

Felix Seibt

Ansteuerungseinheit für Ladeboxen für Elektrofahrzeuge

Bachelorarbeit

Technische Hochschule Köln,

04. März 2024

Betreuer: Prof. Dr. Eberhard Waffenschmidt

Ko-Referent: Prof. Dr. Ingo Stadler

Technology
Arts Sciences
TH Köln

Erklärungen

Name: Felix Seibt

Erklärung zum eigenständigen Verfassen

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst habe. Ich habe keine anderen außer den von mir angegeben Quellen und Hilfsmittel verwendet.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Köln, 04. März 2024

Felix Seibt

Erklärung zur Veröffentlichung

Ich bin damit einverstanden, dass meine Abschlussarbeit ausgeliehen werden darf. Sie darf von meinem Betreuer im Internet veröffentlicht werden.

Köln, 04. März 2024

Felix Seibt

Erklärung zu Bildrechten

Außer den im Folgenden genannten habe ich alle Bilder und Diagramme dieser Abschlussarbeit selbst erstellt.

Für die folgenden Bilder habe ich keine expliziten Nutzungsrechte erhalten. Sie sollten daher vor einer Veröffentlichung der Arbeit unkenntlich gemacht werden.

Abbildung 12 Typ 2 Stecker mit Belegungsvariante 2 und 3

Abbildung 13 Combo Stecker Belegung

Abbildung 14 Isolation der Lademodi

Köln, 04. März 2024

Felix Seibt

Kurzfassung

Die Arbeit gibt einen umfassenden Überblick über die verschiedenen Arten von Wallboxen und deren Ansteuerungsmöglichkeiten. Es werden Vor- und Nachteile der verschiedenen Wallboxen aufgeführt sowie die möglichen Schnittstellen erläutert und die Gründe für die Entscheidung für eine bestimmte Art von Wallbox erläutert. Das Ziel dieser Arbeit ist es, die Leistung einer Wallbox direkt über eine Ansteuerungseinheit und eine Weboberfläche zu steuern und mögliche Werte auszulesen.

Etwa 80% der Wallboxen unterstützen das OCPP-Protokoll, welches am weitesten verbreitet ist. Die Ansteuerung der Wallbox über dieses Protokoll sowie die Kommunikation zwischen Backend und Frontend werden erläutert.

Abstract

This paper provides a comprehensive overview of the different types of wallboxes and their control options. The advantages and disadvantages of the various wallboxes are listed, the possible interfaces are explained and the reasons for deciding in favour of a particular type of wallbox are explained. The aim of this paper is to control the power of a wallbox directly via a control unit and a web interface and to read out possible values.

Around 80% of wallboxes support the OCPP protocol, which is the most widely used. The control of the wallbox via this protocol as well as the communication between backend and frontend are explained.

Danksagung

An dieser Stelle möchte ich allen Danken, die mich bei der Abfassung dieser Arbeit unterstützt haben. Dazu zählen

Peter Fabricius und Robert Brüggemann, mit denen ich die Ergebnisse bis spät in die Nacht diskutiert habe.

Tina Behlen, Johanna Fabricus, Niklas Siegel und Tobias Thirolf, die die vielen Rechtschreibfehler in meiner Arbeit gefunden haben. Wenn noch immer welche übrig sind, habe ich wahrscheinlich eine Korrektur übersehen.

Hanna Grau, die mich beim Zusammenschreiben durch das Essen unterstützt hat und mich damit am Leben erhalten hat.

Inhaltsverzeichnis

Erklärungen	2
Erklärung zum eigenständigen Verfassen	2
Erklärung zur Veröffentlichung	2
Erklärung zu Bildrechten	2
Kurzfassung	3
Abstract	3
Danksagung.....	4
Inhaltsverzeichnis.....	5
1. Einleitung	7
2. Stand der Technik	8
2.1. Steuerung von Wallboxen	8
2.2. Bereits vorhandene Lösungen	8
2.2.1. Mobile Apps.....	9
2.2.2. Open Source Bibliothek Wbec.....	10
2.3. Darstellung und Steuerungsmöglichkeiten.....	11
3. Hintergrund	14
3.1. Wallboxen	14
3.1.1. Was sind Wallboxen	14
3.1.2. Wie funktionieren Wallboxen / Ladepunkte.....	14
3.1.3. Leistungsabhängigkeiten.....	16
3.1.4. Leistungsstufen	17
3.2. Protokolle	17
3.2.1. Mögliche Protokolle.....	17
3.2.2. OCPP.....	18
3.3. Wallbox Anschlussmöglichkeiten.....	20
3.4. Ladestecker.....	20
3.5. Kommunikation zwischen Wallbox und Auto	22
3.6. Isolationssysteme -AC Ladestationen	23
3.7. Ansteuerungseinheit	24
3.7.1. Microprozessoren.....	24
3.8. Programmiersprachen.....	25
3.8.1. Threading	26
3.8.2. Async	26
4. Umsetzung	27
4.1. Voraussetzung für die Umsetzung	27
4.1.1. Auswahl des Microprozessors	27

4.2.	Wahl der Wallbox	28
4.2.1.	Preiswerte Wallbox	30
4.2.2.	Mittelpreisige Wallbox	30
4.2.3.	Hochpreisige Wallbox.....	30
4.3.	Vergleich der Wallboxen.....	30
4.4.	Entscheidung für WB24 mit dem OCPP Protokoll.....	31
4.5.	Umsetzung an der Wallbox	32
4.5.1.	Anschließen der Wallbox.....	32
4.5.2.	Einrichten der Wallbox	33
4.6.	Wahl der Programmiersprache & Umgebung	36
4.7.	Programmierung des Raspberry Pi.....	36
4.7.1.	Interface und Einstellungsmöglichkeiten.....	36
4.7.2.	Anbindung an die Wallbox	38
4.7.3.	Verbindung zwischen den Skripten	42
4.8.	Demo im Labor	44
4.9.	Demo in der Praxis	45
5.	Ergebnis	47
5.1.	Ladeboxen	47
5.2.	Schnittstellen	47
5.3.	Daten lesen und schreiben auf einer Benutzeroberfläche	48
5.4.	Wallbox als Demo im Labor und der Praxis installieren	48
6.	Fazit.....	49
7.	Literaturverzeichnis	50
	Verwendete Abkürzungen.....	55

1. Einleitung

Für das Laden von Elektrofahrzeugen stehen in Deutschland 101.421 öffentliche Ladepunkte [2] zur Verfügung. Darüber hinaus gibt es mindestens 470.000 private Wallboxen [3]. Davon verfügt ein Teil der betrachteten Wallboxen über eine identische Schnittstelle zur Ansteuerung der Wallboxen. Über diese Schnittstelle ist eine Steuerung der Wallbox und ein Auslesen der Daten möglich. Zudem wird es immer wichtiger alle Komponenten in ein dezentrales Lastmanagement einbinden zu können, wie es in dem Konzept Progressus dargestellt ist [4], um planbare Lasten intelligent steuern zu können. Ziel dieser Bachelorarbeit ist es, eine Ansteuerungseinheit zu entwerfen, die es ermöglicht, die Wallbox aus dem internen Netzwerk über ein Webinterface, oder direkt an der Ansteuerungseinheit über ein Display zu steuern. Eine Anforderung dabei ist, dass möglichst viele Wallboxen von der Ansteuerungseinheit unterstützt werden.

Zu Beginn werden die Grundlagen erläutert. Dies beinhaltet eine Definition aller relevanten Geräte, die Funktionsweise einer Wallbox und warum diese für das Laden eines Elektroautos sinnvoll ist. Anschließend werden bereits existierende Lösungen betrachtet und erläutert.

Hier wird darauf eingegangen, warum diese Anwendungen oder Schnittstellen nicht oder nur teilweise für dieses Projekt verwendet werden können. Für die praktische Umsetzung wird eine Wallbox benötigt, die eine möglichst weit verbreitete Schnittstelle bietet. Dazu wird ein Vergleich von mehreren Wallboxen mit den verschiedenen Schnittstellenmöglichkeiten, dem Preis und den Sicherheitsfunktionen durchgeführt. Der Vergleich zeigt, warum die in Abbildung 1 dargestellte Wallbox WB24-EC ausgewählt wurde. Anschließend wird erläutert, wie die Umsetzung erfolgte. Bei der Umsetzung ging es neben der Auswahl der Wallbox und den möglichen Schnittstellen auch um die Möglichkeit, relevante Daten über eine einfache Benutzeroberfläche lesen und schreiben zu können. Des Weiteren soll die Wallbox installiert und als Demonstration im Labor getestet werden. Dabei wird im Kapitel Umsetzung auf die Schwierigkeiten eingegangen. Es wird erklärt, warum ein Einplatinencomputer, gewählt wurde, wie der Programmcode funktioniert und wie die Drossel- und Abschaltfunktionen gestaltet wurden. Abschließend wird das Ergebnis mit den wichtigsten Inhalten und neuen Erkenntnissen dieser Arbeit zusammengefasst. Im Fazit wird ein kurzer Ausblick auf die weitere Vorgehensweise gegeben.



Abbildung 1 Wallbox WB24

2. Stand der Technik

Der vorliegende Stand der Technik, beinhaltet bereits Lösungen, zur Steuerung von Wallboxen und wie eine Oberfläche an einer Ansteuerungseinheit programmiert werden kann. Es wird erläutert, wie diese Lösungen funktionieren und ob diese auf das vorliegende Problem angewendet werden können. Falls dies nicht möglich ist, werden die Unterschiede erläutert, warum diese nicht genutzt werden können oder es nicht sinnvoll ist.

2.1. Steuerung von Wallboxen

Dieser Abschnitt erläutert, warum die Steuerung von Wallboxen sinnvoll sein kann und welche Möglichkeiten beim aktuellen Stand der Technik zur Verfügung stehen. Die Steuerung von Wallboxen ermöglicht es, die gesamte Energie einer PV-Anlage selbst zu nutzen, statt diese in das Stromnetz einzuspeisen. Dies ist möglich, indem nur so viel Ladeleistung freigeschaltet wird, wie der überschüssige Strom vom Dach es zulässt. Zusätzlich könnte vorgegeben werden, wann das Auto fertig geladen sein soll, ob es mit reduzierter Leistung oder z.B. nachts laden soll. Dies könnte das Stromnetz entlasten oder, je nach Stromtarif die Kosten senken und zudem die Lebensdauer der Batterie verlängern. Dazu müssen die Daten aus der Wallbox und bestenfalls aus dem Auto ausgelesen werden, um eine sinnvolle Steuerung zu ermöglichen.

2.2. Bereits vorhandene Lösungen

Markus Wunsch hat bei der Netze BW GmbH bereits eine Lösung zur Steuerung einer Wallbox entwickelt. Die Steuerung im letzten Entwicklungsschritt funktioniert über eine digitale Schnittstelle namens EEBUS welche in Kapitel 3.2.1 kurz erläutert wurde. Über diese Schnittstelle kann die Wallbox stufenlos geregelt werden. Für die Verbindung zum Backend von Netze BW ist eine entsprechende Steuerbox des Herstellers erforderlich. [5] Diese Lösung ist für uns nicht geeignet, da diese auf ein selten genutztes Protokoll zurückgreift und nur über das System der Netze BW genutzt werden kann. Außerdem ist der hierfür genutzte Code nicht öffentlich zugänglich, weshalb eine teilweise Verwendung nicht möglich ist.

Im Rahmen der Arbeit „Algorithmus zur autarken netzdienlichen Steuerung von zeitlich flexiblen Lasten“ wird eine Lösung zur Steuerung von Lasten im Haushalt vorgestellt, zu denen auch Wallboxen gehören. In dieser Arbeit wird die Möglichkeit beschrieben über eine Steuerbox die Lasten mehrerer Verbraucher so zu steuern, dass die Netzlast über den Tag verteilt wird. Die Steuerung erfolgt hierbei über Relais, die freigegeben werden, wie in Abbildung 2 dargestellt ist. Standardmäßig sind diese Relais gesperrt [6]. Diese Lösung ist im Rahmen dieser Arbeit nicht geeignet, da die Arbeit auf das Lastmanagement eines ganzen Hauses ausgelegt ist. Es wird nicht berücksichtigt, ob ein Auto geladen werden soll oder nicht oder ob die Möglichkeit besteht die Ladeleistung zu reduzieren. Außerdem wird nicht beschrieben wie die Konfiguration der Lastenverteilung benutzerfreundlich erfolgt.

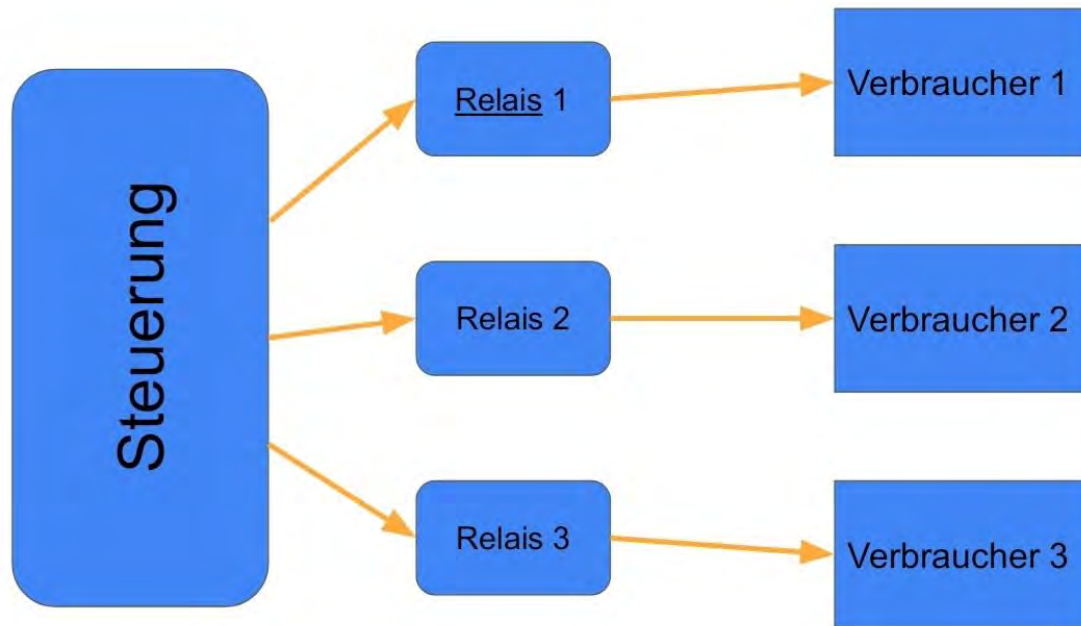


Abbildung 2 Steuerung mehrerer Verbraucher über Relais

Zudem haben verschiedenen Hersteller von Wallboxen Apps entwickelt, mit denen sich die Wallboxen steuern lassen. Die Funktionen sind dabei unterschiedlich umfangreich. Hierauf wird im Folgenden kurz eingegangen.

2.2.1. Mobile Apps

Go-eCharge bietet eine mobile App für Android zur Konfiguration und Steuerung der Wallbox an. Die App zeigt einen Überblick über den aktuellen Status der Wallbox an und ermöglicht eine umfangreiche Konfiguration für verschiedene Ladebedingungen, wie beispielsweise den Strompreis über einem festgelegten Niveau oder die Zielzeit. Außerdem kann die Ladeleistung eingestellt und die bereits geladene Menge abgelesen werden. Die Gesamtleistung, die eine Person geladen hat, wird über einen RFID-Chip erfasst. Des Weiteren besteht hier die Möglichkeit mehrere Wallboxen von E-Charge miteinander zu verbinden und ein sogenanntes Lastmanagement zu konfigurieren [7]. Diese Lösung funktioniert jedoch nur für die Wallboxen von go-e und bietet kein grafisches Interface an der Wallbox oder der Steuereinheit selbst. Die Ansteuerungseinheit soll in dieser Arbeit für möglichst viele Wallboxen funktionieren und es im weiteren Verlauf ermöglichen, mehrere unterschiedliche Wallboxen bzw. deren Steuergeräte miteinander zu vernetzen. Die Lösung bietet diese Möglichkeit nicht.

Der ABB Charger Sync bietet über eine App und ein Webportal den Zugang zu einer Übersicht über den Status und die Konfiguration der Wallbox. Es kann eine Übersicht über alle verbundenen Wallboxen und deren Energieabgabe angezeigt werden. Zusätzlich gibt es eine Übersicht über alle Ladestandorte und eine Nutzerverwaltung. In der Liste der Ladevorgänge können zudem detaillierte Informationen zu einem Ladevorgang abgerufen werden [8]. Diese Lösung bietet sehr viele Möglichkeiten, ist jedoch nur mit den Wallboxen von ABB kompatibel. Eine Steuerung direkt an der Steuereinheit ist nicht möglich, sondern nur über ein Webinterface oder eine App. Aus diesem Grund ist diese Lösung keine Option.

Darüber hinaus bietet die Reev App eine Lösungsmöglichkeit, die durch die Nutzung des Kommunikationsstandards einen Vorteil bietet. Dieser Standard wird in Kapitel 3.2.2 erläutert. Zusätzlich stellt Reev ein Dashboard mit einer Übersicht bereit, auf dem alle Informationen dargestellt werden und die Konfiguration der Wallbox ermöglicht wird. Ein Nachteil besteht darin, dass die gesetzte Anforderung der Bedienung über ein Bedienfeld auf der Ansteuerungseinheit nicht gegeben ist. Zudem wird die Software von Reev kommerziell vertrieben. Daher wäre dies ein Lösungsansatz, aber da es sich nicht um Open Source Software handelt, kann dies nicht für den vorliegenden Zweck modifiziert werden.

2.2.2. *Open Source Bibliothek Wbec*

Die Open-Source-Software Wbec ist eine Softwarelösung, die für die Heidelberg Wallbox Energy Control entwickelt wurde, um diese zu steuern und zu konfigurieren. Dabei ist es möglich über ein Webinterface eine Übersicht anzuzeigen, explizite Ladevorgänge mit Strom von der PV-Anlage ein- und auszuschalten und alle weiteren Informationen im Überblick zu haben. In der Pro Version ist zusätzlich ein Display zur direkten Steuerung integriert [9]. Durch die Modbus RTU Schnittstelle, die in der WbEC Softwarelösung verwendet wird, ist dies für unser vorliegendes Problem, möglichst viele Wallboxen abzudecken, leider nicht denkbar. Jedoch ist dies ein gutes Beispiel zur Umsetzung.

2.3. Darstellung und Steuerungsmöglichkeiten

Zur Darstellung und Steuerung wurden verschiedene Möglichkeiten untersucht unter der Annahme, dass als Steuergerät ein Microprozessor mit Touchdisplay genutzt wird. Die Darstellung und Steuerung soll über ein Webinterface realisiert werden sodass dies über ein beliebiges Endgerät genutzt werden kann.

Eine bekannte Lösung hierfür ist Grafana, welches ein konfigurierbares Dashboard anbietet. In Grafana besteht die Möglichkeit mehrere Datenquellen einzubinden, wobei hier der Fokus auf Datenbanken liegt. Aber auch API Schnittstellen sind möglich. Des Weiteren bietet Grafana eine moderne und ansprechende Oberfläche mit verschiedenen Panels. Auf diesen Panels können Graphiken, Buttons(Add-ons) oder Textfelder eingebunden werden. In Abbildung 3 wird beispielhaft über das Grafana Dashboard ein Graph angezeigt, der einen Ladeverlauf anzeigen könnte, sowie mehrere Buttons, die zum Starten und Stoppen eines Ladevorgangs genutzt werden können. Hier wird nur die Oberfläche beispielhaft dargestellt, ohne eine Funktion. Nachteile von Grafana bestehen in der ungenauen Dokumentation der nutzbaren Add-ons zur Verbindung mit Datenquellen und deren Anwendung, sowie in der eingeschränkten Darstellungsmöglichkeit auf der Oberfläche.



Abbildung 3 Grafana Beispieldashboard

Eine weitere Lösung für eine konfigurierbare Weblösung bietet Zabbix. Zabbix bietet in erster Linie eine Lösung zur Anzeige und Visualisierung von Daten. Hierbei liegt der Fokus allerdings auf dem Monitoring und nicht auf der Steuerung, weshalb die für dieses Projekt notwendigen Funktionen zum Abschalten oder Konfigurieren der Wallbox nicht oder nur eingeschränkt möglich sind.

Des Weiteren wäre es möglich das webbasierte Programm „Node-Red“ zu verwenden. Hier ist es möglich über den Browser einen Flow zu erstellen und in diesen Flow zusätzliche Add-ons einzubinden. Dabei gibt es Add-ons, welche für die Darstellung des „Node-Red-Dashboard“ verantwortlich sind und eine einfache Visualisierung ermöglichen. Hierzu ist in Abbildung 5 ein Flow dargestellt welcher einen http Request startet und die Möglichkeit bietet über einen Button einen festgelegten Text in das Textfeld auf dem Dashboard zu schreiben. In Abbildung 4 wird das Dashboard welches aus dem Flow generiert wurde dargestellt. Es handelt sich um eine Softwarelösung, welche für dieses Projekt konfiguriert werden kann, jedoch wird dabei eine graphische Programmierung verwendet. Graphische Programmierung beinhaltet ein neben der einfachen Click and Play Bedienung auch mehrere Nachteile. Es besteht nur eine eingeschränkte Kontrolle zur Ausführung und bei großen Flows, sind graphische Programmierungen weniger übersichtlich als Code.

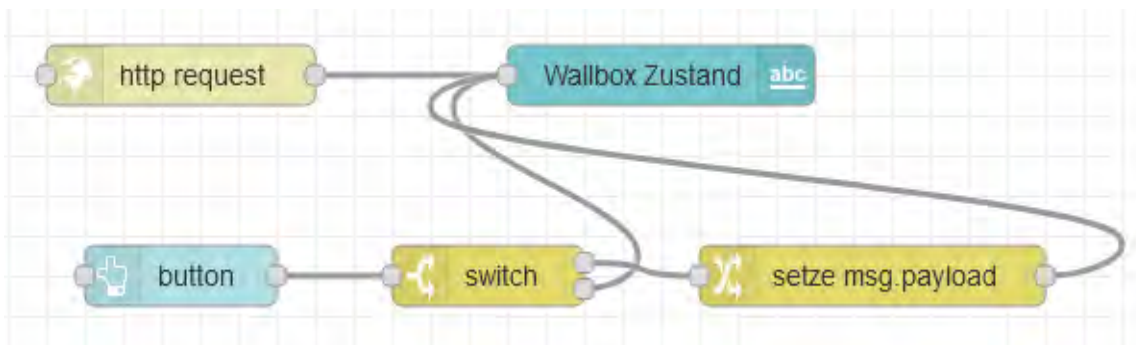


Abbildung 5 Node-Red Beispielflow

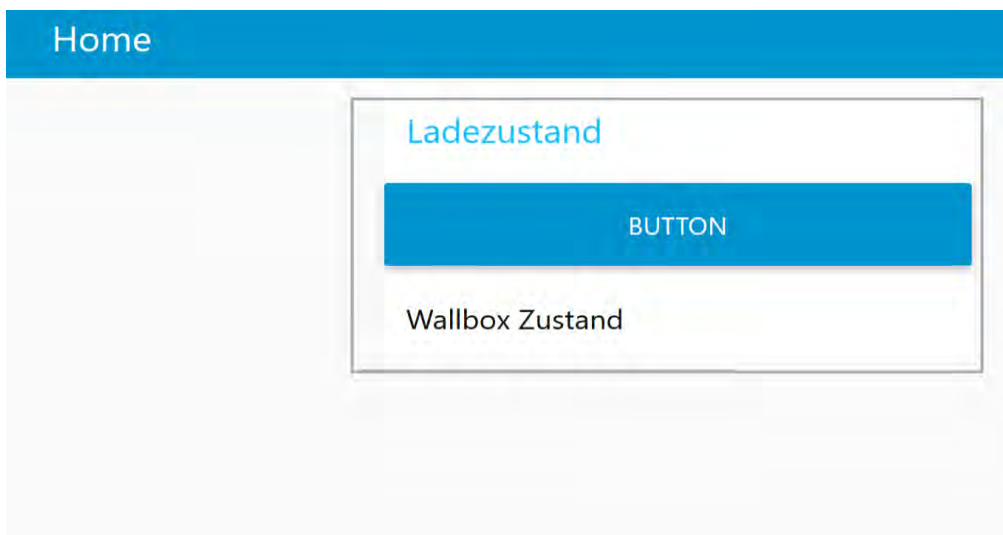


Abbildung 4 Node-Red Beispieldashboard aus dem Flow

Eine Möglichkeit wäre zudem, auf einem Raspberry Pi einen Webserver zu installieren. Als Webserver würde sich Nginx oder Apache anbieten. Diese ermöglichen das Anzeigen von HTML-Dateien, die wie im Abschnitt zu Programmiersprachen beschrieben, mit CSS gestaltet werden können. Um die Seite dynamisch zu gestalten, wird PHP oder Javascript benötigt. Zum Speichern oder Laden von Daten besteht die Möglichkeit, eine Datenbank auf dem Raspberry Pi einzurichten

und über PHP darauf zuzugreifen. Außerdem ist es möglich, über PHP auch Python-Skripte auszuführen [10].

Als Alternative hierzu kann das Python-Web-Framework Flask verwendet werden. Es bietet die Möglichkeit zu testzwecken über ein Python Skript einen Webservice zu starten, der eine Weboberfläche bereitstellt. Diese enthält Funktion wie das Darstellen von Graphen, Textfelder oder Tabellen sowie die Möglichkeit, die Seite in verschiedene Tabs aufzuteilen. Zudem besteht die Möglichkeit, Buttons mit einfachen Funktionen in Python zu hinterlegen wie in Abbildung 6 dargestellt. Hier ist der Code zu sehen, den der Button ausführt und in der Abbildung 7 den Code welcher den Button darstellt durch das Aufrufen der HTML-Seite.

```
@app.route('/')
def home():
    return render_template('index.html')
```

Abbildung 7 Code zum Anzeigen der HTML Seite

```
@app.route('/handle_button_click', methods=['POST'])
def handle_button_click():
    # Hier können Sie jede gewünschte Funktion ausführen
    print("Button wurde geklickt!")
    # Leiten Sie den Nutzer zurück zur Hauptseite oder führen Sie eine andere Aktion durch
    return render_template('index.html', message="Button wurde erfolgreich geklickt!")
```

Abbildung 6 Aktion Button Klick

Ein weiteres Python Web Framework ist Django. Django ist darauf ausgelegt, Anwendungen möglichst schnell vom Konzept bis zur Fertigstellung zu bringen. Hierfür bietet es viele Funktionen, die bei der Webentwicklung benötigt werden. Diese Funktionen bestehen unter anderem aus der Benutzerauthentifizierung, Verwaltung von Inhalten, RSS-Feeds und andern Funktionen. Des Weiteren unterstützt Django der Vermeidung von häufigen Sicherheitsfehlern wie SQL-Injection, Cross-Site Scripting, Cross-Site Request, Forgery und Clickjacking. Auch bietet es ein System für eine sichere Benutzerauthentifizierung [11].

Dash ist ein Open-Source-Web Framework, dessen Umsetzung in Python programmiert wird und die Möglichkeit bietet eine Weboberfläche zu gestalten. In Dash besteht die Möglichkeit Daten einzubinden, welche über Python abrufbar sind und zudem ermöglicht diese Daten zu visualisieren. Zudem können Button Items in jeglicher Form eingebunden werden und über sogenannten Callback Funktionen ausgeführt werden. Zudem können eingegebene Datensätze verwendet oder berechnete Datensätze angezeigt werden. Das Design kann noch angepasst werden, um es anschaulicher zu gestalten. Um das Design entsprechend anzupassen kann HTML und CSS, das Dash Design Kitt(Enterprise Version) oder die Dash Bootstrap Components nach eigenen Wünschen entsprechend genutzt werden.

Die Wahl der Lösung wird im Unterabschnitt 4.7.1 des Kapitels Umsetzung näher erläutert

3. Hintergrund

In diesem Kapitel werden die technischen Hintergründe von Wallboxen und Ansteuerungseinheiten erläutert, um der Thematik im weiteren Verlauf der Arbeit besser folgen zu können.

3.1. Wallboxen

Wie in der Einleitung beschrieben, werden Wallboxen oder Ladeboxen, wie diese auch genannt werden, zum effektiveren Laden von Elektrofahrzeugen benötigt. Hierzu wird in den folgenden Kapiteln erläutert, was Wallboxen sind und wozu diese genutzt werden. Des Weiteren wird auf die Steuerung, Sicherheitsaspekte und die Ladeleistung von Wallboxen eingegangen.

3.1.1. Was sind Wallboxen

Wallboxen werden überwiegend im privaten Umfeld eingesetzt, wie Uwe Götze und Marco Rehme in folgendem Zitat beschreiben: “Der Einsatzort einer Wallbox ist vorwiegend der Innenbereich von Garagen, aber auch an Carports oder Hauswänden kann sie in einer entsprechenden Ausführung montiert werden“ [12]. Dabei gibt es zwei Möglichkeiten des Ladens, Wechselspannung (AC) oder Gleichspannung (DC) [13]. Wallboxen sind überwiegend AC-Ladepunkte. Während DC von Schnellladesäulen genutzt wird. Wie diese im Detail funktionieren, wird in den Abschnitten 3.1.2.1 und 3.1.2.2 näher erläutert.

Wer berechtigt ist, eine Wallbox installieren zu lassen, ist unter anderem im WEG (Wohnungseigentumsgesetz) geregelt. Wie Moritz Zappel in seinem Artikel beschreibt “Gemäß § 16 Abs 2 WEG 2002 ist der Wohnungseigentümer zu Änderungen an seinem Wohnungseigentumsobjekt auf eigene Kosten berechtigt [...] die Errichtung von Strom-, Gas-, Wasser- oder Fernsprechleitungen, Beheizungsanlagen und ähnlichen Einrichtungen kann nicht untersagt werden“ [14]. Das bedeutet, dass jeder Haus- oder Wohnungseigentümer in einem gewissen Rahmen sich eine Wallbox installieren lassen kann.

3.1.2. Wie funktionieren Wallboxen / Ladepunkte

In diesem Unterkapitel wird erläutert, wie Wallboxen bzw. Ladepunkte im Allgemeinen funktionieren. Eine Wallbox ermöglicht das Aufladen eines Autos über ein Kabel oder eine Steckdose. Sollte eine Wallbox nur eine Steckdose haben, so wird ein eigenes Kabel benötigt. Dabei wird Energie aus dem Hausnetz in das Auto geladen. Die Ladepunkte können diesen Ladevorgang steuern und verbinden den Wechselspannungsanschluss sicher mit dem Elektroauto. Um die Sicherheit beim Laden zu gewährleisten enthalten Wallboxen Schutzvorrichtungen gegen Fehlerströme und Überspannungen [15]. Für den korrekten Anschluss durch einen Fachmann wird in Tabelle 1 beschrieben, wie die Leitungen dafür ausgelegt sein müssen. Bei korrekter Leitungsdimensionierung und bestimmungsgemäßer Verwendung treten keine Überbelastungen auf. In den beiden folgenden Kapiteln wird auf die beiden unterschiedlichen Ladearten AC-Laden und DC-Laden eingegangen, warum AC-Laden weniger effektiv ist und warum mit DC-Laden deutlich mehr Leistung übertragen werden kann.

Tabelle 1 Leitungsdimensionierung Abhängig von der Leistung

Spannung [V]	Stromstärke[A]	Netz	max. Ladeleistung [kW]	Leitungsquerschnitt mindestens / empfohlen [mm ²]
230	10	1-phasig	2,3	2,5 / 6
230	16	1-phasig	3,7	2,5 / 6
230	32	1-phasig	7,4	4 / 10
400	16	3-phasig	11	2,5 / 6
400	32	3-phasig	22	4 / 10

3.1.2.1. AC-Ladeboxen

Die Funktionsweise des AC-Ladens wird von Martin Doppelbauer beschrieben, weshalb er hier zitiert wird: „Beim AC-Laden wird das Fahrzeug direkt ans Wechselspannungsnetz angeschlossen. Im Fahrzeug befindet sich ein Ladegerät, das zunächst die Wechsel- in eine Gleichspannung wandelt (AC/DC-Konverter) und dann die Gleichspannung über einen DC/DC-Steller regelt. Es besteht eine galvanische Trennung von Fahrzeug und Netz.“ [16] „Bei der galvanischen Trennung wird die elektrische Leitung zwischen zwei Stromkreisen unterbrochen oder zwei leitfähige Gegenstände voneinander entkoppelt“ [17]

Weiterhin kann sich die Schaltung in der Wallbox vorgestellt werden, wie in Abbildung 8 vereinfacht dargestellt ist. In der vereinfachten Darstellung wird deutlich, dass die Wallbox die Leistung nur dann durchschaltet, wenn diese benötigt wird. Zusätzlich kann die Leistung in der Regel reduziert werden, dies geschieht über den Laderegler im Auto. Je nach Modus kann mehr oder weniger Leistung zur Verfügung gestellt werden, indem die Wallbox dem Auto übermittelt wieviel Leistung das maximum ist. Mehr dazu im Kapitel Leistungsstufen.

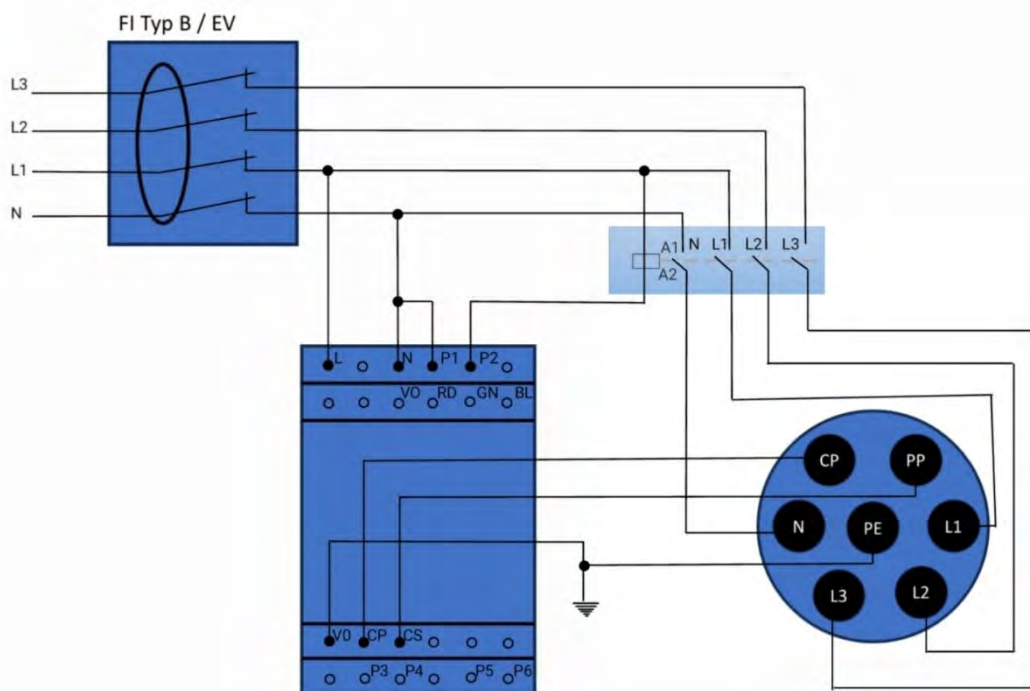


Abbildung 8 Ladeschaltung AC vereinfachte Darstellung [7]

3.1.2.2. DC-Ladeboxen

Martin Doppelbauer beschreibt auch die Funktionsweise des Gleichstromladens: „Beim DC-Laden befindet sich die Ladeschaltung in der Ladesäule. Das Fahrzeug hat einen DC-Anschluss, der direkt auf die Batterie führt. Auch hier ist eine galvanische Trennung von Fahrzeug und Netz möglich.“ [16] Das Laden mit Gleichspannung ist effizienter als mit Wechselspannung und bietet im Vergleich eine höhere Ladeleistung, da die Laderegler für das AC-Laden im Fahrzeug auf maximal 22 kW ausgelegt sind. Währenddessen ist der Laderegler beim DC-Laden in der Ladesäule und der Anschluss führt direkt auf die Batterie. Aufgrund dieser höheren Ladeleistung bis zu 350 kW müssen die Leitungen im Stromnetz für diese Ströme entsprechend dimensioniert sein. Deshalb wird dies überwiegend nur

bei gewerblichen Kunden installiert. In Tabelle 2 Leistungsstufen Ladepunkte wird sichtbar, dass bei einer DC Ladeleistung von 50 kW ein Strom von 125A fließt. Während bei 11 kW Ladeleistung AC an einer Wallbox maximal 16A Strom fließen. Es wird also deutlich, dass bei DC-Schnellladern auch das Stromnetz für diese Leistung ausgelegt sein muss. Ein normales Einfamilienhaus ist jedoch nur für eine Leistung von 16 bis 25 kW Leistung ausgelegt [18], was bei der allgemeinen Netzplanung berücksichtigt werden muss. Das bedeutet für eine Schnellladeanschluss ist ein Hausanschluss, im Normalfall, nicht ausgelegt.

3.1.3. Leistungsabhängigkeiten

Wallboxen können maximal bis zu 22 kW laden, üblich für private Haushalte sind 11 kW maximale Ladeleistung. Die tatsächliche Leistung hängt jedoch nicht nur von der Wallbox ab, sondern auch vom Laderegler im Auto. Hierbei gibt es Autos wie z.B. Hyundai Ioniq 4 die mit einer maximalen AC Leistung von 7,2 kW [19] laden können. Andere Fahrzeuge wie der Tesla Modell 3 bieten interne Laderegler an, die mit 3 Phasen 11 kW [20] laden können. Weitere Faktoren sind der Zustand der Batterie (inklusive den durchlaufenen Lade-Entladezyklen, hohe Ladezustände und tiefe Entladungen, hohe Betriebstemperaturen, hohe Ladespannungen und Ströme und das Alter der Batterie [21]), die Temperatur und der aktuelle Ladestand der Batterie. Ein Vorteil des Schnellladens ist, dass der Verlust im Auto geringer ist, da die On Board Geräte des Autos kürzer eingeschaltet sind und dadurch weniger Energie über die Zeit verbrauchen [22]. Andererseits sollte die Batterie mit Betrachtung auf die Lebensdauer langsam geladen werden, da dies dafür sorgt, dass alle Zellen gleich stark geladen werden und hier ein Spannungsausgleich erfolgen kann [23]. Sollte die Batterie ideale Ladebedingungen aufweisen findet die Ladung wie im Folgenden erklärt statt. In Abbildung 9 Ladeleistung zu State of Charge ist zu sehen, dass der Akku anfangs mit annähernd 100% der Leistung lädt und diese Ladeleistung je nach Modell schneller oder langsamer mit der Zeit absinkt. Wie in der Abbildung sichtbar wird bildet der Mini Cooper SE eine Ausnahme hierzu und steigert die Ladeleistung langsam bis 75% bevor dies auch abfällt. Einen Ähnlichen Ladeverlauf gibt es auch beim Laden an einer Wallbox mit weniger Leistung. An einer Wallbox wird die maximale Ladeleistung jedoch länger gehalten bevor diese absinkt.

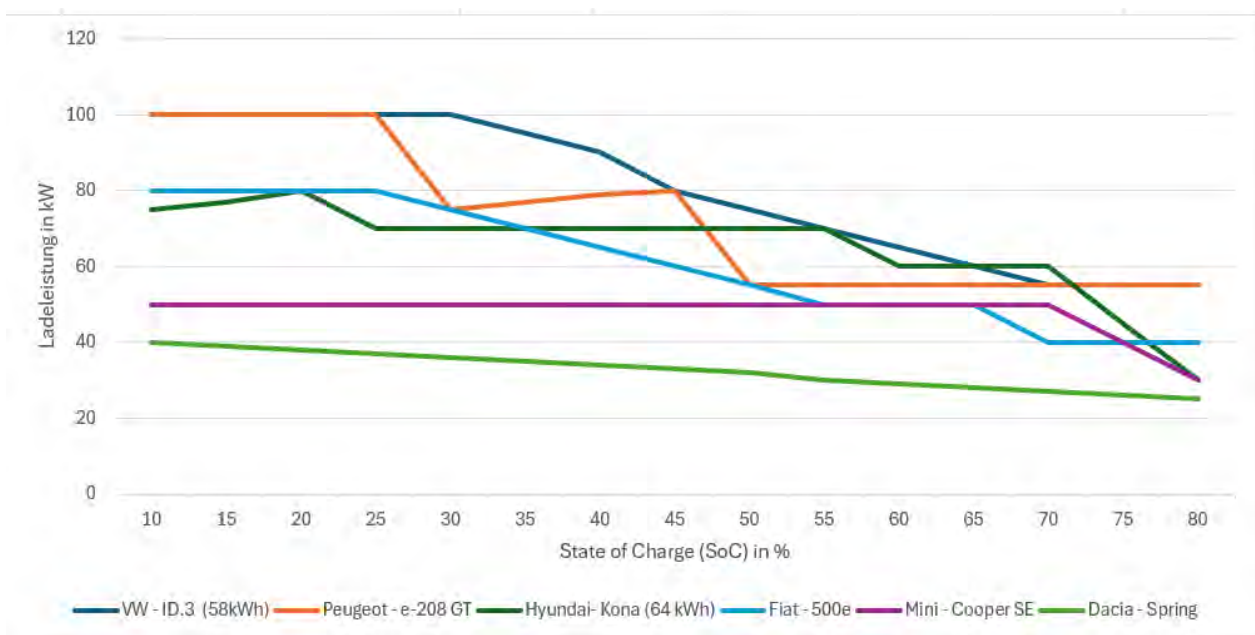


Abbildung 9 Ladeleistung zu State of Charge [30]

3.1.4. Leistungsstufen

Die verschiedenen Leistungsstufen werden aus Tabelle 2 ersichtlich. Hier wird sichtbar, dass es fünf Stufen in den Wechselspannungsladestufen gibt. Diese erstrecken sich von 2,3 kW bis 22 kW Ladeleistung. Zusätzlich gibt es drei Gleichspannungsladestufen, welche eine Ladeleistung von 50 kW bis 350 kW ermöglichen. Hier wird erkennbar, dass Laden mit Gleichspannung (DC) deutlich effektiver ist als das Laden mit Wechselspannung (AC). Dadurch wird ersichtlich, weshalb Schnellladepunkte mit DC betrieben werden. Diese benötigen jedoch zusätzlich eine Ladeelektronik und einen entsprechend starken Anschluss an das Stromnetz [24].

Tabelle 2 Leistungsstufen Ladepunkte [24]

Langsames Laden (<10kW)	1 ~ AC	230 V / 10 A	2,3 kW
	1 ~ AC	230 V / 16 A	3,7 kW
	1 ~ AC	230 V / 32 A	7,4 kW
Beschleunigtes Laden (11, 22 kW)	3 ~ AC	400 V / 16 A	11 kW
	3 ~ AC	400 V / 32 A	22 kW
Schnelles Laden (ab 50 kW)	DC	400 V / 125 A	50 kW
	DC	400 V / 500 A	200 kW
	DC	800 V / 500 A	350 kW

3.2. Protokolle

Protokolle werden benötigt, um eine standardisierte Datenübertragung zwischen zwei Geräten herzustellen. Hierzu ist in der Protokolldefinition festgelegt, wie und in welchem Format miteinander kommuniziert wird [25].

3.2.1. Mögliche Protokolle

Nachfolgend werden Protokolle und deren Funktionsweisen erläutert. Diese sind überwiegend nicht weit verbreitet und werden im Rahmen dieser BA nicht weiter verwendet. Auf das relevanteste Protokoll wird im folgenden Abschnitt näher eingegangen. Die anderen Protokolle werden für Vergleiche im Kapitel 4.4 benötigt. Alle Protokolle sind in Tabelle 3 Protokolle und ihre Funktionsweise aufgelistet und beschrieben.

Tabelle 3 Protokolle und ihre Funktionsweise

Protokoll-Name	Funktionsweise
EEBUS	Über einen eigenen Modbus und einen einheitlichen Kommunikationsstandard können Geräte im SmartHome kommunizieren
Modbus	Kommunikationsprotokoll welches den Datenaustausch zwischen einem Master und mehreren Slaves ermöglicht.
TCP	Standardisierte Vereinbarung zur Datenübertragung zwischen verschiedenen Teilnehmern eines Computernetzwerkes
MQTT	Ein Nachrichtenprotokoll nach dem Publisher- / Subscriber Prinzip über einen zentralen Broker
RTU	Protokoll auf Grundlage vom Modbus

3.2.2. OCPP

Eine passende Definition, wofür das OCPP-Protokoll verwendet werden kann, wird im Folgenden aus dem Buch *Smarte Ladesäulen und Netz- Marktdienliches öffentliches Laden* zitiert:“ Das Open Charge Point Protocol (OCPP) ist ein offenes Protokoll zur Kommunikation zwischen Ladesäulenbetreibern und Ladesäulen. Wichtig ist hierbei das erst ab Version 2.0.1 wichtige Sicherheitsfeatures wie Authentifizierung sowie Verschlüsselte Kommunikationskanäle implementiert worden sind. Weshalb ältere Version des Protokolls als unsicher und manipulationsgefährdet gelten, vor allem die in gängigen Ladestation weit verbreitete Version 1.6. [...] Sofern die Ladestation 2.0.1 unterstützt, läuft die Kommunikation mit der Plattform immer verschlüsselt ab. Der Vorteil des vorliegenden Ansatzes ist, dass hierdurch auch die noch verbreiteten 1.6 nutzenden Ladesäulen zu integrieren sind, die grundsätzliche Architektur des Backends jedoch die neueren Sicherheitskonzepte schon berücksichtigt.“ [26] Welche Funktionen das OCPP-Protokoll bietet und wie sich diese weiterentwickelt haben, zeigt die Tabelle 4 Übersicht der Funktionalitäten nach OCPP-Version. Bereits ab der ersten Version wird ein breites Spektrum an Funktionen angeboten, wie die Tabelle zeigt.

Tabelle 4 Übersicht der Funktionalitäten nach OCPP-Version [27]

OCPP Basisfunktionalität	<ul style="list-style-type: none"> - Autorisierung - Transaktion und Messdaten - Zustandsinformationen - Remote Control (Start / Stopp / Entriegelung) - Over-the-Air Firmware Update
OCPP 1.5 (2012)	<ul style="list-style-type: none"> - Lokale Autorisierungsliste - Reservierung - Personalisierte Messages - Erweiterung Konfiguration
OCPP 1.6 (Release 2015, Edition 2 2017, Edition 3 2019)	<ul style="list-style-type: none"> - Smart Charging - JSON over WebSockets - Trigger Message
OCPP 2.0 (Release 2018)	<ul style="list-style-type: none"> - Kompatibilität mit ISO 15118-2 - Sicherheit (sichere Authentifizierung, TLS-Kommunikation) - Weitere Konfigurationsmöglichkeiten - Überwachung der Ladestation durch Monitoring - verbesserte Verarbeitung der Transaktionen - Erweiterung Smart Charging
OCPP 2.0.1 (Release 2020)	Bugfix und Verbesserung

3.2.2.1. Verbindungsaufbau

Für den Verbindungsaufbau mit OCPP über Websockets wird die URL des zentralen Lademanagementsystems (LC) im Ladepunkt (LP) hinterlegt, dieser fügt anschließend noch seine ID vor dem Verbindungsaufbau hinzu. Durch diese ID kann das LC die Wallbox eindeutig identifizieren. Das Lademanagementsystem muss vor dem LP eingeschaltet sein, da der LP versucht eine Verbindung zu dem LC aufzubauen. Für einen erfolgreichem Verbindungsaufbau sendet das LP eine Boot-Notification mit den wichtigsten Informationen wie in Abbildung 10 dargestellt. Dies setzt sich wie folgt zusammen, aus der MessageTypeId als Integer, der UniqueId als String, der Action als String und dem Payload als JSON mit dem Betreiber und der ChargepointModel. Außerdem wird eine gültige Antwort wie in Abbildung 10 OCPP Beispiel Request dargestellt benötigt, welche darstellt, dass mit dem MessageTyp „3“ und der UniqueID ein Payload übertragen wird. Der Payload beinhaltet den Status, die aktuelle Uhrzeit und wenn gewünscht den Heartbeat-Intervall. Sollte der Verbindungsaufbau nicht erfolgreich verlaufen, wird eine ähnliche Antwort generiert, jedoch mit einem Fehlercode, einer Fehlerbeschreibung und im Payload mit Details zu dem Fehler. Das LC kann bei erfolgreichem Verbindungsaufbau nun mit dem LP kommunizieren. Um die Verbindung aufrechtzuerhalten und festzustellen, dass alle Teilnehmer noch an der Verbindung teilnehmen, gibt es sogenannte periodische Heartbeat-Nachrichten. Heartbeat-Nachrichten sorgen dafür, dass das Lademanagementsystem den aktuellen Status des Ladepunktes hat und weiß, dass dieser noch erreichbar ist. Zur Veranschaulichung ist in Abbildung 10 der Verbindungsaufbau und das anschließende Lebenszeichen dargestellt. Das LC kann bei der ersten Antwort an den Ladepunkt festlegen, wie oft und mit welchen Informationen ein Heartbeat gesendet werden soll.

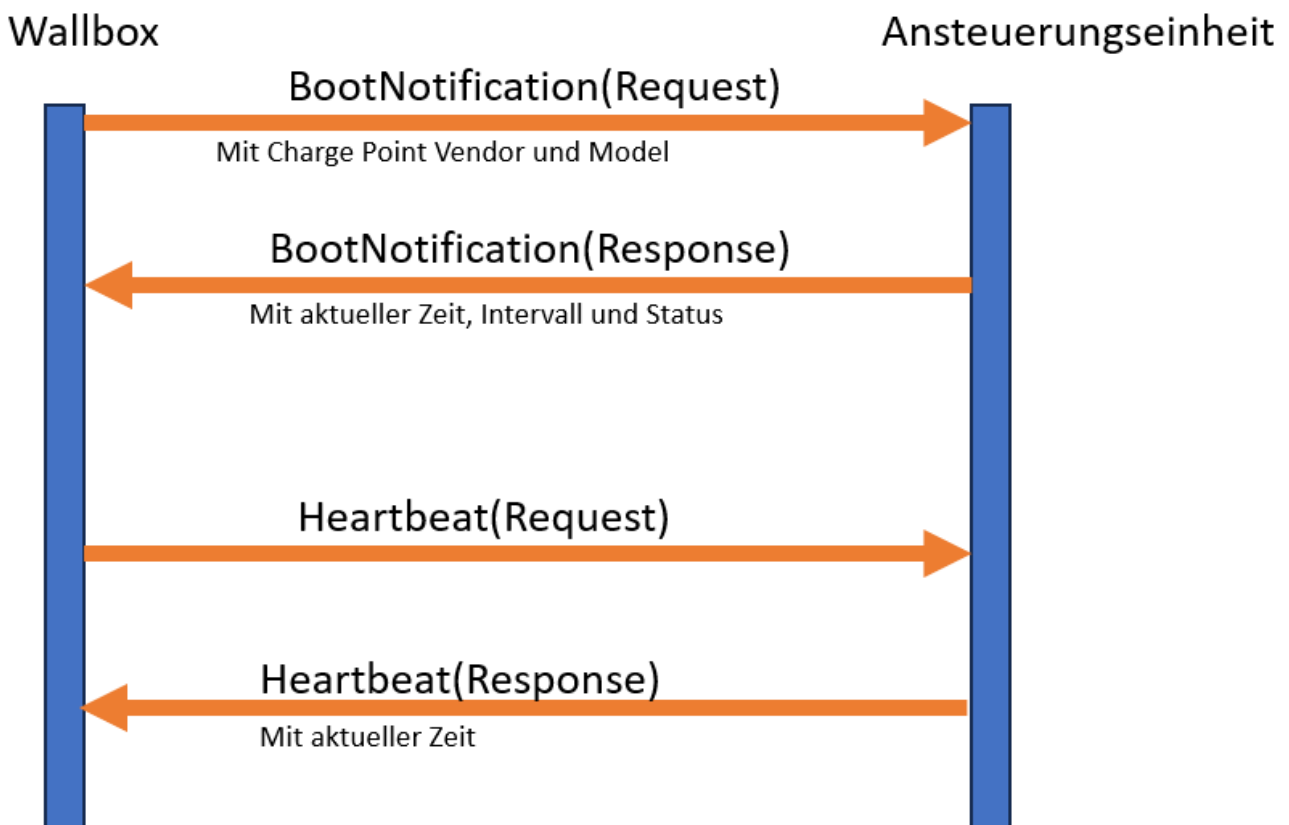


Abbildung 10 OCPP Beispiel Request und Response

3.3. Wallbox Anschlussmöglichkeiten

Jede Wallbox muss an das Stromnetz angeschlossen werden. Um dies zu ermöglichen wird für eine 11 kW Ladeleistung ein dreiphasiger Stromanschluss (400V) am Verteilerschrank benötigt. Hierzu ist bei den Wallboxen von Herstellerseite ein Anschlussbild vorgegeben, welcher Stecker verwendet wird (üblich Typ 2 siehe Abbildung 11 Typ 2 Stecker und Belegung). Des Weiteren müssen je nach Wallbox die Komponenten untereinander noch entsprechend nach Plan miteinander verbunden werden. Da bei 400V Lebensgefahr besteht muss hierbei Fachpersonal die Installation vornehmen.

Des Weiteren muss die Wallbox verpflichtend durch einen Fehlerstrom-Schutzschalter (umgangssprachlich FI-Schalter) gesichert sein. Sollte ein FI-Schalter Typ A bereits vorhanden sein und die Wallbox nicht über eine DC-Fehlerstromerkennung verfügen oder ein Typ A EV Schutzschalter, so wird ein neuer Typ B Schalter benötigt. Typ A erkennt keine Gleichfehlerströme und kann daher nicht abschalten oder durch DC Strom in seiner Funktionalität gestört werden [28].

Zudem muss der Hausanschluss entsprechend dimensioniert sein, sodass zur normalen Nutzung auch die Nutzung der Wallbox nicht zu einer Überbelastung der Leitung führt.

Die Wallbox muss zudem bei dem Netzbetreiber angemeldet werden, dies wird in §19 Abs. 2 NAV (Niederspannungsanschlussverordnung) vorgeschrieben. Ab einer Ladeleistung von mehr als 12 kW ist die Wallbox genehmigungspflichtig durch den Netzbetreiber. [29]

3.4. Ladestecker

Zum Laden eines Elektrofahrzeuges gibt es verschiedene Möglichkeiten im Privaten Bereich. Es gibt den Schuko Stecker, welcher ein haushaltsüblicher Stecker ist, wie an jedem einphasigen Haushaltsgerät. Dieser ermöglicht mit einem passenden Kabel das Auto über die Haushaltssteckdose mit bis zu 3,7 kW zu laden. Hierbei ist allerdings zu beachten, dass gewöhnliche Haushaltssteckdosen nicht für die thermische Dauerbelastung von 3,7 kW ausgelegt sind. Der IEC Typ 2 ist für Ladeleistungen von 3,7 kW bis 43 kW (AC) und bis zu 120 kW (DC) laden ausgelegt. Hierzu gibt es entsprechend 3 verschiedene Kontaktbelegungen, wie in Abbildung 11 und Abbildung 12 dargestellt ist. Der erste Typ ermöglicht AC Laden mit bis zu 3 Phasen, dort wird auf dem Kunststoff die Beschriftung des N Leiter, des L1-L3, die Erdung mit PE und die Kommunikation mit CP und PP erkenntlich. Bei der Kommunikation (AC-Laden) ist der Pilotkontakt (CP) über den Widerstand für den aktuellen Status verantwortlich und der Kontakt zum Proximity-Pilot (PP) für den maximalen Ladestrom, näheres hierzu im nächsten Unterkapitel. Die zweite Steckerbelegung ermöglicht AC und DC Laden wie durch den N-Leiter und L1 und den Plus und Minus Teil dargestellt ist. Des Weiteren gibt es für den Typ 2 eine dritte Anschlussmöglichkeit um nur über DC zu laden. Hierbei gibt es zwei Positive Anschlüsse und zwei negative, welche insgesamt eine Ladeleistung von bis zu 350 kW ermöglichen [30].



Abbildung 11 Typ 2 Stecker und Belegung 1

Abbildung 12 Typ 2 Stecker mit Belegungsvariante 2 und 3 [31]

Zusätzlich gibt es in Europa den Stecker Combo 2 wie in Abbildung 13 dargestellt, wird eine ähnliche Steckerbelegung wie Typ 2 nur ohne den Part des AC-Ladens sichtbar mit einem zusätzlichen Anschluss für positiv und negativ nur für DC. Hierzu gibt es Version 1.0 und 2.0. Die Erste Version ermöglicht laden bis 80 kW während CSS 2.0 bis 450 kW Ladeleistung ermöglicht [32].

Abbildung 13 Combo Stecker Belegung

3.5. Kommunikation zwischen Wallbox und Auto

Um größere Leistungen als 3,7 kW freizugeben muss das Auto mit dem Ladepunkt kommunizieren. Hierzu gibt es wie im vorherigen Unterkapitel bereits kurz erwähnt zwei Leitungen. Zum einen die CP Leitung für den aktuellen Status und die PP Leitung für den maximalen Ladestrom. Die Kommunikation läuft über einen variablen internen Widerstand im Auto, welcher je nach aktuellem Status entsprechend variiert. Die verschiedenen Stadien können der Tabelle 5 entnommen werden. Hierbei bedeutet die Ladefreigabe „D mit Belüftung“, dass sollte der Ladepunkt in einem Raum sein, das in dem Raum die Belüftung eingeschaltet werden muss. Für den maximalen Ladestrom sind in Tabelle 6 die Werte aufgelistet, welcher interne Widerstand welchem Ladestrom zugeordnet ist. Die Kommunikationskanäle haben eine Rechteckschwingung mit 12 V und 1kHz. Diese Spannung wird im Fahrzeug über einen Widerstand und eine Diode auf den Schutzleiter PE zurückgeführt. Bei offenem Stromkreis sind die Ladepunkte grundsätzlich spannungsfrei.

Tabelle 5 Status zu Widerstand einer Wallbox [33]

Widerstand CP-PE	Offen	2700 Ohm	880 Ohm	240 Ohm
Ladefreigabe	A Standby	B Auto erkannt	C bereit, laden	D mit Belüftung

Tabelle 6 maximaler Ladestrom zu Widerstand eines Ladepunktes [33]

Widerstand PP-PE	1500 Ohm	680 Ohm	220 Ohm	100 Ohm
Max. Ladestrom	13 A	20 A	32 A	63 A

Bevor der Ladevorgang freigegeben wird müssen diese Parameter übermittelt worden sein. Generell wird zum Laden nach folgenden Schritten vorgegangen: Erster Schritt ist die Prüfung der Verbindung des Schutzleiters zum Fahrzeug und die Übermittlung des verfügbaren Ladestroms. Anschließend stellt das Fahrzeug den Lader ein. Im dritten Schritt verriegelt das Fahrzeug die Ladesteckvorrichtung und fordert den Start der Ladung an. Anschließend wird im vierten Schritt die Ladesteckvorrichtung am LP verriegelt. Anschließend kann der LP die Steckdose einschalten. Das Beenden der Ladung und die Entriegelung wird über das Fahrzeug gesteuert und über die CP-Leitung an den LP übertragen [33].

Eine HLC Kommunikation findet vorwiegend bei höheren Leistungen im DC Laden statt, wodurch dies für diese Arbeit nicht relevant ist [34].

3.6. Isolationssysteme -AC Ladestationen

In Abbildung 14 sind, die Funktion und Absicherung, drei unterschiedlicher Ladevorgänge dargestellt bei Verwendung zweier verschiedenen Lademodi. Bei der einfachsten Möglichkeit, ein Elektrofahrzeug an der Haushaltssteckdose zu laden (Lademodus 2), wird der Fehlerstrom durch den Adapter gemessen. Im Auslösefall unterbricht die Elektronik die Verbindung. Im Lademodus 3 gibt es in beiden Fällen einen extra Schalter, einen Überstromschutz, eine Fehlerstromerkennung Typ A und eine Gleichstromüberwachung welche bei Strömen größer als 5mA auslöst und zusätzlich einen Überspannungsschutz. Im privaten Umfeld gibt eine Kontrolleinheit welche den Ladevorgang überwacht.

Im öffentlichen Bereich findet zusätzlich noch die Kommunikation mit dem Energieversorger statt, eine Abrechnung und es gibt eine Bedienerschnittstelle [1].

3.7. Ansteuerungseinheit

Wie bereits in der Problemstellung beschrieben soll eine Ansteuerungseinheit entwickelt werden. Diese bildet die Schnittstelle zwischen Mensch und Maschine mit einer graphischen Benutzeroberfläche zum Steuern und Anzeigen von Informationen der Wallbox, sodass es direkt an der Einheit oder über ein lokales Webinterface eine Übersicht über den aktuellen Status der Wallbox gibt und diese gesteuert werden kann. Hierzu gibt es eine Übersicht welche Mikroprozessoren hierzu in Frage kommen könnten, die Wahl des Mikroprozessors wird in Abschnitt 4.1.1 beschrieben.

3.7.1. Mikroprozessoren

In Tabelle 7 sind verschiedene Mikroprozessoren aufgelistet mit der Möglichkeit einer Anbindung an ein Netzwerk und zudem mit genug Leistung für Steuer und Visualisierungsaufgaben. Zudem haben bis auf den RockPiX alle als Standardsoftware eine Linux Grundlage.

Tabelle 7 alternative Einplatinencomputer

Modell	Preis	Features	Quelle
Unihiker	102€	Geringer Arbeitsspeicher, kleines Display, 4 Kern Prozessor 1,2 GHz	[35]
ASUS Tinker Board	120€	Ähnlich wie Raspberry 4B	[36]
Banana Pi	65€	Ähnlich wie Raspberry 4B nur mehr ein und Ausgänge	[37]
RockPi X	140	Intel Atom x5-Z8350-Prozessor, win10 sonst ähnlich zu Raspberry 4 B	[38]
Raspberry Pi	50-250€	Abhängig der Variante	

3.7.1.1. Raspberry Pi

Bei dem Raspberry Pi besteht die Möglichkeit eine Linux Distribution zu installieren. Hierzu gibt es die Standardsoftware Raspbian. Unter Raspbian, dem Betriebssystem, lässt sich zusätzliche Software installieren. Zum Anzeigen und Steuern wird von für den Raspberry ein 7 Zoll Touchdisplay angeboten, wie in Abbildung 16 dargestellt. Dies ermöglicht eine einfache Steuerung über die dargestellte grafische Benutzeroberfläche.

3.7.1.2. Verschiedene Modelle

Von dem Raspberry Pi gibt es verschiedene Modelle. Diese bieten als Grundlage ein Linux Betriebssystem und werden von Modell Zero über Modell 4B und zu Modell 5 von der Leistung stärker und dementsprechend auch teurer in der Anschaffung. Hierbei empfiehlt sich für die meisten Aufgaben der Pi 4B da dieser für Datenverarbeitung und eine visuelle Darstellung genug Leistung zur Verfügung stellt, während Typ 5 für deutlich komplexere Aufgaben genutzt werden kann.

3.8. Programmiersprachen

Zur Programmierung gibt es verschiedene Programmiersprachen, welche im Folgenden mit ihren Vor- und Nachteilen kurz erläutert werden, sodass im Abschnitt 4.6 der Umsetzung für eine möglichst praktikable Programmiersprache entschieden werden kann.

C# ist eine hardwarenahe Programmiersprache welche durch die Entwicklung von Microsoft gut mit der Windows Plattform interagieren kann und hierbei zu einer effizienten Entwicklung in der Windows Umgebung führt. Des Weiteren bietet C# Funktionen welche die Entwicklungszeit verkürzen können und den Code lesbarer machen. Zudem findet eine Starke Typisierung statt, dies reduziert Fehler und verbessert die Codequalität. Ein negativer Punkt ist die Plattformabhängigkeit zu Windows, welches die Portierbarkeit zu anderen Betriebssystemen einschränkt [39].

C++ ist ähnlich zu C# eine hardwarenahe Programmiersprache, ermöglicht die Kontrolle über Hardware-Ressourcen und kann für viele Anwendungen verwendet werden. Zudem besteht ein direkter Zugriff auf den Speicher welcher die Optimierung von den verwendeten Ressourcen ermöglicht. Dadurch, dass C++ bekannt ist für seine Komplexität führt dies zu einem entsprechenden Entwicklungs- und Debugging Aufwand. Zudem können durch die Kontrolle über den Speicher Speicherlecks oder unsicherer Code die Folge sein [40].

Java bietet den Vorteil, dass es auf verschiedenen Plattformen ausgeführt werden kann und es eine objektorientierte Programmiersprache ist. Dies führt zu wiederverwendbarem und modularem Code zudem verfügt Java über umfangreiche Bibliotheken welche die Entwicklung beschleunigen kann. Nachteil ist, dass die Performance von Java langsamer sein kann als eine kompilierte Sprache und nicht direkt auf die Speicherverwaltung zugegriffen werden kann [41].

Die Programmiersprache Python bietet eine einfache Syntax an, wodurch die Lesbarkeit verbessert und die Entwicklung vereinfacht wird. Zudem verfügt sie über sehr umfangreiche Bibliotheken welche für verschiedenste Anwendungsbereiche genutzt werden können. Bei Python kann die Performance geringer sein als bei einer kompilierten Sprache da diese interpretiert wird. Es kann außerdem zu Fehlern in der Typisierung kommen, da es sich hierbei um eine dynamische Typisierung handelt [42].

Zur Weboberflächenprogrammierung kann die Auszeichnungssprache HTML verwendet werden mit den Zusätzen CSS zum Styling und JavaScript zum Ausführen von Aufgaben. Ein Vorteil von HTML ist, dass es leicht anzuwenden ist und von allen gängigen Browsern unterstützt wird, wodurch es plattformübergreifend genutzt werden kann. Es ist für statische Inhalte verantwortlich und die Strukturierung dieser. Um dynamisch das Design zu optimieren wird CSS und JavaScript benötigt [43].

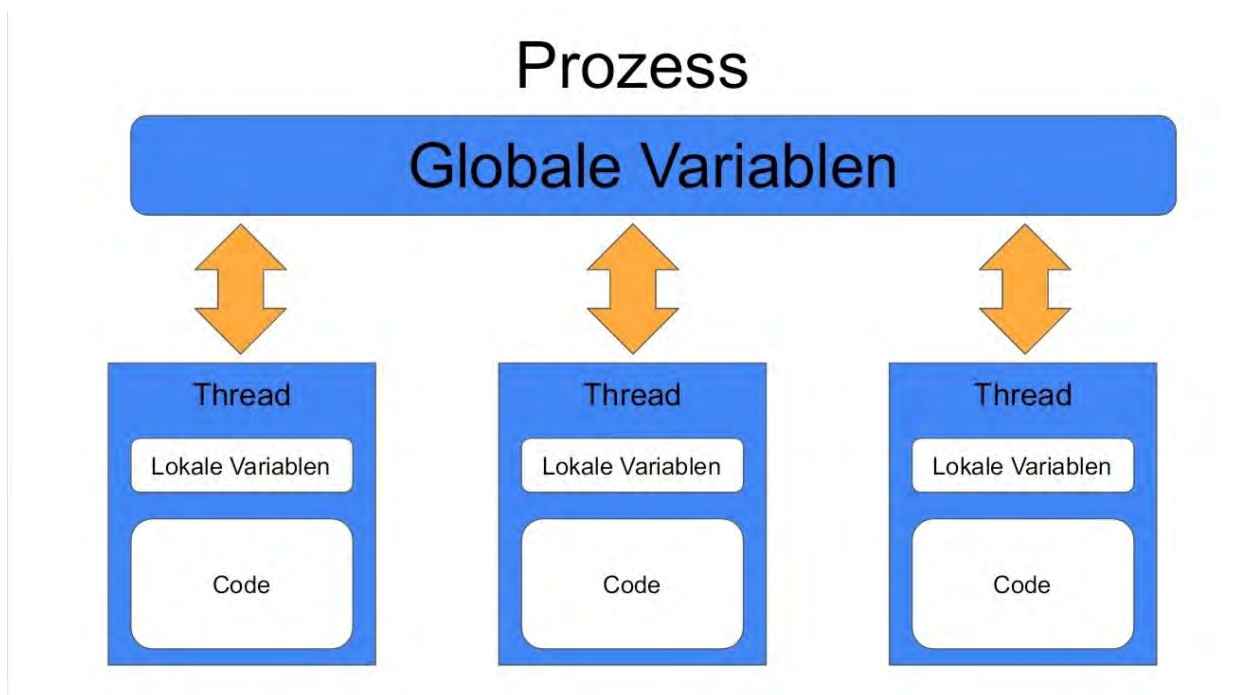
CSS ermöglicht die Trennung von HTML Inhalten und dem visuellen Design, zudem ermöglicht CSS, dass das selbe Design auf mehreren oder sogar allen Seiten identisch ist zudem bietet es die Möglichkeit das Design an verschiedene Bildschirmgrößen anzupassen. Ein Nachteil an CSS ist, dass unterschiedliche Browser dies unterschiedlich interpretieren könnten und es zu verlängerten Ladezeiten führen kann [44].

JavaScript ermöglicht die Erstellung von interaktiven Benutzeroberflächen und Inhalten, es kann für die Webentwicklung als auch für die Entwicklung im Backend genutzt werden. Zusätzlich bietet es viele Bibliotheken und Frameworks an welche die Entwicklung erleichtern können. Ein Nachteil ist, ähnlich zum CSS, dass verschiedene Browser dies unterschiedlich interpretieren könnten und JavaScript für Sicherheitslücken anfällig sein kann [42].

3.8.1. Threading

In Python besteht die Möglichkeit Threading zu nutzen hierbei gibt es zwei Arten Kernel-Threads und User-Threads. Kernel-Threads sind ein Teil des Betriebssystems und User-Threads sind vom Anwender. Hierbei sind für uns User-Threads sinnvoll. Ein Thread ist als Funktionsaufruf zu verstehen. Jeder Prozess besteht aus mindestens einem Thread. Ein Prozess kann mehrere Threads starten und diese laufen scheinbar gleichzeitig ab. Zudem wird aus Abbildung 15 klar, dass verschiedene Threads welche von einem Prozess aus gestartet wurden denselben Speicherbereich für globale Variablen nutzt und diese deshalb Thread übergreifend genutzt werden können. Sollten globale Variablen in mehreren Threads genutzt werden und diese verändert werden besteht die Möglichkeit die Variable vor der Nutzung zu sperren und anschließend wieder freizugeben. So können in der Zeit, in welcher diese genutzt werden, keine Änderungen geschehen. Andere Threads welche diese Variable nutzen warten, sollte diese gesperrt sein [45].

Abbildung 15 Multithreading Variablen



3.8.2. Async

Statt Threading besteht in Python auch die Möglichkeit asynchrone Funktionen zu bauen. Hierzu muss bei der betreffenden Funktion vor dem „def“ das Schlüsselwort `async` stehen sodass diese entsprechend definiert ist. Zudem kann jetzt in diesem Prozess eine weitere asynchrone Funktion (auch Coroutine genannt) aufgerufen werden. Auf welche mit `await` gewartet wird, bis das Ergebnis von der Ausführung vorhanden ist. Hierbei ist es nicht sinnvoll eine Funktion ohne ein `await` zu bauen, aber trotzdem möglich. Eine Asynchrone Funktion wird mit einem `asyncio.run(Funktion)` ausgeführt. Zudem sollte hierbei vermieden werden durch Schreibzugriffe auf Dateien einen Prozess zu blockieren. Da dies Zeit benötigt, bis eine solche Anfrage an das System eine Rückmeldung bekommt.

4. Umsetzung

In diesem Kapitel wird die Umsetzung dieser Arbeit erläutert, welche Voraussetzungen erfüllt werden müssen, wie der Microprozessor ausgewählt wurde, wie der Wallboxen Vergleich stattgefunden hat und was bei den Entscheidungen ausschlaggebend war. Zusätzlich wird noch erläutert, wie die Programmierung der Steuerungseinheit ablief.

4.1. Voraussetzung für die Umsetzung

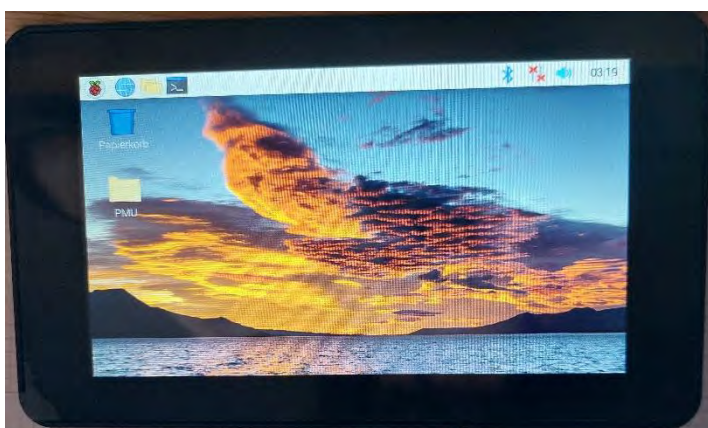
Als Voraussetzung wird eine Darstellung und Steuerungsmöglichkeit benötigt, hierzu muss sich für einen Steuergerät entschieden werden. Hierbei wurde die Vorauswahl bereits auf einen Einplatinencomputer entschieden, da diese kompakt sind und die benötigte Leistung zur Verfügung stellen. Des Weiteren muss sich zur Umsetzung, für eine Wallbox mit einer passenden Schnittstelle entschieden werden. Aus den Hintergrundinfos zu den Protokollen geht hervor, dass OCPP ein verbreitetes Protokoll ist, welches überwiegend in Version 1.6 implementiert ist.

4.1.1. Auswahl des Microprozessors

Die Wahl des Microprozessors ist auf den Raspberry Pi 4B gefallen, da dieser wie in Abschnitt 3.7.1 beschrieben mit dem Standard Betriebssystem Raspbian einfach zu bedienen ist. Mit den Komponenten entsprechend genug Leistung zur Steuerung und Visualisierung bietet und zu dem Unterschied des BananaPi weniger Ein- und Ausgänge besitzt, welche nicht benötigt werden. Zudem gibt es zum Raspberry Pi ein 7 Zoll Touchdisplay, wie in Abbildung 16 dargestellt. Dies ermöglicht eine einfache Steuerung über die dargestellte grafische Benutzeroberfläche.

Der Raspberry Pi bietet zusätzlich die OnBoard-Schnittstellen Ethernet und WLAN, die ein Webinterface ermöglichen. Zur Nutzung wird eine Software mit Webserver oder lediglich ein Webserver mit den passenden Inhalten benötigt. Des Weiteren kann bei Bedarf ein größerer Monitor angeschlossen werden oder über SSH oder VNC über das Netzwerk auf den Raspberry Pi zugegriffen werden.

Abbildung 16 Raspberry PI 7" Touchdisplay



4.2. Wahl der Wallbox

In diesem Unterkapitel wird auf die Unterschiede zwischen den verschiedenen Wallboxen eingegangen. Dabei wird erst auf alle Wallboxen, welche betrachtet wurden eingegangen und anschließend der Unterschied zwischen der preiswerten Variante und den Vorteilen der mittel- und hochpreisigen Ausführungen erläutert. Dazu wurde für die günstige Variante aus der Tabelle 8 die „a-TroniX Wallbox Home plus“ ausgewählt. Für die Mittelklasse wurde der „go-eCharger HOMEfix“ und für die hochpreisige Variante die „ABB Terra Wallbox“ gewählt. Zu der Entscheidung, welche Wallbox genutzt wird, wird in Abschnitt 4.4 eingegangen.

Zur Betrachtung der Wallboxen wurden Wallboxen gewählt, welche eine Ansteuerung unterstützen, da es auf dem Markt auch Wallboxen gibt, welche eine externe Steuerung nicht zulassen.

Eine dieser Wallboxen war die EV Box Elvi V2, welche es in unterschiedlichen Ausführungen gibt. Diese bietet unabhängig von der Leistung nur eine Backendanbindung mit OCPP 1.6. Zudem ist eine Autorisierung immer möglich. Diese gibt es mit einer Leistung von 11 kW und 22 kW. Es gibt auch Wallboxen wie die go-eCharge Homefix, welche mehr als eine Art der Anbindung unterstützen. Auf diese wird im folgenden Abschnitt Mittelpreisig näher eingegangen.

Des Weiteren wird aus der Tabelle ersichtlich, dass je nach Wallbox, neben Unterschiedlichen Protokollen, auch unterschiedliche Möglichkeiten geboten werden diese zu steuern, ob dies über eine App oder ein Webportal möglich ist. Oder es bei diesen Einstellungsmöglichkeiten die Autorisierung über ein RFID Chip ist. Hierbei ist möglich die Leistung der Wallbox frei zu wählen oder bis wann das Auto vollgeladen sein soll. Dies funktioniert je nach Hersteller unterschiedlich. Ein gemeinsamer Nenner bei den aufgelisteten Wallboxen ist das OCPP-Protokoll über welches die Wallbox gesteuert werden kann. Und dies ist Wallbox-Hersteller unabhängig möglich.

Wallboxart	Protokolle	Leistung	Kosten	Autorisierung möglich	DC Fehlerstromerkennung	Bewertung
EVBox Elvi V2 (E3160-A45062-10.2)	OCPP 1.6-	11kw-22kw	404€	Ja, RFID	Ja	1 (günstig mit Autorisierung und bis 22kw)
go-eCharger HOMEfix	HTTP API (Lokal + Cloud), MQTT, OCPP 1.6, Modbus TCP, Netzbetreiber API	11kw oder 22kw	599€	Ja, RFID	Ja	
ABB Terra Wallbox [6]	OCPP1.6 RS485/P1 für Energiezähleranschluss Ethernet RJ45	22kw drosselbar auf 11	1000 €	Ja, RFID, App	Ja	Sehr teuer, hohe Nutzerfreundlichkeit
Vestel EVC04 [7]	WiFi, LTE mit OCPP 1.6 zertifizierter MID, Ethernet, Smart Charging ISO15118	11kw	600€	Ja, RFID, App	Ja	
WB24 Wallbox	EEBUS, OCPP 1.5 & 1.6, modbus TCP SMA SEMP	11-22 kw	759€	optional	Ja	Bietet funktionale extras
Wallbox Pulsar Plus	OCPP	4,2 bis 22kw	680€	App oder Portal	Ja	
Entratek Wallbox Power Dot Pro	OCPP 1.6	22kw	740€	Ja, RFID	Ja	
Wallbox copper SB	OCPP 1.6	22kw	730€	RFID, App, Portal	Ja	
a-TroniX Wallbox Business	OCPP 1.6	22kw	800€	RFID	Ja	
a-TroniX Wallbox Home plus	Ocpp 1.6	11kw	400€	RFID	Ja	Günstigstes nur 11kw möglich

Tabelle 8 Wallboxvergleich

4.2.1. *Preiswerte Wallbox*

Die preisgünstigste Version bietet die im Folgenden genannten Zusatzfunktionen für den Ladevorgang an. Es ist möglich bis zu 11 kW zu laden, der Ladevorgang kann mit einer RFID Karte freigeschaltet werden. Ein serienmäßiger FI-Schutzschalter und ein 3 poliger Stecker sind enthalten. Zusätzlich wird das OCPP 1-6 Protokoll unterstützt, welches die Anbindung von zusätzlichen Features ermöglicht.

4.2.2. *Mittelpreisige Wallbox*

Die gewählte Wallbox der Mittelklasse bietet eine maximale Ladeleistung von 22 kW. Dies ist doppelt so viel wie bei der günstigen Variante. Zusätzlich bietet diese den Vorteil der Steuerung und Planung des Ladevorgangs über eine App. Es wird zudem ein statisches Lastmanagement und eine Anbindung an eine Photovoltaikanlage über eine API-Schnittstelle angeboten. Die Verbindung selbst muss vom Endnutzer erst programmiert werden, es wird nur die Möglichkeit geboten diese Schnittstelle zu nutzen. Durch eine API-Schnittstelle können Geräte über ein Netzwerk miteinander kommunizieren. Durch eine solche Anbindung kann tagsüber der Strom von der PV-Anlage effektiv genutzt werden. Des Weiteren besteht bei manchen Stromanbietern die Möglichkeit nachts vergünstigt Strom zu nutzen, welcher hierüber dann in das Fahrzeug geladen werden kann [46].

4.2.3. *Hochpreisige Wallbox*

Die gewählte hochpreisige Version beinhaltet identische Funktionalitäten wie bei der mittelpreisigen Versionen. Zusätzlich ist dieses Modell über ein Webportal konfigurierbar, bietet einen integrierten Energiezähler und ermöglicht die Integration eines externen Energiezählers für ein dynamisches Lastmanagement. Damit ist diese Wallbox für die Integration in moderne, intelligente Gebäudeenergiesysteme ausgelegt. Dies beinhaltet das Überschussladen, dass nur der Überschüssige Strom welcher von der PV-Anlage in das Hausnetz gespeist und im Haus nicht benötigt, in das Auto geladen wird und kein Strom aus dem Netz zugekauft wird.

4.3. **Vergleich der Wallboxen**

Zum Vergleichen der Wallboxen wurden die relevantesten Faktoren der Wallboxen zusammengetragen und aufgeteilt um diese detaillierter zu erläutern.

Jede Wallbox hat eine maximale Ladeleistung welche diese zur Verfügung stellen kann. Für eine normale Haushaltswallbox ist eine Leistung von 11 kW üblich, je nach Wallbox und Anschluss können diese aber bis zu 22 kW Ladeleistung freischalten. Bei der unterschiedlichen Leistung müssen zudem die örtlichen Gegebenheiten passen.

Bei Wallboxen werden zur Anbindung an andere Systeme unterschiedliche Protokolle, wie bereits in Absatz 3.2 erläutert, zur Kommunikation unterstützt. Hierbei ist ein Ziel dieser Arbeit, dass die Ansteuerungseinheit möglichst viele Wallboxen unterstützt. Weshalb bei den Protokollen eines gewählt werden muss, welches von möglichst vielen Wallboxen unterstützt wird und es zudem in der Zukunft weiterhin unterstützt wird.

Zur Sicherheit hat jede Wallbox verschiedene Sicherheitsaspekte welche eine wichtige Rolle spielen hierbei handelt es sich um eine Gleichspannungsfehlerstromerkennung, welche verhindert, das Gleichspannung ins Wechselspannungsnetz gerät und FI-Schalter außer Kraft setzt.

Um eine fremde Nutzung zu verhindern oder um den Verbrauch der Nutzer zur späteren Abrechnung zu tracken ist es zudem Interessant, ob die Wallbox über Autorisierungsmöglichkeiten verfügt und wie diese möglich sind. Hierbei gibt es mehrere Möglichkeiten: die Wallbox mit einem RFID Chip freischalten, über eine App des Herstellers oder über ein Webinterface. Über einen RFID-Chip kann

ein Nutzer einfach identifiziert werden, über eine App oder über das Webinterface müsste sich hierzu angemeldet werden.

4.4. Entscheidung für WB24 mit dem OCPP Protokoll

Die Entscheidung für eine Wallbox mit dem OCPP-1.6 Protokoll wurde getroffen, da von 15 Wallboxen 11 das OCPP-Protokoll unterstützen und somit ca. 73% der gewählten Wallboxen die Steuerungseinheit unterstützen. Weiterhin soll dies standardmäßig in immer mehr Wallboxen verbaut werden [47].

Die Wallbox WB24 der EC Serie wurde gewählt, da diese aufgrund der modularen Bauweise eine einfache Erweiterung der Funktionalitäten bietet, sowie kostensparende Reparaturen möglich sind. Dies ist bei anderen Vergleichsmodellen nicht der Fall. Zudem ist bei dieser Wallbox der Ladestecker direkt mitinbegriffen und muss nicht zusätzlich erworben werden.

4.5. Umsetzung an der Wallbox

In folgenden Abschnitten wird darauf eingegangen, wie die Wallbox angeschlossen wurde und wie diese nach dem Anschluss an ein Netzwerk über die Weboberfläche konfiguriert wurde.

4.5.1. Anschließen der Wallbox

Beim Anschließen der Wallbox mussten mehrere Dinge beachtet werden. Wie in aus dem Datenblatt (siehe Anhang) erkennbar ist muss die gesamte Wallbox mit Dreiphasenwechselstrom wie in Kapitel 3.3 bereits erläutert angeschlossen werden. Bei den Komponenten, welche mitgeliefert wurden handelt es sich um die Teileliste welche im Anhang hinterlegt ist. Die wichtigsten Teile sind ein Installationsschutz mit 4 Schließer Kontakten, ein Leergehäuse 2-reihig, ein Bender Messstromwandler, ein Bender Laderegler und ein ABB Hauptschalter.

Diese Teile sind wie in Abbildung 17 dargestellt, im Leergehäuse eingebaut und im Labor montiert. Hierbei wird zudem erkenntlich, dass dies sehr kompakt ist und einem kleinen Sicherungskasten sehr ähnlich sieht. Die Wallbox wird über einen CEE-Stecker mit 16A 5 Polig, wie in Abbildung 18 dargestellt, angeschlossen und über ein Ethernet Kabel mit dem internen Netzwerk des Labors verbunden.



Abbildung 17 Wallbox zusammgebaut

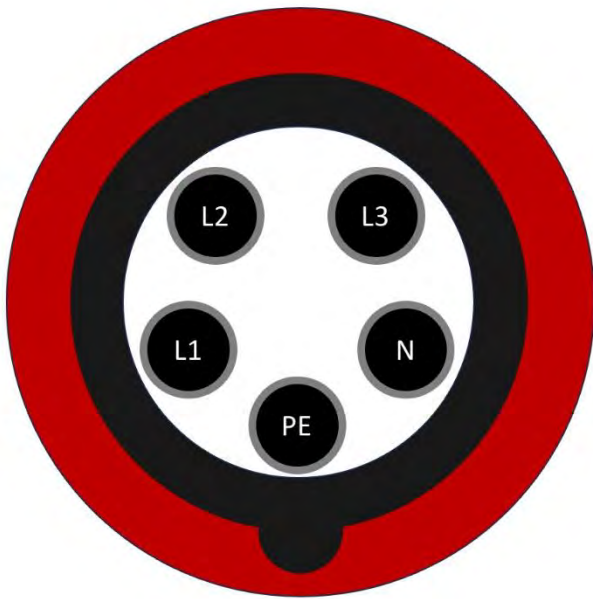


Abbildung 18 Steckerbelegung CEE-Stecker 400V 16A

4.5.2. Einrichten der Wallbox

Nachdem die Wallbox an die Spannungsversorgung angeschlossen wurde, musste diese so eingerichtet werden, dass diese sich mit der Ansteuerungseinheit verbindet. Hierzu muss der Nutzer sich auf dem Webinterface der Wallbox als Operator einloggen und verschiedene Anpassungen vornehmen. Die Einstellungsmöglichkeiten sind in Abbildung 19 zu sehen. Diese wurden so umgesetzt, wie in der Abbildung erkenntlich ist. Ganz oben muss angegeben werden wie die Wallbox in das Netzwerk eingebunden werden soll. Hier wurde Ethernet gewählt. Zudem wird die ChargePointID angezeigt und kann verändert werden. Es muss auch festgelegt werden, über welche Version des OCPP-Protokolls kommuniziert werden soll, hierzu gibt es wie in Kapitel 3.2.2 bereits erläutert mehrere Möglichkeiten. In diesem Fall, wie in der Abbildung zu sehen, ist die Version 1.6 gewählt worden. Diese Version bietet die Kommunikation im JSON-Format über Websockets und im SOAP-Format an. In der neueren Version 2.0 wird ausschließlich die Kommunikation im JSON-Format über Websockets angeboten. Dadurch bietet es sich an, Version 1.6 im JSON-Format zu nutzen, da dies noch am weitesten verbreitet ist und bei Bedarf auf Version 2.0 upgrade bar ist. Das Upgrade auf 2.0 kann hierbei einfach umgesetzt werden, indem die Version von 1.6 auf 2.0 angepasst wird.

Des Weiteren muss die URL des Central-Management-Systems angegeben werden. Die einzelnen Teile wurden bereits in Abschnitt 3.2.2 erläutert. Hier ist eine IP-Adresse angegeben, welche aber auch durch einen Host-Namen ersetzt werden kann. Zudem können nähere Details zu der Verbindung hier eingestellt werden. Dies wird in der Abbildung 19 ersichtlich. Die Titel der Einstellungsmöglichkeiten sind soweit selbsterklärend, sodass eine Erläuterung nicht notwendig ist.

BACKEND

Verbindung

Verbindungstyp

OCPP

OCPP ChargeBoxIdentity (ChargePointID)

OCPP Modus

WebSockets JSON OCPP URL des Backends

Websockets-Proxy

WebSockets Keep-Alive-Intervall

HTTP Basic Authentication Passwort

Heartbeat Nachrichten immer senden

Sende informative StatusNotifications

Sende StatusNotifications für Fehler

USB-Fehler über StatusNotifications senden

Strategie für StatusNotification-Zustandsübergänge

Langes Abrufen von Konfigurationsschlüsseln erlauben

Laden unterbinden bei andauernder Backend-Störung

Zustand 'verfügbar' gegenüber dem Backend erzwingen

Andere

Timeout der Backend-Verbindung

Anzahl der Versandversuche von transaktionsrelevanten Nachrichten

Anzahl der Sendeveruche von transaktionsrelevanten Eichrecht Nachrichten

SSL Modus als Client



TCP Watchdog Timeout

Backend-Verbindungsausfall als Fehler signalisieren

Abbildung 19 Wallbox einrichten

Zum Ersten kann in der Wallbox noch festgelegt werden, wie die Freigabe des Ladens gesteuert werden kann. In Abbildung 20 ist dargestellt, ob kostenloses Laden und die Authentifizierung stattfinden kann. Zudem gibt es noch nähere Einstellungen zu der Verbindung, wie Timeout zum Fahrzeug und RFID-Einstellungen.



AUTORISIERUNG**Kostenloses Laden**

Kostenloses Laden		An
Kostenloses Laden Modus		Mit OCPP-Statusmeldung ohne Authentifizierung

Überblick

Timeout für die Fahrzeugverbindung		45
Sende OCPP Authorize für RemoteStart Anfragen		An
Transaktionsmodus stoppen		Normal
Aktuator nur bei Autorisierung schließen		Aus

RFID Einstellungen

RFID Tag Groß/Kleinschreibung		Kleinschreibung
Sprache der Display-Anzeige		Multi-Language EN-DE-FR-NL

RFID Whitelists

Lokale-Whitelist aktivieren		Aus
OCPP-Whitelist aktivieren		An
OCPP-Whitelist-Ablaufmodus		Ende der Epoche 2038 (Standard)
Lokale Vorautorisierung		An
Lokale Offlineautorisierung		An

Abbildung 20 Wallbox Autorisierungs-Einstellungen

Zum Zweiten bietet die Wallbox ein sogenanntes Dashboard, auf welchem der aktuelle Systemstatus angezeigt wird. Für uns besonders relevant, ist bei dem aktuellen Systemstatus, die Charge-Point-ID, der OCPP-Status, der Verbindungsstatus und ob es aktuell Fehler gibt. Die relevanten Zeilen sind in Abbildung 21 die folgenden, OCPP-Status, Verbindungsstatus, Fehler und Netzwerk.

Systemstatus	
Name	Value
OCPP ChargeBoxIdentity (ChargePointID)	+49*839*00000000001
OCPP Status	FREI (verfügbar)
Status des Type2 Anschlusses	(A) Fahrzeug nicht verbunden Angeschlagenes Kabel AMCC: (-/-/-)
Angebotener Strom	0 A
Verbindungsstatus (Backend)	Not Connected (Verbindungsfehler) Offline seit 0:00:18:26 (d:h:m:s) (verbinde neu in 457 Sekunden)
Kostenloses Laden	On (No OCPP)
Fehler	No errors
RDC-M (RCMB) Status	DC: OK, RDC-M (RCMB) Device Status (IEC 62955): OK Last transaction maximum DC: 0,1 mA Values DC: 0,1 mA
Schaltzyklen des Lastschützes Typ2	4/10.000.000
Steckzyklen des Typ 2-Anschlusses	2/50.000
Netzwerk	eth0: [E0:AE:B2:08:5A:52] IP: 192.168.11.132 (ocpp)
Eichrechtsfunktionen	Ladepunkt ohne Eichrecht

Abbildung 21 Dashboard der Wallbox

4.6. Wahl der Programmiersprache & Umgebung

Zur Entwicklung der Ansteuerungseinheit musste sich für eine Programmiersprache und eine Programmierumgebung entschieden werden. Hierzu gibt es wie in Abschnitt 3.8 beschrieben mehrere Möglichkeiten, welche hier verwendet werden können. Dabei ist eine Windows nahe Programmiersprache ungeeignet, da deren Ausführung, auf Linux basierten Betriebssystemen, nicht ohne erheblichen Mehraufwand möglich ist. Des Weiteren spricht gegen Java, die umständliche Syntax und die Syntax von Python hierzu einfacher ist. Des Weiteren ist ein Raspberry Pi vor allem hierzu ausgelegt Python zu unterstützen weshalb dies das Naheliegendste ist. Ein weiterer Vorteil besteht in den umfangreichen Python-Bibliotheken, welche in dieser Arbeit genutzt werden können.

Als Programmierumgebung wird die IDE Pycharm Community Edition genutzt. Diese bietet einen intelligenten Python-Editor, einen grafischen Debugger und Test-Runner, Möglichkeiten zur Navigation und Refaktorisierung, Codeinspektion und VCS-Unterstützung. Zudem kann zur Versionsverwaltung ein Git-Dienst verwendet werden.

4.7. Programmierung des Raspberry Pi

Für die Programmierung des Raspberry Pi mussten zwei Dinge beachtet werden, einmal die Anbindung an die Wallbox, sowie die Visualisierung auf dem Raspberry Pi und die Visualisierung des Webinterfaces.

4.7.1. *Interface und Einstellungsmöglichkeiten*

In Abschnitt 2.3 ist beschrieben welche Frameworks für das Webinterface genutzt werden können. Hierbei sind Zabbix und Grafana ausgeschlossen worden, da es mit diesen zwar möglich ist das Projekt umzusetzen, diese jedoch mehr zum Anzeigen von Daten gemacht sind und nicht um etwas zu steuern. Ein anderer Ansatz mit Node-Red bietet sich an, ist jedoch unübersichtlicher als Code, da es eher graphische Programmierung ist. Zudem könnte das System Django genutzt werden, dies ist mit dem breiten Funktionsumfang für diese Demo zu überdimensioniert und fällt ebenfalls heraus. Weshalb sich für das Framework Dash entschieden wurde. Hierbei ist es einfach eine Oberfläche darzustellen und Buttons mit Python Skripten zu kombinieren.

Für die Visualisierung auf dem Pi sowie der Weboberfläche wurde sich zu Demozwecken für ein Webserver entschieden, welcher aus der Bibliothek Dash heraus gestartet werden kann. Hierbei besteht die Möglichkeit einfach Graphen, Buttons, Tabs und vieles mehr auf einem Webinterface darzustellen. Zur Darstellung auf dem Touch-Interface, ist es gedacht den Standardwebbrowser des Raspbian Systems, im Autostart mit der passenden Startseite zu starten. Das Interface ist somit direkt an der Steuerungseinheit geöffnet. Dies erleichtert, im Rahmen von Softwareerweiterungen, eine Anpassung, da diese nur im Webinterface gesehen müssen.

Das Webinterface besteht, wie Abbildung 22 zu entnehmen, aus zwei Teilen: Einer Übersicht und einem Verlauf. In der Übersicht wird der aktuelle Status angezeigt und es ist möglich auf verschiedene Funktionen zuzugreifen wie Abschalten oder Starten des Ladevorganges. Diese aktuellen Informationen werden periodisch aktualisiert. Hierzu gibt es im Verlauf des Unterkapitels noch nähere Informationen.

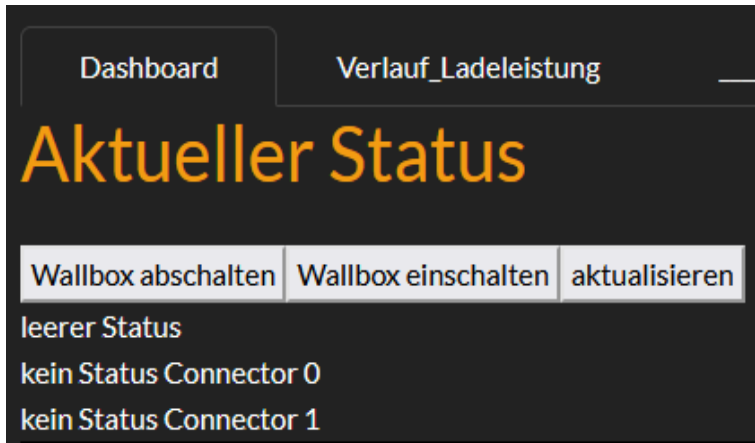


Abbildung 22 Dashboard Übersicht

Diese Einstellungsmöglichkeit ist nur geboten, wenn eine aktive Verbindung vorhanden ist. Zum konstanten Aktualisieren der Oberfläche, wird in Dash die Möglichkeit geboten, über `dcc.Interval` in regelmäßigen Zeitabständen, eine `CallBack` Funktion ausführen zu lassen. Diese wird genutzt um die aktuellen Statusinformationen des OCPP-Servers zu laden und zu überprüfen, ob dieser noch eine Verbindung hat. Das Intervall wird in Millisekunden angegeben, es muss ein Startwert angegeben werden, als Anzahl wie oft dies bereits gelaufen ist. In Abbildung 23, ist diese Konfiguration gezeigt. Es wird alle Sekunde abgefragt und startet bei 0 Intervallen [48].

```
dcc.Interval(  
    id='interval-component',  
    interval=1*1000, # in Millisekunden  
    n_intervals=0  
)
```

Abbildung 23 Intervalle in Dash

Zum Aktualisieren der Inhalte wird das sogenannte Threading eingesetzt. Dies sorgt dafür, dass Prozesse nahezu gleichzeitig ausgeführt werden können, wie bereits in Kapitel 3.8.1 erläutert. In diesem Fall findet es Anwendung um die Daten welche von der Wallbox empfangen werden zu empfangen und visuell darzustellen. Durch das Threading werden diese Daten kontinuierlich geladen, mit einer geringeren Pause als das Intervall, welcher dies anzeigt.

4.7.2. Anbindung an die Wallbox

Die Anbindung der Ansteuerungseinheit an die Wallbox, geschieht über ein Ethernet Kabel. Hierbei können beide in ein bestehendes Netzwerk eingebunden werden. Bei einer direkten Verbindung ist es wichtig die IP-Adressen so zu konfigurieren, dass diese sich im selben Subnetz befinden und sich auch finden können. Kommuniziert wird über das OCPP-Protokoll, hierbei bietet der Raspberry Pi einen Websocket, mit welchem sich die Wallbox verbinden kann. Das genaue Vorgehen ist bereits in Kapitel 3.2.2 beschrieben. Hierbei wird, während dem Verbindungsaufbau, über die URL die Client ID übergeben.

Jetzt besteht entweder die Möglichkeit, dass die Wallbox verschiedene Befehle an das Central-Management-System sendet. In Tabelle 9 ist kurz erläutert welche Befehle es gibt und wofür diese nützlich sind.

Tabelle 9 Von dem Ladepunkt mögliche Befehle mit Beschreibung

Anfrage	Beschreibung
Authorize	Anfrage ob ein idTag laden darf
Boot Notification	Registriert sich am CMS
Data Transfer	Überträgt Informationen an CMS
Diagnostics Status Notification	Diagnose Status Informationen
Firmware Status Notification	Firmware Status Information
Heartbeat	Lebenszeichen an das CMS
Meter Values	Messwerte der Wallbox
Start Transaction	Information Laden gestartet an Connector
Stop Transaction	Information Laden gestoppt an Connector
Status Notification	Status Information mit Status, connectorId and error Code

Des Weiteren gibt es verschiedene Befehle, welche vom Central-Management-System aus, gesendet werden können. Diese sind in Tabelle 10 Befehle des CMS an die Wallbox mit Beschreibung aufgelistet und kurz beschrieben

Tabelle 10 Befehle des CMS an die Wallbox mit Beschreibung

Anfrage	Beschreibung
Cancel Reservation(reservationID)	Eine Reservierung mit der ID widerrufen
Change Availability(connectorID,type)	Ändert die Verfügbarkeit eines Connectors
Change Configuration(kex, value)	Änderungen an der Konfiguration mit dem key, value Prinzip
Clear Cache	löscht den Cache der Autorisierungen
Clear Charging Profile	löscht alle oder die übergebenen Ladeprofile
Data Transfer(vendorID, messageID, Data)	Zum übertragen von Daten die von OCPP nicht unterstützt werden
Get Composite Schedule	Ruft Ladezeitpläne ab
Get Configuration	Ruft die Konfiguration für den übergebenen Parameter ab oder gibt eine Liste mit allen zurück
Get Diagnostics	Fragt eine Diagnose an für einen Zeitraum
Get Local List Version	Fragt die Autorisierungen als Liste ab
Remote Start Transaction	Startet Laden
Remote Stop Transaction	Stoppt das Laden
Reserve Now	Reserviert einen Ladepunkt für einen IDTag
Reset	Für einen weichen oder harten Reset des Ladepunktes
Send Local List	Fragt die Lokale Autorisierungsliste ab
SetChargingProfile(connector, csLadeprofil)	Setze ein Ladeprofil
Trigger Message	Triggert eine Ausgabe von dem übergebenen Nachrichten Typ an
Unlock Connector	Um die Verbindung manuell zum Auto zu lösen
Update Firmware	Den Ladepunkt an einem bestimmten Datum zum Updaten der Firmware auffordern

Damit die Wallbox eine Verbindung mit dem Raspberry Pi herstellen kann, muss auf dem Pi erstmals der Server gestartet werden. Hierzu sieht man in Abbildung 24 Codeausschnitt OCPP-Server initialisieren, dass angegeben wird auf welche IP-Adressen gelauscht wird und auf welchem Port dies empfangen werden kann. Hierzu bedeutet bei der IP-Adresse dieses 0.0.0.0, dass auf alle eingehenden Verbindungen gelauscht wird, während die 8001 dies auf Port 8001 begrenzen. Des Weiteren sorgt die Angabe des Sub-Protocol dazu, dass über OCPP 1.6 kommuniziert wird. Dies bedeutet nach der ersten Verbindungsanfrage wird die Kommunikation auf OCPP 1.6 geändert. Der ws_handler sorgt zudem dafür, dass die eingehende Verbindung verarbeitet werden. Hierzu wird der Handler in folgenden nochmals betrachtet.

```
# Initialisiere den WebSocket-Server
server = await websockets.serve(
    ws_handler,
    '0.0.0.0',
    8001,
    subprotocols=['ocpp1.6']
)
```

Abbildung 24 Codeausschnitt OCPP-Server initialisieren

Sobald nun eine Verbindungsanfrage aufkommt, wird diese an den ws_handler geleitet, dieser extrahiert, aus dem Pfad, die Charge-Point-ID und gibt die Verbindung an die Charge-Point-Klasse weiter. Dieses Element der Charge-Point-Klasse wird anschließend in eine globale Liste connected_charge_points geschrieben und anschließend gestartet. Wie in Abbildung 25 gezeigt, wird sobald die Verbindung wieder beendet wird, das Element aus der Liste wieder entfernt.

```
async def ws_handler(websocket, path):
    charge_point_id = path.strip("/")
    cp = ChargePoint(charge_point_id, websocket)
    connected_charge_points.append(cp)
    print("Connected")
    try:
        await cp.start()
    finally:
        connected_charge_points.remove(cp)
```

Abbildung 25 Programmcode Verbindungsanfragen handler

Sobald die Verbindung mit der Wallbox hergestellt wurde, besteht die Möglichkeit vom Central-Management-System Nachrichten zu empfangen und auf diese zu reagieren. Hierbei ist in Abbildung 26 dargestellt, wie eine Boot-Notification empfangen, verarbeitet und beantwortet wird. In Abbildung 27 ist dargestellt, wie eine Anfrage zu einem Status update übertragen wird. Hierbei wird bei dem Boot-Notification sichtbar, dass die Nachricht von der Wallbox aus getriggert wird und eine Antwort des CMS benötigt wird. Sollte diese Antwort ausbleiben, erkennt die Wallbox, dass keine Verbindung mehr besteht und versucht diese erneut herzustellen. Hierbei verlängert sich die Zeit zwischen den Verbindungsversuchen immer weiter, solange keine erfolgreiche Verbindung hergestellt wird.

Wie bereits kurz erwähnt wird in Abbildung 27 sichtbar, wie vom CMS aus eine getriggerte Nachricht verarbeitet wird. Hierzu sendet das CMS eine Nachricht an die Wallbox, in unserem Fall, mit einem Trigger-Message, die Wallbox sendet anschließend eine Bestätigung an uns, dass die Anfrage akzeptiert oder abgelehnt wurde. Sobald die Änderung in der Wallbox umgesetzt wurde wird nun von der Wallbox ein Status an das CMS versendet.

Um die erste initiale Verbindung mit dem OCPP-Server zu bestätigen muss diese Nachricht beantwortet werden. Über das genutzte OCPP-Framework, wird die bereitgestellte mit einem Dekorators („@on(BootNotification)“) markierte Funktion aufgerufen und verarbeitet die Anfrage zur Verbindung. Nach der Verarbeitung wird wie in Abbildung 26 dargestellt mit der aktuellen Zeit, in welchen Interfällen ein Heartbeat gesendet werden soll in Sekunden und das die Verbindung akzeptiert wurde geantwortet.

```
@on('BootNotification')
async def on_boot_notification(self, charge_point_vendor, charge_point_model, **kwargs):
    logging.debug(f"BootNotification received: {charge_point_vendor} {charge_point_model}")
    return call_result.BootNotificationPayload(
        current_time=datetime.now(timezone.utc).isoformat(),
        interval=20,
        status=RegistrationStatus.accepted
```

Abbildung 26 Boot-Notification Empfangen

In einer identischen Art und Weise werden die übrigen Nachrichten empfangen und verarbeitet, je nach empfangener Nachricht wird ein Teil des Inhaltes an die Weboberfläche zum Anzeigen gesendet.

Es besteht jedoch auch die Möglichkeit, den aktuellen Status explizit von der Wallbox anzufragen. Dies ist mit der Funktion welche in Abbildung 27 dargestellt ist möglich. Hier wird erst die Nachricht zusammengesetzt und anschließend an die Wallbox gesendet. Diese liefert anschließend erst ein Accepted zurück und die Antwort wird mit in das Logging geschrieben. Kurze Zeit später wird eine Status-Notification empfangen und an die Weboberfläche weitergeleitet.

```
async def trigger_status_notification(self):
    request = call.TriggerMessagePayload(
        requested_message=MessageTrigger.status_notification
    )
    response = await self.call(request)
    logging.debug(f"Trigger Message Response: {response}")
```

Abbildung 27 Anforderung einer StatusNotification

4.7.3. Verbindung zwischen den Skripten

Die Verbindung zwischen den Skripten der Weboberfläche und des OCPP-Servers, läuft über RabbitMQ. Damit können diese einfach kommunizieren, der Webserver kann bei Bedarf auf ein anderes Gerät umgezogen werden und nach einer kurzen Konfiguration wieder entsprechend genutzt werden.

Da der OCPP-Server nicht direkt die ankommenden Nachrichten des RabbitMQ verarbeiten kann, geschieht dies über einen Asynchronen Prozess, welcher prüft, ob eine neue Nachricht vorhanden ist und ob dies sich in eine Befehlsqueue einreicht. Aus dieser Befehlsqueue nimmt der OCPP-Server die Befehle, welche übertragen wurden und führt diese aus. In Abbildung 28 wird hierzu sichtbar, dass der Prozess einmal gestartet wird und anschließend den Server regelmäßig nach neuen Nachrichten abfragt. Sollte eine Nachricht empfangen werden, wird diese verarbeitet und die Aktion extrahiert. Anschließend wird die Aktion in eine Befehlswarteschlange geschrieben. Wie die Umsetzung hierzu stattgefunden hat wird im nächsten Abschnitt erläutert.

```
async def rabbitMQ_Empfaenger():
    logging.debug("Starte Prozess um Nachrichten über RabbitMQ zu empfangen.")
    while True:
        # Verbinde zum RabbitMQ Server asynchron
        connection = await aio_pika.connect_robust("amqp://guest:guest@localhost/")
        # Erstelle einen asynchronen Channel
        async with connection:
            channel = await connection.channel()

            # Stelle sicher, dass die Queue existiert
            queue = await channel.declare_queue('ocpp_commands')

            # Definiere die Callback-Funktion für eingehende Nachrichten
            # FelixSeibt
            async def on_message(message: aio_pika.IncomingMessage):
                async with message.process():
                    command = message.body.decode()
                    logging.debug(f"Received command from RabbitMQ: {command}")
                    json_command = json.loads(command)
                    befehl = json_command["action"]
                    logging.debug(f"erkannter Befehl: {befehl}")

                    # Schreibe den Befehl in die Warteschlange
                    try:
                        await befehlswarteschlange.put(befehl)
                    except Exception as e:
                        logging.warning(f"Es gibt folgenden Fehler: {e}")
                    logging.debug(f"Folgende Befehle sind jetzt in der Warteschlange: {befehlswarteschlange}")

            # Starte den Consumer
            await queue.consume(on_message)
```

Abbildung 28 Empfangen von Nachrichten über RabbitMQ

In dem Abschnitt gerade wurde kurz erläutert, wie die Nachrichten von der RabbitMQ in die Befehlswarteschlange gelangen. Nun wird erläutert, wie die Befehle von der Warteschlange aus ausgeführt werden. Hierzu ist zum leichteren Verständnis der Programmcode in Abbildung 29 dargestellt. Es gibt hierzu eine Asynchrone Funktion, welche in einer unendlichen Schleife versucht Befehle abzurufen. Sollten Befehle vorhanden sein wird geprüft, um welchen Befehl es sich handelt. Anschließend wird aus einer globalen Liste mit verbundenen Wallboxen, zu Demo Zwecken, die erste Wallbox genommen und für diese der entsprechende Befehl ausgeführt

```
async def befehle_schleife():
    print("Starte schleife für befehle")
    while True:
        try:
            aktuellerBefehl = await abrufe_befehl()
            if aktuellerBefehl != "leer" and connected_charge_points:
                print(f"Ausgeführter Befehl: {aktuellerBefehl}")
                if aktuellerBefehl == "status":
                    await connected_charge_points[0].trigger_status_notification()
                elif aktuellerBefehl == "start":
                    await connected_charge_points[0].inside_remote_start_transaction(1)
                elif aktuellerBefehl == "stopp":
                    await connected_charge_points[0].inside_remote_stop_transaction(1234567)
                elif aktuellerBefehl == "reduce":
                    await connected_charge_points[0].send_data_transfer_request()
                else:
                    print("Unknown command or no connected ChargePoints.")
            except:
                print("Fehler bei den Befehlen")
```

Abbildung 29 Funktion Befehls Schleife

In der folgenden Abbildung 30 wird dargestellt, wie ein Befehl entgegengenommen wird, in der Warteschlange als erledigt markiert und anschließend der Befehl zurückgegeben wird. Sollte in der Warteschlange ein Fehler auftreten, da zum Beispiel keine Befehle mehr vorhanden sind, so wird als Befehl leer eingetragen und es werden 4 Sekunden gewartet.

```
async def abrufe_befehl() -> str:
    try:
        befehl = await befehlswarteschlange.get()
        # Markiert den Befehl in der Warteschlange als verarbeitet
        logging.debug(f"befehl_bekommen:{befehl}")
        befehlswarteschlange.task_done()
    except:
        befehl = "leer"
        await asyncio.sleep(4)
    return befehl
```

Abbildung 30 Funktion Befehle Abrufen

4.8. Demo im Labor

In der Demo im Labor, konnte die Verbindung zwischen den beiden Geräten, das erste Mal eingerichtet werden und benötigte Einstellungen vorgenommen werden. Hierzu wurde in der Fritz Box jedem Gerät eine feste IP Adresse zugewiesen und in der Wallbox die IP Adresse des Backends (des Raspberry Pi) eingetragen. Sonstige nähere Einstellungen konnten festgelegt werden, welche die Verbindung betreffen. Hierbei handelt es sich um Einstellungen, wie oft Heartbeat Nachrichten gesendet werden sollen, welche Arten von Status-Notification gesendet werden sollen, die Dauer des Timeouts der Backend Verbindung und der Versand von transaktionsrelevanten Nachrichten. Im Labor gab das System immer eine Ablehnung von den Start / Stop Transactions Befehlen, da kein Auto angeschlossen werden konnte. Zudem wurden keine Status-Notifications versendet, da aus Sicht der Wallbox keine Änderungen stattgefunden haben. Die Wallbox wurde mit einem Starkstromanschluss im Labor angeschlossen und über ein Ethernet mit einem internen Netzwerk verbunden.



Abbildung 31 Wallbox Anschluss im Labor

4.9. Demo in der Praxis

Bei der Demo in der Praxis wurde als Testfahrzeug ein Renault Zoe von 2015 verwendet. Die Wallbox wurde in einem Privathaushalt, an eine passende Steckdose, in einer Garage angeschlossen und über eine Power-Line Verbindung mit dem Netzwerk verbunden, da in der Garage keine Netzwerkanbindung vorhanden ist. Der Raspberry Pi mit dem Display wurde ebenfalls in das Netzwerk eingebunden. Anschließend musste die Wallbox nochmals auf den Raspberry Pi konfiguriert werden, da sich die IP-Adresse geändert hat. Nun konnte die Wallbox sich wieder mit dem Backend des Raspberry Pi verbinden. Zudem konnte in der Praxis ein Elektroauto an die Wallbox angeschlossen werden, wie in Abbildung 32 dargestellt.



Abbildung 32 Ladestecker im Auto

Durch das Anschließen des Autos hat die Wallbox Änderungen registriert. Hierzu hat die Konsole des OCPP-Servers und die Weboberfläche entsprechende Rückmeldungen gegeben. Dies lief wie folgt ab:

- Zu Beginn eine Nachricht, dass eine Wallbox verbunden ist (Boot-Notification).
- Der regelmäßige Heartbeat, welcher zu informativen Zwecken auch auf der Weboberfläche dargestellt wird, da es sich um eine Demo handelt.
- Die Wallbox liefert eine Status-Notification sobald ein Auto mit der Wallbox verbunden wurde. Diese Status-Notification wird zweimal mal empfangen für Connector0 und Connector1. Bei vorhandener Wallbox ist jedoch nur ein Connector verfügbar und zwar Connector1. Diese wechselt nun in den Status prepare,
- Sollte in der Wallbox konfiguriert sein, dass keine Freigabe des CMS benötigt wird als nächstes eine Start-Transaction Message empfangen, welche positiv beantwortet werden muss und anschließend ein Status Update mit Charging. Bei jeder Status-Notification wird zudem mit gesendet, ob es einen Fehler gibt und wenn ja welchen.
- In vorliegendem Fall wurde das automatische starten des Ladevorgangs in der Wallbox deaktiviert, sodass der Raspberry Pi welcher das CMS System bildet die Kontrolle hierüber hat. Um nun einen Ladevorgang zu starten muss eine Remote-Start-Transaction Befehl mit den benötigten Parametern, welcher Ladepunkt und mit welcher Authentifizierung in diesem Fall „freeCharging“ an die Wallbox

gesendet werden. Diese prüft den Authentifizierungsparameter und schickt entweder ein Rejected oder ein Accepted zurück und startet den Ladevorgang.

- Sobald der Ladevorgang läuft wird wieder eine Status-Notification mit dem neuen Status empfangen „charging“.

- Währenddessen wurde das Auto vollgeladen und der Status hat sich auf SuspendedEV geändert.

- Zudem kann der Ladevorgang auch über das CMS beendet werden, mit einem sogenannten Remote-Stop-Transaction. Hierbei wird der Ladevorgang von der Wallbox aus beendet und anschließend gibt es nach der Bestätigung des Befehls, eine neue Status-Notification mit dem neuen Status Finished und der Ladevorgang ist beendet.

In der Praxis hat sich zudem herausgestellt, dass die Wallbox den aktuellen Ladestatus des Autos abfragen kann und dies auch an das CMS übermitteln kann aber es das Auto auch unterstützen muss. Das Testfahrzeug hat dies nicht unterstützt. Über die zyklischen Meter-Values, welche die Wallbox sendet kann jedoch auch ausgelesen werden, wieviel Leistung bereits an das Fahrzeug übertragen worden ist und auf welcher Phase der Wert erfasst wurde. Aufgrund des Fahrzeuges wurde nur einphasiges AC laden unterstützt. Diese Werte werden entsprechend auf der Weboberfläche als Graph dargestellt, sodass ein Verlauf sichtbar wird. Bei einem neueren Hybrid Auto wurde über die Meter-Values ebenfalls der aktuelle Ladestand des Autos übermittelt und angezeigt.

5. Ergebnis

In diesem Kapitel wird darauf eingegangen, welche Informationen vor der Arbeit bereits bekannt waren und welche neuen Erkenntnisse hieraus gewonnen werden konnten. Hierbei wird auf die Ladeboxen eingegangen, auf die Schnittstellen der Zentralen Management Systemen, auf die Möglichkeit Daten auf der Benutzeroberfläche darzustellen und die neuen Erkenntnisse welche bei der Demo im Labor entstanden sind.

5.1. Ladeboxen

Bei den Ladeboxen war vor der Arbeit nicht allzu viel bekannt, es war durch das allgemeine Wissen bereits klar, dass diese verschiedenen Leistungen haben können bis maximal 22 kW, genehmigungspflichtig sind und verschiedene Steuerungsmöglichkeiten, je nach Hersteller besitzen. Zudem das es verschiedene Stecker gibt.

Während der Arbeit hat sich herausgestellt, dass es sehr viele unterschiedliche Arten von Wallboxen gibt und jeder Hersteller viele verschiedene Varianten anbietet, ohne das auf den ersten Blick ein genauer Unterschied hierbei erkennbar ist.

Hierbei hat sich herausgestellt es gibt Wallboxen ab einer Leistung von 3,7 kW, dies entspricht 16A an einer Phase und es gibt Wallboxen mit einer Leistung bis zu 22 kW. Lademöglichkeiten mit einer höheren Ladeleistung sind im privaten Umfeld nicht üblich. Zudem wurde klar, dass es für die Ladeleistung einen Unterschied macht, mit wie vielen Phasen ein Auto geladen werden kann und das eine 3 phasige Wallbox mit 11 kW Ladeleistung ein Auto, welches nur einphasiges Laden unterstützt auch nur mit 3,7 kW laden kann. Zudem ist das Ladeverhalten zunehmend vom Auto abhängig und nicht nur von der bereitgestellten Leistung der Wallbox.

Ein weiterer neuer aber logischer Punkt ist es, dass je nach Wallbox Modell eine Wallbox entweder eine Typ 2 Steckdose anbietet oder direkt ein fest angeschlagenes Kabel mit einem Typ 2 Stecker vorhanden ist. Hierbei ist es überwiegend so, dass ein Kabel angeschlagen ist.

Des Weiteren ist bekannt, dass Wallboxen gesteuert und in ein modernes Smart Home System eingebunden werden können. Neu hierbei ist jedoch, dass es viele Möglichkeiten gibt, wie dies geschehen kann und manche Modelle keinerlei Steuerung anbieten. Hierbei gehen die Möglichkeiten von einer App-Steuerung, über die Einbindung in ein Smart-Home System, ein Lastmanagement, verschiedene Arten von Zugangskontrollen und die Möglichkeit die Wallbox zu überwachen.

Ein weiterer Punkt sind die verschiedenen Schutzfunktionen, welche eine Wallbox bietet oder welche zusätzlich erworben und installiert werden müssen. Hierzu zählt die DC-Fehlerstromerkennung, der Überlastschutz und der Witterungsschutz. Der Witterungsschutz ist je nach Installationsort vernachlässigbar.

5.2. Schnittstellen

Zusätzlich wurden in dieser Arbeit die Schnittstellen und unterstützte Protokolle der Wallboxen untersucht. Hierzu war vorher nur bekannt, dass diese Schnittstellen besitzen, jedoch nicht welche und mithilfe welcher Protokolle diese nutzbar sind.

Während dieser Arbeit ist hat sich herausgestellt, dass nicht jede Wallbox über eine Schnittstelle zur Steuerung verfügt. Je nach Hersteller werden verschiedene Möglichkeiten zur Verbindung mit der Wallbox ermöglicht. Das meist unterstützte Protokoll war hierbei OCPP 1.6 welches über eine Netzwerkschnittstelle genutzt werden kann. Hierbei ist nicht relevant, ob die Verbindung über einen Ethernet Schnittstelle oder eine WLAN Schnittstelle zustande kommt. Zudem etabliert sich OCPP immer mehr und wird von immer mehr Wallboxen unterstützt. Hierzu wird der Kommunikationsstandard immer weiterentwickelt und bietet mehr Sicherheit bei mehr Funktionen. Die möglichen Funktionen wurden im Rahmen dieser Arbeit in Abschnitt 3.2.2 näher erläutert.

5.3. Daten lesen und schreiben auf einer Benutzeroberfläche

Ein weiterer Teil der Arbeit war es, die Inhalte des OCPP-Servers welcher das Backend bildet auf der Weboberfläche darzustellen und Befehle von der Weboberfläche an den OCPP-Server zu übermitteln, dass dieser die Befehle an die Wallbox weiterleiten konnte.

Hierzu bekannt war, dass es mehrere Frameworks gibt um eine Weboberfläche zu erstellen und das eine Möglichkeit existiert um mit der Wallbox zu kommunizieren. Wie diese Möglichkeit genau aussah war nicht bekannt.

Neuer aus dieser Arbeit ist, dass es eine Bibliothek OCPP gibt, welche es erleichtert eine Verbindung über OCPP aufzubauen und zu halten. Zudem erleichtert diese Bibliothek Befehle an die Wallbox zu senden, zu empfangen und diese zu verarbeiten. Des Weiteren unterstützt die Bibliothek auch neuere Versionen des OCPP-Protokolls, sodass in Zukunft einfach eine neuere Version genutzt werden kann. Zudem wurde während der Programmierung klar, dass die Kommunikation zwischen dem Asynchronen OCPP-Server Prozess und dem der Weboberfläche nicht ganz trivial ist und über eine Message-Queue vereinfacht werden kann. In Tabelle 11 ist aufgelistet, welche Daten ausgelesen werden können und über welchen Name es empfangen wird.

Tabelle 11 Mögliche Daten zum Empfangen

Empfangsparameter	Daten
MeterValue	Ladestand des Autos
MeterValue	Bereits geladene Leistung
StatusNotification	Aktueller Status, Connector ID

5.4. Wallbox als Demo im Labor und der Praxis installieren

In Bezug auf die Installation der Wallbox im Labor und in der Praxis gibt es mehrere Aspekte zu berücksichtigen. Im Labor wurde insbesondere festgestellt, dass ohne Elektroauto nur die Verbindungen getestet werden können. Es war bereits bekannt, dass je nach Wallbox eine Leistungssteuerung möglich ist, jedoch nicht, wie diese funktioniert. Als Ergebnis können wir festhalten, wie die Steuerung mithilfe des OCPP-Protokolls 1.6J funktioniert und welche Möglichkeiten dieses Protokoll bietet: Starten, Stoppen und Reduzieren der maximalen Ladeleistung durch ein geändertes Charging Profil. In Tabelle 12 ist dargestellt, mit welchem Commando welche Befehle ausgeführt oder welche Nachrichten abgefragt werden können. Außerdem können die aktuellen Werte der Wallbox ausgelesen und der aktuelle Ladestand des Autos abgefragt werden. Bei vorliegender Wallbox gibt es leider nicht alle Möglichkeiten, da nur das Auslesen der Werte und das Starten und Stoppen des Ladevorgangs möglich ist.

Tabelle 12 mögliche Befehle und Reaktionsbeschreibung

Befehl	Reaktion
Trigger Status Notification	Gibt den aktuellen Status zurück
RemoteStartTransaction	Startet den Ladevorgang
RemoteStoppTransaction	Stoppt den Ladevorgang

6. Fazit

Das Ziel dieser Bachelorarbeit ist es, eine Ansteuerungseinheit zu entwerfen, die es ermöglicht, die Wallbox aus dem internen Netzwerk über ein Webinterface oder direkt an der Ansteuerungseinheit über ein Display zu steuern. Eine Anforderung dabei ist, dass möglichst viele Wallboxen von der Ansteuerungseinheit unterstützt werden. Hierzu wurden einerseits passende Komponenten für die Ansteuerungseinheit gewählt, ein Raspberry Pi und ein 7“ Display hierzu. Des Weiteren musste ein Schnittstellenprotokoll gewählt werden, welches am weitesten verbreitet ist und auch in Zukunft noch verwendet wird. Hierbei ist die Wahl auf OCPP 1.6 gefallen, welches in 80% der Schnittstellenfähigen Wallboxen genutzt werden kann. Neuere Wallboxen unterstützen bereits OCPP 2.0.1, welches mehr Sicherheit bietet. Ist in der Nutzung jedoch sehr Ähnlich zu 1.6. Nach der Wahl der Schnittstelle, musste noch eine Wallbox gewählt werden. Hierbei ist die Wahl auf die Wallbox WB24EC gefallen, welche den Vorteil bietet, dass diese sehr modular aufgebaut ist und kaputte Teile im Zweifel günstig ersetzt werden können oder zusätzliche Funktionalitäten hinzu gebaut werden können. Zur Steuerung wurde nun eine Anwendung entworfen, welche eine Weboberfläche zur Verfügung stellt und einen sogenannten OCPP-Server zum Verbinden mit der Wallbox. Sobald eine Wallbox sich mit dem OCPP-Server verbindet besteht nun die Möglichkeit das Starten und Stoppen zu steuern und zusätzlich alle möglichen Werte zu visualisieren.

Im gesamten kann gesagt werden, dass die Anforderung soweit umgesetzt wurde. Nach anfänglichen Schwierigkeiten, die Funktionen ohne ein Auto zu testen. Mithilfe eines geliehenen Fahrzeuges die Umsetzung in der Praxis getestet werden konnte und auch verifiziert werden konnte, dass das Fahrzeug mit dem Laden beginnt oder stoppt.

7. Literaturverzeichnis

1. Bender. [Online] Januar 2020. [Zitat vom: 16. Februar 2024.] https://www.bender.de/fileadmin/content/Products/b/d/Emobility_PROSP_de.pdf.
 2. Bundesnetzagentur. [Online] 1. August 2023. https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/Unternehmen_Institutionen/E-Mobilitaet/Ladesaeulenkarte/start.html.
 3. Fryzel, Markus. Energielösungen. [Online] 8. Dezember 2021. <https://www.energieloesung.de/magazin/statistiken-und-zahlen-zur-elektromobilitat-in-deutschland/>.
 4. Waffenschmidt, Eberhard. 100pro-erneuerbare. [Online] 26. Okt 2023. [Zitat vom: 26. Februar 2024.] http://www.100pro-erneuerbare.com/projekte/2020-04_Progressus/progressus.htm.
 5. Wunsch, Markus. Vorbereitung des Stromnetzes für die Mobilitätswende. *MTZ - Motortechnische Zeitschrift*. 13. November 2020.
 6. Veronika, Barta, et al. *Algorithmus zur autarken netzdienlichen Steuerung*. TU Graz : s.n.
 7. go-e. Go-E. [Online] [Zitat vom: 21. 12 2023.] Kapitel 12 App. https://go-e.com/fileadmin/user_upload/go-e-charger-homefix-installations-und-bediungsanleitung.pdf.
 8. ABB Guides e Mobility. [Online] [Zitat vom: 21. 12 2023.] https://guides.e-mobility.abb.com/chargersync_portal_DE/#!/lessons/yi0UMa6xc7yFRaz7yYIObaIAuE4cQ7IR.
 9. Ferstl, Stefan. github. [Online] [Zitat vom: 22. 12 2023.] <https://steff393.github.io/wbec-site/>.
 10. selfhtml. [Online] [Zitat vom: 16. Februar 2024.] https://wiki.selfhtml.org/wiki/Raspberry/Webserver_mit_Apache#SQL-Datenbank.
 11. django projekt. [Online] [Zitat vom: 2. Januar 2024.] <https://www.djangoproject.com/start/overview/>.
 12. Götze, Uwe und Rehme, Marco. *Elektromobilität – Herausforderungen und Lösungsansätze*. Chemnitz : TU Chemnitz, 2011. Seite 32.
 13. angegeben, kein Autor. wallbox. [Online] [Zitat vom: 16. Februar 2024.] https://wallbox.com/de_de/faqs-ladestrom-unterschied-ac-dc.
 14. Zappel, Moritz. Ladestation für Elektroauto und WEG. *NR Nachhaltigkeitsrecht*. März 2021.
 15. Salzburg Ag. [Online] [Zitat vom: 05. 12 2023.] <https://www.salzburg-ag.at/emobilitaet/elektromobilitaet/wallboxen/funktionsweise.html>.
 16. Doppelbauer, Martin. *Ladesysteme*. Wiesbaden : Springer Fachmedien Wiesbaden, 2020. Kapitel 12.4.2 und 12.4.3.
 17. Delphin Technology AG. Delphin Galvanische Trennung. [Online] [Zitat vom: 10. 12 2023.] <https://www.delphin.de/loesungen/universelle-messdatenerfassung/galvanische-trennung/>.
 18. RCPlan.de. [Online] [Zitat vom: 10. 12 2023.] <https://replan.de/hausanschluss-dimensionieren/>.
 19. EV Database Ioniq 4. [Online] [Zitat vom: 10. 12 2023.] <https://ev-database.org/de/pkw/1165/Hyundai-IONIQ-Elektro>.
 20. evbox tesla 3. [Online] [Zitat vom: 10. 12 2023.] <https://evbox.com/de-de/elektroautos/tesla/tesla-model-3>.
 21. Lan, Liu. *Einfluss der privaten Elektrofahrzeuge auf Mittel- und Niederspannungsnetze*. TU Darmstadt : s.n. Kapitel 2.3.2.3.
 22. Adac.de. [Online] 13. September 2022. [Zitat vom: 16. Februar 2024.] <https://www.adac.de/rund-ums-fahrzeug/elektromobilitaet/laden/ladeverluste-elektroauto-studie/>.
-

-
23. angegeben, kein Autor. e-mobileo. [Online] [Zitat vom: 26. Februar 2024.] <https://www.e-mobileo.de/10-tipps-wie-der-akku-laenger-haelt/>.
 24. P3-Group. [Online] Juli 2022. [Zitat vom: 16. Februar 2024.] https://www.p3-group.com/p3-charging-index-vergleich-der-schnelllade-faehigkeit-verschiedener-elektrofahrzeuge-aus-nutzerperspektive_07-22/.
 25. EVbox. [Online] [Zitat vom: 16. Februar 2024.] <https://evbox.com/de-de/lade-guide>.
 26. techopedia. [Online] 27. November 2023. [Zitat vom: 16. Februar 2024.] <https://www.techopedia.com/de/definition/protokoll>.
 27. Stollenwerk, Dominik, et al. *Smarte Lades{ "a }ulen | Netz- und Marktdienliches { "o }ffentliches Laden*. Wiesbaden : Springer Fachmedien Wiesbaden, 2023.
 28. Santoire, Julien. *OCPP: Von der Ladeabrechnung bis hin zum Smart Charging*. Blog : Vispiron Systems, 2021.
 29. Kring, Friedhelm. elektrofachkraft. [Online] 24. Juni 2022. [Zitat vom: 16. Februar 2024.] <https://www.elektrofachkraft.de/sicheres-arbeiten/typenvielfalt-von-fehlerstrom-schutzeinrichtungen-rcd>.
 30. sh-netz. sh-netz. [Online] 2022. [Zitat vom: 19. Februar 2024.] https://www.sh-netz.com/content/dam/revu-global/sh-netz/Documents/Energie_anschliessen/Stromnetz/Neuanschluss_ans_Stromnetz/Niederspannung/sh-netz_partner_ladeeinrichtung_anschliessen_229i_0422_1022_web.pdf.
 31. goingelectric. [Online] 2023. [Zitat vom: 16. Februar 2024.] <https://www.goingelectric.de/wiki/Ladung-und-Ladestecker/>.
 32. wikipedia. [Online] [Zitat vom: 16. Februar 2024.] https://de.wikipedia.org/wiki/IEC_62196_Typ_2.
 33. jh-profishop. [Online] [Zitat vom: 16. Februar 2024.] <https://www.jh-profishop.de/profi-guide/ccs-stecker-belegung/>.
 34. Going Electric. [Online] [Zitat vom: 16. Februar 2024.] <https://www.goingelectric.de/wiki/Typ2-Signalisierung-und-Steckercodierung/>.
 35. vector. [Online] [Zitat vom: 16. Februar 2024.] <https://www.vector.com/de/de/know-how/smart-charging/ladeschnittstellen/#>.
 36. Botland.de UNIHAKER. [Online] [Zitat vom: 02. 12 2023.] <https://botland.de/sbc-minicomputer/23454-unihiker-einplatinencomputer-iot-python-mit-28-touchscreen-display-dfrobot-dfr0706-en-6959420923618.html>.
 37. Asus Tinker Board. [Online] [Zitat vom: 10. 12 2023.] https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiGo_ah5puDAxUZgv0HHe7CAAkQFnoECA8QAQ&url=https%3A%2F%2Fwww.asus.com%2Fde%2Fnetworking-iot-servers%2Faiot-industrial-solutions%2Fall-series%2Ftinker-board%2F&usg=AOvVa.
 38. Marotronics Banana Pi. [Online] [Zitat vom: 10. 12 2023.] <https://www.marotronics.de/Banana-Pi-BPI-M4-Vierkern-Einplatinencomputer>.
 39. Amazon RockPI X. [Online] [Zitat vom: 10. 12 2023.] <https://www.amazon.de/Pi-Einplatinencomputer-Windows-x5-Z8350-CPU-Win10-Produktschl%C3%BCssel/dp/B08MKL2Q6K>.
 40. Informatik verstehen. [Online] [Zitat vom: 16. Februar 2024.] <https://www.informatik-verstehen.de/tutorials/csharp-tutorial/c-sharp-einfuehrung/>.
 41. Software developer india. [Online] [Zitat vom: 16. Februar 2024.] <https://www.software-developer-india.com/de/vor-und-nachteile-der-programmiersprache-c/>.
 42. *Vorzüige und Nachteile von Java*. Nazarevich, Dmitry. s.l. : Innowise, 2021.
 43. lernprogrammieren. [Online] [Zitat vom: 16. Februar 2024.] <https://lernprogrammieren.de/python-vs-javascript/>.
-

-
44. Ionos. [Online] [Zitat vom: 16. Februar 2024.] <https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-html/>.
 45. checkdomain. [Online] [Zitat vom: 16. Februar 2024.] <https://www.checkdomain.de/hosting/lexikon/css/>.
 46. python-kurs. [Online] [Zitat vom: 16. Februar 2024.] <https://www.python-kurs.eu/threads.php>.
 47. go-e. [Online] 2023. [Zitat vom: 16. Februar 2024.] <https://go-e.com/de-de/produkte/go-e-charger-homefix>.
 48. go-e. [Online] 05. April 2023. [Zitat vom: 16. Februar 2024.] <https://go-e.com/de-de/magazin/elektroautos-laden-mit-ocpp-wallbox>.
 49. plotly. [Online] [Zitat vom: 16. Februar 2024.] <https://dash.plotly.com/dash-core-components/interval>.
 51. Schulze, Olaf. *Elektrisch Laden: private nicht öffentliche und öffentliche Ladepunkte. In: Elektromobilität – ein Ratgeber für Entscheider, Errichter, Betreiber und Nutzer.* Wiesbaden : Springer, 2022.
 52. wallbox24.de. [Online] [Zitat vom: 05. 12 2023.] Schaltung AC _Laden Bild. <https://www.wallbox24.de/wb24-ev-lade-controller-epc-10a-32a-einstellbar-fuer-festanschlusskabel-wallbox24-wb24-ev-lade-controller-epc-10a-32a-einstellbar-fuer-festanschlusskabel-wallbox24/a-1047656>.
 53. Berrybase Pi 4B. [Online] [Zitat vom: 10. 12 2023.] <https://www.berrybase.de/raspberry-pi-4-computer-modell-b-4gb-ram>.
 54. Berrybase Pi 5. [Online] [Zitat vom: 10. 12 2023.] <https://www.berrybase.de/raspberry-pi-5-8gb-ram?c=319>.
 55. Berrybase. [Online] [Zitat vom: 10. 12 2023.] <https://www.berrybase.de/raspberry-pi-400-de?c=2411>.
 56. berrybase Pi Zero. [Online] [Zitat vom: 10. 12 2023.] <https://www.berrybase.de/raspberry-pi-zero-w>.
-

Verzeichnis der Abbildungen

Abbildung 1 Wallbox WB24	7
Abbildung 2 Steuerung mehrerer Verbraucher über Relais	9
Abbildung 3 Grafana Beispieldashboard	11
Abbildung 4 Node-Red Beispieldashboard aus dem Flow	12
Abbildung 5 Node-Red Beispielflow	12
Abbildung 6 Aktion Button Klick	13
Abbildung 7 Code zum Anzeigen der HTML Seite	13
Abbildung 8 Ladeschaltung AC vereinfachte Darstellung [7]	15
Abbildung 9 Ladeleistung zu State of Charge [30]	16
Abbildung 10 OCPP Beispiel Request und Response	19
Abbildung 11 Typ 2 Stecker und Belegung 1	21
Abbildung 12 Typ 2 Stecker mit Belegungsvariante 2 und 3 [31]	21
Abbildung 13 Combo Stecker Belegung	22
Abbildung 14 Isolation der Lademodi [18]	23
Abbildung 15 Multithreading Variablen	26
Abbildung 16 Raspberry PI 7" Touchdisplay	27
Abbildung 17 Wallbox zusammgebaut	32
Abbildung 18 Steckerbelegung CEE-Stecker 400V 16A	33
Abbildung 19 Wallbox einrichten	34
Abbildung 20 Wallbox Autorisierungs-Einstellungen	35
Abbildung 21 Dashboard der Wallbox	35
Abbildung 23 Dashboard Übersicht	37
Abbildung 24 Intervalle in Dash	37
Abbildung 25 Codeausschnitt OCPP-Server initialisieren	40
Abbildung 26 Programmcode Verbindungsanfragen handler	40
Abbildung 27 Boot-Notification Empfangen	41
Abbildung 28 Anforderung einer StatusNotification	41
Abbildung 29 Empfangen von Nachrichten über RabbitMQ	42
Abbildung 30 Funktion Befehls Schleife	43
Abbildung 31 Funktion Befehle Abrufen	43
Abbildung 32 Wallbox Anschluss im Labor	44
Abbildung 33 Ladestecker im Auto	45

Verzeichnis der Tabellen

Tabelle 1 Leitungsdimensionierung Abhängig von der Leistung	14
Tabelle 2 Leistungsstufen Ladepunkte [24]	17
Tabelle 3 Protokolle und ihre Funktionsweise	17
Tabelle 4 Übersicht der Funktionalitäten nach OCPP-Version [27]	18
Tabelle 5 Status zu Widerstand einer Wallbox [33]	22
Tabelle 6 maximaler Ladestrom zu Widerstand eines Ladepunktes [33]	22
Tabelle 7 alternative Einplatinencomputer	24
Tabelle 8 Wallboxvergleich	29
Tabelle 9 Von dem Ladepunkt mögliche Befehle mit Beschreibung	38
Tabelle 10 Befehle des CMS an die Wallbox mit Beschreibung	39
Tabelle 11 Mögliche Daten zum Empfangen	48
Tabelle 12 mögliche Befehle und Reaktionsbeschreibung	48

Verwendete Abkürzungen

A	Strom, allgemein
AC	Wechselspannung
API	Application Programming Interfaces
BA	Bachelorarbeit
CMS	Content Management System
CP	Pilotkontakt (Kontakt am Typ 2 Stecker)
DC	Gleichspannung
kW	kilo Watt
LC	Lademanagementsystem
LP	Ladepunkt
OCPP	Open Charge Point Protocol
Pi	Raspberry Pi
PV-Anlage	Photovoltaik Anlage
PP	Proximity-Pilot (Kontakt am Typ 2 Stecker)
SoC	State of Charge
SSH	Secure Shell
U	Spannung, allgemein

Anhang
Teileliste

2	WB24 WB24PLUGHOLDERTYP2	WB24 Winkel Steckerwandhalterung Typ 2 Ladekabel Wallbox Ladestation Wallbox24 Plug Holder	0,00
3	1,00 WB24SCHUETZ63A	Wallbox24 Installationsschütz 63A 4S AC 230/400V mit 4 Schließer-Kontakten, für EV-Ladegeräte	0,00
4	1,00 WB24SIGN8	Wallbox24 Aufkleber Parkplatz E-Fahrzeuge Nr. 8 selbstklebende Folie Zubehör Elektromobilität	0,00
5	WB24 WB24KABELHALTERTYP2	WB24 Wandhalterung für Ladekabel Stecker Typ 2 geeignet Ladestation Wallbox24	0,00
6	1,00 BSEVP001F5M	EV Ladekabel 16A 3ph. Typ 2, Stecker/Auto 5m und freiem Kabelende 400V 11kW 16A Typ 2 für Eigenbau Wallbox	0,00
7	1,00 WB24KABEL5X25	Wallbox24 Gummikabel schwarz 5x2,5mm ² je Meter	0,00
8	WB24LEERGEREHKUNSTSTOFF2R	Wallbox24 Leergehäuse2-reihig Kunststoff grau Stromverteiler IP66 2x13 Module Verteilerkasten Aufputz	0,00
9	WB24 WB24CTBC17KABEL325	Wallbox24 Bender Anschlusskabel inkl Gehäuse für CTBC17P 6 Pins 325mm	0,00
10	1,00 WB24CTBC17P03	Wallbox24 Bender Messtromwandler CTBC17P 03 für Wallbox Ladestation Home Energy Management	0,00

Pos.	Menge	Artikelnummer	Bezeichnung	NETTO
11	1,00	WB24B94060129	Wallbox24 Bender Stecker Kit für CC613 Laderegler	0,00
12	1,00	WB24HDR1512	Wallbox24 MeanWell HDR-15-12 LED Netzteil 15W 12V DC für Hutschiene	0,00
13	WB24 WB24HAUPTSCHALTER	Wallbox24 ABB Hauptschalter E463/3-KB 3x63A Trennschalter Leistungstrenner Elektromobilität	0,00	
14	1,00	WB24CC613ELPRM	Wallbox24 Bender Laderegler CC613 ELPR M mit OCPP Schnittstelle für Wallbox Ladestation Home Energy Management	0,00