         MEASURE_OFF;
7         count_ms_ticks = 0;
8     }
9 }
```

Code 48: Slave C-Code zur Taktmessung

In der Timerinterrupt Routine werden die Anzahl der Interrupts gezählt bis die gewünschte Messzeit abgelaufen ist. In Folge werden die entsprechenden Interrupts wieder ausgschalten.

### 7.5.10 Layout des Slave-Gerätes

Das Layout des Slaves wurde in Eagle<sup>17</sup> erstellt. Während der Entwicklungsphase gab es mehrere Versionen des Slaves. In der ersten war das PCB nicht sehr platzeffizient und benutzte für den Mikrocontroller keine SMD Variante. Unten ist ein Bild des ersten Prototyps mit bestückten Bauteilen und Gehäuse.

<sup>17</sup><https://www.autodesk.com/products/eagle/>



Abbildung 84: erster Slave Prototyp

Um einzelne Fehler auszubessern und um weitere Eigenschaften anzubringen wurde im späteren Verlauf eine weitere Version des Slaves erstellt. Diese Version ist im Vergleich zur letzten kleiner und verwendet durchgehend SMD Varianten der ICs. In der unteren Abbildung sehen wir rechts den alten Prototyp und links die neueste Version des Slaves.

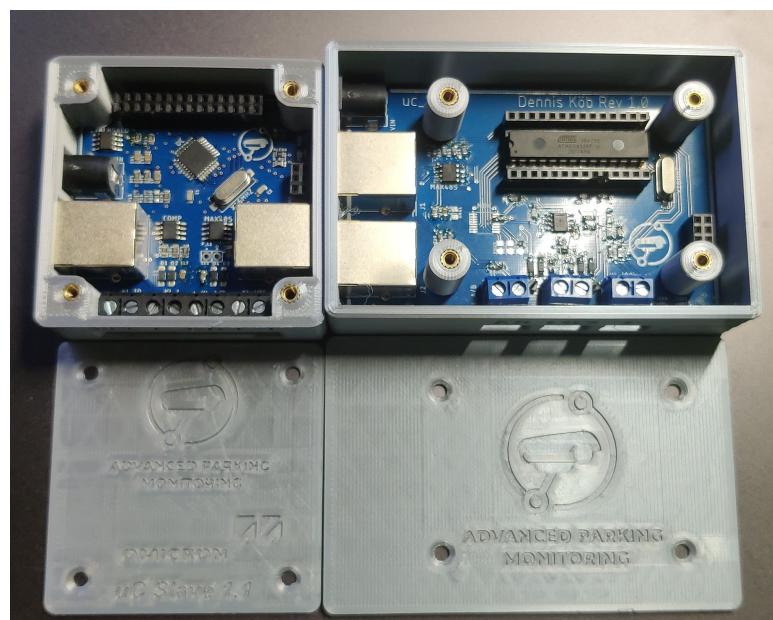


Abbildung 85: beide Slave Prototypen

### 7.5.11 Gehäuse

Das Gehäuse wurde in Fusion360 16 erstellt und die dazu das Slave-Gerät besser vor Umwelteinflüssen zu schützen ohne dabei die Funktionalität des Gerätes einzuschränken. Es existieren insgesamt drei Teile, die zusammenverschraubt das Gehäuse bilden. Das Material, welches verwendet wurde ist ein 3D-druckbares PLA Polyester. Für die Gewinde wurden spezielle Gewindegänge eingesetzt mit einem metrischen Nenndurchmesser von 3 mm. Die verwendeten Schrauben entsprechen der DIN-Norm 965.

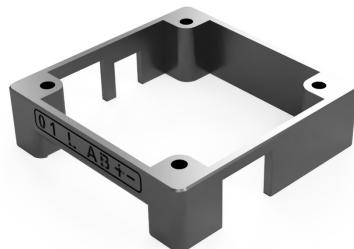


Abbildung 86: Boden des Slave Gehäuses   Abbildung 87: Wände des Slave Gehäuses   Abbildung 88: Deckel des Slave Gehäuses

## 7.6 USB-Master

Die Aufgabe des Masters ist es die Kommunikation am Bus zu steuern, ihn mit Spannung zu versorgen und die Daten an das Webinterface weiterzugeben. In der folgenden Abbildung ist der logische Aufbau des Masters zu sehen.

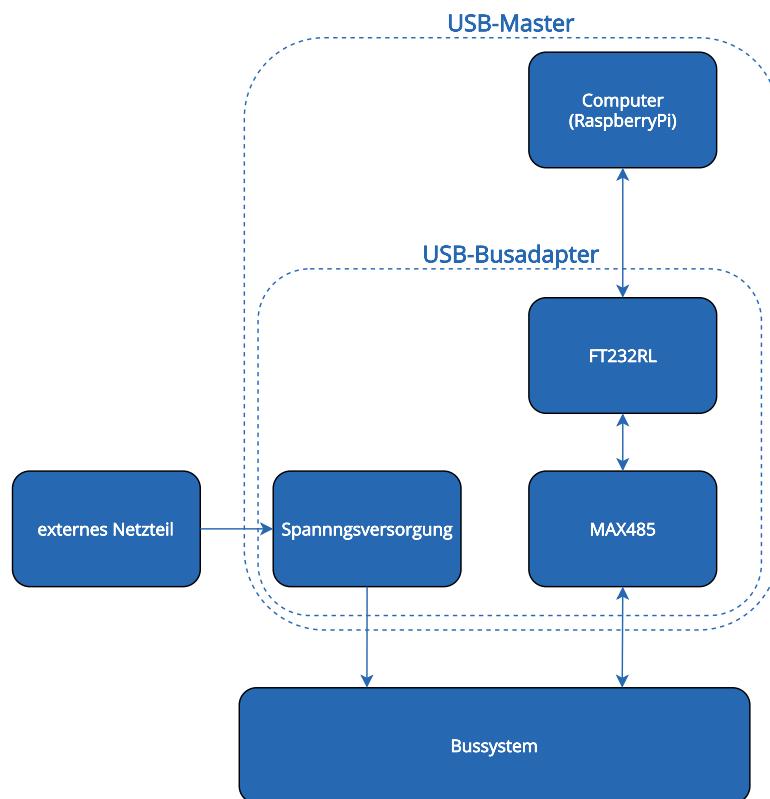


Abbildung 89: Logischer Aufbau des Masters

## 7.6.1 USB-Bussadapter Gerät

### 7.6.1.1 FT232RL

Der FT232RL Ic ist ein USB auf UART Wandler IC, der keine programmieren des internen Speichers nötig hat. Er kann ohne weiteres über einen USB-Stecker an einen Computer angeschlossen werden. Neben den für USB relevanten Anschlüssen besitzt der Chip zusätzliche Pins um mehrere Funktionen zu erfüllen. In seinem Datenblatt ist eine Beispielschaltung vorgegeben für die Verwendung eines RS485-Pegelwandlers.

## 7.2 USB to RS485 Converter

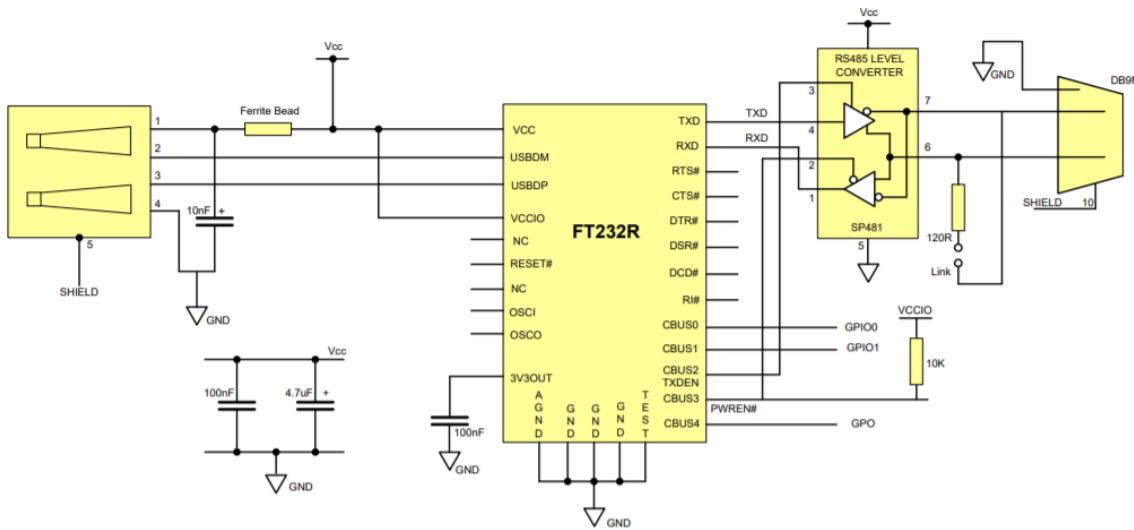


Abbildung 90: Beispielschaltung eines FT232 mit einem RS485 Pegelwandlers

In der folgenden Abbildung ist der FT232 im Schaltplan des Busadapters dargestellt. Ist sei zu beachten, dass die LEDs falsch angeschlossen sind. Die Anschlüsse CBUS0 und CBUS1 funktionieren als Stromsenken. Der Fehler wurde mittels Drahtbrücken und Umplatzierung der Bauteile korrigiert.

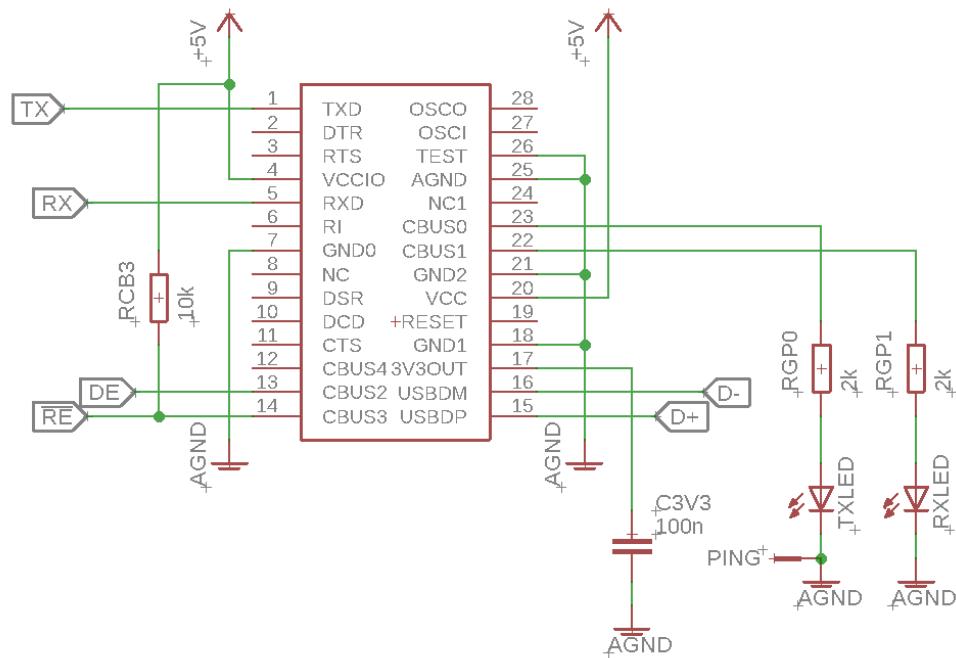


Abbildung 91: FT232RL im Schaltplan

### 7.6.1.2 USB-C Anschluss

Als USB-Stecker ist der Typ C aufgrund seiner besonderen Eigenschaften ausgewählt worden. Er ist geometrisch sehr klein und er ist symmetrisch seiner Drehachse entlang des Kabels. Im Vergleich zum Typ A oder Typ B Stecker kann man ihn nicht verkehrt einstecken. Augrund dieser Symmetrie besitzt er alle Anschlüsse doppelt. Die Anschlüsse im Schalplan sehen wie folgt aus.

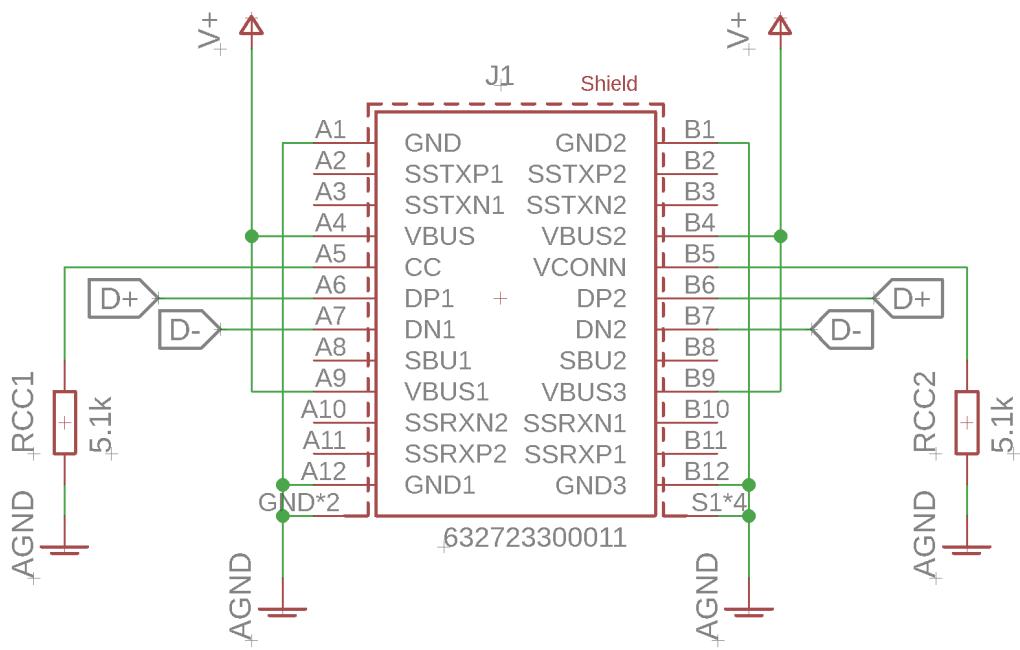


Abbildung 92: USB Typ C Stecker im Master Schaltplan

Es werden nur die normalen D+ und D- Pins verwendet. Damit das USB-Gerät am Anderen Ende weis, dass das Gerät für die serielle Kommunikation zur Verfügung steht sind zwei  $5\text{ k}\Omega$  an den Pins VCONN und CC nötig.

#### 7.6.1.3 Layout des Master-Geräts

Genau gleich wie das Layout des Slaves wurde dieses in Eagle erstellt. Es gibt den zuvor erwähnten USB Typ C Stecker, einen RJ45 Anschluss für das Bussystem und einen Anschluss für eine Netzteil mit 9 V bis 30 V.

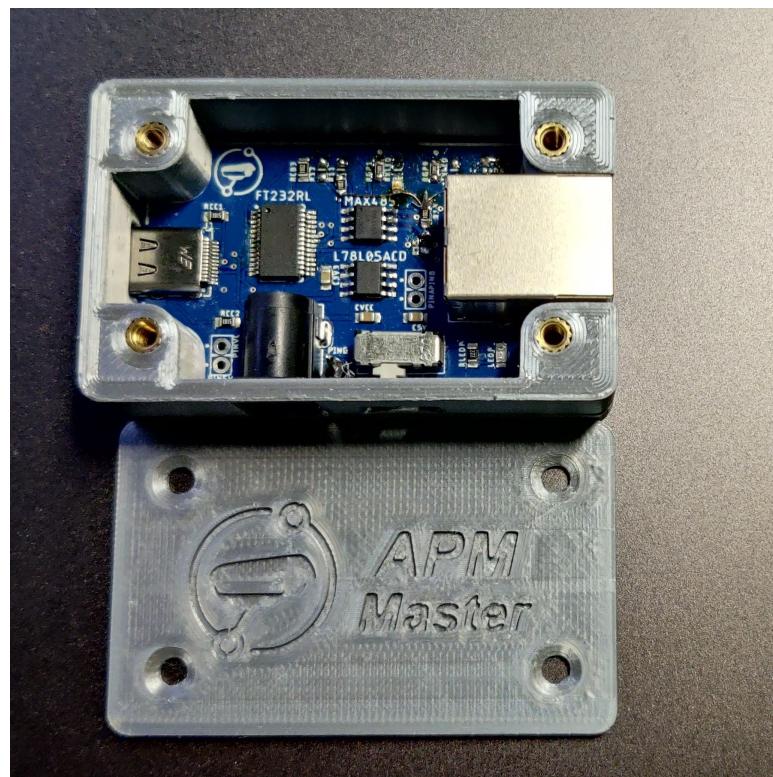


Abbildung 93: USB-Bussadapet mit Gehäuse

**7.6.1.4 Gehäuse** Das Gehäuse ist wie das Slave-Gehäuse in Fusion360 gezeichnet worden und mit einem 3D-Drucker ausgedruckt worden. Es existieren insgesamt 3 Teile, die zusammen das Gehäuse bilden. Die verwendeten Gewinde sind die gleichen Gewindestütze wie bei Slave und es wurden die gleichen Schrauben nach DIN-Norm 965 verwendet.



Abbildung 94: Boden des Ma-  
ster Gehäuses



Abbildung 95: Wände des Ma-  
ster Gehäuses



Abbildung 96: Deckel des Ma-  
ster Gehäuses

## 7.6.2 Master Programm

### 7.6.2.1 Erkennung des USB-Geräts

Um den Bus ansteuern zu können ist es zuerst nötig den Busadapter zu finden. Alle registrierten Geräte die an einem COM-Anschluss hängen können eindeutig durch ihre VID und PID identifiziert werden. Es ist jedoch auch möglich mithilfe des Herstellers des Gerätes den richtigen COM-Anschluss zu finden. Der untere Code geht im Master Programm alle verfügbaren Geräte durch bis es eines mit dem richtigen Hersteller findet. In folge wird dann der entsprechende COM-Anschluss zurückgegeben.

```
1 def get_port_vidpid(vid_pid):
2     vp = vid_pid.split(':')
3     pid = vp[-1]
4     vid = vp[0]
5     for device in devices:
6         if device.vid != None or device.pid != None:
7             if format(device.vid, 'x') == vid and format(device.pid, 'x') ==
8                 pid:
9                 return device.device
10                break
```

Code 49: Master Code zur Erkennung des USB-Bussadapters

### 7.6.2.2 Frequenzauslesung

Für die Frequenzauslesung muss der Master die Form des entsprechenden Datenframes beachten (siehe Abbildung ??). Er muss zusätzlich noch eine MEsszeit für den Slave angeben. Der Slave gibt jedoch nicht die gemessene Frequenz zurück, sondern die Anzahl der Takte über dem vorgegebenen Zeitraum. Das folgende Unterprogramm gibt bei erfolgreicher Messung die gemessene Frequenz zurück und gibt bei einem Fehler einen Text in die Kommandozeile.

```
1 def get_frequency(adress:int, measure_time):  
2     temp = ser.timeout  
3     ser.timeout = measure_time * (2 /10)  
4     write_frame(adress = adress, controll = CTRL.GET_FRQ.value,  
5                  ↴ argument_1 = measure_time)  
6     (s, data, error_sent, error_received) = read_frame(True)  
7     if error_received == ERROR.NONE:  
8         ser.timeout = temp  
9     try:  
10         return int(data,16) / (measure_time / 10)  
11     except:  
12         print(s)  
13     print(s)  
14     return -1
```

Code 50: Frequenzsteuerbefehl des Masters

### 7.6.2.3 Auswertung

Die Auswertung der Frequenz kann auf unterschiedliche Weise erfolgen. Die einfachste Variante ist es eine Detektion festzustellen, wenn die gemessene Frequenz von der eines leeren Parkplatzes abweicht. Dies setzt voraus, dass die Frequenz bei leerer Parklücke bekannt ist. Der untere Code detektiert eine prozentuelle Frequenzänderung und gibt je nach dem eine Ausgabe an die Kommandozeile und einen Post an die API.

```
1 def update():
2     global timer_iterations
3     print(timer_iterations)
4     for adress in f_dict:
5         frq = Serial.get_frequency(adress, 60)
6         print(frq)
7         if frq * df_relative > f_dict[adress]:
8             print(post(adress, 1))
9             print("detected\n")
10        else:
11            print(post(adress, 0))
12            print("not detected\n")
13        timer_iterations += 1
14    threading.Timer(1.0, update).start()
```

Code 51: Detektionscode des Masters

#### 7.6.2.4 API-Post

Für den Post an die API ist ein API-Token nötig. Bei Detektion eines Fahrzeuges wird die Parklücke mit der entsprechenden Adresse als belegt gekennzeichnet. Der untere Code übernimmt die Adresse der Parklücke und den Detektionszustand und übergibt sie der API. Eine Verbindung zum Internet ist jedoch nötig.

```

1 def post(adress: int, occupied: int) -> str:
2     url = "http://dev.philipp-kraft.com/api/v1/lots/3/spots"
3     payload = {'address': str(adress), 'occupied': str(occupied)}
4     files = []
5     headers = { 'Authorization': 'TOKEN' }
6     response = requests.request("POST", url, headers=headers,
7         ↳ data=payload, files=files)
8
9     return response.text

```

Code 52: API Post Code des Masters

### 7.6.3 RaspberryPi als Mastergerät

#### 7.6.3.1 SSH Remote Zugriff

Um den RaspberryPi besser zu Konfigurieren und verwalten zu können ist ein Zugriff aus der Ferne sehr praktisch. Eine Möglichkeit ist der Zugriff mit SSH über einen VPS. Dafür benötigen alle Geräte beteiligten Gerät Zugriff auf das Internet. Der RaspberryPi baut zuerst einen Reverse SSH Tunnel zum VPS auf, da man vom VPS auf den RaspberryPi zugreifen will. Ein externer Benutzer greift über den VPS mit einer SSH auf den RaspberryPi zu.



Abbildung 97: Zugriff über VPS

Für das erstellen des Reversetunnel muss folgender Befehl beim RaspberryPi ausgeführt werden. Dabei muss die IP-Adresse des VPS *VPSIP* bekannt sei und es muss ein Umleitungsport *VPSPORT* festgelegt werden.

```

1 ssh -R VPSPORT:localhost:22 Nutzer@VPSIP

```

Code 53: Öffnen des Reverse SSH Tunnels

Der externe Nutzer kann mit dem untenstehenden Befehl sich bei Benutzer 'pi' über SSH einloggen.

```
1 ssh -p VPSPORT pi@VPSIP -o StrictHostKeyChecking=no
```

Code 54: Zugriff auf den Raspberry über SSH

### 7.6.3.2 Code Deployment

Für die Entwicklungsphase war es sehr angenehm den fertigen Code auf den RaspberryPi hinaufzuladen zu können. Github hat eine gewisse Funktionalität namens Github Actions. Mit diesen kann bei einer bestimmten Aktion ein Linux-Server gestartet werden, der je nach Anwendung bestimmte Shell-Befehle durchführt. Mit diesem Linux-Server lässt sich auch über SSH auf unseren RaspberryPi zugreifen. Der untenstehende Befehl aktualisiert das lokale Repository des RaspberryPi über SSH. Github braucht dafür diverse Daten wie das Kennwort des Benutzers 'pi', die nicht in Klartext im Code stehen dürfen. Sie können als sogenannte Repository-Secrets deklariert werden. Aus diesen Grund steht im unteren Befehl nicht etwa die IP des VPS sondern secrets\_VPSHOST.

```
1 sshpass -p ${ secrets.USER_PASSWORD } ssh -o StrictHostKeyChecking=no
  -p ${ secrets.VPS_PORT } ${ secrets.USER }@$ ${ secrets.VPS_HOST }
  \
2 "cd /home/pi/Advanced_Parking_Monitoring_Raspberry_Master \
3 && sudo git checkout . "
```

Code 55: Zugriff auf den Raspberry über SSH

Bei erfolgreicher GithubAction sieht man folgendes in der Benutzeroberfläche von Github sehen.

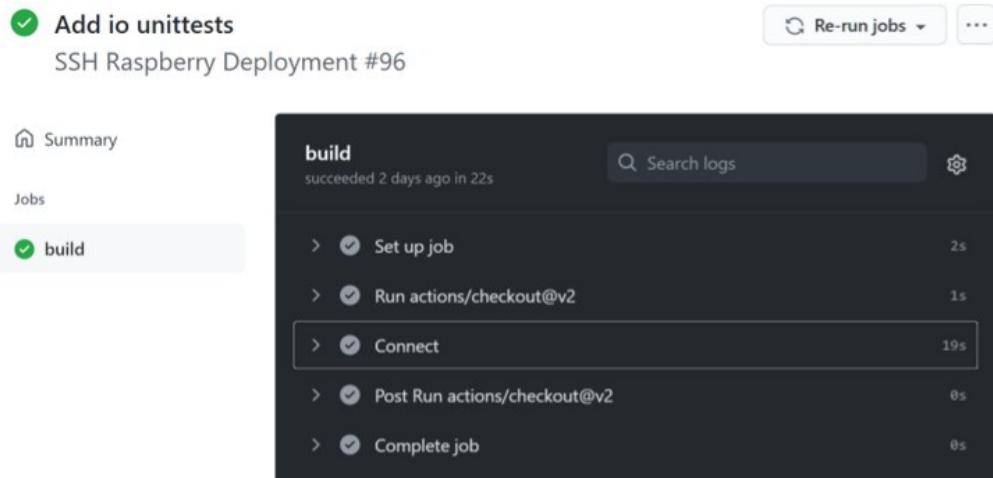


Abbildung 98: Github Action im Github Master repository

## 8 Webinterface

### 8.1 Einleitung

Das Webinterface ist die Web-Applikation, die auf der einen Seite die Aufgabe hat, die Kommunikation mithilfe einer eigenen Programmierschnittstelle mit der Kennzeichenerkennung und Fahrzeugerkennung sicherzustellen und auf der anderen Seite die Verwaltung und Darstellung der gewonnenen Daten für den Betreiber des Parkplatzes bzw. der Parkplätze.

Ein Benutzer besitzt im Webinterface eine Vielzahl von Konfigurationsmöglichkeiten und Funktionen, dazu zählen:

- Mobile-Friendly
- Dashboard
- Neuigkeiten
- Benutzerverwaltung
- Rechte- und Rollenverwaltung
- Parkplatzverwaltung
- Kennzeichenverwaltung
- Erkennungsverlauf
- Seiten Einstellungen
- Profiloptionen
- Lokalisierung

## 8.2 Verwendete Technologien

### 8.2.1 HTML

HTML steht hierbei für Hypertext Markup Language und ist eine Auszeichnungssprache welche vom World Wide Web Consortium (W3C)<sup>18</sup> definiert wird. Hypertext Markup Language (HTML) ist De-Facto-Standard um Inhalte in Browsern darzustellen. HTML ist dabei aber nicht für die visuelle Darstellung verantwortlich, sondern nur für die semantische Struktur<sup>19</sup>. Der Sinn dahinter ist, dass der Inhalt und die Vorgaben an die Darstellung möglichst gut getrennt ist. Für die Formatierung kommt die Stylesheet-Sprache Cascading Style Sheets (CSS) zum Einsatz, welche ebenfalls vom World Wide Web Consortium entwickelt wird. Die aktuellste Version der HTML Spezifikation ist HTML5<sup>20</sup> und wurde am 28. Oktober 2014 vom W3C vorgelegt.



Abbildung 99: HTML5 Logo von <https://www.w3.org/html/logo>

#### 8.2.1.1 Beispielhafte HTML Seite

Eine HTML Seite setzt sich aus einer Vielzahl von sogenannten Elementen zusammen. Ein Element besteht aus einem Start Tag und aus einem End Tag, der Inhalt wird zwischen diese Tags geschrieben. Nun folgt eine einfache HTML Seite, welche die grundlegenden Funktionen von HTML und CSS darlegen soll.

<sup>18</sup><https://www.w3.org>

<sup>19</sup>WHATWG. *HTML Living Standard*. [Online; Abgerufen am 02. April 2021]. 2021. URL: <https://html.spec.whatwg.org>.

<sup>20</sup><https://www.w3.org/2014/10/html5-rec.html.en>

```
1  <!DOCTYPE html>
2
3  <html>
4  <head>
5    <title>Titel</title>ü
6
7  </head>
8
9  <body>
10
11   <h1>Überschrift</h1>
12   <p>Paragraph</p>
13
14  </body>
15
16  </html>
```

Code 56: index.html

Das Element `<!DOCTYPE html>` deklariert, dass die folgende Seite den HTML5 Standard verwendet. Danach folgt mit `<html>` das Wurzelement, dass alle anderen Elemente beinhaltet. Das `<head>` Element beinhaltet verschiedene Metadaten. Im oben gezeigten Code 56 wird nur der Titel des Dokuments gesetzt, dieser wird im Browser Tab angezeigt. Es können aber auch noch andere Daten gesetzt bzw. eingebunden werden:

- Character Set
- Styles
- Scripts
- Viewport
- Sonstige Metainformationen (Author, Keywords)

## Überschrift

Paragraph

Abbildung 100: Einfache HTML Seite von <https://www.w3.org/html/logo>

### 8.2.2 CSS

Wie bereits angesprochen ist Cascading Style Sheets (CSS) für die Formatierung bzw. die visuelle Darstellung der einzelnen HTML-Elemente verantwortlich. Der Standard wird wie bei HTML vom W3C spezifiziert und die aktuellste Version ist CSS3, was so viel bedeutet wie CSS Level 3<sup>21</sup>, wobei nur einzelne Teile als Empfehlung durch das W3C vorgelegt wurden, beispielweise das CSS Color Module Level 3<sup>22</sup>. Um die Funktion darzustellen wird die vorherige HTML Seite nun mit CSS ergänzt.

<sup>21</sup>World Wide Web Consortium. *Cascading Style Sheets (CSS) Snapshot 2007*. [Online; Abgerufen am 25. März 2021]. 2011. URL: <https://www.w3.org/TR/css-beijing/#css3>.

<sup>22</sup><https://www.w3.org/TR/css-color-3>

```
1  body {  
2      background-color: deepskyblue;  
3  }  
4  
5  h1 {  
6      color: white;  
7      text-align: center;  
8      font-family: verdana;  
9  }  
10  
11 p {  
12     color: wheat;  
13     font-family: verdana;  
14     font-size: 20px;  
15 }
```

Code 57: style.css

Nun muss dieses Stylesheet nur noch im <head> Tag mit

```
<link rel="stylesheet" href="style.css">
```

 eingebunden werden.

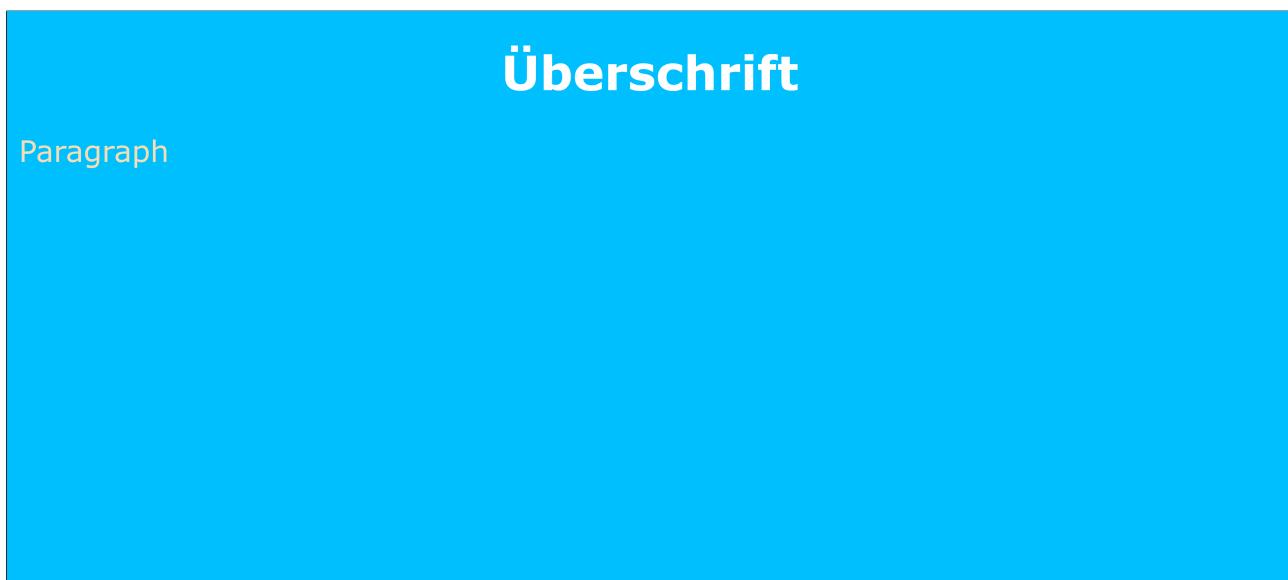


Abbildung 101: Einfache HTML Seite mit CSS

Es lässt sich nun erkennen, dass sich die Webseite deutlich verändert hat. Dabei bieten HTML und CSS noch viel mehr Funktionen, eine ausführliche Dokumentation der Funktionen sind auf der Website w3schools<sup>23</sup> zu finden.

### 8.2.3 JavaScript

Es folgt nun eine weitere sehr wichtige Technologie im Bereich der Webentwicklung und die meist verwendete Programmiersprache überhaupt laut der Stack Overflow Developer Survey 2020<sup>24</sup>. JavaScript (JS) ermöglicht es dynamische Webseiten zu erstellen, dabei wird der Code direkt lokal im Browser ausgeführt. Jedoch ist JavaScript nicht mehr nur auf das Frontend<sup>25</sup> beschränkt, es ist möglich mit Frameworks wie *Node.js* auch Backend<sup>26</sup> Applikationen zu schreiben und somit ist es möglich Full-Stack<sup>27</sup>-Anwendungen vollständig mit JavaScript zu entwickeln. Eine der wichtigsten Anwendungsbereiche ist die Manipulation von Elementen über das Document Object Model (DOM). Der Standard wird unter dem Namen ECMA Script von der Organisation Ecma International<sup>28</sup> veröffentlicht und die aktuellste Version ist ECMA-262.

<sup>23</sup><https://www.w3schools.com/html> und <https://www.w3schools.com/css>

<sup>24</sup><https://insights.stackoverflow.com/survey/2020>

<sup>25</sup>Präsentationsebene in Form der grafischen Benutzeroberfläche

<sup>26</sup>Verarbeitung von Daten auf beispielsweise einem Server

<sup>27</sup>Front- und Backend

<sup>28</sup><https://www.ecma-international.org>

#### 8.2.4 PHP

Hypertext Preprocessor (PHP) ist eine serverseitige Skriptsprache, um dynamische Webseiten zu realisieren, jedoch wird nicht wie bei JavaScript der Code auf dem Client ausgeführt, sondern auf dem Server, dort wird die HTML-Ausgabe generiert und dem Client zugesendet. Darum ist es nicht möglich, den Code als Benutzer zu betrachten. Der Syntax von PHP erinnert an C und Java und ermöglicht es schnell und einfache dynamische Webseiten zu programmieren. Die aktuellste Version ist PHP 8<sup>29</sup>.

#### 8.2.5 TailwindCSS

Es gibt eine Vielzahl von CSS-Frameworks, das Ziel ist ein einfacheres und schnelleres Erstellen von Webseiten, dazu gehören neben TailwindCSS folgende relevanten Frameworks.

- Bootstrap (<https://getbootstrap.com>)
- Foundation (<https://get.foundation>)
- Materialize (<https://materializecss.com>)

Viele von diesen CSS-Frameworks verwenden vorgefertigte Komponenten, welche direkt verwendet werden können, dies führt dazu, dass die Entwicklung sehr rasch ist. Dort unterscheidet sich TailwindCSS von den anderen CSS-Frameworks, dort existieren sogenannte Utility-Classes, diese können direkt im HTML auf die einzelnen Elemente angewendet werden und ermöglichen somit eine bessere Anpassbarkeit.

#### 8.2.6 Laravel

Laravel<sup>30</sup> ist ein Open-Source PHP-Framework, es erleichtert die Entwicklung und erhöht die Sicherheit. Durch die zahlreichen First-Party-Packages bietet Laravel ein sehr hochwertiges Ecosystem<sup>31</sup> für Entwickler.

---

<sup>29</sup><https://www.php.net/docs.php>

<sup>30</sup><https://laravel.com/>

<sup>31</sup>Vielzahl von Tools und Software

Da Laravel ein sehr wichtiger Bestandteil des Webinterfaces ist, wird später noch genauer auf die einzelnen Funktionen des Frameworks eingegangen. Laravel wird seit Juni 2011 entwickelt und erhält jährlich eine neue Version, aktuell ist Laravel 8 die neuste Version.

Das Framework folgt dem sogenannten Model-View-Controller (MVC) Muster<sup>32</sup> das bedeutet, dass die Programmierlogik in drei verschiedene Teile unterteilt wird. Der Sinn dahinter ist, dass die Anwendung dadurch sehr flexibel ist und später leichter erweitert werden kann oder einzelne Teile einfach wiederverwendet werden können.

- **Model**

Hier befindet sich die Datenstruktur der Anwendung. In Laravel kann dies beispielsweise das Model *User* sein.

- **View**

Die View ist die Präsentationsebene, im Fall von Laravel sind das Komponenten und Layouts.

- **Controller**

In den Controllern der Anwendung befindet sich die Logik, um beispielsweise durch das Absenden eines Formulares einen Eintrag in der Datenbank zu erstellen.

Ein Anfrage auf einer Webseite läuft in 6. Schritten ab:

- **1. Schritt:** Benutzer führt eine HTTP-Anfragemethode aus
- **2. Schritt:** Anfrage wird geroutet
- **3. Schritt:** Controller interagiert mit dem Model
- **4. Schritt:** Model greift auf die Datenbank zu
- **5. Schritt:** Controller liefert die Daten an eine View
- **6. Schritt:** View mit den Daten werden an den Client gesendet

<sup>32</sup>Matt Stauffer. *Laravel: Up & Running: A Framework for Building Modern PHP Apps.* O'Reilly Media, 2019.

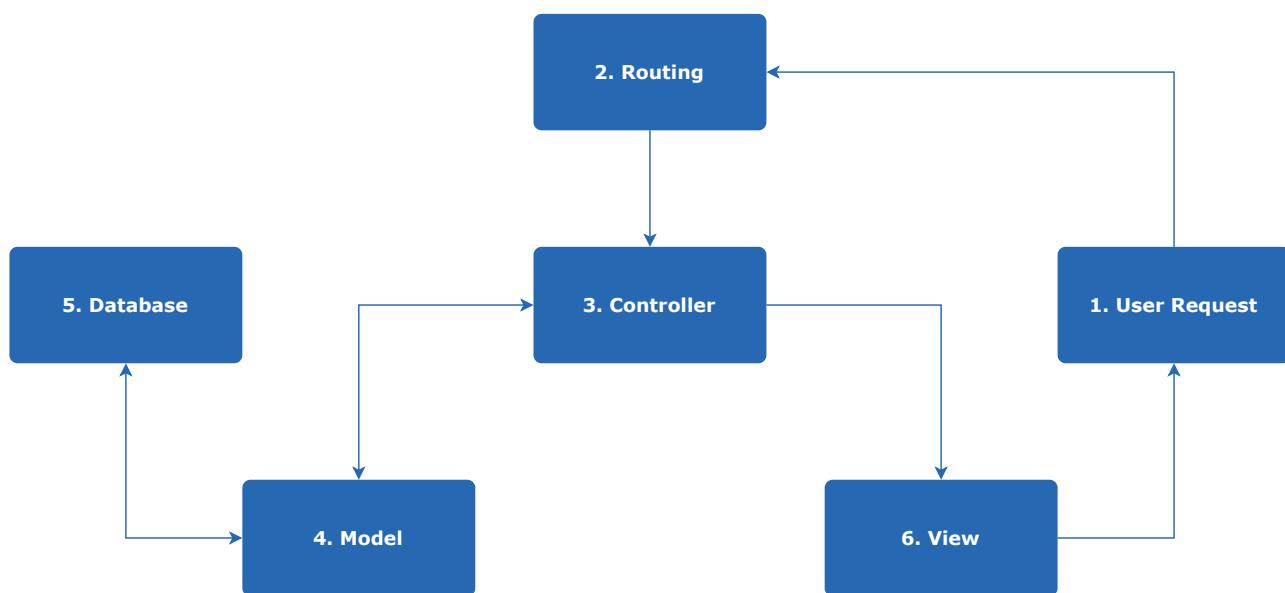


Abbildung 102: Laravel MVC Muster

## 8.3 Lokale Entwicklungsumgebung mit Laragon

### 8.3.1 Einleitung

Für die Programmierung des Webinterfaces müssen zuerst einige Vorehrungen getroffen werden, dazu zählt zum einem die Installation von benötigter Software und deren Konfiguration.

### 8.3.2 Benötigte Software

- **Laragon** (<https://laragon.org>)

Beinhaltet mehrere Softwarepakete die für die Entwicklung notwendig sind.

- Apache HTTP Server
- MySQL
- PHP

- **phpMyAdmin** (<https://www.phpmyadmin.net>)

Webinterface für MySQL

- **Composer** (<https://getcomposer.org>)

Paketmanager für PHP

- **Git** (<https://git-scm.com>)  
Versionskontrolle
- **Visual Studio Code** (<https://code.visualstudio.com>)  
Quelltext-Editor

### 8.3.3 Konfiguration von PHP

Um PHP Befehle von der Kommandozeile auszuführen, muss die PHP Installation zuerst in den Windows Path Variables hinzugefügt werden.

Dies erfolgt durch die **Advanced System Settings** ► **Environment Variables** ► **System Variables**. Dort kann nun die Path Variable editiert werden und der Pfad hinzugefügt werden in welchem die `php.exe` liegt.

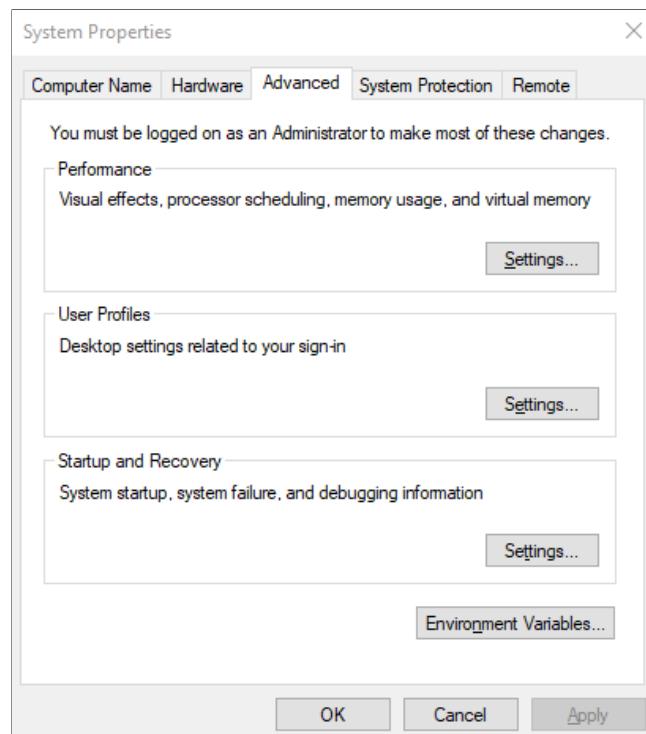


Abbildung 103: Advanced System Settings

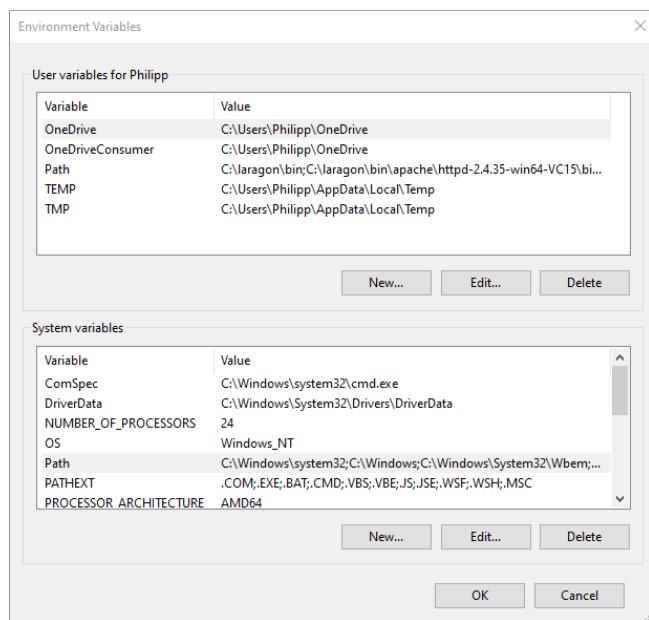


Abbildung 104: System Variables

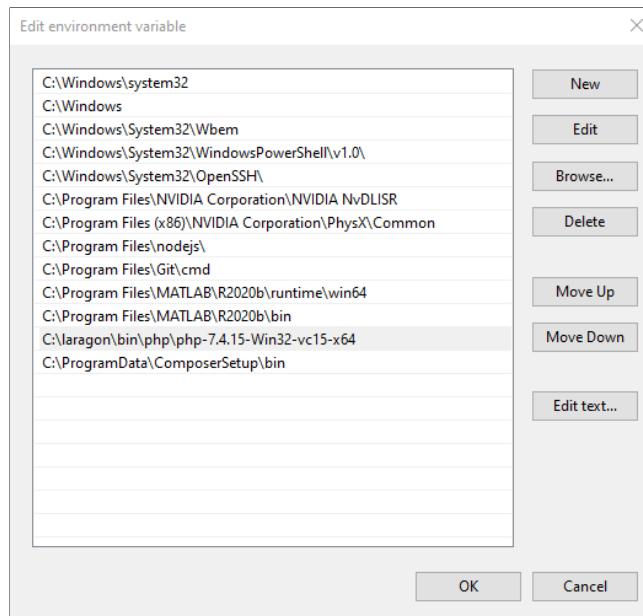


Abbildung 105: Environment Variables

Ob die Konfiguration korrekt ist kann durch die Kommandozeile überprüft werden, dort muss der Befehl `php -v` ausgeführt werden. Dabei ist zu beachten, dass nach dem Hinzufügen der Path Variable die gewählte Kommandozeile neu gestartet werden muss.

```
C:\Users\Philipp>php -v
PHP 7.4.15 (cli) (built: Feb  2 2021 20:47:45) ( ZTS Visual C++ 2017 x64 )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
```

Abbildung 106: PHP Version

Da eine Antwort mit der installierten Versionsnummer folgt, ist PHP korrekt installiert und konfiguriert.

### 8.3.4 Installation von phpMyAdmin

phpMyAdmin ist ein Tool, welches den Umgang mit MySQL Datenbanken mit einem Webinterface erleichtert. Die aktuellste Version lässt sich von <https://www.phpmyadmin.net/downloads> downloaden. Dieses Archiv muss entpackt werden und ausgehend vom Laragon Root Verzeichnis in das Verzeichnis /etc/apps kopiert werden. Um die Installation zu überprüfen, muss der Apache HTTP Server und der MySQL Server gestartet werden, nun sollte bei einer korrekten Installation das Webinterface von phpMyAdmin unter <http://localhost/phpmyadmin> erreichbar sein.

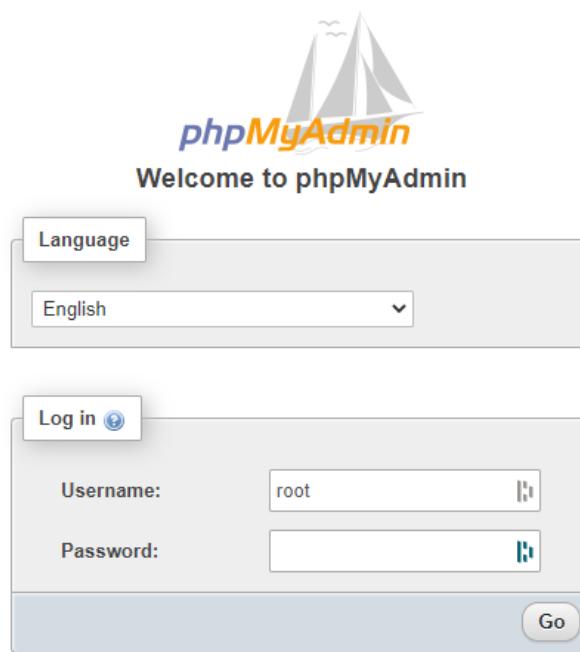


Abbildung 107: phpMyAdmin Webinterface

Es ist nicht notwendig ein Passwort einzugeben, da standardmäßig dem *root* Benutzer kein Passwort gesetzt wird.

## 8.4 Lokale Entwicklungsumgebung mit WSL und Docker

### 8.4.1 Einleitung

Eine Alternative zu der Entwicklungsumgebung mit Laragon, ist die Entwicklungsumgebung mit WSL und Docker. Der Vorteil von dieser Umgebung ist, dass die Anwendung sich in einem virtualisierten Container befindet und somit die exakt gleichen Bedingungen wie später auf dem Production Server vorhanden sind. Ein weiterer Vorteil ist die Übersichtlichkeit und die Wartung, auf dem Entwickler PC muss kein PHP, kein Apache usw. installiert werden, all diese Software wird nur im Container installiert.

Windows Subsystem for Linux (WSL) ist eine Kompatibilitätsschicht zum Ausführen von Linux Programmen. WSL 2 virtualisiert hierbei mithilfe von Hyper-V einen vollständigen Linux-Kernel, ein Nachteil davon ist, dass durch den Einsatz von Hyper-V auch das Host-Betriebssystem virtualisiert wird, jedoch ist das wenig bis gar nicht spürbar für den Anwender.

### 8.4.2 Benötigte Software

- **Docker** (<https://www.docker.com>)  
Ermöglicht Isolation von Anwendungen mit Containervirtualisierung
- **WSL** (<https://docs.microsoft.com/en-us/windows/wsl>)  
Kompatibilitätsschicht für Linux Anwendungen unter Windows 10
- **Visual Studio Code** (<https://code.visualstudio.com>)  
Quelltext-Editor

### 8.4.3 Installation von WSL

Zuerst müssen einige Einstellungen in Windows getroffen werden, damit später eine Linux-Distribution heruntergeladen werden kann. Die folgenden Befehle müssen über eine Kommandozeile mit Administrativen Rechten ausgeführt werden.

#### 8.4.3.1 1. Schritt: WSL aktivieren

```
1  dism.exe /online /enable-feature  
   ↳ /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

Code 58: WSL Feature aktivieren

#### 8.4.3.2 2. Schritt: Virtual Machine aktivieren

```
1  dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all  
   ↳ /norestart
```

Code 59: Virtual Machine Feature aktivieren

Nach diesem Schritt ist ein Neustart des Computers notwendig.

#### 8.4.3.3 3. Schritt: Linux Kernel Update

Nun muss ein Linux Kernel Update installiert werden, die aktuelle Version ist unter <https://aka.ms/wsl2kernel> zu finden. Die Installation ist sehr einfach, da nur ein Setup ausgeführt werden muss.

#### 8.4.3.4 4. Schritt: WSL 2

Nach dem Neustart des Computers sollte es nun möglich sein WSL 2 als Version auszuwählen.

```
1  wsl --set-default-version 2
```

Code 60: WSL 2 auswählen

#### 8.4.3.5 5. Schritt: Linux-Distribution herunterladen

Zuletzt kann eine Linux-Distribution aus dem Windows Store heruntergeladen werden, in diesem Fall Debian (<https://www.microsoft.com/de-de/p/debian>).

#### 8.4.4 Installation von Docker

Die aktuellste Version von Docker Desktop für Windows lässt sich am einfachsten über die offizielle Website von Docker herunterladen (<https://docker.com>). Nach der Installation muss noch die WSL Integration aktiviert werden. Dazu muss in den Einstellungen unter Resources ► WSL Integration ein Haken bei *Enable integration with my default WSL distro* gesetzt werden und die installierte Linux-Distribution muss unten aktiviert werden.

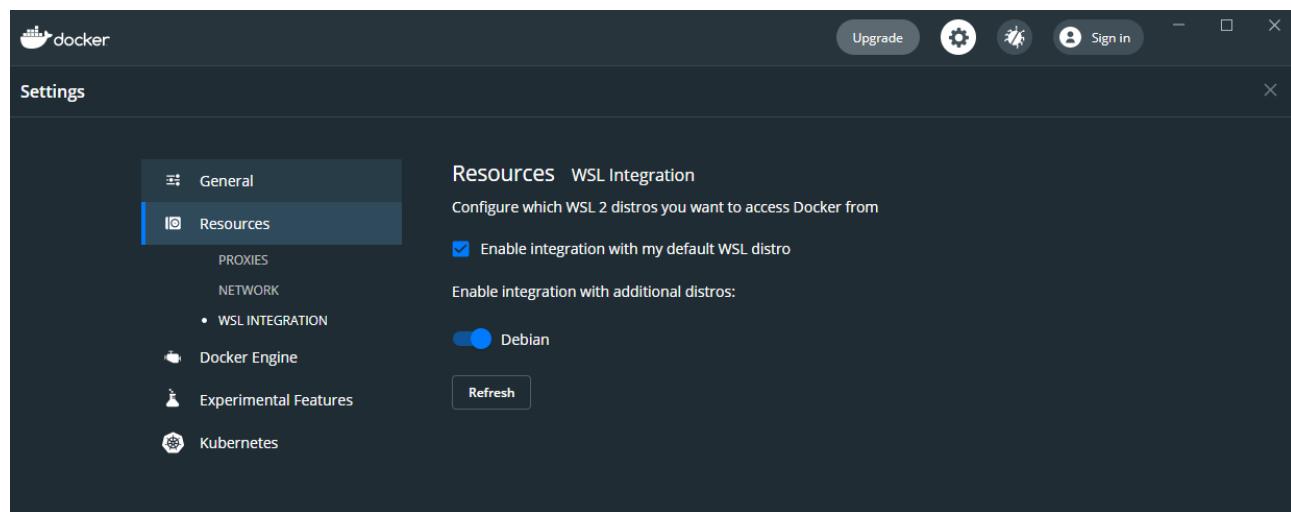


Abbildung 108: Docker WSL Integration

Somit ist die lokale Entwicklungsumgebung mit WSL und Docker abgeschlossen, weitere benötigte Software wird später automatisch durch Laravel Sail in einem Docker Container installiert.

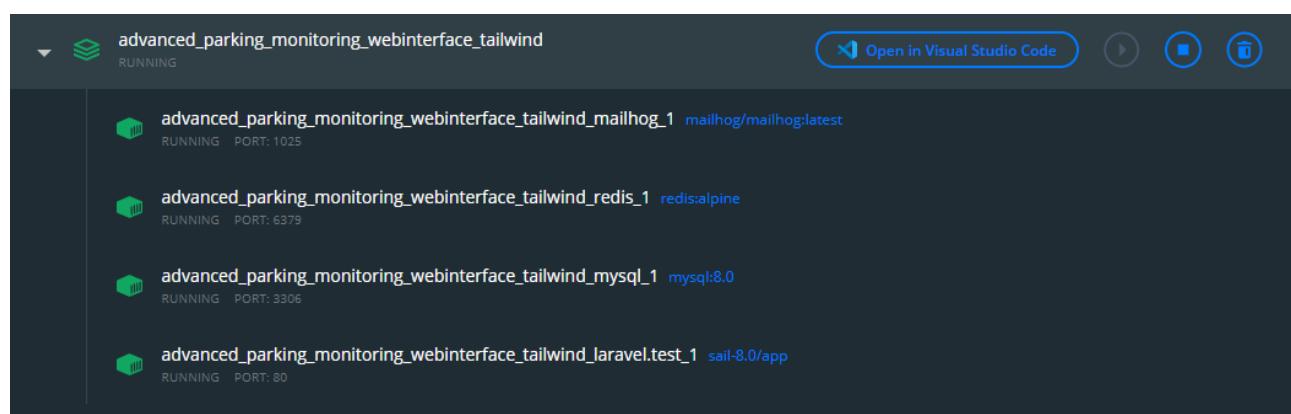


Abbildung 109: Docker Container Steuerung

Es ist somit möglich die Services welche im Container in der Linux-Distribution laufen, über die Docker Desktop Anwendung zu steuern.

### 8.4.5 Laravel Sail

Laravel Sail ist ein Kommandozeilen Tool für das Entwickeln in einer Docker Umgebung. Sail wird standardmäßig in neuen Laravel Anwendungen mitinstalliert, es ist deshalb keine manuelle Installation notwendig. Um Sail zu verwenden ist es sinnvoll ein Bash alias<sup>33</sup> zu setzen:

```
1 alias sail='bash vendor/bin/sail'
```

Code 61: Sail Bash alias

Nun ist es möglich mit dem Befehl `sail up` die Docker Virtualisierung zu starten, dabei wird MySQL, Apache, Redis und PHP gestartet. Ist es doch gewünscht an der Docker Virtualisierung etwas anzupassen kann dies über die Datei `docker-compose.yml` im Root Verzeichnis der Laravel Anwendung vorgenommen werden.

## 8.5 Production Server

### 8.5.1 Einleitung

Der Production Server bzw. der Live Server ist der Server, wo sich die Webanwendung befindet und die Endbenutzer zugreifen, dieser Server wird auch einfach mit Production abgekürzt. Der Production Server ist in diesem Fall ein Virtual Private Server mit dem Betriebssystem Debian 10, welcher bei einem Internet-Hosting Unternehmen mit Sitz in Deutschland gehostet wird.

### 8.5.2 Benötigte Software

Für den Live Server wird der sogenannte „LAMP“ Stack verwendet. LAMP steht dabei für die Software **L**inux, **A**pache, **M**ySQL und **P**HP.

- **Apache Web Server** (<https://httpd.apache.org>)

HTTP Server

- **MariaDB** (<https://mariadb.org>)

Fork von MySQL

---

<sup>33</sup>Definiert Kurznamen für Befehle

- **PHP** (<https://www.php.net>)  
Hypertext Preprocessor (PHP)
- **phpMyAdmin** (<https://www.phpmyadmin.net>)  
Webinterface für MySQL
- **Composer** (<https://getcomposer.org>)  
Paketmanager für PHP
- **Git** (<https://git-scm.com>)  
Versionskontrolle

### 8.5.3 Installation des LAMP Stacks

Bevor die Softwarepakete installiert werden, sollte die Linux Software Repository geupdatet werden.

```
1 apt-get update && apt-get upgrade
```

Code 62: Repositorys updaten

#### 8.5.3.1 Apache

Nun kann der Apache Web Server installiert werden.

```
1 apt install apache2
```

Code 63: Apache installieren

Die Installation kann nun leicht überprüft werden, indem man im Browser die IP bzw. die dazugehörige Domain aufruft, in diesem Fall: (<http://dev.philipp-kraft.com>).

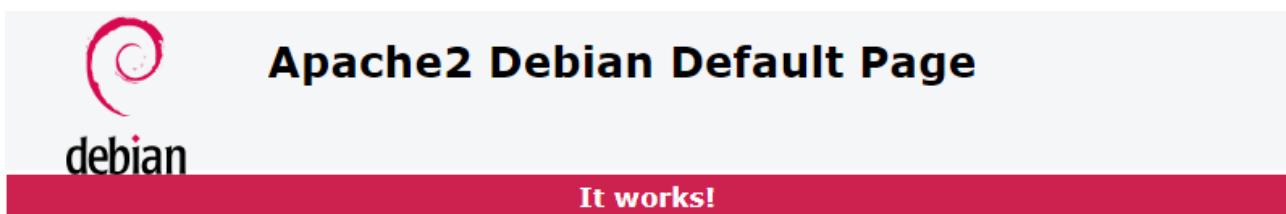


Abbildung 110: Debian Default Page

Erscheint die Debian Default Page ist Apache korrekt installiert.

#### 8.5.3.2 PHP

Neben PHP werden auch einige PHP Extensions benötigt, diese können leicht mit dem folgenden Befehl installiert werden.

```
1 apt install wget php php-cgi php-mysqli php-pear php-mbstring php-gettext
  ↳ libapache2-mod-php php-common php-phpseclib php-mysql
```

Code 64: PHP installieren

Die Installation kann nun wie bei Windows mit dem Befehl `php -v` überprüft werden.

#### 8.5.3.3 MariaDB

Die Installation von MariaDB ist ähnlich der von Apache und erfolgt mit einem Befehl.

```
1 apt install mariadb-server
```

Code 65: Mariadb installieren

Nun muss MariaDB noch konfiguriert werden.

```
1 mysql_secure_installation
```

#### Code 66: MariaDB Secure Installation

Dabei wird dem root MySQL User ein Passwort gesetzt, anonyme Benutzer gelöscht und allfällige Test Datenbanken gelöscht.

Nun wird ein neuer Benutzer mit dem Namen *admin* mit *root* Berechtigungen erstellt.

```
1 mysql
2 GRANT ALL ON *.* TO 'admin'@'localhost' IDENTIFIED
3 BY 'password' WITH GRANT OPTION;
4 flush privileges;
5 exit
```

#### Code 67: MariaDB konfiguration

##### 8.5.3.4 phpMyAdmin

Die aktuelle Version von phpMyAdmin kann von (<https://www.phpmyadmin.net/downloads>) bezogen werden und mit dem wget Befehl heruntergeladen werden.

```
1 wget https://files.phpmyadmin.net/phpMyAdmin/5.0.4
2 /phpMyAdmin-5.0.4-all-languages.tar.gz
```

#### Code 68: phpMyAdmin Download

Anschließend muss das Archiv entpackt werden.

```
1 tar xvf phpMyAdmin-5.0.4-all-languages.tar.gz
```

#### Code 69: phpMyAdmin Entpacken

Als Nächstes muss das entpackte Archiv in einen anderen Pfad verschoben werden und zusätzlich müssen einige Rechte und Verzeichnisse angepasst werden.

```
1 mv phpMyAdmin-5.0.4-all-languages /usr/share/phpmyadmin
2 mkdir -p /var/lib/phpmyadmin/tmp
3 chown -R www-data:www-data /var/lib/phpmyadmin
4 mkdir /etc/phpmyadmin/
```

#### Code 70: phpMyAdmin Rechte und Verzeichnisse

Nun muss eine Konfigurationsdatei erstellt werden und dort muss ein Blowfish Secret<sup>34</sup> angegeben werden und der Pfad für ein temporäres Verzeichnis.

```
1 cp /usr/share/phpmyadmin/config.sample.inc.php
→ /usr/share/phpmyadmin/config.inc.php
```

#### Code 71: phpMyAdmin Konfigurationsdatei erstellen

Am Ende dieser Datei müssen folgende zwei Zeilen eingefügt werden.

```
1 $cfg['blowfish_secret'] = 'H20xcGXxf1Sd8JwrwVlh6KW6s2rER63i';
2 $cfg['TempDir'] = '/var/lib/phpmyadmin/tmp';
```

#### Code 72: phpMyAdmin Blowfish Secret und TempDir

Zuletzt muss der Apache Web Server konfiguriert werden.

Im Verzeichnis /etc/apache2/sites-available muss eine neue Konfigurationsdatei mit dem Namen `phpmyadmin.conf` angelegt werden.

<sup>34</sup>32 Zeichen String für Cookie-Authentifizierung

```
1 Listen 9000
2
3 <VirtualHost *:9000>
4     ServerName localhost
5
6     <Directory /usr/share/phpmyadmin>
7         AllowOverride None
8         Require all granted
9     </Directory>
10
11    DocumentRoot /usr/share/phpmyadmin
12
13    ErrorLog ${APACHE_LOG_DIR}/phpmyadmin.error.log
14    CustomLog ${APACHE_LOG_DIR}/phpmyadmin.access.log combined
15 </VirtualHost>
```

Code 73: phpmyadmin.conf

Nun kann diese Virtual Host Konfigurationsdatei aktiviert werden und danach muss der Apache Web Server neu gestartet werden, damit die Konfiguration wirksam ist.

```
1 a2ensite phpmyadmin
2 systemctl restart apache2
```

Code 74: Virtual Host Konfiguration aktivieren

Diese Konfiguration ermöglicht es, dass das Webinterface von phpMyAdmin über den Port 9000 (<http://dev.philipp-kraft.com:9000>) erreichbar ist und nicht wie standardmäßig vorgesehen über das Verzeichnis /phpmyadmin (<http://dev.philipp-kraft.com/phpmyadmin>), dies bietet einen Sicherheitsvorteil, da potenzielle Angreifer nicht einmal den Zugang finden.

### 8.5.3.5 Webinterface Virtual Host

Nun muss noch eine Virtual Host Konfiguration für das Webinterface selbst erstellt werden.

```
1 <VirtualHost *:80>
2   ServerAdmin webmaster@localhost
3   DocumentRoot /var/www/apm/public
4
5   <Directory />
6     Options FollowSymLinks
7     AllowOverride All
8   </Directory>
9
10  <Directory /var/www/apm>
11    Options Indexes FollowSymLinks MultiViews
12    AllowOverride All
13    Order allow,deny
14    allow from all
15  </Directory>
16
17  ErrorLog ${APACHE_LOG_DIR}/error.log
18  CustomLog ${APACHE_LOG_DIR}/access.log combined
19 </VirtualHost>
```

Code 75: apm.conf

Diesmal wird auf den Standard HTTP Port 80 gehört und dieser führt in das Verzeichnis /var/www/apm/public. Zusätzlich werden noch Directives gesetzt, damit das Standard .htaccess File von Laravel richtig funktionieren kann.

### 8.5.3.6 Installation von Composer

Die Installation von Composer gestaltet sich relativ einfach.

```
1 wget -O composer-setup.php https://getcomposer.org/installer
```

Code 76: Download Composer Installer

Nun muss das Setup ausgeführt werden und damit das `composer` Befehl Global verfügbar ist, wird Composer in den Pfad `/usr/local/bin` verschoben.

```
1 php composer-setup.php --install-dir=/usr/local/bin --filename=composer
```

Code 77: Composer Setup

#### 8.5.4 Deployment mit Github Actions

Unter Deployment versteht man die automatische Installation von Software, in diesem Fall auf einem Linux Server. Erreicht wird das durch zwei Bash Scripts und mit Github Actions (<https://github.com/features/actions>).

##### 8.5.4.1 Git Setup

Sollte auf dem Server noch kein Git installiert sein, lässt sich das wie folgt installieren.

```
1 apt install git
```

Code 78: Git Installation

Im Verzeichnis `/var/www/apm` soll sich später das Webinterface befinden, deshalb muss in diesem Pfad Git konfiguriert werden. Dazu wird die Remote URL definiert.

```
1 git config remote.origin.url 'https://{{TOKEN}}@github.com/  
→ Philipp-Kraft/Advanced_Parking_Monitoring_Webinterface.git'
```

Code 79: Git Remote Origin

Da beim Github Account eine Zwei-Faktor-Authentisierung verwendet wird, muss die Authentifizierung mit einem Personal Access Token erfolgen. Dieser kann unter <https://github.com/settings/tokens> erstellt werden, der erstellte Token kann dann einfach in der Remote URL eingefügt werden.

#### 8.5.4.2 Deploy Script

Das Deploy-Script wird auf der lokalen Entwicklermaschine ausgeführt. Das Script wechselt in den Production Branch und merged mit dem Main Branch, dieser Push in den Production Branch löst dann die Github Action aus.

```
1 #!/bin/sh  
2 set -e  
3  
4 #vendor/bin/phpunit  
5  
6 (git push) || true  
7  
8 git checkout production  
9 git merge main  
10  
11 git push origin production  
12  
13 git checkout main
```

Code 80: Lokales Deploy Script

### 8.5.4.3 Server Deploy Script

Das Server Deploy Script `server_deploy.sh` versetzt die Laravel Applikation in den Wartungsmodus und lädt sich vom Deploy Branch den Code auf den Server herunter, danach werden einige Befehle ausgeführt.

```
1  #!/bin/sh
2
3
4  echo "Deploying application . . ."
5
6  # Enter maintenance mode
7  php artisan down
8
9  # Update codebase
10 git fetch origin deploy
11 git reset --hard origin/deploy
12
13 # Install dependencies based on lock file
14 composer install --no-interaction --prefer-dist --optimize-autoloader
15
16 # Migrate database
17 php artisan migrate:refresh --seed
18
19 # Clear cache
20 php artisan optimize
21
22 # Exit maintenance mode
23 php artisan up
24
25 echo "Application deployed!"
```

## Code 81: Server Deploy Script

#### 8.5.4.4 Github Actions

Github Actions ist ein Projekt von Github, welches es ermöglicht, Automatisierungen in den Bereichen Entwicklung, Testing und Deployment durchzuführen.

Als erstes muss ein Workflow erstellt werden, dieser wird im Root-Verzeichnis des Projekts unter dem Pfad APM\.github\workflows\main.yml erstellt.

```
1  name: Deploy Laravel app
2
3  on:
4    push:
5      branches: [ production ]
6
7  jobs:
8    deploy:
9      runs-on: ubuntu-latest
10     steps:
11       - uses: actions/checkout@v2
12         with:
13           token: ${{ secrets.PUSH_TOKEN }}
14       - name: Set up Node
15         uses: actions/setup-node@v1
16         with:
17           node-version: '12.x'
18       - run: npm install
19       - run: npm run production
20       - name: Commit built assets
21         run: |
22           git config --local user.email "action@github.com"
23           git config --local user.name "GitHub Action"
```

```

24      git checkout -B deploy
25
26      git add -f public/
27      git commit -m "Build front-end assets"
28      git push -f origin deploy
29
30      - name: Deploy to production
31          uses: appleboy/ssh-action@master
32
33          with:
34
35              username: root
36
37              host: dev.philipp-kraft.com
38
39              password: ${{ secrets.SSH_PASSWORD }}
40
41              script: 'cd /var/apm && ./server_deploy.sh && chown -R
42                  www-data.www-data /var/apm && chmod -R 755 /var/apm && chmod
43                  -R 777 /var/apm/storage'
```

Code 82: main.yml

Dieser Workflow setzt einen Ubuntu Server in der Cloud bei Github auf und baut dort die Frontend Assets zusammen, damit die Downtime auf dem Production Server möglichst gering ist. Danach wird eine SSH Verbindung zum Production Server aufgebaut und dort wird das Bash-Script `server_deploy.sh` ausgeführt. Gleichzeitig werden einige Berechtigungen angepasst.

In der Repository muss nun noch das SSH-Passwort des Servers und der Personal Access Token hinterlegt werden. Dies geschieht über **Settings ▶ Secrets**. Dort können nun über *New repository secrets* die Secrets hinterlegt werden.

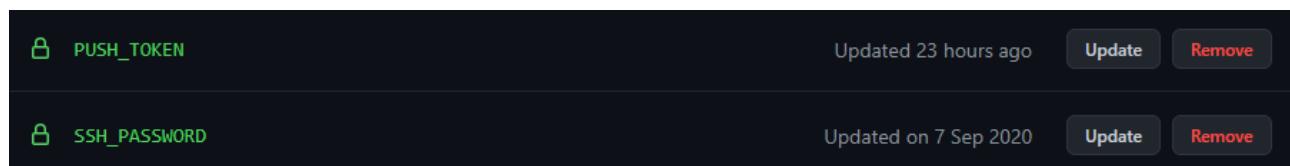


Abbildung 111: Action Secrets

```

deploy
succeeded 23 hours ago in 1m 10s

> ✓ Set up job 5s
> ✓ Build appleboy/ssh-action@master 5s
> ✓ Run actions/checkout@v2 3s
> ✓ Set up Node 0s
> ✓ Run npm install 20s
> ✓ Run npm run production 20s
> ✓ Commit built assets 3s
> ✓ Deploy to production 14s
> ✓ Post Run actions/checkout@v2 0s
> ✓ Complete job 0s

```

Abbildung 112: Github Action Übersicht

## 8.6 Backend

### 8.6.1 Einleitung

Als Backend bezeichnet man Systeme, die sich mit der Datenverarbeitung im Hintergrund beschäftigen. Für das Webinterface wurde als Backend PHP in Verbindung mit Laravel gewählt. PHP und Laravel in Kombination wurden aus folgenden Gründen gewählt:

- **Quelloffen:**

PHP und Laravel sind quelloffen d.h. der Quelltext ist öffentlich zugänglich.

- **Plattformunabhängig:**

Es spielt keine Rolle welches Betriebssystem verwendet wird.

- **Verbreitung:**

PHP ist die meist genutzte Server-Programmiersprache und Laravel das beliebteste PHP-Framework

- **Sicherheit:**

Besonders in Kombination mit Laravel sind viele Gefahren wie Cross-Site-Scripting<sup>35</sup> und SQL Injection<sup>36</sup> kein Problem mehr.

<sup>35</sup>Ausnutzen einer Computersicherheitslücke, oft mit JavaScript

<sup>36</sup>Datenbankbefehle einschleusen aufgrund mangelnder Maskierung

- **Stabilität:**

Fehler und Sicherheitslücken werden innerhalb kürzester Zeit von den Entwicklern behoben.

## 8.6.2 Routing

### 8.6.2.1 Routing in Laravel

In Laravel erfolgt das Routing durch ein Routefile unter `routes/web.php`. API-Anfragen haben ein eigenes Routefile unter `routes/api.php` und erhalten direkt einen `/api` Prefix in der URL. Die Route Files werden automatisch durch einen Service Provider von Laravel beim Start der Anwendung geladen.

Der Laravel Router erlaubt folgende HTTP-Anfragemethoden:

- **GET** (Fordert Ressource an)

```
Route::get($uri, $callback);
```

- **POST** (Neue Ressource erstellen)

```
Route::post($uri, $callback);;
```

- **PUT** (Ressource ersetzen oder erstellen)

```
Route::put($uri, $callback);
```

- **PATCH** (Ressource ändern)

```
Route::patch($uri, $callback);
```

- **DELETE** (Löscht Ressource)

```
Route::delete($uri, $callback);
```

- **OPTIONS** (Liste von unterstützten Methoden des Servers)

```
Route::options($uri, $callback);
```

### 8.6.2.2 Verfügbare Web Routen

Im Webinterface sind die Routen in drei Gruppen unterteilt:

- **Öffentliche Routen**

Diese Routen können von jedem Benutzer erreicht werden.

- **Verified Routen**

Verified Routen sind nur durch eingeloggte Benutzer erreichbar.

- **Admin Routen**

Ob Admin Routen erreichbar sind, hängt davon ab ob ein Benutzer die passenden Rechte besitzt.

Insgesamt besitzt das Webinterface **52** verschiedene Routen. Die Middleware *verified* überprüft hierbei ob ein Benutzer eingeloggt ist, die Option *prefix* sorgt dafür, dass *admin* vor allen Routen in der URL angeführt wird.

Es gibt für folgende Seiten Routen, wobei Routen mit verschachtelungen nur einmalig genannt werden:

- Locale
- Privacy Policy
- Terms of Service
- Imprint
- Profile
- Posts
- Users
- Roles
- Plates
- Lots
- Detections
- Displays
- Settings
- Tokens

Ein Ausschnitt dieser Routen ist im Code 83 zu sehen, hierbei sind alle Routen der Benutzerverwaltung dargelegt.

```

1  <?php
2
3  Route::group(['middleware' => 'verified', 'prefix' => 'admin'],
4
5      function () {
6
7          Route::get('users', [UserController::class,
8
9              'index'])->name('users.index');
10
11         Route::get('users/create', [UserController::class,
12
13             'create'])->name('users.create');
14
15         Route::post('users', [UserController::class,
16
17             'store'])->name('users.store');
18
19         Route::get('users/{user}', [UserController::class,
20
21             'show'])->name('users.show');
22
23         Route::get('users/{user}/edit', [UserController::class,
24
25             'edit'])->name('users.edit');
26
27         Route::patch('users/{user}', [UserController::class,
28
29             'update'])->name('users.update');
30
31         Route::delete('users/{user}', [UserController::class,
32
33             'destroy'])->name('users.destroy');
34
35     });

```

Code 83: web.php

### 8.6.3 Controller

Es besteht theoretisch die Möglichkeit, die komplette Logik für die Bearbeitung von Anfragen direkt in den Route Files zu platzieren, dies ist jedoch sehr unübersichtlich und deshalb ist es sinnvoll, diese Logik in Controller Klassen auszulagern. Dabei werden ähnliche Anfragen zusammengeführt, beispielsweise werden alle Anfragen im Bezug auf die Benutzerverwaltung wie im Code 83 im gleichen Controller zusammengeführt.

Das Webinterface besitzt neben den Controllern für die Authentifizierung und die API folgende Controller:

- DashboardController
- DetectionController
- LicensePlateController
- ParkingLotController
- ParkingSpotController
- PermissionController
- PostController
- ProfileController
- RoleController
- SettingController
- TokenController
- UserController

Ein Ausschnitt des UserController ist im folgenden Code 84 zu erkennen.

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5
6 class UserController extends Controller
7 {
8
9     public function index()
10    {
11        $this->authorize('index', User::class);
12
13        $users = User::orderBy('id', 'asc')->paginate(25, ['*'],
14            ['users']);
15
16        return view('users.index')->with('users', $users);
17    }
18
19    . . .
```

Code 84: UserController.php

#### 8.6.4 Artisan CLI

Artisan ist ein Kommandozeilentool von Laravel und zentraler Bestandteil vom Framework. Die Artisan CLI stellt eine Fülle an sehr nützlichen Befehlen dem Entwickler zur Verfügung. Mithilfe von

Artisan können neue Controller, Models, Migrations und viel mehr, sehr einfach erstellt werden. Da es sehr viele Befehle gibt, folgen hier nur die wichtigsten Befehle:

- **php artisan list**

Zeigt eine Liste aller verfügbaren Befehle an

- **php artisan serve**

Startet einen PHP Development Server unter dem Port 8000

- **php artisan make:model <name>**

Erstellt ein neues Model

- **php artisan make:migration <name>**

Erstellt eine neue Migration

- **php artisan make:controller <name>**

Erstellt ein neuer Controller

- **php artisan migrate**

Führt die Datenbank Migration Files aus

## 8.6.5 Datenbank

### 8.6.5.1 Seeder

Um ein bestimmtes Feature zu testen, beispielsweise die Benutzerverwaltung, sind Testdaten notwendig. Da es sehr mühsam wäre mehrere hunderte Benutzer selbst zu erstellen werden Seeder verwendet. Seeder erlauben, sehr einfach, vorher definierte Daten in die Datenbank zu seeden bzw. schreiben. Neben Testdaten für das Testen von bestimmten Features werden auch die vordefinierten Rechte mithilfe von Seeder in die Datenbank geschrieben.

Das Webinterface besitzt folgende Seeder:

- DetectionSeeder

- ParkingLotSeeder

- LicensePlateSeeder

- ParkingSpotSeeder

- PermissionSeeder
- PostSeeder
- RoleSeeder
- SettingSeeder
- UserSeeder

```
1 <?php
2
3 public function run()
4 {
5
6     Role::query()->delete();
7
8
9     Role::create(['name' => 'Owner', 'slug' => 'owner'])
10    ->permissions()->attach(Permission::all());
11
12
13 }
```

Code 85: Rollen Datenbank Seeder

### 8.6.5.2 Migrations

Datenbank Migrations sind praktisch Vorlagen, wie einzelne Tabellen in der Datenbank auszusehen haben. Migrations erleichtern die Handhabung von SQL Datenbanken im Team, da nun die Möglichkeit besteht alles bis auf den Inhalt der Datenbank mit Git zu verwalten.

```

1  <?php
2
3  class CreateUsersTable extends Migration
4  {
5
6      public function up()
7      {
8
9          Schema::create('users', function (Blueprint $table) {
10
11             $table->id();
12
13             $table->string('first_name');
14
15             $table->string('last_name');
16
17             $table->string('email')->unique();
18
19             $table->string('avatar')->default('default.png');
20
21             $table->timestamp('email_verified_at')->nullable();
22
23             $table->string('password');
24
25             $table->string('last_login_ip')->nullable();
26
27             $table->timestamp('last_login_at')->nullable();
28
29             $table->rememberToken();
30
31             $table->timestamps();
32
33         });
34
35     }
36
37
38     public function down()
39     {
40
41         Schema::dropIfExists('users');
42
43     }
44
45 }

```

Code 86: create\_users\_table.php

Im Code 86 sind zwei Methoden zu erkennen, up und down, in der up Methode werden neue Tabellen und Spalten in der Datenbank erstellt, bei der down Methode wird alles von der up Methode rückgängig gemacht, in diesem Fall wird die komplette Tabelle gelöscht. Um nun die Datenbank zu

migrieren, muss in der Artisan CLI der Befehl `php artisan migrate` ausgeführt werden, dabei werden alle Migrations im Verzeichnis `database/migrations` ausgeführt.

Das Webinterface besitzt folgende Migration Files:

- `create_users_table`
- `create_password_resets_table`
- `create_failed_jobs_table`
- `create_personal_access_tokens_table`
- `create_license_plates_table`
- `create_parking_lots_table`
- `create_detections_table`
- `create_roles_table`
- `create_permissions_table`
- `create_users_roles_table`
- `create_roles_permissions_table`
- `create_settings_table`
- `create_parking_spots_table`
- `create_posts_table`

### 8.6.6 Eloquent ORM

Eloquent ist ein Object–relational mapping (ORM) Tool d.h. um auf die SQL Datenbank zuzugreifen, müssen keine Raw SQL Queries ausgeführt werden. Im Code erscheint die Datenbank als objektorientierte Datenbank und erleichtert somit dem Umgang mit der Datenbank. Um nun mit der Datenbank zu interagieren, werden im Code Models verwendet, beispielsweise ein *User Model*. In solch einem Model sind auch die Beziehungen zu anderen Models festgelegt (1:n, n:m usw.). Ist es nun gewünscht, auf die Daten in der Datenbank zuzugreifen, wird im Code mit dem passenden Model gearbeitet. Der große Vorteil, der beim Verwenden von Eloquent entsteht, ist der leicht lesbare Code, jedoch wird die Ausführungszeit leicht erhöht, was jedoch keine große Rolle bei kleinen- bis mittelgroßen Webseiten spielt.

```
1 <?php
2 User::where('first_name', 'Philipp')->first();
```

Code 87: Eloquent Query

```
1  SELECT * FROM `users` WHERE `first_name` = Philipp
```

Code 88: Raw SQL Query

Beim Vergleich zwischen dem Code 87 und Code 88 ist es erkenntlich, dass der Query mit Eloquent verständlicher ist. Besonders bei komplexeren Queries ist Eloquent den Raw SQL Queries im Bezug auf die Lesbarkeit stark überlegen.

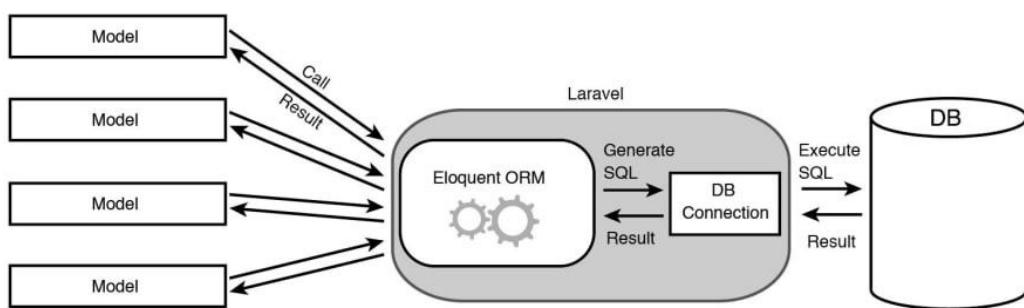


Abbildung 113: Eloquent ORM Workflow von <https://dev.to/xenoxdev/mastering-laravel-eloquent-orm-the-eloquent-journey-part-1-1571>

### 8.6.7 Laravel Sanctum

Laravel Sanctum ist ein First-Party-Package von den Entwicklern von Laravel und ermöglicht eine einfache Authentifizierung mithilfe von API-Tokens. Dabei wird der vom Benutzer erstellte API-Token im HTTP Header `Authorization` mitgeliefert und somit wird die Anfrage bei korrektem Token autorisiert. Da es sich hierbei um ein sicherheitskritisches Modul handelt ist es besonders wichtig, dass dieser Code nahezu fehlerfrei ist und dies wird durch kontinuierliche Updates und Patches von den Laravel Entwicklern gewährleistet.

### 8.6.8 Sessions

Das HTTP Netzwerkprotokoll ist ein zustandsloses Protokoll. Da aber immer über einen Benutzer Informationen über mehrere Seiten beibehalten werden soll, wird ein System benötigt, um diese Informationen zu speichern. Sessions lösen dieses Problem und funktionieren wie folgt. Ein Benutzer besucht eine Seite und besitzt im lokalen Speicher (Cookie) eine Session ID vom Server. Wenn

der Benutzer nun eine andere Seite der gleichen Webseite besucht, wird diese Session ID dem Server wieder gesendet und der Server kann den Benutzer passend zuordnen. Laravel verwendet als Backend für die Sessions Memcached oder Redis.

### 8.6.9 Middleware

Eine Middleware erleichtert es HTTP Anfragen zu filtern bzw. zu begutachten. Beispielsweise wird im Webinterface eine Middleware für das passende Setzen der Sprache verwendet. Bei jedem Aufruf einer Seite überprüft die Middleware welcher Wert in der Session des Benutzers steht, um so die passende Sprache auszuwählen.

### 8.6.10 Service Provider

Service Provider sind da, um bestimmte Funktionen in der Anwendung zu initialisieren. Beispielsweise wird im Webinterface ein *SettingsServiceProvider* verwendet um die Einstellungen aus der Datenbank in das Laravel Einstellungssystem zu übernehmen.

### 8.6.11 Form Validation

Für bestimmte Formulare ist es notwendig zu überprüfen, ob die Angaben korrekt sind, beispielsweise dass ein Passwort eine bestimmte Länge hat oder ob eine E-Mail ein @-Zeichen enthält. Um diese HTTP-Anfragen zu überprüfen, müssen bestimmte Regeln festgelegt werden, diese werden in Laravel in *Request Files* festgelegt.

Im Code 89 werden Regeln für den Vornamen, Nachnamen, die E-Mail und für das Passwort festgelegt. Die verfügbaren Regeln sind durch Laravel definiert. Liefert eine Request Methode *false* wird der Controller nicht weiter ausgeführt und die nicht korrekt ausgefüllten Felder werden in der Session gespeichert und können dem User als Fehlernachricht dargestellt werden.

```
1 <?php
2 public function rules()
3 {
4     return [
5         'first_name' => ['string', 'max:255'],
6         'last_name' => ['string', 'max:255'],
7         'email' => ['string', 'email', 'max:255'],
8         'password' => ['nullable', 'string', 'min:8', 'confirmed'],
9     ];
10 }
```

Code 89: UserUpdate Request

## 8.7 Frontend

### 8.7.1 Einleitung

Da das Ziel ist, dass das Frontend des Webinterfaces möglichst modular aufgebaut ist und so wenig Code wie möglich wiederholt wird, dies wird mithilfe von Components ermöglicht.

### 8.7.2 Blade Templates

Blade ist eine Template Engine, die mit Laravel mitgeliefert wird. Blade erleichtert und vereinfacht das Verwenden von PHP Code in HTML, dabei wird der Blade Syntax in normalen PHP Code kompiliert. Blade Dateien werden mit der Extension .blade.php erstellt. Damit der Inhalt der einzelnen Seiten dynamisch ist, müssen die Blade Dateien mit Inhalt bzw. mit Daten gefüllt werden, dies erfolgt über die Controller.

Um nun übergebenen Inhalt eines Controllers innerhalb einer Blade Datei anzuzeigen, verwendet Blade doppelt geschweifte Klammern.

```
1 Hallo, {{ $user->full_name }}.
```

Code 90: example.blade.php

Im Code 90 wird nun auf das übergebene Model *User* zugegriffen und der Name ausgegeben.

### 8.7.3 Components

Components sind praktisch kleine Bausteine, aus denen die komplette Seite aufgebaut ist. Components sind kein natives Feature von HTML/CSS oder PHP, diese Funktion wird von der Template Engine Blade bereitgestellt, deshalb werden diese auch oft Blade Components genannt.

#### 8.7.3.1 Anonymous Components

Es gibt viele verschiedene Möglichkeiten Components zu erstellen und verschiedene Konventionen, am einfachsten sind aber die *Anonymous Components*, diese haben den Vorteil, dass diese in einer Datei verwaltet werden können und somit sehr einfach zu handhaben sind.

#### 8.7.3.2 Components erstellen

Das Erstellen von einem Component wird nun anhand eines Buttons gezeigt. Da dieser als Anonymous Component angelegt wird, muss dieser mit keiner Klasse assoziiert werden, es wird einfach im Pfad `resources/views/components` ein Ordner mit dem Namen *buttons* angelegt und darin ein Blade File mit dem Namen `primary.blade.php` angelegt. Dort kann nun der gewünschte HTML Code platziert werden.

```
1 <button type="submit" class="inline-flex items-center px-4 py-2
  ↳ bg-apm-blue...">
2   {{ $slot }}
3 </button>
```

Code 91: primary.blade.php

Im Code 91 wird eine Variable mit dem Namen slot verwendet. Diese Variable wird später beim verwenden automatisch mit dem Inhalt im HTML Tag ersetzt.

### 8.7.3.3 Components verwenden

Es stellt sich nun die Frage, wie man dieses erstellen Component nun verwendet. Die Blade Components verwenden den gleichen Syntax wie ein normales HTML Element mit dem Unterschied, dass ein x- vor dem Namen des Components angeführt werden muss. Da sich der vorhin erstellte Component in einem Ordner befindet, muss dieser auch angegeben werden, dabei wird kein Slash wie üblich um einen Pfad anzugeben verwendet sondern ein Punkt, es muss auch keine Datei Extension angegeben werden.

```
1 <x-buttons.primary>Press me!</x-buttons.primary>
```

Code 92: Verwendung eines Button Components

### 8.7.3.4 Attribute übergeben

Auch wenn viele Components ohne Problem überall ohne Veränderung verwendet werden können, ist es gewünscht, bei manchen Components beispielweise eine zusätzliche Klasse, anzugeben um beispielsweise die Größe des Elements zu verändern. Erreicht wird das mit der attributes Variable im Blade File des Components, da aber oft schon Attribute definiert sind, ist es möglich, mit der merge Methode die Attribute zusammenzuführen.

```
1 <button {{ $attributes->merge(['type' => 'submit', 'class' => 'inline-flex
  ↵ items-center px-4 py-2 bg-apm-blue...']) }}>
2   {{ $slot }}
3 </button>
```

Code 93: Modularer Button Component

Somit ist dieser Button Component nun vollständig modular.

#### 8.7.4 Layouts

Da auf den meisten Seiten des Webinterfaces fast das gleiche Layout beibehalten wird, ist es sinnvoll diesen Inhalt in ein Component umzuwandeln. Auch Layouts sind Components.

Es gibt im Webinterface folgende Layouts:

- **app.blade.php**

Layout für allgemeine Seiten mit Footer

- **admin.blade.php**

Layout für die administrativen Seiten mit einer Sidebar, Content Header und ohne Footer

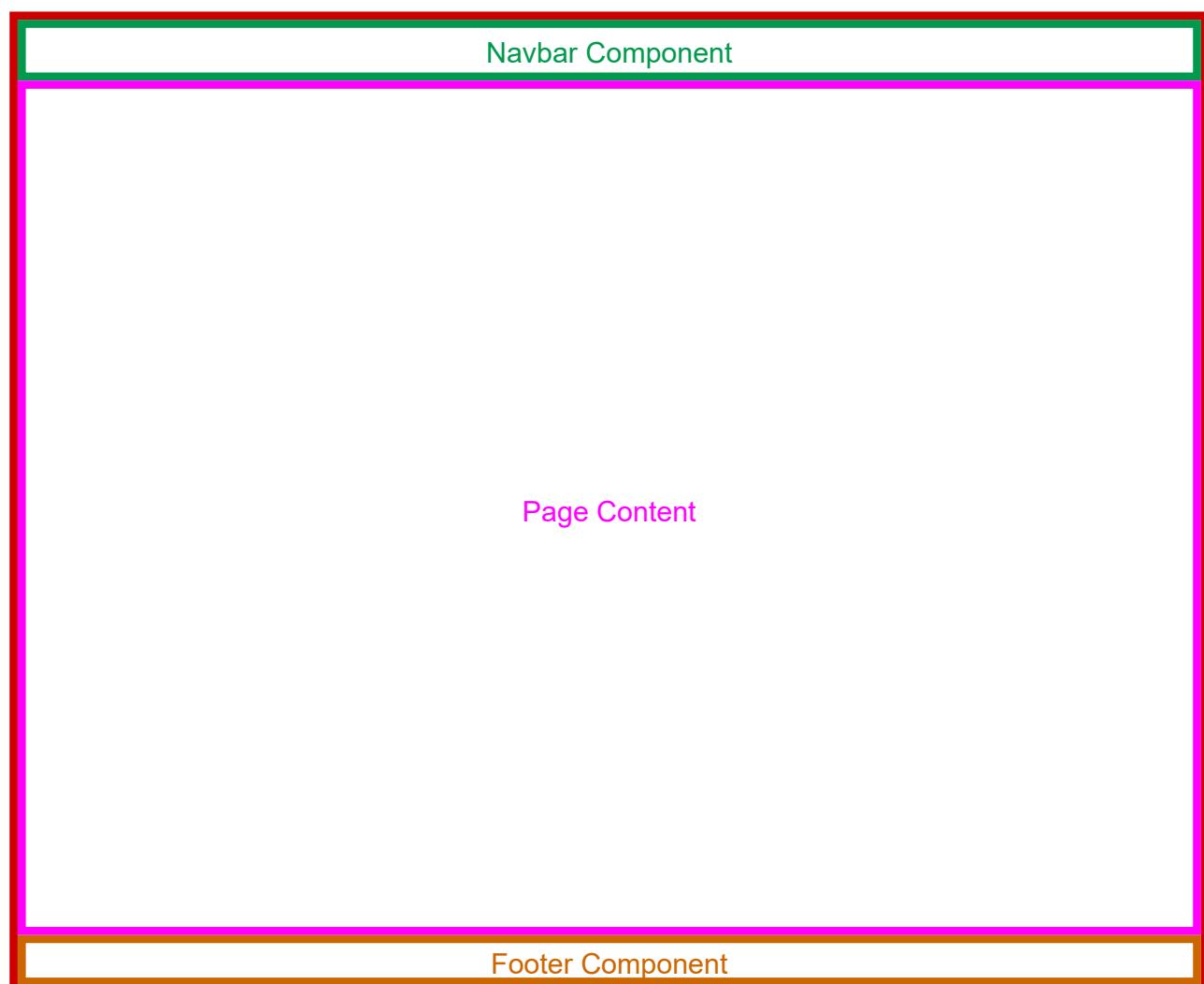


Abbildung 114: App Layout

Das App Layout ist das standardmäßige Layout, es besteht aus der Navigationsleiste und dem Footer, dazwischen befindet sich der Inhalt der Webseite.

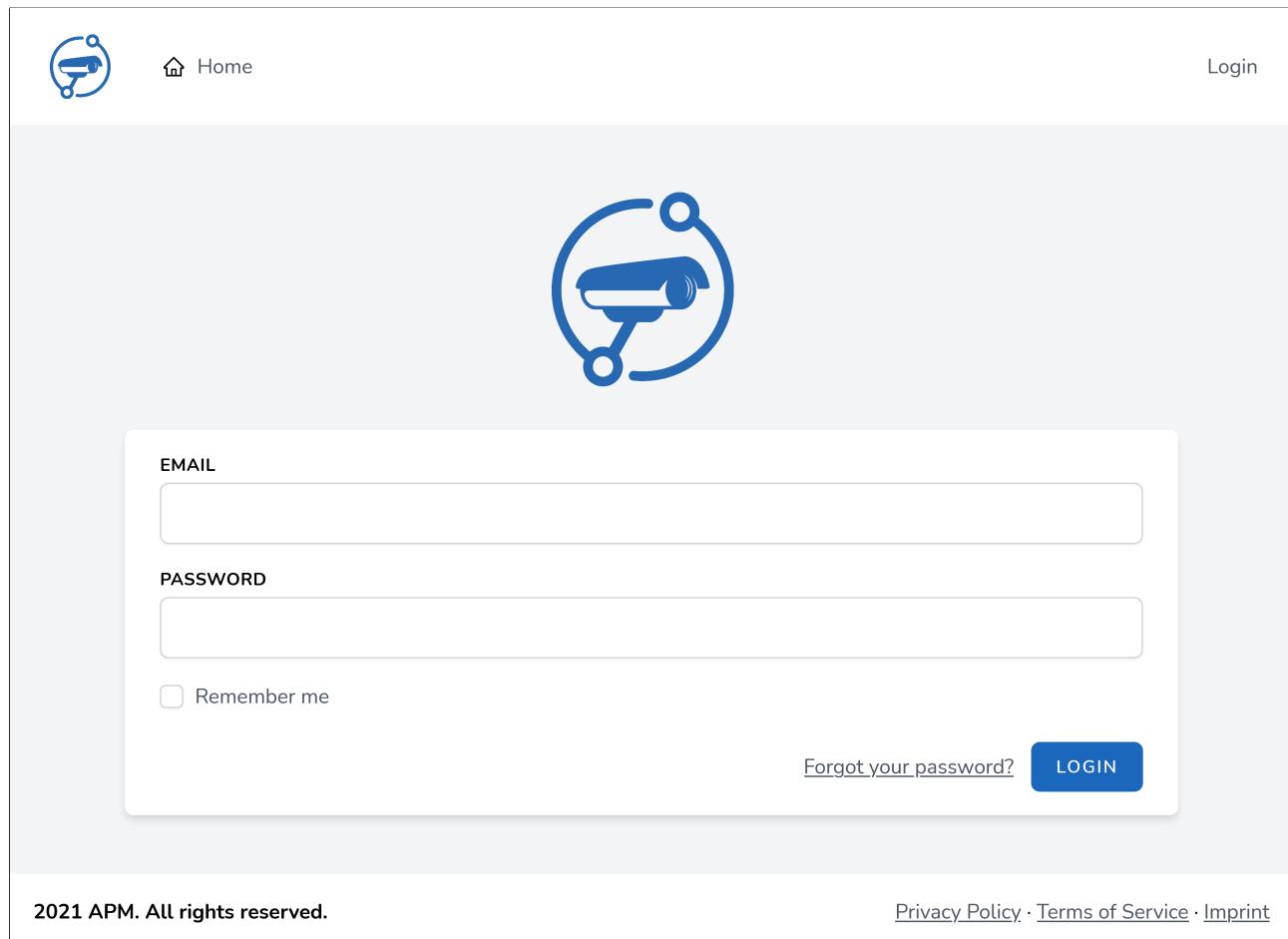


Abbildung 115: Login Seite mit App Layout

In der Abbildung 115 wird das App Layout bei der Login Seite verwendet.

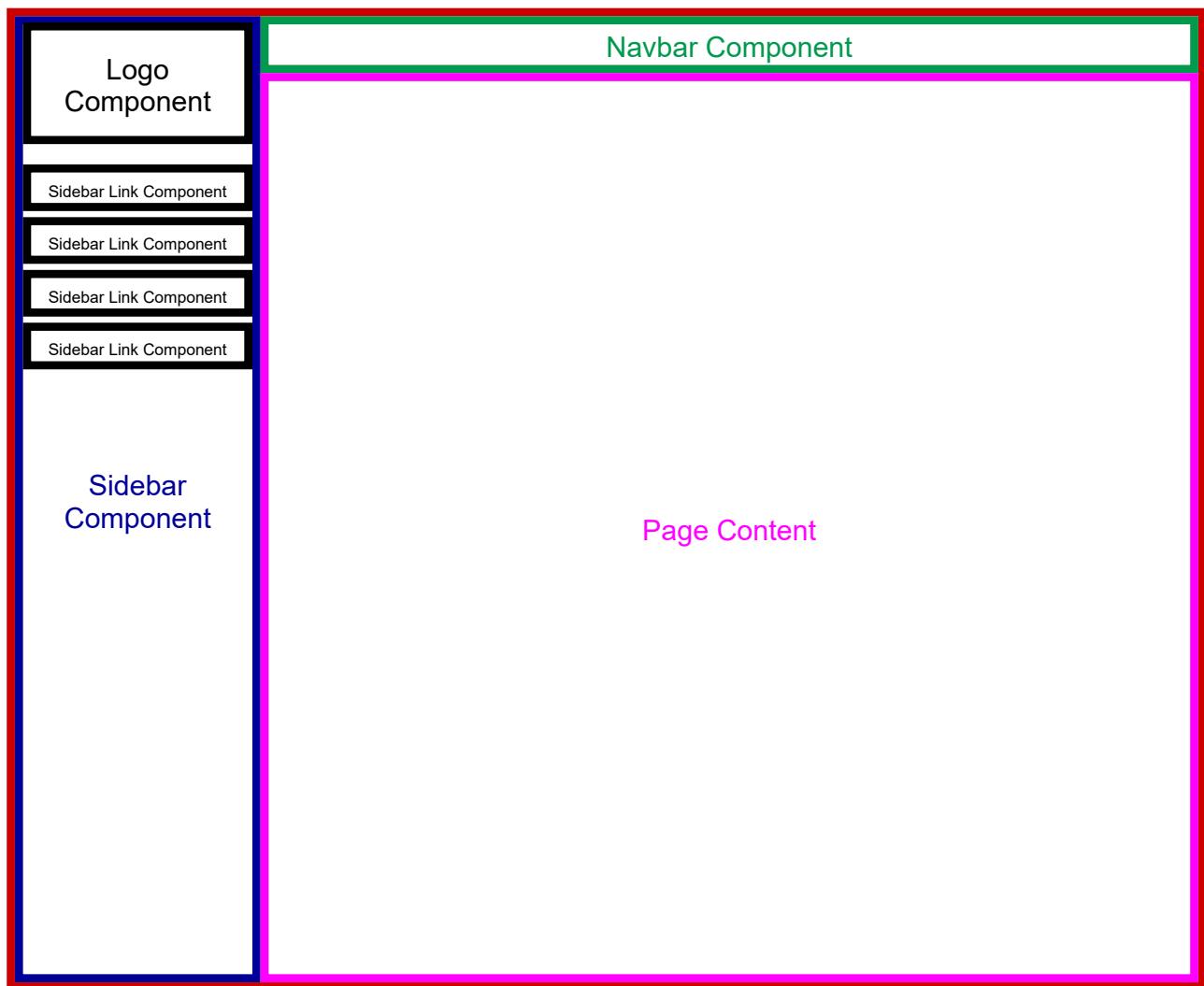


Abbildung 116: Admin Layout

Das Admin Layout besitzt eine zusätzliche Sidebar mit Links zu verschiedenen administrativen Seiten, im Vergleich zum App Layout, besitzt das Admin Layout keinen Footer und eine leicht veränderte Navbar<sup>37</sup>.

<sup>37</sup>Navigation Bar engl. für Navigationsleiste

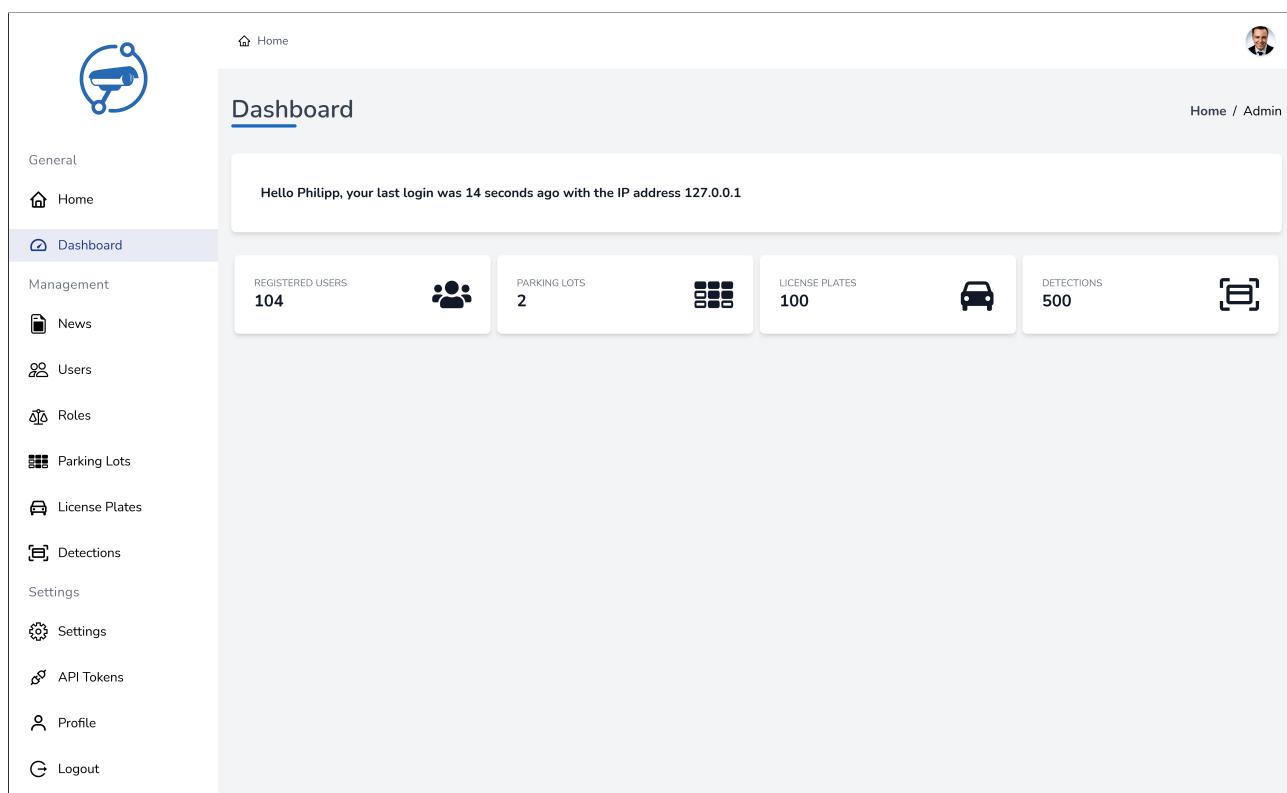


Abbildung 117: Dashboard mit Admin Layout

Im Dashboard wird das Admin Layout verwendet, somit befindet sich auf der linken Seite nun eine Sidebar mit Links zu anderen administrativen Seiten.

Um nun die Funktion der Webseite besser zu verstehen, wird nun auf den Code einzelner wichtiger Components genauer eingegangen.

### 8.7.5 Navbar

Die Navigationsleiste, auch Navbar genannt, ist ein zentrales Element der Webseite, um verschiedene Seiten anzusteuern. Dabei veränderte sich diese dynamisch, je nachdem, ob man eingeloggt ist oder ob der Betreiber der Seite ein bestimmtes Feature wie das Registrieren von neuen Benutzern deaktiviert hat.

```
1 @if (!(Request::is('admin/*') || Request::is('admin')))  
2     <a class="flex title-font font-medium items-center text-gray-900 mr-8">  
3         <x-application-logo class="max-h-12" />  
4     </a>  
5     @if (!Auth::guest())  
6         <a href="{{ route('dashboard') }}>{{ __('Dashboard') }}</a>  
7     @endif  
8     @endif
```

Code 94: Ausschnitt 1 Navigationsleiste

Der Code 94 regelt das Aussehen der Navigationsleiste, wenn ein Benutzer eingeloggt ist. Dabei wird kein Logo angezeigt, wenn der Benutzer eine administrative Seite öffnet, da das Logo bereits in der Sidebar vorhanden ist. Ebenfalls wird auf nicht administrativen Seiten ein *zurück zum Dashboard Link* angezeigt.

[Dashboard](#) [Home](#)

Abbildung 118: Navbar Variante 1

[Home](#)

Abbildung 119: Navbar Variante 2

```

1  @if (Auth::guest())
2    <div class="flex items-center">
3      <a href="{{ route('login') }}>{{ __('Login') }}</a>
4      @if (Route::has('register'))
5        <a href="{{ route('register') }}>{{ __('Register') }}</a>
6      @endif
7    </div>
8  @endif

```

Code 95: Ausschnitt 2 Navigationsleiste

Im zweiten Ausschnitt der Navigationsleiste geht es um die Login/Register Links, diese werden nur nicht eingeloggten Benutzern angezeigt. Ob der Register Link angezeigt wird ist abhängig, davon ob die Registrierung in den Seiten Einstellungen erlaubt ist.



Abbildung 120: Navbar Variante 3

### 8.7.6 Content Header

Um die Navigation zu erleichtern, besitzt das Webinterface bei Seiten mit dem Admin Layout unter der Navigationsleiste einen Content Header. Der Content Header setzt sich aus dem Titel der jeweiligen Seite und den sogenannten Breadcrumbs<sup>38</sup> zusammen. Dabei befindet sich ganz links der aktuelle Seitentitel, welcher ebenfalls im Browser Tab angezeigt wird und rechts die Brotkrümelnavigation. Die Breadcrumbs geben dabei dem Benutzer an, in welcher Verzweigung er sich aktuell befindet.

Die Breadcrumbs werden aus der URL aufgebaut, dabei werden die einzelnen Segmente zerlegt und in einer Liste horizontal mit Hyperlinks zusammengeführt.

<sup>38</sup>engl. für Brotkrümel, entspricht einer Navigation

```

1 <li><a href="/" class="text-gray-700 font-bold">Home</a></li>
2 <?php $link = ''; ?>
3 @for ($i = 1; $i <= count(Request::segments()); $i++)
4 @if (($i < count(Request::segments())) & ($i > 0))
5 <?php $link .= '/' . Request::segment($i); ?>
6 <li><span class="mx-2"/></li>
7 <li><a href="<?= $link ?>" class="text-gray-700 font-bold">{{
8   ucwords(str_replace('-', ' ', Request::segment($i))) }}</a></li>
9 @else
10 <li><span class="mx-2"/></li>
11 <li>{{ ucwords(str_replace('-', ' ', Request::segment($i))) }}</li>
12 @endif
13 @endfor

```

Code 96: Erstellung der Breadcrumbs

## Edit License Plate

Home / Admin / Plates / 1 / Edit

Abbildung 121: Content Header

### 8.7.7 Session Alerts

Um einen Benutzer zu informieren, dass eine Anfrage erfolgreich ausgeführt wurde oder ob es einen Fehler gab, werden *Session Alerts* verwendet.

Wird beispielsweise ein neues Kennzeichen im System angelegt, so wird der Benutzer nicht nur zu der Liste aller Kennzeichen zurück geleitet, sondern es wird in der Session des jeweiligen Benutzers eine Nachricht mitgespeichert.

```

1 <?php
2 return redirect(route('plates.index'))->with('success', __('The license
→ plate was successfully created!'));

```

Code 97: Controller mit success Nachricht

Um diese Nachricht nun anzuzeigen wurde ein Component entworfen welcher nur angezeigt wird, wenn eine solche Session Nachricht vorhanden ist. Neben einer *success* Nachricht gibt es auch noch eine *error* Nachricht, diese wird beispielsweise bei fehlerhaften Angaben in einem Formular angezeigt.

```

1 @if (session('success'))
2 <div class="bg-green-200 px-6 py-4 mb-4 rounded-md text-lg flex
→ items-center justify-items-stretch">
3   <span class="text-green-800">{{ session('success') }}</span>
4 </div>
5 @endif

```

Code 98: Session Alert Component

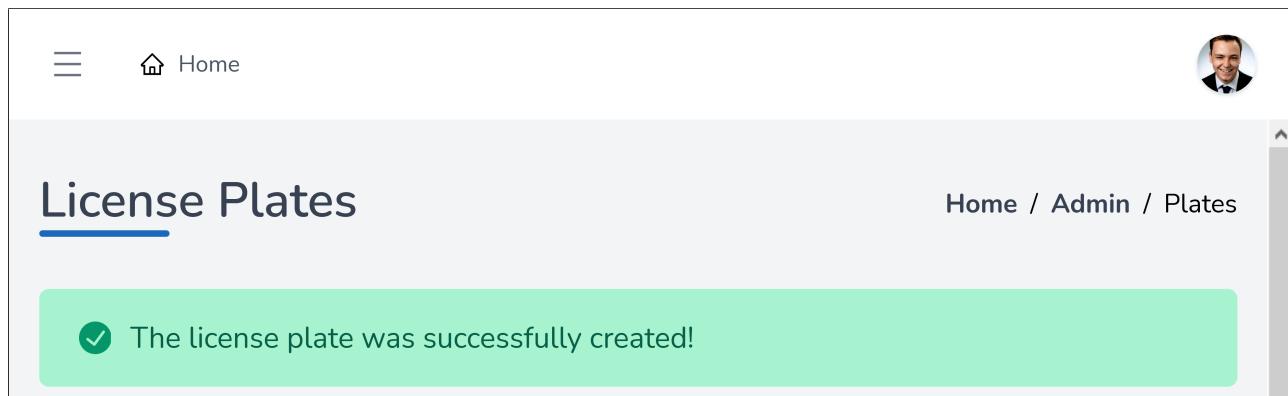


Abbildung 122: Success Session Alert

### 8.7.8 Sidebar

Die Sidebar ist das zentrale Element für administrative Benutzer des Webinterfaces. Über die Sidebar sind alle wichtigen Seiten auf einen Blick zu sehen und erreichen, dabei spielt es keine Rolle ob auf dem Smartphone oder auf dem Desktop, denn die Sidebar wurde so implementiert, dass sie auf kleineren Auflösungen eingeklappt wird und nur auf Wunsch mit einem Klick auf das Hamburger-Menü-Symbol<sup>39</sup> in der Navigationsleiste geöffnet wird, somit bleibt mehr Platz für das Darstellen anderer Elemente.

Für das dynamische Verhalten der Sidebar wird JavaScript verwendet. Es wird kein Vanilla<sup>40</sup> JavaScript verwendet, sondern AlpineJS<sup>41</sup>. Dabei folgt das Lightweight Framework einem ähnlichen Prinzip wie TailwindCSS nur eben für JavaScript.

Die Sidebar Einträge bestehen aus zwei Components, dem Sidebar Header Component und dem Sidebar Link Component. Der Sidebar Header Component ist da, um die verschiedenen Links in verschiedene Sektionen aufzuteilen, damit die Übersichtlichkeit gewährt bleibt.

1    <**x-sidebar-header**>{{ \_\_('General') }}</**x-sidebar-header**>

Code 99: Sidebar Header

Der Sidebar Link Component ist für die Links zuständig. Dieser besteht aus einem SVG-Icon und dem Namen der Seite. Die Route wird dem Component über :href übergeben und nicht mit href, da die URL mit der Helper Methode route angegeben wird und sonst von Blade nicht kompiliert wird und schlussendlich der zu kompilierende Code im HTML steht.

<sup>39</sup>Drei waagrechte Striche ähneln einem Hamburger

<sup>40</sup>Bedeutet so viel wie ohne Zusätze

<sup>41</sup><https://github.com/alpinejs/alpine>

```
1 <x-sidebar-link :href="route('home')"  
2   ↳ :active="request()->routeIs('home')">  
3   <span class="iconify h-6 w-6" data-icon="bx:bx-home"  
4     ↳ data-inline="false"></span>  
5   <span class="mx-3">{{ __('Home') }}</span>  
6 </x-sidebar-link>
```

Code 100: Sidebar Link

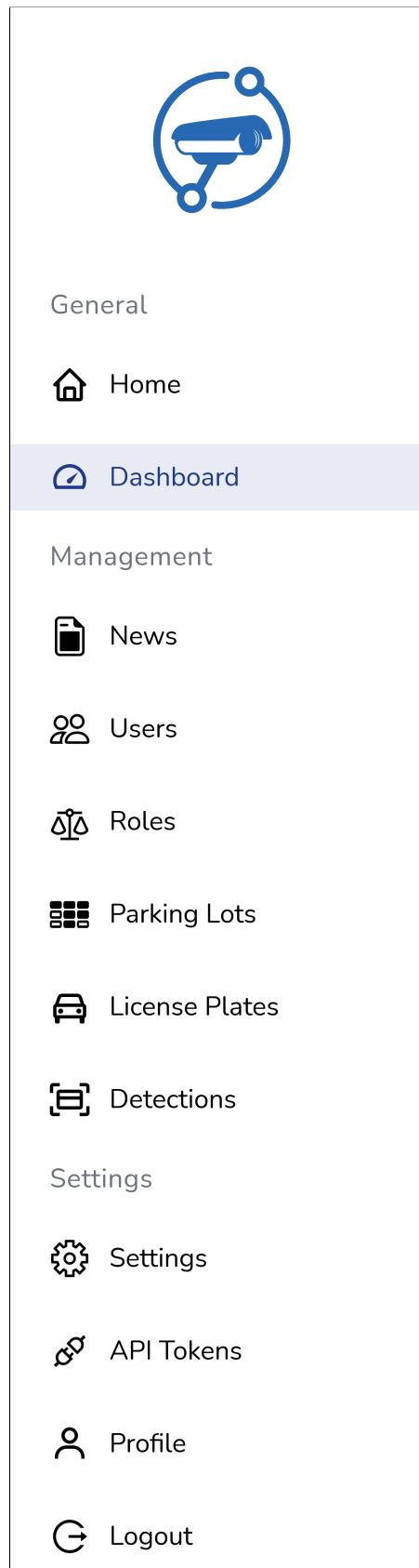


Abbildung 123: Sidebar Desktop

### 8.7.9 Footer

Der Footer ist dazu da, um Informationen rund um Impressum, Datenschutzverweis, Geschäftsbedingungen und Urheberrechtshinweis darzustellen.

Grundsätzlich besitzt der Footer des Webinterfaces auf der linken Seite einen variablen Text der sich frei über die Seiten Einstellungen konfigurieren lässt und auf der rechten Seiten Verweise auf die Datenschutzbestimmungen, allgemeine Geschäftsbedingungen und Impressum, welche sich ebenfalls über die Seiten Einstellungen konfigurieren lassen. Befindet sich auf diesen Seiten kein Inhalt, werden diese auch nicht im Footer angezeigt.

2021 APM. All rights reserved.

[Privacy Policy](#) · [Terms of Service](#) · [Imprint](#)

Abbildung 124: Footer

## 8.8 Funktionen

### 8.8.1 Einleitung

Das Webinterface besitzt eine große Anzahl an Features und Konfigurationsmöglichkeiten. In den folgenden Seiten werden alle Funktionen ausführlich mit dem technischen Hintergrund erklärt.

### 8.8.2 Dashboard

Das Dashboard ist praktisch die Startseite für eingeloggte Benutzer, als Erstes wird dem Benutzer eine Sicherheitsnachricht über den letzten Login angezeigt. Dabei wird die Zeit seit dem letzten Login angegeben und die dabei verwendete IP-Adresse.

Um diese Daten zu speichern, wird im `AuthenticatedSessionController` die `store` Methode ergänzt. Dabei wird, nachdem sich der Benutzer erfolgreich authentifiziert hat, die IP-Adresse und der aktuelle Zeitpunkt in die Datenbank geschrieben.

```
1 <?php
2 $user = Auth::user();
3 $user->last_login_at = Carbon::now()->toDateTimeString();
4 $user->last_login_ip = $request->getClientIp();
5 $user->save();
```

Code 101: AuthenticatedSessionController.php Store Methode

The screenshot shows a web application dashboard. At the top left is the word "Dashboard" underlined in blue. At the top right is a "Home / Admin" link. Below this, there is a large white box containing a message: "Hello Philipp, your last login was 31 minutes ago with the IP address 127.0.0.1".

Abbildung 125: Dashboard Sicherheitsnachricht

Neben der Sicherheitsnachricht werden einige statistische Zahlen über das System dargestellt:

- Anzahl der registrierten Benutzer
- Anzahl der Parkplätze im System
- Anzahl der bekannten Kennzeichen
- Anzahl der erkannten Kennzeichens

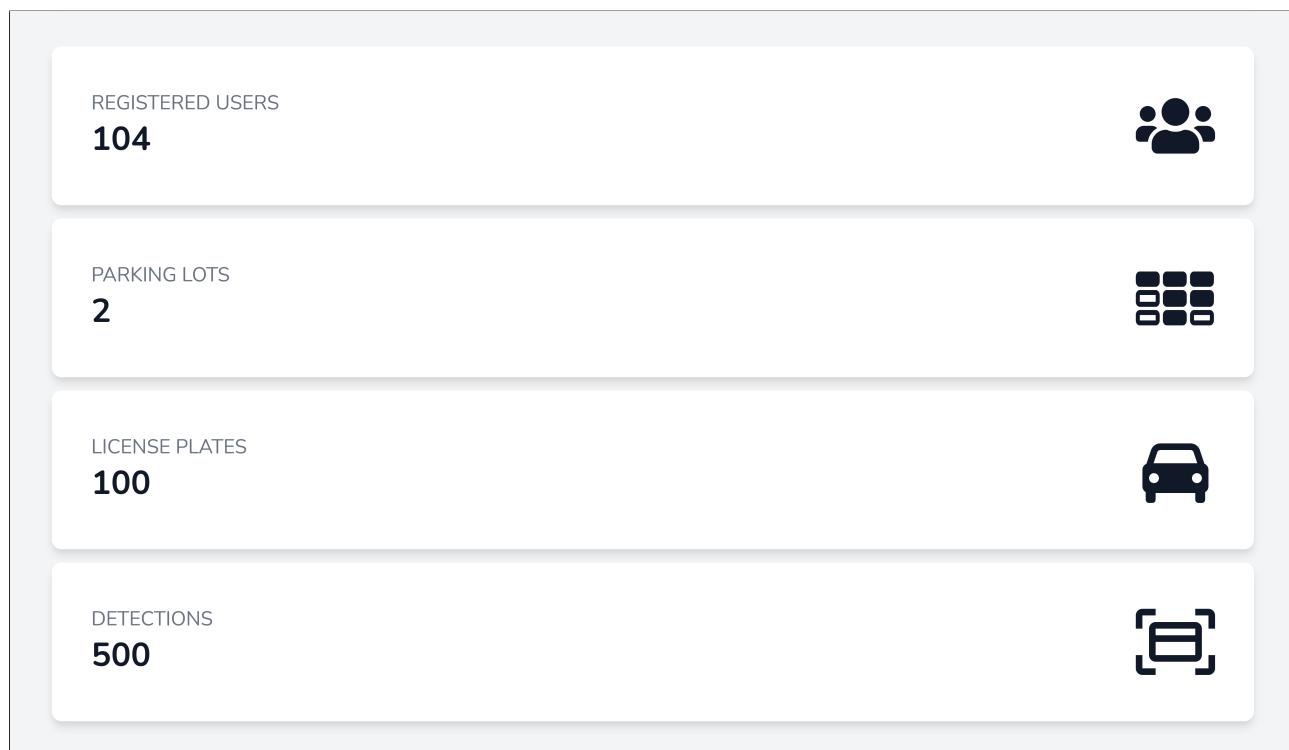


Abbildung 126: Dashboard Statistik

### 8.8.3 Benutzerverwaltung

Die Benutzerverwaltung ist ein zentrales Steuerelement des Webinterfaces und wichtig für die Zugriffskontrolle. Für die Benutzerverwaltung sind alle CRUD-Operationen<sup>42</sup> verfügbar.

<sup>42</sup>Grundlegende Operationen Create, Read, Update und Delete von Daten

Users
id (bigint)
first_name (string)
last_name (string)
email (string)
avatar (string)
email_verified_at (datetime)
password (string)
remember_token (string)
created_at (datetime)
updated_at (datetime)

Abbildung 127: Benutzer ER

#### 8.8.3.1 Liste aller Benutzer

Die Startseite der Benutzerverwaltung ist eine Liste von allen Benutzern im System. Dort sind die wichtigsten Informationen direkt abzulesen wie Name, Rolle und E-Mail.

Users						Home / Admin / Users
USERS						CREATE NEW USER
User	Roles	Email	Created	Updated	Actions	
 Philipp Kraft	<span>Owner</span>	Mail@Philipp-Kraft.com	1 day ago	1 hour ago	  	
 Dennis Köb	<span>Owner</span>	Dennis.Koeb@gmail.com	1 day ago	1 day ago	  	
 Samuel Bleiner	<span>Owner</span>	Bleiner.Samuel@gmail.com	1 day ago	1 day ago	  	
 Christoph Stüttler	<span>Owner</span>	Christoph.Stuettler@htl-rankweil.at	1 day ago	1 day ago	  	

Abbildung 128: Liste aller Benutzer

### 8.8.3.2 Benutzer erstellen/editieren

Neue Benutzer können mit dem Button rechts oben über Create new user angelegt werden. Einzelne Benutzer besitzen drei mögliche Aktionen in der letzten Spalte, dabei gibt es Betrachten (Blau), Editieren (Gelb) und Löschen (Rot). Es können folgende Informationen eines Benutzers konfiguriert werden:

- Profilbild
- Vorname
- Nachname
- Email
- Passwort
- Rollen

## Edit User

Home / Admin / Users / 1 / Edit



Philipp Kraft

### General Settings

FIRST NAME

Philipp

LAST NAME

Kraft

EMAIL

Mail@Philipp-Kraft.com

### Account Security

NEW PASSWORD

CONFIRM NEW PASSWORD

### Roles

Owner  
 Editor

Administrator  
 Viewer

[DELETE USER](#)

[SAVE USER](#)

Abbildung 129: Benutzer editieren

### 8.8.3.3 Profilbilder

Jeder Benutzer hat die Möglichkeit, ein Profilbild zu besitzen und dies wird überall verwendet, wo ein Benutzer erwähnt wird.

Um ein Bild hochzuladen, muss beim editieren bzw. erstellen, auf das aktuelle Profilbild bzw. auf das Standardprofilbild geklickt werden, es öffnet sich nun ein Upload-Dialog des Betriebssystems und erlaubt es ein Bild auszuwählen.

```

1  <div class="flex justify-center items-center mt-6">
2    <label for="avatar">
3      <input type="file" name="avatar" id="avatar" style="display:none;" />
4      
6    </label>
7  </div>

```

Code 102: Profilbild Upload

Bevor das Bild nun auf dem Server gespeichert wird, ist es notwendig, das Bild mit der PHP Image Library **Intervention**<sup>43</sup> auf die passende Größe zu skalieren. Nach diesem Schritt wird das Bild unter dem Pfad `/uploads/avatars/` mit einem zufälligem Namen mithilfe des aktuellen Datums abgespeichert und mit dem Benutzer in der Datenbank assoziiert.

```

1  <?php
2
3  if ($request->hasFile('avatar')) {
4
5    $avatar = $request->file('avatar');
6
7    $filename = time() . '.' . $avatar->getClientOriginalExtension();
8
9
10   Image::make($avatar)->resize(600, 600, function ($constraint) {
11
12     $constraint->aspectRatio();
13
14   })->save(public_path('/uploads/avatars/' . $filename));
15
16
17   $user->avatar = $filename;
18
19 }

```

Code 103: UserController.php Profilbild Upload

---

<sup>43</sup><http://image.intervention.io/>

#### 8.8.4 Rollen- und Rechteverwaltung

Die Rollen- und Rechteverwaltung ist eng mit der Benutzerverwaltung verbunden. Dabei ist das System so aufgebaut, dass es verschiedene Rechte für bestimmte Anfragen gibt (z.B. Benutzer editieren), diese Vielzahl an Rechten können verschiedenen Gruppen zugewiesen werden. Schliessendlich kann man diese Gruppen einzelnen Benutzern zuweisen.

Es gibt nun drei relevante Modelle (*User*, *Role* und *givePermissionBySlug*), dabei besteht zwischen dem User und dem Role Model eine *m zu n* Beziehung. Zwischen dem Role Model und dem Permission Model besteht auch eine eine *m zu n* Beziehung.

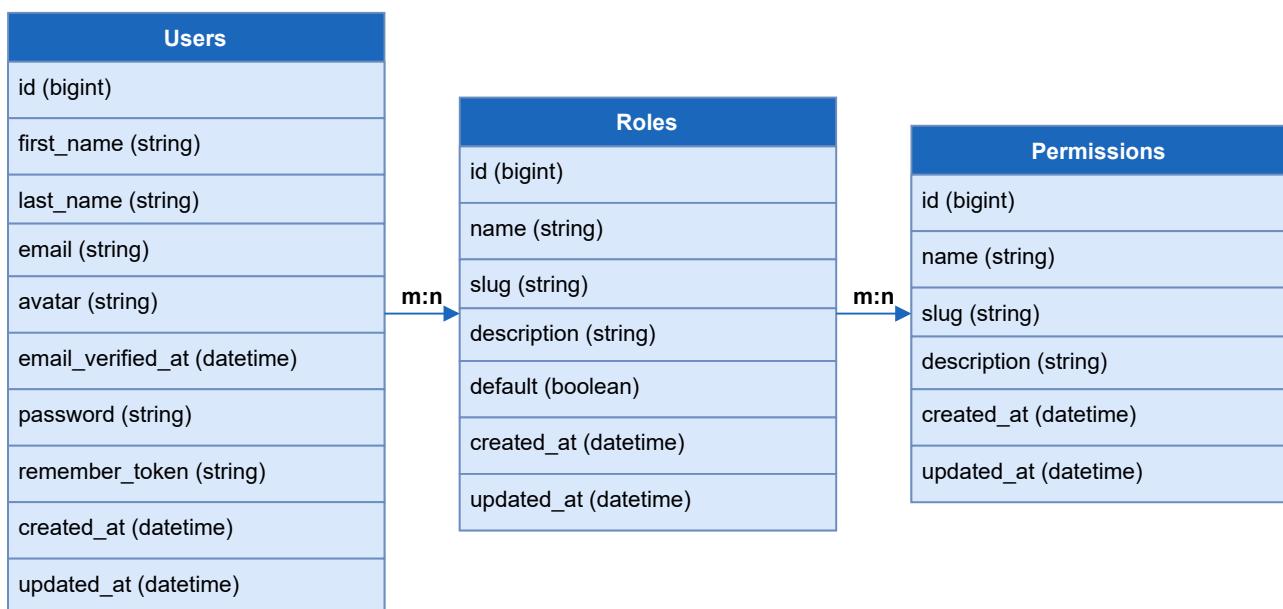


Abbildung 130: Benutzer, Rollen und Rechte ER

##### 8.8.4.1 Liste der verfügbaren Rechte

Die Rechte von einzelnen Funktionen sind ähnlich aufgebaut:

- **Index <Model>**: Liste aller einzelnen Elemente der Model
- **Create <Model>**: Formular für das Erstellen eines neuen Elements der Model
- **Store <Model>**: Neues Element der Model in die Datenbank speichern

- **Show <Model>**: Einzelnes Element der Model anzeigen
- **Edit <Model>**: Formular für das Editieren eines Elements der Model
- **Update <Model>**: Vorhandenes Element der Model in der Datenbank aktualisieren
- **Destroy <Model>**: Element der Model aus der Datenbank löschen

Das Webinterface besitzt 43 verschiedene Rechte.

Name	Slug	Beschreibung
Index Post	index-post	Liste aller Neuigkeit anzeigen
Create Post	create-post	Neuigkeit erstellen
Store Post	store-post	Neuigkeit speichern
Show Post	show-post	Neuigkeit anzeigen
Edit Post	edit-post	Neuigkeit editieren
Update Post	update-post	Neuigkeit aktualisieren
Destroy Post	destroy-post	Neuigkeit löschen
Index User	index-user	Liste aller Benutzer anzeigen
Create User	create-user	Benutzer erstellen
Store User	store-user	Benutzer speichern
Show User	show-user	Benutzer anzeigen
Edit User	edit-user	Benutzer editieren
Update User	update-user	Benutzer aktualisieren
Destroy User	destroy-user	Benutzer löschen
Index License Plate	index-license-plate	Liste aller Kennzeichen anzeigen
Create License Plate	create-license-plate	Kennzeichen erstellen
Store License Plate	store-license-plate	Kennzeichen speichern
Show License Plate	show-license-plate	Kennzeichen anzeigen
Edit License Plate	edit-license-plate	Kennzeichen editieren
Update License Plate	update-license-plate	Kennzeichen aktualisieren
Destroy License Plate	destroy-license-plate	Kennzeichen löschen
Index Detection	index-detection	Liste aller Erkennungen anzeigen

Index Parking Lot	index-parking-lot	Liste aller Parkplätze anzeigen
Create Parking Lot	create-parking-lot	Parkplätze erstellen
Store Parking Lot	store-parking-lot	Parkplätze speichern
Edit Parking Lot	edit-parking-lot	Parkplätze editieren
Update Parking Lot	update-parking-lot	Parkplätze aktualisieren
Destroy Parking Lot	destroy-parking-lot	Parkplätze löschen
Store Parking Spot	store-parking-spot	Parkplätze speichern
Destroy Parking Spot	destroy-parking-spot	Parkplätze löschen
Index Role	index-role	Liste aller Rollen anzeigen
Create Role	create-role	Rolle erstellen
Store Role	store-role	Rolle speichern
Show Role	show-role	Rolle anzeigen
Edit Role	edit-role	Rolle editieren
Update Role	update-role	Rolle aktualisieren
Destroy Role	destroy-role	Rolle löschen
Index Setting	index-setting	Liste aller Einstellungen anzeigen
Update Setting	update-setting	Einstellungen aktualisieren
Index Token	index-token	Liste aller API Schlüssel anzeigen
Store Token	store-token	API Token speichern
Destroy Token	destroy-token	API Token löschen

Tabelle 7: Verfügbare Rechte

#### 8.8.4.2 Vergabe von Rechten und Rollen mit Slugs

Da es bei der großen Anzahl von verschiedenen Rechten kompliziert wäre, die Rechte und Rollen mit einer ID zu vergeben wird ein Slug<sup>44</sup> verwendet.

<sup>44</sup>Benutzerfreundlicher Name

```
1 <?php
2 public function givePermissionBySlug($slugs)
3 {
4     if (!isset($slugs))
5         return;
6
7     foreach ($slugs as $slug) {
8         $permission = Permission::where('slug', $slug)->first();
9         $this->permissions()->attach($permission);
10    }
11 }
```

Code 104: Permission Vergabe Methode

Mit der `givePermissionBySlug` Methode ist es nun möglich Rollen eine Vielzahl von Rechten zuzuordnen, dabei muss nur ein Array mit den Slugs der einzelnen Rechte übergeben werden.

Im Code 105 vom RoleSeeder wird eine neue Rolle mit dem Namen **Administrator** und dem Slug **admin** erstellt, gleichzeitig werden dieser Gruppe einige Rechte zugewiesen.

```
1 <?php
2 Role::create(['name' => 'Administrator', 'slug' => 'admin'])
3 ->givePermissionBySlug([
4     'index-user', 'create-user', 'store-user', 'show-user', 'edit-user'
5 ]);
```

Code 105: Rolle erstellen und Rechte zuweisen

Eine ähnliche Methode wird verwendet, um einem Benutzer eine oder mehrere Rollen zuzuordnen.

#### 8.8.4.3 Authentisierung

Für das Überprüfen, ob ein Benutzer die korrekten Rechte besitzt, werden Laravel Policies verwendet.

Es wird für jedes Modell eine Policy erstellt und für jedes Recht eine Methode. Diese Methoden werden nun innerhalb eines Controllers aufgerufen, liefert die Methode `true` wird die Anfrage fortgesetzt, liefert die Methode `false` erhält der Benutzer einen 403 Forbidden HTTP Error.

```
1 <?php
2 public function index(User $user)
3 {
4     $permission = Permission::where('slug', 'index-user')->first();
5     return $user->hasRole($permission->roles);
6 }
```

Code 106: UserPolicy Index

```
1 <?php
2 public function index()
3 {
4     $this->authorize('index', User::class);
5     . . .
6 }
```

Code 107: UserController Index

#### 8.8.4.4 Rolle erstellen

Eine neue Rolle kann über Roles ► Create new role erstellt werden. Dabei können folgende Informationen konfiguriert werden:

- Rollename
- Slug
- Beschreibung
- Rechte

- Standardrolle

Die gewünschten Rechte werden mithilfe von Checkboxen ausgewählt. Die zusätzliche Option Standardrolle erlaubt es, diese Rolle automatisch neu registrierten Benutzern zuzuweisen.

### Create Role

Home / Admin / Roles / Create

**New role**

**General Settings**

NAME	Tester	SLUG	tester
DESCRIPTION	Eine Rolle für Tester		

**Permissions**

<input checked="" type="checkbox"/> Index Post <input checked="" type="checkbox"/> Store Post <input checked="" type="checkbox"/> Edit Post <input type="checkbox"/> Destroy Post <input type="checkbox"/> Create User <input type="checkbox"/> Show User <input type="checkbox"/> Update User <input type="checkbox"/> Index License Plate <input type="checkbox"/> Store License Plate <input type="checkbox"/> Edit License Plate <input type="checkbox"/> Destroy License Plate <input type="checkbox"/> Index Parking Lot <input type="checkbox"/> Store Parking Lot <input type="checkbox"/> Edit Parking Lot <input type="checkbox"/> Destroy Parking Lot <input type="checkbox"/> Destroy Parking Spot <input type="checkbox"/> Create Role <input type="checkbox"/> Show Role <input type="checkbox"/> Update Role <input type="checkbox"/> Index Setting <input type="checkbox"/> Index Token <input type="checkbox"/> Destroy Token	<input checked="" type="checkbox"/> Create Post <input checked="" type="checkbox"/> Show Post <input checked="" type="checkbox"/> Update Post <input type="checkbox"/> Index User <input type="checkbox"/> Store User <input type="checkbox"/> Edit User <input type="checkbox"/> Destroy User <input type="checkbox"/> Create License Plate <input type="checkbox"/> Show License Plate <input type="checkbox"/> Update License Plate <input type="checkbox"/> Index Detection <input type="checkbox"/> Create Parking Lot <input type="checkbox"/> Show Parking Lot <input type="checkbox"/> Update Parking Lot <input type="checkbox"/> Store Parking Spot <input type="checkbox"/> Index Role <input type="checkbox"/> Store Role <input type="checkbox"/> Edit Role <input type="checkbox"/> Destroy Role <input type="checkbox"/> Update Setting <input type="checkbox"/> Store Token
---	---

**Other Settings**

DEFAULT ROLE

Add new users automatically to this role

SAVE NEW ROLE

Abbildung 131: Benutzer erstellen

### 8.8.5 Neuigkeiten

Das Neuigkeiten Feature ist eine Möglichkeit, um Besucher des Webinterfaces rasch über wichtige Neuigkeiten zu informieren. Dabei ist das System ähnlich wie ein Blog aufgebaut und zeigt die einzelnen News auf der Startseite der Webseite an. Es sind alle CRUD-Operationen verfügbar.

## Edit News

[Home](#) / [Admin](#) / [Posts](#) / [1](#) / [Edit](#)

### Post Number 1

#### General Settings

**TITLE**

Post Number 1

**CONTENT**

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

body p

#### Other Settings

**PUBLISHED** Display this news on the home site**ANONYMOUS** Display this news with no author name[DELETE NEWS](#)[SAVE NEWS](#)

Abbildung 132: Neuigkeit editieren

Neben dem Titel und Inhalt gibt es noch zwei besondere Einstellungen:

- **Published**

Um eine Neuigkeit auf der Startseite Besucher anzuzeigen muss diese Einstellung aktiviert

sein.

- **Anonymous**

Ist es nicht erwünscht, dass der Name und das Profilbild des Autors auf der Startseite angezeigt wird, kann dies mit dieser Option deaktiviert werden. Anstatt des Namens wird *Anonymous user* angezeigt und das Standardprofilbild wird verwendet.

Für das editieren des Inhalts wird der quelloffene WYSIWYG-HTML-Editor CKEditor 5<sup>45</sup> verwendet.

Die Startseite besitzt einen modifizierten Content Header mit dem akutellen Datum und der Uhrzeit.

The screenshot shows the homepage of the HTBLuVA Rankweil website. At the top, it displays the date "TODAY 31. March 2021 16:13". Below this is a header with the word "NEWS" underlined. Two news items are listed:

- Post Number 1** (30. March 2021) by Philipp Kraft: The post content is a placeholder text in Latin: "Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet."
- Post Number 2** (30. March 2021) by Anonymous user: The post content is another placeholder text in Latin: "Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet."

Abbildung 133: Home Neuigkeiten

## 8.8.6 Parkplatzverwaltung

Ein Benutzer kann beliebig viele Parkplätze im Webinterface anlegen, ein Parkplatz besteht dabei aus einem Namen und einer Beschreibung, einem Parkplatz kann ebenfalls ein Bild hinzugefügt werden, ähnlich wie bei den Benutzern. In der Abbildung 134 ist zu erkennen, dass ein Parkplatz eine 1:n Beziehung zu den Parklücken (Parking Spots), sowie zu den Erkennungen (Detections) besitzt.

<sup>45</sup><https://ckeditor.com>



Abbildung 134: Parking Lots, Parking Spots und Detections ER

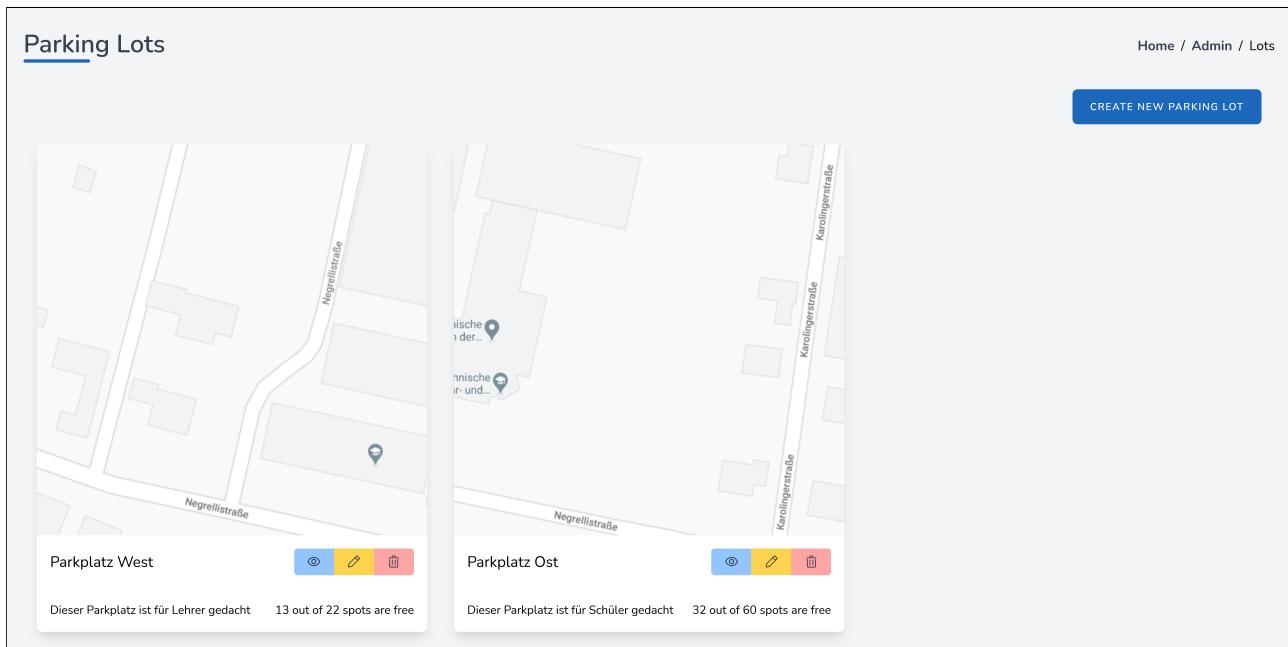


Abbildung 135: Übersicht aller Parkplätze

### 8.8.6.1 Parklücken

Eine Parklücke besteht aus einer Adresse und dem Status. Die Adresse ist eine von der Hardware gesetzte Zahl, um die einzelnen Parklücken mit der API zu identifizieren und anzusteuern. Der Status ist dafür da, um anzuzeigen, ob eine Parklücke besetzt oder frei ist.

### Show Parking Lot

[Home](#) / [Admin](#) / [Lots](#) / 3

PARKING SPOTS

ID	ADDRESS	STATUS	UPDATED	ACTIONS
3	100	Occupied	2021-02-10 13:41:36	
4	110	Free	2021-02-02 12:45:11	
5	120	Free	2021-02-08 02:20:36	
6	130	Occupied	2021-02-08 10:18:43	

DETECTIONS

LICENSE PLATE	PARKING LOT	NEW PARKING STATUS	ALLOWANCE STATUS	ASSOCIATED USER	DETECTED
B509DYY	No image available	Parking Space 1	Outside	Disallowed	Kristofer Bayer 2021-03-23 02:10:57
FK802HT	No image available	Parking Space 1	Outside	Allowed	Adonis Runolfsdottir 2021-03-16 05:27:25
FK12BR2I	No image available	Parking Space 1	Outside	Disallowed	Darrel Walsh 2021-03-03 06:09:57

Abbildung 136: Parklücken und Erkennungen eines Parkplatzes

### 8.8.7 Kennzeichenverwaltung

Die Kennzeichenverwaltung ist ein wichtiger Teil des Webinterfaces und für die Kennzeichenerkennung, die Kennzeichenverwaltung beherrscht alle CRUD-Operationen und steht in starker Abhängigkeit mit anderen Features wie dem Erkennungsverlauf oder der Parkplatzverwaltung.

Philipp Kraft

209

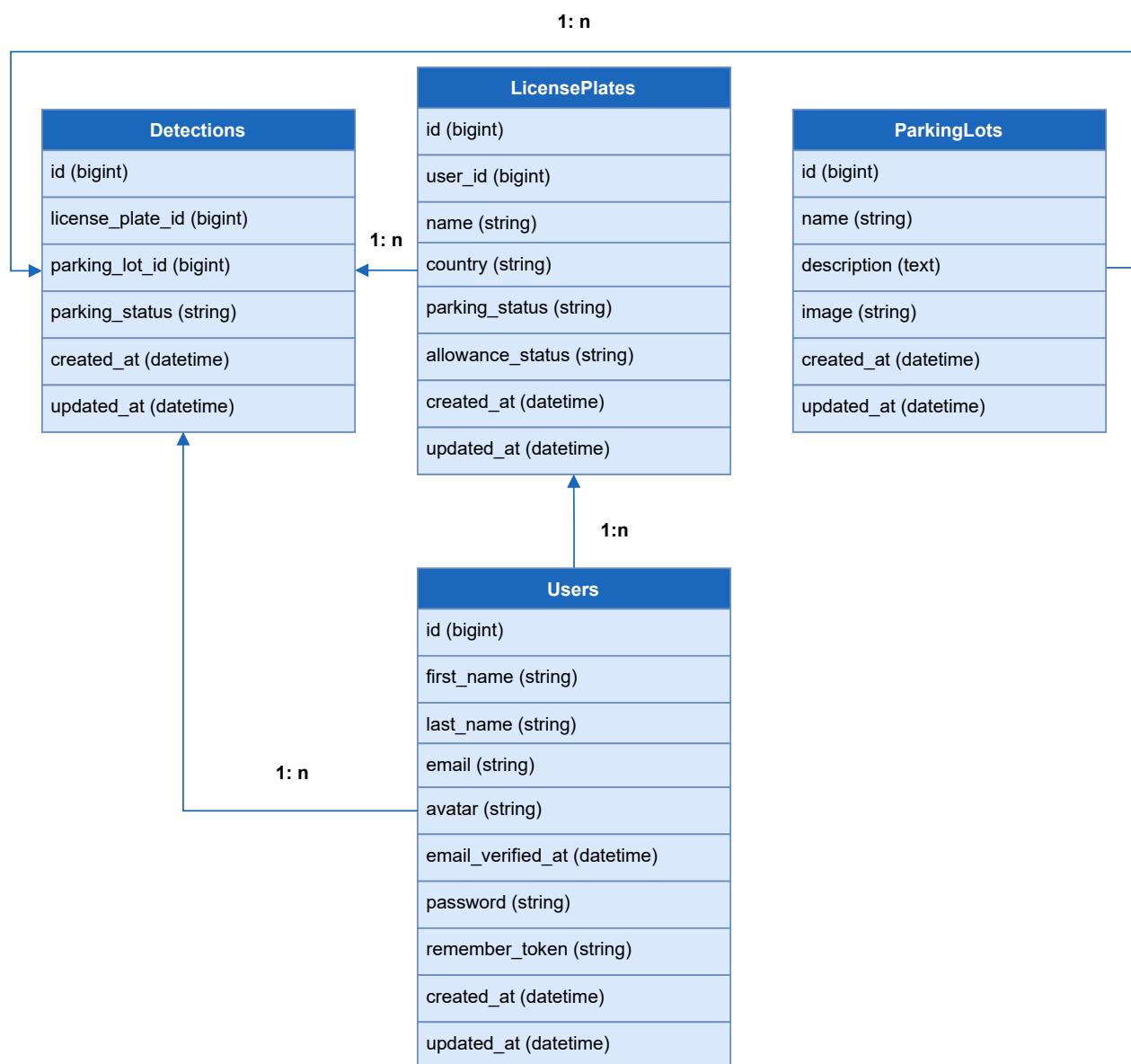


Abbildung 137: Kennzeichen, Parkplätze, Erkennungen und Benutzer Beziehung

Ein Kennzeichen besitzt folgende Informationen:

- **Assoziierter Benutzer**

Es besteht die Möglichkeit ein Kennzeichen einem Benutzer zuzuweisen.

- **Name des Kennzeichens**

Der Name des Kennzeichens wird durch die Kennzeichenerkennung vorgegeben, oder kann manuell verändert bzw. eingetragen werden.

- **Park Status** Der Park Status gibt an, ob sich ein Fahrzeug mit dem Kennzeichen gerade auf einem Parkplatz befindet (Inside) oder nicht (Outside).
- **Allowance Status** Der Allowance Status gibt an, ob ein Kennzeichen erlaubt, unerlaubt oder unbekannt ist.

Betrachtet man ein einzelnes Kennzeichen im Webinterface, ist es möglich, diesem einen Benutzer zuzuordnen bzw. manuelle Einstellungen vorzunehmen, beispielsweise bei einem fehlerhaften Park Status. Ebenfalls ist es direkt möglich alle aufgenommenen Erkennungen des Kennzeichens inklusive Bild zu betrachten.

### Show License Plate

Home / Admin / Plates / 1

**FK8VD0**

LICENSE PLATE  
FK8VD0

ASSOCIATED USER  
graham.greyson@example.org (No Role)

ALLOWANCE STATUS  
Unknown

OVERWRITE PARKING STATUS  
Inside

**EDIT LICENSE PLATE**

DETECTIONS					
LICENSE PLATE	PARKING LOT	NEW PARKING STATUS	ALLOWANCE STATUS	DETECTED	
FK8VD0	No image available	Parkplatz Ost	Inside	Unknown	2021-03-12 18:18:40
FK8VD0	No image available	Parkplatz West	Outside	Unknown	2021-03-07 17:37:40
FK8VD0	No image available	Parkplatz Ost	Inside	Unknown	2021-03-04 14:46:09

Abbildung 138: Liste aller Kennzeichen

License Plates						
LICENSE PLATES						CREATE NEW LICENSE PLATE
LICENSE PLATE	LAST PARKING LOT	PARKING STATUS	ALLOWANCE STATUS	ASSOCIATED USER	LAST DETECTED	ACTIONS
FK8VD0	Parkplatz Ost	Inside	Unknown	Favian Dicki	2021-03-12 18:18:40	
FK7770RE	Parkplatz West	Inside	Unknown	Theron Langosh	2021-03-20 02:29:09	

Abbildung 139: Einzelnes Kennzeichen

### 8.8.8 Erkennungsverlauf

Unter einer Erkennung bzw. Detection wird im Webinterface eine einzelne Kennzeichenaufnahme eines Fahrzeugs verstanden. Das bedeutet, ein Kennzeichen kann viele Erkennungen besitzen.

Eine erkennung besteht praktisch aus dem neuen Park Status, dem Foto des Kennzeichens von der Kennzeichenerkennung und einem genauen Zeitpunkt. Eine Erkennung kann nicht manuell über das Webinterface bearbeitet werden und ist nur durch die API veränderbar. Eine Erkennungen wird nicht nur mit einem Kennzeichen assoziiert und somit mit einem Benutzer, sondern auch mit einem Parkplatz. Die Kennzeichenerkennungen gibt bei einer API-Request einen Parkplatz an und verknüpft somit die Erkennung mit dem jeweiligen Parkplatz.

DETECTIONS						
LICENSE PLATE	PARKING LOT	NEW PARKING STATUS	ALLOWANCE STATUS	ASSOCIATED USER	DETECTED	
BZ12345	No image available	Parkplatz West	Inside	Unknown	No association	2021-04-01 09:25:54
BZ12345	No image available	Parkplatz West	Outside	Unknown	No association	2021-04-01 09:25:52
BZ12345	No image available	Parkplatz West	Outside	Unknown	No association	2021-04-01 09:25:51
BZ12345	No image available	Parkplatz West	Inside	Unknown	No association	2021-04-01 09:25:51
BZ12345	No image available	Parkplatz West	Inside	Unknown	No association	2021-04-01 09:25:47
DO6371K	No image available	Parkplatz West	Outside	Allowed	Zoie Jones	2021-03-23 15:37:40
B675UQH	No image available	Parkplatz West	Outside	Allowed	Kelton Hyatt	2021-03-23 15:23:20
DO743F7L	No image available	Parkplatz West	Inside	Disallowed	Adonis Runolfsdottir	2021-03-23 12:17:09

Abbildung 140: Liste aller Erkennungen

### 8.8.9 Seiten-Einstellungen

In den Seiten-Einstellungen ist es möglich, als Benutzer bestimmte Aspekte des Webinterfaces zu konfigurieren dazu zählt:

- Seiten Titel (key: site\_title)
- Standardsprache (key: default\_language)
- Seiten Logo
- Erlauben, dass sich neue Benutzer registrieren (key: allow\_new\_users)
- Footer Einstellungen (key: footer\_content)
- Inhalt der Datenschutzbestimmungsseite (key: privacy\_policy)
- Inhalt der Allgemeinen Geschäftsbedingungen (key: terms\_of\_service)
- Inhalt des Impressums (key: imprint)

### 8.8.9.1 Funktion der Einstellungen

Settings
id (bigint)
key (string)
value (text)

Abbildung 141: Einstellunge Tabelle

Die Einstellungen werden in der Datenbank gespeichert, in der Datenbank besteht eine Einstellung aus einem key und einer value. Der key ist dabei vorgegeben und die value kann frei durch die grafische Benutzeroberfläche konfiguriert werden.

Damit der Browser des Benutzers über die aktuellen Einstellungen Bescheid weiß, wird mithilfe eines Service Providers die in der Datenbank vorhandenen Einstellungen in die Laravel Konfiguration übernommen.

```

1 <?php
2
3 public function boot()
4 {
5     if (Schema::hasTable('settings')) {
6         config()->set('settings', \App\Models\Setting::pluck('value',
7             'key')->all());
8     }
9 }
```

Code 108: SettingsServiceProvider

Nun ist es möglich, im Projekt überall mit der config() Helper Methode die Seiten Einstellungen abzurufen. Somit ist es beispielsweise möglich, den Seiten Titel des Webinterfaces in den Layouts anzugeben.

```
1 <title>{{ config('settings.site_title') }}</title>
```

## Code 109: Seiten Titel Konfiguration

### 8.8.10 Profil

Das Profil ist dazu da, um seine eigenen Benutzerinformationen zu betrachten und zu bearbeiten. Es sind die gleichen Einstellungen wie in der Benutzerverwaltung einstellbar, jedoch nur über seinen eigenen Account. Eine zusätzliche Einstellung stellt die UI-Sprache dar, dort kann jeder Benutzer seine gewünschte UI-Sprache konfigurieren.

The screenshot shows the 'General Settings' section of a user profile. It includes fields for First Name ('Philipp'), Last Name ('Kraft'), and Email ('Mail@Philipp-Kraft.com'). A note below the email field states: 'We will never share your email with anyone'. The 'Account Security' section contains fields for New Password and Confirm New Password. The 'Other Settings' section includes a UI Language dropdown set to 'English' and a 'SAVE PROFILE' button.

General Settings	
FIRST NAME	Philipp
LAST NAME	Kraft
EMAIL	Mail@Philipp-Kraft.com
<small>We will never share your email with anyone</small>	
Account Security	
NEW PASSWORD	
CONFIRM NEW PASSWORD	
Other Settings	
UI LANGUAGE	English
<b>SAVE PROFILE</b>	

Abbildung 142: Profil

Erreichbar ist das Profil über die Navigationsleiste oben rechts mit einem Klick auf das Profilbild, es öffnet sich ein Dropdownmenü mit dem Eintrag Profil.

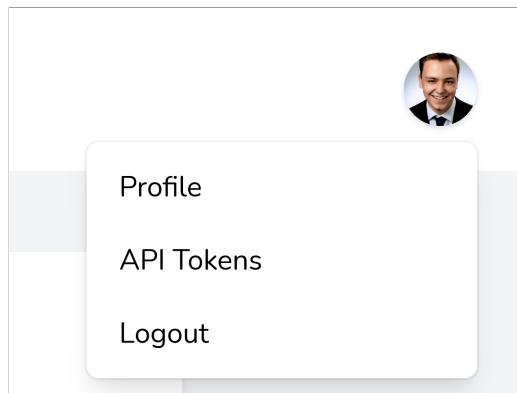


Abbildung 143: Navigationsleiste Dropdown

### 8.8.11 Lokalisierung

Lokalisierung ist sehr ähnlich zur Übersetzung, aber anstatt einer wörtlichen Übersetzung wird die Sprache an das Zielpublikum angepasst. Gegebenenfalls werden auch Maßeinheiten, Datums- und Uhrzeitformate angepasst.

Das Webinterface besitzt die Möglichkeit, zwischen Deutsch und Englisch auszuwählen, hierbei wurde das Webinterface zuerst in Englisch verfasst und dann in das Deutsche lokalisiert.

#### 8.8.11.1 Funktion der Sprachauswahl

Wie bereits in der Sektion 8.8.10 erklärt, kann ein Benutzer in den Profileinstellungen sich eine Sprache je nach Bedürfnis auswählen, dabei wird diese Sprache nur lokal bei diesem Benutzer angepasst.

Bei der Sprachauswahl wird ein `select` Element verwendet, wenn ein Benutzer nun eine andere Option auswählt, wird dieser weitergeleitet, beispielsweise bei einem Klick auf English wird dieser auf `/locale/en` weitergeleitet.

```

1  <?php
2
3  <select name="language" onChange="location = this.value;">
4
5  <option {{ Session::get('locale') == 'en' ? 'selected' : '' }}>
6    English</option>
7
8  <option {{ Session::get('locale') == 'de' ? 'selected' : '' }}>
9    Deutsch</option>
10
11 </select>

```

Code 110: Spracheauswahl

Beim Aufruf dieser Route wird in die Session des Benutzers die gewünschte Sprache geschrieben und der Benutzer wird direkt wieder zu der Ursprungsseite zurückgeschickt. Dieser Vorgang geht so schnell, dass ein Benutzer nicht bemerkt, dass die Seite gewechselt wurde.

```

1  <?php
2
3  Route::get('locale/{locale}', function ($locale) {
4
5      Session::put('locale', $locale);
6
7      return redirect()->back();
8
9  })->name('locale');

```

Code 111: Spracheauswahl Route

Nun steht in der Session des Benutzers die gewünschte Sprache, diese muss nun angewendet werden. Für das Anwenden der Sprache wird eine Middleware verwendet. Die Localization Middleware wird bei jedem Aufruf einer Route ausgeführt, dabei wird geprüft, ob der Benutzer den Eintrag locale besitzt, welcher vorhin durch den Service Provider gesetzt wurde. Ist dort ein Eintrag vorhanden wird die Seite in die passende Sprache lokalisiert.

```
1 <?php
2 if (\Session::has('locale')) {
3     \App::setlocale(\Session::get('locale'));
4 } else {
5     \App::setlocale(config('settings.default_language'));
6     \Session::put('locale', config('settings.default_language'));
7 }
```

Code 112: Localization Middleware

Besitzt der Benutzer keine locale Variable in seiner Session, bedeutet das, dass dieser noch keine Sprache festgelegt hat, somit wird automatisch die ausgewählte Standardsprache aus den Seiten Einstellungen ausgewählt.

## 8.9 API

### 8.9.1 Einleitung

Die Application programming interface (API) ist der Teil des Webinterfaces der mit der Hardware kommuniziert. Diese Programmierschnittstelle ist sehr einfach zu verwenden und wird mit Laravel Sanctum abgesichert. Die API ermöglicht ein Bearbeiten und lesen von Daten aus dem Webinterface mithilfe von HTTP-Anfragen.

### 8.9.2 Autorisierung mit Bearer Token

Wie bereits besprochen erfolgt die Autorisierung der HTTP-Anfragen mit Laravel Sanctum. Sanctum verwendet dabei einen sogenannten Bearer Token. Dieser Bearer Token wird bei Anfragen einfach im HTTP-Header mitgeschickt und der Server gleicht dies mit den verschlüsselten Token in der Datenbank ab und erlaubt oder verweigert den Zugriff.

### 8.9.3 API Tokens erstellen

Über das Webinterface kann entweder über die Sidebar oder über das Dropdownmenü in der Navigationsleiste das Menü *API Tokens* erreicht werden. Dort können nun neue API Tokens erstellt werden, zur späteren Identifikation kann dieser API Token mit einem Namen versehen werden, dieser Name spielt aber ansonsten keine Rolle.

The screenshot shows a web interface for managing API tokens. At the top, there's a navigation bar with 'Home / Admin / Tokens'. Below it, a section titled 'Create API Token' contains a 'TOKEN NAME' input field and a 'CREATE NEW TOKEN' button. Below this, another section titled 'Active API Tokens' lists a single token named 'Mein API Token' with the note 'LAST USED: Never' and a 'DELETE TOKEN' button.

Abbildung 144: API Token

Beim Erstellen des API Tokens öffnet sich einmalig ein Popup mit dem Bearer Token, dieser muss nun abgespeichert werden, denn der Token wird aus Sicherheitsgründen nur einmal angezeigt.

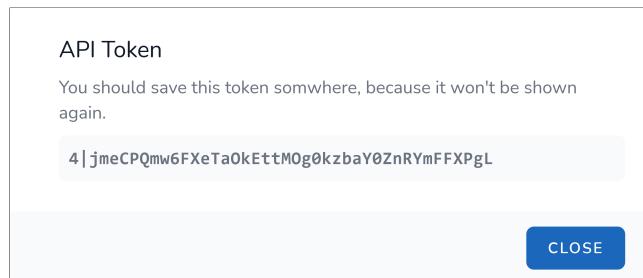


Abbildung 145: API Token Popup

## 8.9.4 Erkennungen

### 8.9.4.1 GET /api/v1/detections

Erkennungen GET	
Pfad	/api/v1/detections
Beschreibung	
HTTP-Methode	GET
Authentifizierung	Bearer Token
Rückgabetyp	application/json
Path Parameter	
None	
Query Parameter	
None	

Tabelle 8: GET /api/v1/detections

Die Rückgabe entspricht der Rückgabe von GET /api/v1/detections/{id}, jedoch wird ein Array von Erkennungen geliefert.

### 8.9.4.2 GET /api/v1/detections/{id}

<b>Erkennungen GET</b>	
<b>Pfad</b>	/api/v1/detections
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	GET
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
id	Required: true
	Type: int
	Description: Detection Id
<b>Query Parameter</b>	
None	

Tabelle 9: GET /api/v1/detections/{id}

```
1  {
2      "id": 1,
3      "license_plate_id": 69,
4      "parking_lot_id": 3,
5      "parking_status": "Outside",
6      "image": null,
7      "created_at": "2021-03-23T02:10:57.000000Z",
8      "updated_at": "2021-03-09T02:08:54.000000Z",
9      "license_plate": {
10          "id": 69,
11          "user_id": 74,
12          "name": "B509DYY",
13          "country": "Austria",
14          "parking_status": "Outside",
15          "allowance_status": "Disallowed",
16          "created_at": "2020-11-19T17:26:03.000000Z",
17          "updated_at": "2021-03-20T21:12:24.000000Z"
18      },
19      "parking_lot": {
20          "id": 3,
21          "name": "Parking Space 1",
22          "description": null,
23          "image": "default.png",
24          "created_at": "2021-03-31T22:40:24.000000Z",
25          "updated_at": "2021-03-31T22:42:57.000000Z"
26      }
27 }
```

Code 113: Beispielhafte GET /api/v1/detections/{id} Rückgabe

#### 8.9.4.3 POST /api/v1/detections

<b>Erkennungen POST</b>	
<b>Pfad</b>	/api/v1/detections
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	POST
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Query Parameter</b>	
name	Required: true
	Type: string
	Description: License plate name
parking_lot_id	Required: true
	Type: int
	Description: Parking Lot Id
<b>Query Parameter</b>	
None	

Tabelle 10: POST /api/v1/detections

Die Rückgabe entspricht der Rückgabe von GET /api/v1/detections/{id}.

#### 8.9.5 Kennzeichen

##### 8.9.5.1 GET /api/v1/plates

<b>Kennzeichen GET</b>	
<b>Pfad</b>	/api/v1/plates
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	GET
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
None	
<b>Query Parameter</b>	
None	

Tabelle 11: GET /api/v1/plates

Die Rückgabe entspricht GET /api/v1/plates/{id}, jedoch wird ein Array mit einzelnen Kennzeichen geliefert.

#### 8.9.5.2 GET /api/v1/plates/{id}

<b>Kennzeichen GET</b>	
<b>Pfad</b>	/api/v1/plates
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	GET
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
id	Required: true
	Type: int
	Description: License Plate Id

Tabelle 12: GET /api/v1/plates/{id}

```
1  {
2      "id": 1,
3      "user_id": 8,
4      "name": "FK8VD0",
5      "country": "Austria",
6      "parking_status": "Inside",
7      "allowance_status": "Unknown",
8      "created_at": "2020-12-05T07:04:31.000000Z",
9      "updated_at": "2021-03-22T09:31:46.000000Z",
10     "detections": [
11         {
12             "id": 149,
13             "license_plate_id": 1,
14             "parking_lot_id": 2,
15             "parking_status": "Inside",
16             "image": null,
17             "created_at": "2021-03-12T18:18:40.000000Z",
18             "updated_at": "2021-03-22T11:46:44.000000Z"
19         },
20         ...
21     ]
22 }
```

Code 114: Beispielhafte GET /api/v1/plates/{id} Rückgabe

## 8.9.6 Parkplätze

### 8.9.6.1 GET /api/v1/lot

<b>Parkplätze GET</b>	
<b>Pfad</b>	/api/v1/lots
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	GET
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
None	
<b>Query Parameter</b>	
None	

Tabelle 13: GET /api/v1/lots

Die Rückgabe entspricht GET /api/v1/lots/{i\}, jedoch wird ein Array von einzelnen Parkplätzen geliefert.

#### 8.9.6.2 GET /api/v1/lots/{id}

<b>Parkplätze GET</b>	
<b>Pfad</b>	/api/v1/lots
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	GET
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
id	Required: true
	Type: int
	Description: Parking Lot Id
<b>Query Parameter</b>	
None	

Tabelle 14: GET /api/v1/lots/{id}

```
1  {
2      "id": 3,
3      "name": "Parking Space 1",
4      "description": null,
5      "image": "default.png",
6      "created_at": "2021-03-31T22:40:24.000000Z",
7      "updated_at": "2021-03-31T22:42:57.000000Z",
8      "detections": [
9          {
10             "id": 1,
11             "license_plate_id": 69,
12             "parking_lot_id": 3,
13             "parking_status": "Outside",
14             "image": null,
15             "created_at": "2021-03-23T02:10:57.000000Z",
16             "updated_at": "2021-03-09T02:08:54.000000Z"
17         },
18         ...
19     ]
20 }
```

Code 115: Beispielhafte GET /api/v1/lots/{id} Rückgabe

## 8.9.7 Parklücken

### 8.9.7.1 GET /api/v1/spots

<b>Parklücke GET</b>	
<b>Pfad</b>	/api/v1/spots
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	GET
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
None	
<b>Query Parameter</b>	
None	

Tabelle 15: GET /api/v1/spots

Die Rückgabe entspricht GET /api/v1/spots/{id}, jedoch wird ein Array von einzelnen Parklücken geliefert.

#### 8.9.7.2 GET /api/v1/spots/{id}

<b>Parklücke GET</b>	
<b>Pfad</b>	/api/v1/spots
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	GET
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
id	Required: true
	Type: int
	Description: Parking Spot Id
<b>Query Parameter</b>	
None	

Tabelle 16: GET /api/v1/spots/{id}

```
1   {
2     "id": 40,
3     "address": null,
4     "parking_lot_id": 2,
5     "occupied": 0,
6     "created_at": "2021-03-22T17:51:10.000000Z",
7     "updated_at": "2021-02-13T23:23:55.000000Z",
8     "parking_lot": {
9       "id": 2,
10      "name": "Parkplatz Ost",
11      "description": "Dieser Parkplatz ist für Schüler gedacht",
12      "image": "parkplatz_ost.png",
13      "created_at": "2021-03-30T15:44:19.000000Z",
14      "updated_at": "2021-03-30T15:44:19.000000Z"
15    }
16 }
```

Code 116: Beispielhafte GET /api/v1/spots/{id} Rückgabe

#### 8.9.7.3 POST /api/v1/lots/{id}/spots

<b>Parklücken POST</b>	
<b>Pfad</b>	/api/v1/lots/{id}/spots
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	POST
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
<b>id</b>	Required: true
	Type: int
	Description: Parking Lot Id
<b>Query Parameter</b>	
<b>address</b>	Required: true
	Type: int
	Description: Hardware Adresse
<b>occupied</b>	Required: true
	Type: bool
	Description: 1 for occupied 0 for free

Tabelle 17: POST /api/v1/lots/{id}/spots

```
1   {
2     "id": 101,
3     "address": 1337,
4     "parking_lot_id": 1,
5     "occupied": 0,
6     "created_at": "2021-03-31T22:27:21.000000Z",
7     "updated_at": "2021-03-31T22:27:21.000000Z",
8     "parking_lot": {
9       "id": 1,
10      "name": "Parkplatz West",
11      "description": "Dieser Parkplatz ist für Lehrer gedacht",
12      "image": "parkplatz_west.png",
13      "created_at": "2021-03-30T15:44:19.000000Z",
14      "updated_at": "2021-03-30T15:44:19.000000Z"
15    }
16 }
```

Code 117: Beispielhafte POST /api/v1/lots/{id}/spots Rückgabe

## 8.9.8 Einstellungen

### 8.9.8.1 GET /api/v1/settings

<b>Einstellungen GET</b>	
<b>Pfad</b>	/api/v1/settings
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	GET
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
None	
<b>Query Parameter</b>	
None	

Tabelle 18: GET /api/v1/settings

```

1   [
2     {
3       "id": 1,
4       "key": "site_title",
5       "value": "APM"
6     },
7     ...
8   ]

```

Code 118: Beispielhafte GET /api/v1/settings Rückgabe

### 8.9.9 Benutzer

#### 8.9.9.1 GET /api/v1/users

<b>Benutzer GET</b>	
<b>Pfad</b>	/api/v1/users
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	GET
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
None	
<b>Query Parameter</b>	
None	

Tabelle 19: GET /api/v1/users

Die Rückgabe entspricht GET /api/v1/users/{id}, jedoch wird ein Array von einzelnen Benutzern geliefert.

#### 8.9.9.2 GET /api/v1/users/{id}

<b>Benutzer GET</b>	
<b>Pfad</b>	/api/v1/users
<b>Beschreibung</b>	
<b>HTTP-Methode</b>	GET
<b>Authentifizierung</b>	Bearer Token
<b>Rückgabetyp</b>	application/json
<b>Path Parameter</b>	
id	Required: true
	Type: int
	Description: User Id
<b>Query Parameter</b>	
None	

Tabelle 20: GET /api/v1/users/{id}

```

1   {
2     "id": 1,
3     "first_name": "Philipp",
4     "last_name": "Kraft",
5     "email": "Mail@Philipp-Kraft.com",
6     "avatar": "philipp.png",
7     "email_verified_at": "2021-03-30T15:44:18.000000Z",
8     "last_login_ip": "127.0.0.1",
9     "last_login_at": "2021-04-01 09:22:01",
10    "created_at": "2021-03-30T15:44:18.000000Z",
11    "updated_at": "2021-04-01T09:22:01.000000Z",
12    "license_plates": []
13 }
```

Code 119: Beispielhafte GET /api/v1/users/{id} Rückgabe

## 9 Zusammenfassung und Ausblick

Zusammenfassend ist zu erwähnen, dass alle drei Teile durchaus realisierbar sind. Die Kennzeichenerkennung ist in der Lage einzelne Fahrzeugschilder mit ausreichender Genauigkeit auszulesen. Die Detektion der Fahrzeuge auf den einzelnen Parklücken ist auch sehr zuverlässig und erfüllt daher auch seine Grundaufgabe. Die API erfüllt ebenfalls die gewünschte datenübermittelnde Funktion mit dem Webinterface. Es ist möglich in diesem die einzelnen Parkplätze, deren Fahrzeuge und freie Parklücken einzusehen und zu verwalten. Diese Web-Applikation ist auch für jeden Kunden anpassbar.

Abschließend kann gesagt werden, dass dieses Verwaltungssystem dem Benutzer viele Vorteile bringt und das Überwachen des Parkplatzes vereinfacht.

## 10 Anhang

## Abbildungsverzeichnis

1	Arbeitspakete Teil 1 . . . . .	13
2	Arbeitspakete Teil 2 . . . . .	14
3	Gantt-Chart Teil 1 . . . . .	15
4	Gantt-Chart Teil 2 . . . . .	16
5	Vor Bilateraler Filterung . . . . .	21
6	Nach Bilateraler Filterung . . . . .	21
7	Graustufenbild . . . . .	22
8	Binäres Bild nach Thresholding . . . . .	22
9	Binäres Bild nach Thresholding . . . . .	23
10	Nach Erosion . . . . .	23
11	Ablaufdiagramm der Kennzeichenerkennung . . . . .	27
12	Beispiel einer Anwendung von Matplotlib . . . . .	33
13	GPIO-Pins des Raspberry Pi . . . . .	39
14	Visualisierung mit Matplotlib . . . . .	48
15	Visualisiertes Ergebnis der Kennzeichenerkennung . . . . .	49
16	Raspberry Pi . . . . .	51
17	Raspberry Pi mit Kamera . . . . .	52
18	Fehler bei Konturerkennung . . . . .	55
19	Resultat mit maschinellem Lernen . . . . .	55
20	Ablauf von WPOD-NET . . . . .	57
21	Beispiel eines Zeichens aus dem Datensatz . . . . .	58
22	Verlauf des Modell Trainings . . . . .	61
23	Boden des Gehäuses . . . . .	62
24	Mittelteil des Gehäuses . . . . .	62
25	Deckel des Gehäuses . . . . .	63
26	Gesamtesansicht des Gehäuses . . . . .	63
27	Gedruckte Einzelteile des Gehäuses . . . . .	64
28	Gesamtesansicht des gedruckten Gehäuses . . . . .	64
29	Testaufbau . . . . .	65

30	Erster Test . . . . .	66
31	Zweiter Test   Bild 1 . . . . .	67
32	Zweiter Test   Bild 2 . . . . .	68
33	Zweiter Test   Bild 3 . . . . .	69
34	Zweiter Test   Bild 4 . . . . .	70
35	Zweiter Test   Bild 5 . . . . .	71
36	Installation der Spule . . . . .	74
37	LTSpice Blockbild zweier RL-Glieder . . . . .	76
38	Stromkurven zweier RL-Glieder . . . . .	77
39	RL-Oszillator mit NE555 . . . . .	78
40	RL-Oszillator mit NE555 Zeitsignale . . . . .	79
41	Vergleich des NE555-Oszillators bei unterschiedlichen L . . . . .	80
42	Eddy Currents in einem Leiter . . . . .	81
43	Parallelschwingkreis in LTSpice . . . . .	82
44	Gedämpfte Schwingung . . . . .	83
45	Colpitts LC-Glied . . . . .	85
46	Aktiver Colpitts-Oszillator . . . . .	86
47	Zeitsignale Colpitts-Oszillator . . . . .	87
48	Beispiel einer UART Zeichenübertragung . . . . .	89
49	RS485 Widerstandsnetzwerk . . . . .	89
50	Logische Tabelle für $U_{AB}$ . . . . .	91
51	Beispiel einer RS485 Zeichenübertragung . . . . .	91
52	RJ45 Steckerbuchse und Kabel . . . . .	92
53	Logische Übersicht des Bussystems . . . . .	93
54	Physische Übersicht des Bussystems . . . . .	94
55	Anordnung der Slave-Geräte mit Adresslogikleitung . . . . .	95
56	RJ45 Pinbelegung des Slave-Geräts . . . . .	96
57	RJ45 Pinbelegung des Master-USB-Geräts . . . . .	96
58	Elektrische Eigenschaften des L78L05ACD . . . . .	97
59	Ablauf von Steueranfragen . . . . .	99
60	Master Datenframe . . . . .	100

61	Slave Response Datenframe . . . . .	101
62	gemessene Slave Response bei Steuerbefehl IO1 . . . . .	102
63	Slave-Geräte mit Gehäuse . . . . .	104
64	Slave-Geräte mit Gehäuse von vorne . . . . .	104
65	Logischer Aufbau des Slave-Gerät . . . . .	105
66	Atmega328PU in DIP28 . . . . .	106
67	absolute Grenzwerte des L78L05ACD . . . . .	107
68	Colpitts Oszillator mit Schmitttrigger . . . . .	109
69	LM393D im Slave Schaltplan . . . . .	110
70	Colpitts Kondensatoren im Slave Schaltplan . . . . .	110
71	Slave-Gerät mit Gehäuse von links . . . . .	111
72	SPI Anschluss . . . . .	112
73	Pin-Netz Zuordnung des Atmega328PB im Slave Schaltplan . . . . .	113
74	Low Fuse-Byte . . . . .	114
75	Einstelleoptionen für den internen Takt des Atmega328PB . . . . .	114
76	Quartz im Slave Schaltplan . . . . .	115
77	High Fuse-Byte . . . . .	115
78	Extended Fuse-Byte . . . . .	116
79	ArduinoISP Sketch auswählen . . . . .	117
80	Arduino Upload Einstellungen für ArduinoISP Sketch . . . . .	117
81	Anschluss eines externen SPI Programmer . . . . .	118
82	Slave UART-Buffer Struktur . . . . .	121
83	Flussdiagramm der Datenframeerkennung . . . . .	123
84	erster Slave Prototyp . . . . .	128
85	beide Slave Prototypen . . . . .	128
86	Boden des Slave Gehäuses . . . . .	129
87	Wände des Slave Gehäuses . . . . .	129
88	Deckel des Slave Gehäuses . . . . .	129
89	Logischer Aufbau des Masters . . . . .	130
90	Beispielschaltung eines FT232 mit einem RS485 Pegelwandlers . . . . .	131
91	FT232RL im Schaltplan . . . . .	132

92	USB Typ C Stecker im Master Schaltplan . . . . .	133
93	USB-Bussadapet mit Gehäuse . . . . .	134
94	Boden des Master Gehäuses . . . . .	134
95	Wände des Master Gehäuses . . . . .	134
96	Deckel des Master Gehäuses . . . . .	134
97	Zugriff über VPS . . . . .	138
98	Github Action im Github Master repository . . . . .	140
99	HTML5 Logo von <a href="https://www.w3.org/html/logo">https://www.w3.org/html/logo</a> . . . . .	142
100	Einfache HTML Seite von <a href="https://www.w3.org/html/logo">https://www.w3.org/html/logo</a> . . . . .	144
101	Einfache HTML Seite mit CSS . . . . .	146
102	Laravel MVC Muster . . . . .	149
103	Advanced System Settings . . . . .	150
104	System Variables . . . . .	151
105	Environment Variables . . . . .	151
106	PHP Version . . . . .	152
107	phpMyAdmin Webinterface . . . . .	152
108	Docker WSL Integration . . . . .	155
109	Docker Container Steuerung . . . . .	155
110	Debian Default Page . . . . .	158
111	Action Secrets . . . . .	167
112	Github Action Übersicht . . . . .	168
113	Eloquent ORM Workflow von <a href="https://dev.to/xenoxdev/mastering-laravel-eloquent-orm-the-eloquent-journey-part-1-1571">https://dev.to/xenoxdev/mastering-laravel-eloquent-orm-the-eloquent-journey-part-1-1571</a> . . . . .	177
114	App Layout . . . . .	182
115	Login Seite mit App Layout . . . . .	183
116	Admin Layout . . . . .	184
117	Dashboard mit Admin Layout . . . . .	185
118	Navbar Variante 1 . . . . .	186
119	Navbar Variante 2 . . . . .	186
120	Navbar Variante 3 . . . . .	187
121	Content Header . . . . .	188

122	Success Session Alert . . . . .	189
123	Sidebar Desktop . . . . .	192
124	Footer . . . . .	193
125	Dashboard Sicherheitsnachricht . . . . .	194
126	Dashboard Statistik . . . . .	195
127	Benutzer ER . . . . .	196
128	Liste aller Benutzer . . . . .	197
129	Benutzer editieren . . . . .	198
130	Benutzer, Rollen und Rechte ER . . . . .	200
131	Benutzer erstellen . . . . .	205
132	Neuigkeit editieren . . . . .	206
133	Home Neuigkeiten . . . . .	207
134	Parking Lots, Parking Spots und Detections ER . . . . .	208
135	Übersicht aller Parkplätze . . . . .	208
136	Parklücken und Erkennungen eines Parkplatzes . . . . .	209
137	Kennzeichen, Parkplätze, Erkennungen und Benutzer Beziehung . . . . .	210
138	Liste aller Kennzeichen . . . . .	211
139	Einzelnes Kennzeichen . . . . .	212
140	Liste aller Erkennungen . . . . .	213
141	Einstellunge Tabelle . . . . .	214
142	Profil . . . . .	215
143	Navigationsleiste Dropdown . . . . .	216
144	API Token . . . . .	219
145	API Token Popup . . . . .	220

## Tabellenverzeichnis

1	Zeitaufwendungen Übersicht . . . . .	17
2	RJ45 allgemeine Pinbelegung . . . . .	95
3	allgemeiner Aufbau eines Datenframes . . . . .	98
4	ausgelesene Bytes aus einem gemessenen Datenframe . . . . .	102
5	Netzbelegung des SPI Anschluss . . . . .	112
6	Konfiguration der Fuse Bits . . . . .	113
7	Verfügbare Rechte . . . . .	202
8	GET /api/v1/detections . . . . .	220
9	GET /api/v1/detections/{id} . . . . .	221
10	POST /api/v1/detections . . . . .	223
11	GET /api/v1/plates . . . . .	224
12	GET /api/v1/plates/{id} . . . . .	224
13	GET /api/v1/lots . . . . .	226
14	GET /api/v1/lots/{id} . . . . .	227
15	GET /api/v1/spots . . . . .	229
16	GET /api/v1/spots/{id} . . . . .	230
17	POST /api/v1/lots/{id}/spots . . . . .	232
18	GET /api/v1/settings . . . . .	234
19	GET /api/v1/users . . . . .	235
20	GET /api/v1/users/{id} . . . . .	236

## Codeverzeichnis

1	PIP Installation von Jupyter Notebook . . . . .	30
2	PIP Installation von Numpy . . . . .	30
3	PIP Installation von OpenCV . . . . .	31
4	Installation benötigter Tools und Bibliotheken für OpenCV . . . . .	32
5	Klonen von OpenCV von GitHub . . . . .	32
6	Kompilieren von OpenCV . . . . .	32
7	Abschließende Installation von OpenCV . . . . .	33
8	PIP Installation von Matplotlib . . . . .	34
9	PIP Installation von Keras . . . . .	34
10	PIP Installation von Tensorflow . . . . .	35
11	Benötigte Tools für Tensorflow . . . . .	35
12	Tensorflow von GitHub herunterladen . . . . .	35
13	Entpacken von Tensorflow . . . . .	36
14	Zusätzlich erforderliche Librarys . . . . .	36
15	Kompilieren von Tensorflow . . . . .	36
16	Abschließen der Installation von Tensorflow . . . . .	36
17	PIP Installation von Scikit-learn . . . . .	37
18	PIP Installation von Requests . . . . .	38
19	PIP Installation von Gpiozero . . . . .	39
20	PIP Installation von Picamera . . . . .	40
21	WPOD-NET Modell laden . . . . .	40
22	Anwendung des WPOD-NET Modells . . . . .	41
23	get_plate . . . . .	41
24	Bild vorbereiten für WPOD-NET . . . . .	42
25	Kennzeichen lokalisieren . . . . .	42
26	Bild aufnehmen . . . . .	43
27	Konturen finden . . . . .	44
28	Konturen sortieren . . . . .	44
29	Filterung der korrekten Konturen . . . . .	45

30	Bildverarbeitungsalgorithmen . . . . .	46
31	Anwendung einer Visualisierung mit Matplotlib . . . . .	47
32	predict_from_model . . . . .	48
33	Anwendung der Zeichenerkennung . . . . .	49
34	Senden der Ergebnisse an die Datenbank . . . . .	50
35	Erstellen des Modells basierend auf MobileNetV2 . . . . .	59
36	Trainieren des Modells und Einstellungen anpassen . . . . .	60
37	Response C-Code des Slave für IO1 . . . . .	103
38	AVRDUDE Argumente für die Fuse-Bits . . . . .	118
39	PlatformIO Einstellungen . . . . .	119
40	allgemeine Registereinstellungen . . . . .	120
41	Timer Registereinstellungen . . . . .	120
42	UART Registereinstellungen . . . . .	121
43	Speichern eines Bytes auf den Buffer . . . . .	122
44	Auswertung der Daten eines Datenframes . . . . .	124
45	Slave C-Code zur Taktmessung . . . . .	125
46	Slave C-Code zur Taktmessung . . . . .	126
47	Slave C-Code zur Taktmessung . . . . .	126
48	Slave C-Code zur Taktmessung . . . . .	127
49	Master Code zur Erkennung des USB-Bussadapters . . . . .	135
50	Frequenzsteuerbefehl des Masters . . . . .	136
51	Detektionscode des Masters . . . . .	137
52	API Post Code des Masters . . . . .	138
53	Öffnen des Reverse SSH Tunnels . . . . .	138
54	Zugriff uf den Raspberry über SSH . . . . .	139
55	Zugriff uf den Raspberry über SSH . . . . .	139
56	index.html . . . . .	143
57	style.css . . . . .	145
58	WSL Feature aktivieren . . . . .	154
59	Virtual Machine Feature aktivieren . . . . .	154
60	WSL 2 auswählen . . . . .	154

61	Sail Bash alias . . . . .	156
62	Respositories updaten . . . . .	157
63	Apache installieren . . . . .	157
64	PHP installieren . . . . .	158
65	Mariadb installieren . . . . .	158
66	MariaDB Secure Installation . . . . .	159
67	MariaDB konfiguration . . . . .	159
68	phpMyAdmin Download . . . . .	159
69	phpMyAdmin Entpacken . . . . .	159
70	phpMyAdmin Rechte und Verzeichnisse . . . . .	160
71	phpMyAdmin Konfigurationsdatei erstellen . . . . .	160
72	phpMyAdmin Blowfish Secret und TempDir . . . . .	160
73	phpmyadmin.conf . . . . .	161
74	Virtual Host Konfiguration aktivieren . . . . .	161
75	apm.conf . . . . .	162
76	Download Composer Installer . . . . .	163
77	Composer Setup . . . . .	163
78	Git Installation . . . . .	163
79	Git Remote Origin . . . . .	164
80	Lokales Deploy Script . . . . .	164
81	Server Deploy Script . . . . .	165
82	main.yml . . . . .	167
83	web.php . . . . .	171
84	UserController.php . . . . .	172
85	Rollen Datenbank Seeder . . . . .	174
86	create_users_table.php . . . . .	175
87	Eloquent Query . . . . .	176
88	Raw SQL Query . . . . .	177
89	UserUpdate Request . . . . .	179
90	example.blade.php . . . . .	180
91	primary.blade.php . . . . .	180

92	Verwendung eines Button Components . . . . .	181
93	Modularer Button Component . . . . .	181
94	Ausschnitt 1 Navigationsleiste . . . . .	186
95	Ausschnitt 2 Navigationsleiste . . . . .	187
96	Erstellung der Breadcrumbs . . . . .	188
97	Controller mit success Nachricht . . . . .	189
98	Session Alert Component . . . . .	189
99	Sidebar Header . . . . .	190
100	Sidebar Link . . . . .	191
101	AuthenticatedSessionController.php Store Methode . . . . .	194
102	Profilbild Upload . . . . .	199
103	UserController.php Profilbild Upload . . . . .	199
104	Permission Vergabe Methode . . . . .	203
105	Rolle erstellen und Rechte zuweisen . . . . .	203
106	UserPolicy Index . . . . .	204
107	UserController Index . . . . .	204
108	SettingsServiceProvider . . . . .	214
109	Seiten Titel Konfiguration . . . . .	215
110	Spracheauswahl . . . . .	217
111	Spracheauswahl Route . . . . .	217
112	Localization Middleware . . . . .	218
113	Beispielhafte GET /api/v1/detections/{id} Rückgabe . . . . .	222
114	Beispielhafte GET /api/v1/plates/{id} Rückgabe . . . . .	225
115	Beispielhafte GET /api/v1/lots/{id} Rückgabe . . . . .	228
116	Beispielhafte GET /api/v1/spots/{id} Rückgabe . . . . .	231
117	Beispielhafte POST /api/v1/lots/{id}/spots Rückgabe . . . . .	233
118	Beispielhafte GET /api/v1/settings Rückgabe . . . . .	234
119	Beispielhafte GET /api/v1/users/{id} Rückgabe . . . . .	236

## Abkürzungsverzeichnis

**PK** Philipp Kraft

**DK** Dennis Köb

**SB** Samuel Bleiner

**APM** Advanced Parking Monitoring

**HTML** Hypertext Markup Language

**W3C** World Wide Web Consortium

**CSS** Cascading Style Sheets

**PHP** Hypertext Preprocessor

**JS** JavaScript

**DOM** Document Object Model

**MVC** Model-View-Controller

**ORM** Object-relational mapping

**WSL** Windows Subsystem for Linux

**CLI** Command-line interface

**SQL** Structured Query Language

**UART** Universal Asynchronous Receiver Transmitter

**IIC** Inter-Integrated Circuit

**SPI** Serial Peripheral Interface

**ASCII** American Standard Code for Information Interchange

**DIN** Deutsches Institut für Normung

**IC** Integrated Circuit

**DIP** Dual Inline Package

**VID** Vendor Identification Number

**PID** Product Identification Number

**PCB** Printed Circuit Board

**SMD** Surface Mount Device

**SMT** Surface Mount Technology

**IO** Input Output

**SVG** Scalable Vector Graphics

**USB** Universal Serial Bus

**WYSIWYG** What You See Is What You Get

**CRUD** Create Read Update Delete

**MSB** Most Significant Byte

**LSB** Least Significant Byte

**API** Application programming interface

## Literaturverzeichnis

- Montazzolli, Sérgio und Claudio Jung. »License Plate Detection and Recognition in Unconstrained Scenarios«. In: Sep. 2018. URL: [https://www.researchgate.net/figure/Detailed-WPOD-NET-architecture\\_fig3\\_327861610](https://www.researchgate.net/figure/Detailed-WPOD-NET-architecture_fig3_327861610).
- Redmon, Joseph und Ali Farhadi. »YOLO9000: Better, Faster, Stronger«. In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: <http://arxiv.org/abs/1612.08242>.
- Sandler, Mark u. a. »Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation«. In: *CoRR* abs/1801.04381 (2018). arXiv: 1801.04381. URL: <http://arxiv.org/abs/1801.04381>.
- Stauffer, Matt. *Laravel: Up & Running: A Framework for Building Modern PHP Apps*. O'Reilly Media, 2019.
- WHATWG. *HTML Living Standard*. [Online; Abgerufen am 02. April 2021]. 2021. URL: <https://html.spec.whatwg.org>.
- World Wide Web Consortium. *Cascading Style Sheets (CSS) Snapshot 2007*. [Online; Abgerufen am 25. März 2021]. 2011. URL: <https://www.w3.org/TR/css-beijing/#css3>.
- . *HTML5 Logo*. [Online; Abgerufen am 31. März 2021]. 2014. URL: <https://www.w3.org/html/logo>.
- XenoX. *Eloquent ORM Workflow*. [Online; Abgerufen am 31. März 2021]. 2020. URL: <https://dev.to/xenoxdev/mastering-laravel-eloquent-orm-the-eloquent-journey-part-1-1571>.