

# Diplomarbeit

Höhere Technische Bundes- Lehr- und Versuchsanstalt Salzburg  
Abteilung für Elektrotechnik

## Entwicklung eines emissionsfreien Sportmotorrades

### Entwicklung der Zentralsteuerung / Projektleitung

Martin Kronberger 5AHET Betreuer: Dipl.-Ing. (FH) Johannes Ferner

### Entwicklung des Antriebssystems

Jakob Lackner 5AHET Betreuer: Prof. Dipl.-Ing. MBA Adolf Reinhart

### Entwicklung des Akkusystems

Simon Kern 5AHET Betreuer: Prof. Dipl.-Ing. Reinhold Benedikter

### Entwicklung der mechanischen Komponenten

Tobias Schmeisser 5AHET Betreuer: Prof. Dipl.-Ing. Peter Lindmoser

Höhere Technische Bundeslehr-  
und Versuchsanstalt Salzburg  
Itzlinger Hauptstraße 30  
A-5022 Salzburg  
[www.htl-salzburg.ac.at](http://www.htl-salzburg.ac.at)





## Eidesstaatliche Erklärung

Wir erklären an Eides statt, dass wir die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht haben. Wir versichern, dass wir dieses Diplomarbeitsthema bisher weder im In- noch im Ausland (einer Beurteilerin oder einem Beurteiler) in irgendeiner Form als Prüfungsarbeit vorgelegt haben.

## Gendererklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Diplomarbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

---

Martin Kronberger

Ort, Datum

---

Jakob Lackner

Ort, Datum

---

Simon Kern

Ort, Datum

---

Tobias Schmeisser

Ort, Datum



# Vorwort

In immer mehr Großstädten werden Fahrzeuge mit Verbrennungsmotoren verboten. Viele Motorräder und Autos können nicht mehr produziert werden, da sie die immer strenger werdenden Abgasnormen nicht mehr einhalten können und das Thema der Klimaerwärmung wird immer präsenter und immer mehr Menschen versuchen ihren „carbon footprint“ zu verkleinern.

Doch leider gibt es für Motorradfahrer zumeist keine wirklichen alternativen, um für ihr Hobby auf eine emissionsfreie Alternative umzusteigen. Denn zumeist ist das Preis-Leistungsverhältnis, oder auch das Produkt selbst, nicht sehr einladend. Daher ist unser Ziel die Entwicklung in diesem Bereich voranzutreiben und dadurch den Markt zu vergrößern, wodurch immer mehr und bessere Produkte angeboten werden können.



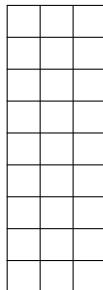
# Danksagung

TEXT DANKSAGUNG



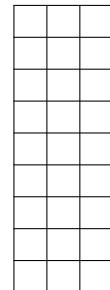
# DIPLOMARBEIT

## DOKUMENTATION



# DIPLOMA THESIS

## DOCUMENTATION





# Erklärung

Die unterfertigten Kandidaten haben gemäß §34 (3) SchUG im Verbindung mit §22 (1) Zi. 3 lit. b der Verordnung über die abschließenden Prüfungen in den berufsbildenden mittleren und höheren Schulen, BGBl. II Nr. 70 vom 24.02.2000 (Prüfungsordnung BMHS), die Ausarbeitung einer Diplomarbeit mit der umseitig angeführten Aufgabenstellung gewählt. Die Kandidaten nehmen zur Kenntnis, dass die Diplomarbeit in eigenständiger Weise und außerhalb des Unterrichtes zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichtes mit einbezogen werden können. Die Abgabe der vollständigen Diplomarbeit hat bis spätestens

03.04.2020

beim zuständigen Betreuer zu erfolgen. Die Kandidaten nehmen weiters zur Kenntnis, dass gemäß §9 (6) der Prüfungsordnung BMHS nur der Schulleiter bis spätestens Ende des vorletzten Semesters den Abbruch einer Diplomarbeit anordnen kann, wenn diese aus nicht beim Prüfungskandidaten / bei den Prüfungskandidaten gelegenen Gründen nicht fertiggestellt werden kann.

Kandidaten / Kandidatinnen	Unterschrift
Martin Kronberger	
Jakob Lackner	
Simon Kern	
Tobias Schmeisser	

---

Prof. Dipl.-Ing. Reinhold Benedikter  
Prüfer

Dipl.-Ing. (FH) Johannes Ferner  
Prüfer

---

Prof. Dipl.-Ing. MBA Adolf Reinhart  
Prüfer

Lindmoser, Prof. Dipl.-Ing. Peter  
Prüfer

---

Prof. Dipl.-Ing. (FH) Roland Holzer  
Abteilungsvorstand

Dipl.-Ing. Dr.techn. Franz Landertshamer  
Direktor

# Inhaltsverzeichnis

<b>I Einführung</b>	<b>2</b>
1 Projektteam . . . . .	2
2 Projektbetreuer . . . . .	3
3 Aufgabeneinteilung . . . . .	3
<b>II Einleitung</b>	<b>5</b>
1 Motivation . . . . .	5
2 Zielsetzung . . . . .	5
3 Topologie des Gesamtsystems . . . . .	5
4 Leitfaden . . . . .	5
<b>III Stand der Technik</b>	<b>6</b>
1 Steuereinheiten . . . . .	6
1.1 Raspberry PI . . . . .	6
2 Bussysteme . . . . .	6
2.1 SPI Bus . . . . .	6
2.2 CAN Bus . . . . .	6
<b>IV Mechanische Umsetzung</b>	<b>7</b>
<b>V Human-Computer Interaction System</b>	<b>8</b>
1 Übersicht . . . . .	8
1.1 Grundfunktionen des Systems . . . . .	8
1.2 Steuereinheit . . . . .	9
1.3 Grundaufbau des Systems . . . . .	10
2 Versorgung . . . . .	11
2.1 Aufbau des Versorgungssystems . . . . .	11
2.2 Spannungswandler . . . . .	11
2.2.1 5V Versorgungssystem . . . . .	11
2.2.2 12V Versorgungssystem . . . . .	11
3 Steuerung der Peripherie . . . . .	12
3.1 Hardware . . . . .	12
3.1.1 Input . . . . .	12
3.1.2 Output . . . . .	13
3.2 Software . . . . .	13
3.2.1 GPIO Zero . . . . .	13
3.2.2 Threading . . . . .	13
4 Benutzeroberfläche . . . . .	14
4.1 Hardware . . . . .	14
4.1.1 Befestigung . . . . .	15
4.2 Software . . . . .	15
4.2.1 Aufbau . . . . .	15
4.3 Komponenten . . . . .	16
4.3.1 Navigations Menu . . . . .	16
4.3.2 Balken Anzeige . . . . .	16
4.3.3 Graph . . . . .	16

4.4	Program Fenster . . . . .	17
4.4.1	Login . . . . .	17
4.4.2	Fahrdaten . . . . .	17
4.4.3	Akku- und Ladedaten . . . . .	18
4.4.4	Fahrdaten Diagnose . . . . .	18
4.4.5	Errors . . . . .	19
4.5	Realisierung der Benutzeroberfläche . . . . .	20
4.5.1	QML . . . . .	20
4.5.2	Qt-Quick . . . . .	20
4.5.3	Slots und Signals . . . . .	20
4.5.4	Bridge . . . . .	21
5	Kommunikation . . . . .	22
5.1	Hardware . . . . .	22
5.1.1	CAN-Modul . . . . .	22
5.1.2	Netzwerkstruktur . . . . .	22
5.2	Listener . . . . .	23
5.2.1	Receive Data . . . . .	23
6	Fahrdatenspeicher . . . . .	24
6.1	Datenbankstruktur . . . . .	24
6.1.1	Login System . . . . .	24
6.1.2	Motor Daten . . . . .	24
6.1.3	Akku Daten . . . . .	24
6.2	Handler . . . . .	24
6.2.1	SELECT Befehl . . . . .	24
6.2.2	INSERT Befehl . . . . .	24
<b>VI Antriebsstrang</b>		<b>25</b>
<b>VIAkku und Ladekonzept</b>		<b>26</b>
<b>VIIHndergebnis</b>		<b>27</b>
<b>A Arbeitsnachweis</b>		<b>28</b>
1 Zeitplan . . . . .		28
2 Kosten . . . . .		28
<b>B Programmier-Code</b>		<b>29</b>
<b>C CAD-Zeichnungen</b>		<b>30</b>
<b>D Schaltpläne</b>		<b>31</b>
<b>Literaturverzeichnis</b>		<b>31</b>
<b>Abbildungsverzeichnis</b>		<b>31</b>
<b>Tabellenverzeichnis</b>		<b>32</b>
<b>Codeverzeichnis</b>		<b>33</b>

# Kapitel I

## Einführung

### 1 Projektteam



Martin Kronberger



Jakob Lackner



Simon Kern



Schmeisser Tobias

## 2 Projektbetreuer

**Prof. Dipl.-Ing. Reinhold Benedikter**

unterstützte Jakob Lackner bei der Entwicklung des Akku- und Ladesystems

**Dipl.-Ing. (FH) Johannes Ferner**

unterstützte Martin Kronberger bei der Entwicklung des Human-Computer Interaction Systems

**Prof. Dipl.-Ing. MBA Adolf Reinhart**

unterstützte Jakob Lackner bei der Entwicklung des Antriebssystems

**Lindmoser, Prof. Dipl.-Ing. Peter**

unterstützte Tobias Schmeisser bei der Entwicklung der mechanischen Komponenten

## 3 Aufgabeneinteilung

**Martin Kronberger**

- Projektleitung
- Projektfindung und Projektplanung
- Projektaufteilung
- Erstellen der Einreichdokumente
- Entwickeln der Hardware des Human-Computer Interaction Systems
- Entwickeln der Software des Human-Computer Interaction Systems
- Planung und Umsetzung der elektrischen Installation
- Verfassen der Dokumentation

**Jakob Lackner**

- Projektleitung
- Projektfindung und Projektplanung
- Projektaufteilung
- Entwicklung des Antriebssystems
- Entwicklung der Software des Motorsteuergerätes
- Erstellen der Einreichdokumente
- Verfassen der Dokumentation

**Simon Kern**

- Projektleitung
- Projektfindung und Projektplanung
- Projektaufteilung
- Entwicklung des Akkusystems
- Erstellen der Einreichdokumente
- Verfassen der Dokumentation

**Tobias Schmeisser**

- Projektleitung
- Projektfindung und Projektplanung
- Projektaufteilung
- Entwicklung der mechanischen Komponenten
- Entwicklung der Getriebemechanik
- Erstellen der Einreichdokumente
- Verfassen der Dokumentation

## Kapitel II

# Einleitung

- 1 Motivation**
- 2 Zielsetzung**
- 3 Topologie des Gesamtsystems**
- 4 Leitfaden**

# Kapitel III

# Stand der Technik

## 1 Steuereinheiten

### 1.1 Raspberry PI

## 2 Bussysteme

### 2.1 SPI Bus

### 2.2 CAN Bus

## Kapitel IV

# Mechanische Umsetzung

# Kapitel V

# Human-Computer Interaction System

## 1 Übersicht

Das Human-Computer Interaction System ist, wie der Name schon verrät, die Komponente, welche als Schnittstelle zwischen dem Nutzer und dem gesamten elektrischen System dient. Durch es sollte die fehlerfreie Nutzung der Funktionen des Motorrades gewährleistet sein, ebenso sollte es wichtige Fahrdaten und andere Informationen speichern und dem User anzeigen können. Wichtig ist das System, trotz der großen Komplexität, so intuitiv und nutzerfreundlich wie möglich zu gestalten.

### 1.1 Grundfunktionen des Systems

Die geplanten Funktionen des HCIS lassen sich grob in vier Grundfunktionen einteilen.

- **Steuerung der Peripherie**

Die Schalter und Buttons am Lenker, welche zuvor über den Kabelbaum die Leuchten, Blinker oder der Hupe gesteuert haben. Werden nun über die General-purpose input/output (GPIO) anschlüsse des Raspberry Pi Micro Computers gesteuert.

- **Graphische Benutzeroberfläche**

Dient der Anzeige wichtiger Fahr- und Ladedaten welche entweder in echtzeit oder über die Datenbankschnittstelle abgerufen und graphisch angezeigt werden können.

- **Kommunikation mit den Steuereinheiten des Motorrades**

Über CAN-Bus werden Daten von dem Batterie Management Systems (BMS) und der Curtis Motorsteuerung empfangen und an die Benutzeroberfläche zur echtzeit verwertung und an die Datenbankschnittstelle zur Langzeitsicherung der Fahrdaten weiter gegeben.

- **Speichern der relevanten Fahrdaten über die Datenbankschnittstelle**

Die über den CAN-Bus empfangenen Daten werden sofort an die Datenbankschnittstelle (Hander) weitergegeben um für Datenauswertung und Testberichte die Daten zu speichern. Ebenso bezieht das Diagnosesystem der Benutzeroberfläche die Daten über diese Schnittstelle.

## 1.2 Steuereinheit

Als Basis zur Auswahl der Steuereinheit wurden die zuvor erläuterten Grundfunktionen herangeszogen genommen. Die Ausgewählte Steuereinheit sollte diese erfüllen können und ebenso Potential zur Erweiterung der Funktionen bieten. Genauso wichtig war das eine große Flexibilität und Individualität erreicht werden kann, um nicht in der Umsetzung unserer Ideen eingeschränkt zu sein. Zur Auswahl standen verschiedene Speicherprogrammierbare Steuerungen und Microcomputer, doch letzten endes überzeugte der Microcomputer Raspberry Pi.

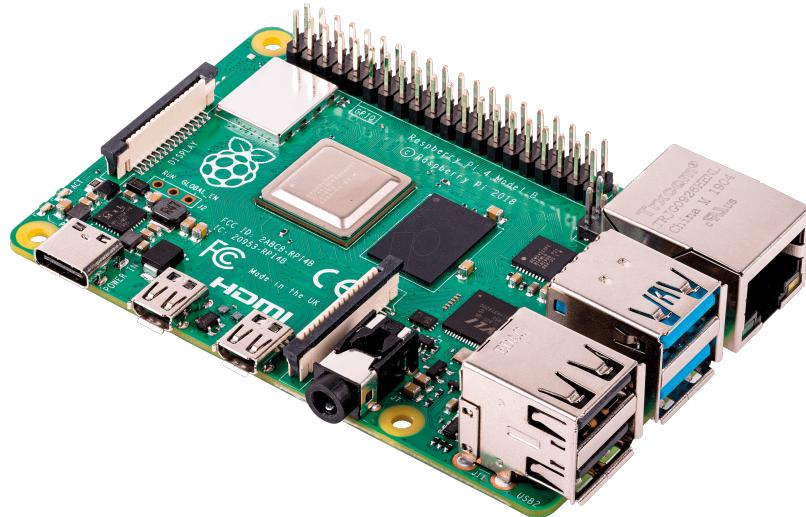


Abbildung V.1: Raspberry Pi - Steuereinheit des HCIS

Durch

### 1.3 Grundaufbau des Systems

In der Abbildung wird der Grundaufbau des Systems und die Datenverbindungen der folgenden Komponenten veranschaulicht.

- Raspberry Pi - Die Steuereinheit des Systems.  
Kommuniziert über CAN-Bus mit den anderen Steuerkomponenten des Motorrades.
- User Input - Die vorhandenen Buttons am Lenker des Motorrads werden über pull down Widerstände mit den Inputs des Raspberry Pi verbunden.
- Peripherie - Die Grundkomponenten des Motorrades wie Scheinwerfer oder Hupe.
- Dashboard - Der Bildschirm zur Anzeige der Verarbeiteten Informationen.

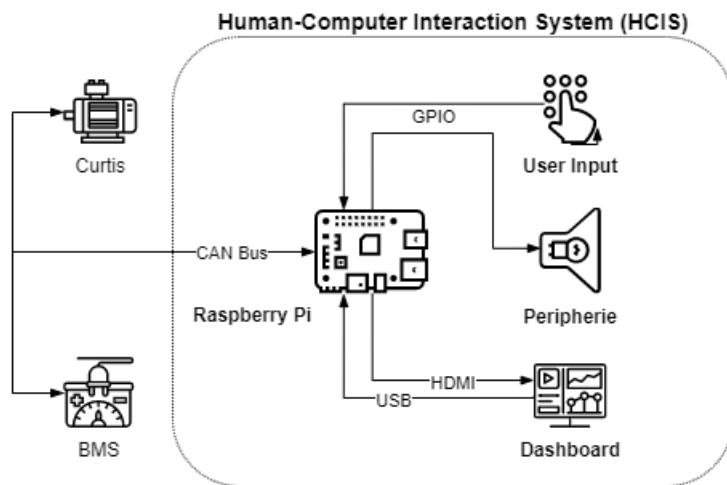


Abbildung V.2: Grundaufbau des Human-Computer Interaction Systems

Nicht in der Abbildung dargestellt ist die Versorgung der einzelnen Komponenten, welche in dem folgenden Abschnitt noch genauer erläutert wird.

## 2 Versorgung

### 2.1 Aufbau des Versorgungssystems

### 2.2 Spannungswandler

#### 2.2.1 5V Versorgungssystem

#### 2.2.2 12V Versorgungsystem

### 3 Steuerung der Peripherie

Die Grundfunktionen wie Beleuchtung, Hupe und Blinker werden hier als Peripherie bezeichnet. Diese sollten so einfach wie möglich und vom Lenker aus zu bedienen sein. Ebenso müssen sie verlässlich gesteuert werden können. Daher haben wir uns entschieden diese Funktionen ebenso über den Raspberry zu steuern, da dieser bei einem Fehler der Motorsteuerung über den eingebauten Puffer gespeist werden kann und daher diese wichtigen Funktionen bis zu einem sicheren Stillstand weiter betrieben und gesteuert werden können.

Dennoch ist in Planung in Zukunft die Motorsteuerung, welche ebenso in der Lage wäre die Ausgänge abhängig von den Inputs zu schalten, diese Aufgabe übernehmen zu lassen, solange die Ausfallsicherheit ebenso gegeben wäre. Der Vorteil dieser Methode wäre die Schaffung einer Zentralen Steuereinheit, welche alle Steueraufgaben in einem Bauteil vereinen kann.

#### 3.1 Hardware

##### 3.1.1 Input

Man einen GPIO Pin entweder als Eingang oder als Ausgang betreiben. Als Eingang kann er die Zustände "High" und "Low" einnehmen. Zum Beispiel von einem Schalter oder Taster. In der Regel beschaltet man die GPIOs des Raspberry Pi mit Widerständen, um Eingänge auf einen definierten Pegel zu setzen oder um den Strom zu begrenzen. Standardmäßig werden 10k Widerstände benutzt. Ob Pullup oder Pulldown ist grundsätzlich gleichgültig. Wir benutzen für das Einlesen unserer Eingänge 10k Pulldown Widerstände um die Eingänge des Raspberry Pis nicht immer an einer Spannung angeschlossen zu haben.

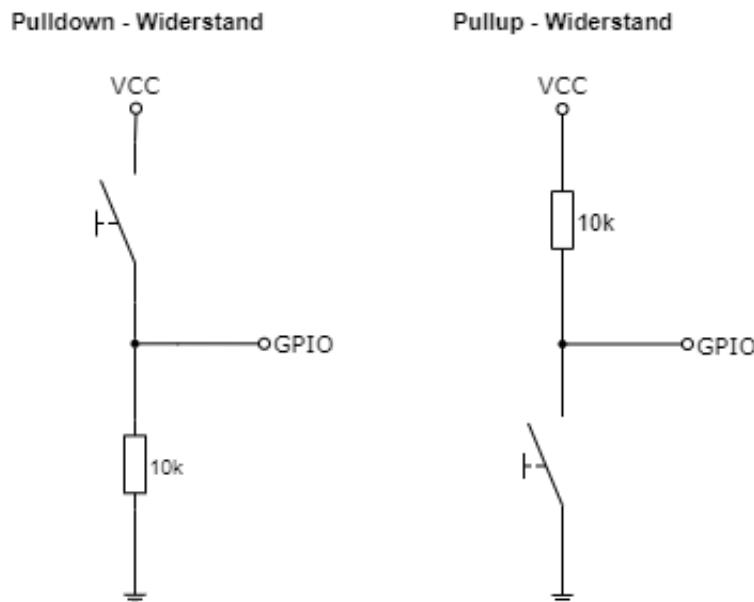


Abbildung V.3: Anschlussplan Inputs

##### Pullup Widerstand

Bei den Nutzen eines Pullup Widerstands wird der GPIO Pin mit einem Widerstand auf die Spannung von VCC gezogen. Der Grundzustand des Eingangs ist dann "High". Mit einem Schalter oder Taster wird der Eingang dann gegen Ground gezogen. Das heißt er hat solange der Schalter geschlossen ist den Zustand logisch "Low".

##### Pulldown Widerstand

Bei den Nutzen eines Pulldown Widerstands wird der GPIO Pin mit einem Widerstand auf die Spannung von Ground gezogen. Der Grundzustand des Eingangs ist dann logisch "Low". Mit einem Schalter oder Taster wird der Eingang dann gegen VCC gezogen. Das heißt er hat solange der Schalter geschlossen ist den Zustand logisch "High".

### 3.1.2 Output

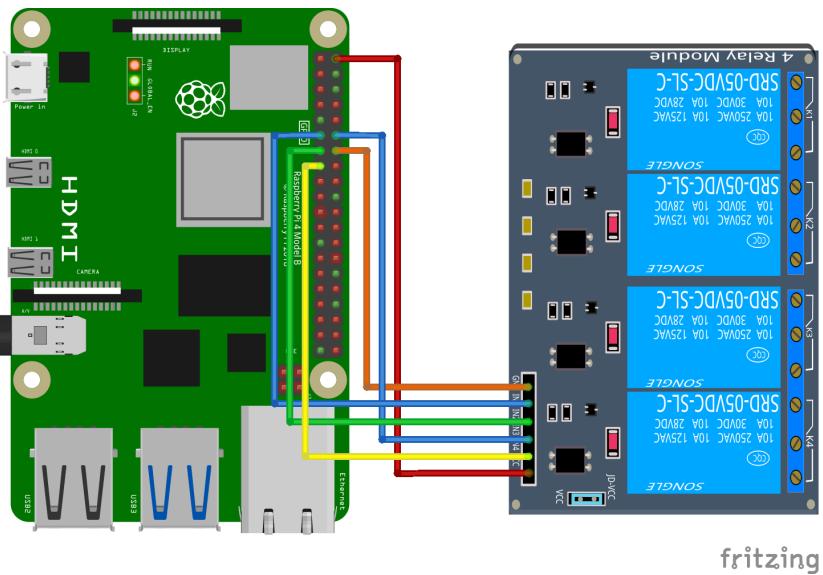


Abbildung V.4: Anschlussplan Relai

## 3.2 Software

### 3.2.1 GPIO Zero

### 3.2.2 Threading

## 4 Benutzeroberfläche

Die Benutzeroberfläche stellt die Verbindung zwischen dem Nutzer und dem Motorrad dar. Sie sollte während der Fahrt die Instrumententafel des Motorrades zu ersetzen und dem Nutzer die wichtigsten Fahrinformationen anzeigen. Sobald man zum Stillstand gekommen ist, wird es möglich Einstellungen zu ändern und die aufgezeichneten Fahrdaten anzeigen zu lassen. Ebenso kann der Akkuladestatus und Informationen über Fehler im System entnommen werden.

### 4.1 Hardware

Zur Anzeige und Bedienung wird ein 11.6 Zoll Kapazitives Touch LCD Display verwendet. Es besitzt eine FullHD Auflösung (1920x1080), was für eine professionelle Darstellung essentiell ist. Ebenso hat es ein schützendes ABS Gehäuse, welches trotz fehlender IPX Zertifizierung das Abdichten ermöglicht. Die Versorgungsspannung beträgt 12V, was ident zu den anderen Komponenten am Motorrad ist und daher die Versorgung sehr vereinfacht, kann also über den gleichen Spannungswandler versorgt werden.



Abbildung V.5: Touch Panel Maße

Die Auflösung und die Größe des Touch Panels wirkt sich stark auf das Design der Benutzeroberfläche aus. Es muss die Größe der Icons und der anderen Designelemente so angepasst werden, dass sie einerseits gut ersichtlich und andererseits einfach über Touch zu bedienen sind.

#### 4.1.1 Befestigung

Das Touchpanel besitzt M4 verschraubungen in einem Abstand von 75mm x 75mm und kann daher einfach an Wänden oder Platten verschraubt werden. Um Den Bildschirm nun in einer ähnlichen Position wie die Instrumententafel zu befestigen wird eine 120mm x ...mm x 1mm Aluminium Platte, wie in der Abbildung zu sehen, gebogen und mit Löchern versehen. Um diese Halterung nun an dem Motorrad zu befestigen werden die Verschraubungen der alten Instrumententafel verwendet.



Abbildung V.6: Befestigung des Displays

#### 4.2 Software

Bevor die Software für die Benutzeroberfläche verfasst wurde, musste das Design, Funktionen sowie die angezeigten Informationen geplant werden, um einen reibungslosen Workflow beim Entwickeln des Front Ends zu gewährleisten. Design Elemente wurden zuvor in Adobe Illustrator vorgefertigt. In den Folgenden Seiten wird das Ergebnis dieses Prozesses dargeboten.

##### 4.2.1 Aufbau

Die nachfolgende Abbildung zeigt den grundsätzlichen Programmaufbau der Benutzeroberfläche. Die Einzelnen Fenster werden als Tabelle mit ihren angezeigten informationen Dargestellt. Dies ist wichtig da jede dieser Informationen vom back end an das front end gesendet werden müssen. Ebenso sind in den letzten Zeilen der Tabellen die QML-Elemente zur Navigation zwischen den einzelnen Fenstern niedergeschrieben. Diese müssen auch schon in der frühen phase der Entwicklung der Benutzeroberfläche definiert werden.

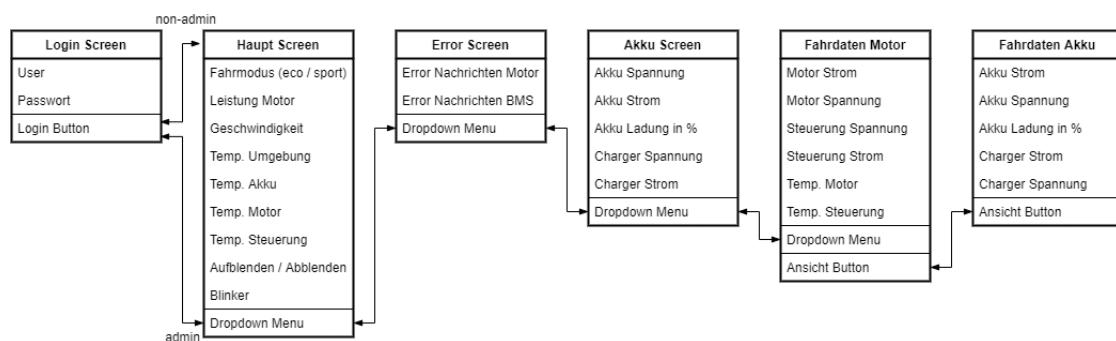


Abbildung V.7: Aufbau der Graphischen Benutzeroberfläche

### 4.3 Komponenten

Komponenten sind wiederverwendbare, gekapselte QML-Elemente mit genau definierten Schnittstellen. Komponenten werden häufig durch Komponentendateien definiert, das heißt durch QML-Dateien. Wichtig ist dabei die Definierung von Schnittstellen sowie Properties und Signals (Siehe 4.5.3).

#### Properties

Der Property eines Objektes kann ein statischer Wert zugewiesen werden, der konstant bleibt, bis ihm explizit ein neuer Wert zugewiesen wird. Um QML und seine integrierte Unterstützung für dynamisches Objektverhalten optimal zu nutzen, verwenden die meisten QML-Objekte jedoch Propertiesbindings.

Propertiesbindings sind eine Kernfunktion von QML, mit der Beziehungen zwischen verschiedenen Objekteigenschaften festgelegt werden können. Wenn sich die Abhängigkeiten einer Property im Wert ändern, wird die Eigenschaft automatisch gemäß der angegebenen Beziehung aktualisiert. Hinter den Kulissen überwacht die QML-Engine die Abhängigkeiten der Eigenschaft. Wenn eine Änderung erkannt wird, wertet die QML-Engine den Bindungsausdruck erneut aus und wendet das neue Ergebnis auf die Eigenschaft an.

##### 4.3.1 Navigations Menu



Abbildung V.8: GUI Komponente - Navigation Menu

##### 4.3.2 Balken Anzeige

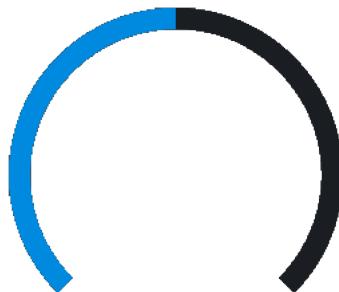


Abbildung V.9: GUI Komponente - Balken Anzeige

##### 4.3.3 Graph

## 4.4 Program Fenster

### 4.4.1 Login

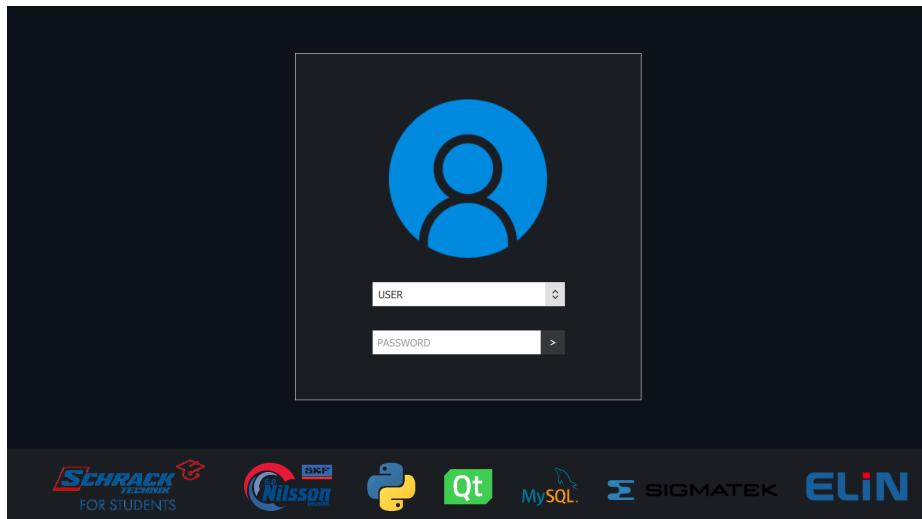


Abbildung V.10: GUI Fenster - Login Menu

### 4.4.2 Fahrdaten

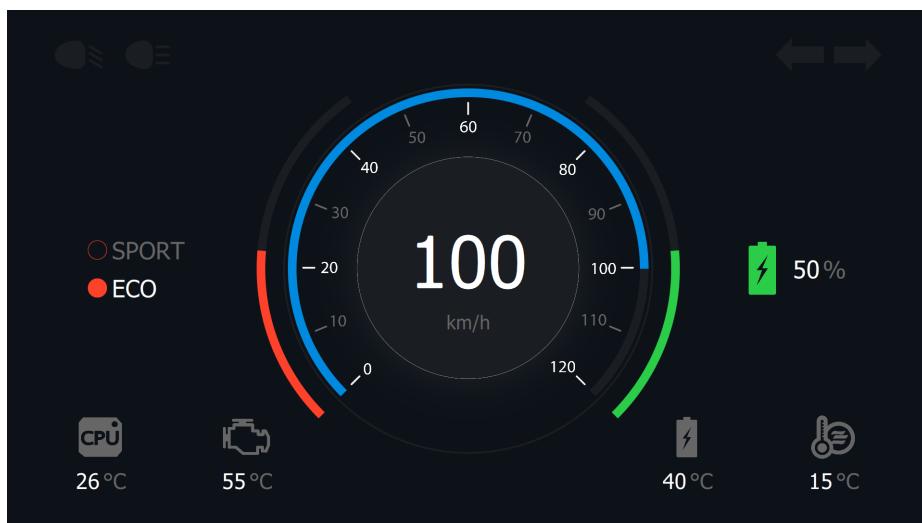


Abbildung V.11: GUI Fenster - Fahrdaten

#### 4.4.3 Akku- und Ladedaten

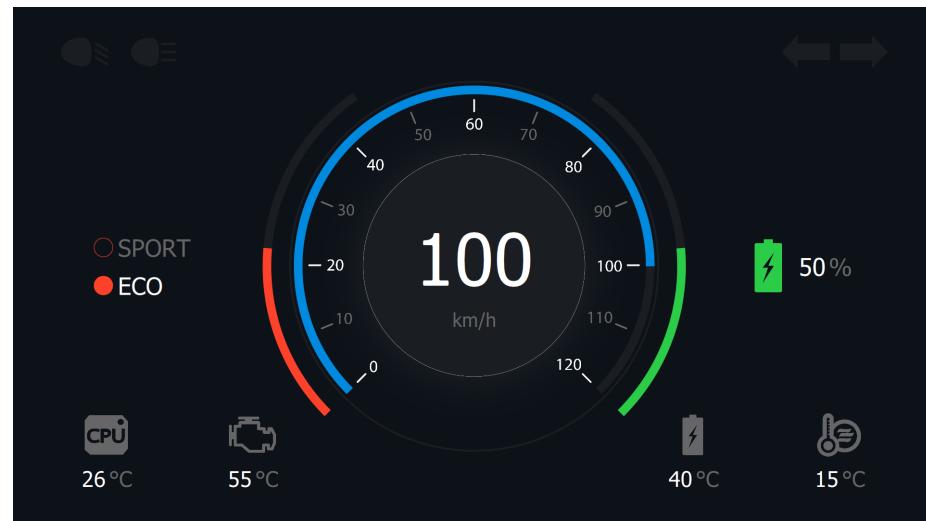


Abbildung V.12: GUI Fenster - Akkudaten

#### 4.4.4 Fahrdaten Diagnose

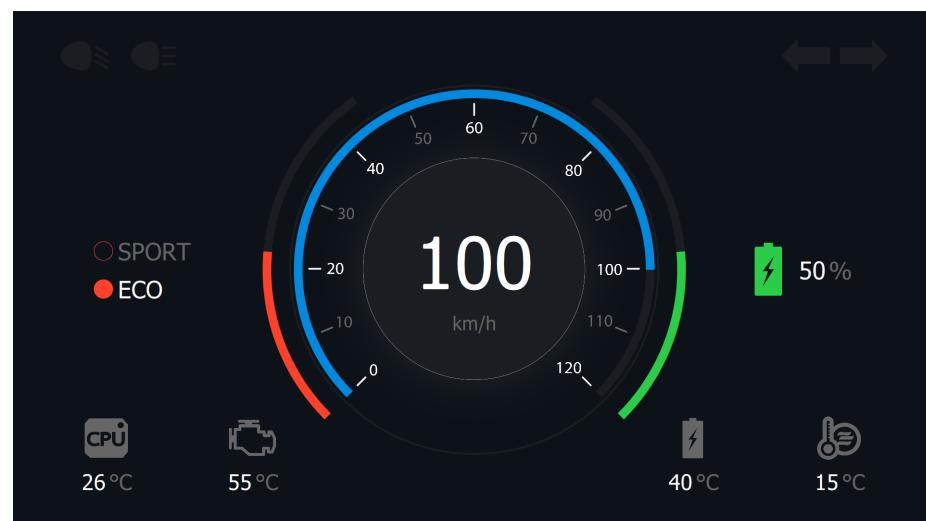


Abbildung V.13: GUI Fenster - Fahrdaten Diagnose

#### 4.4.5 Errors

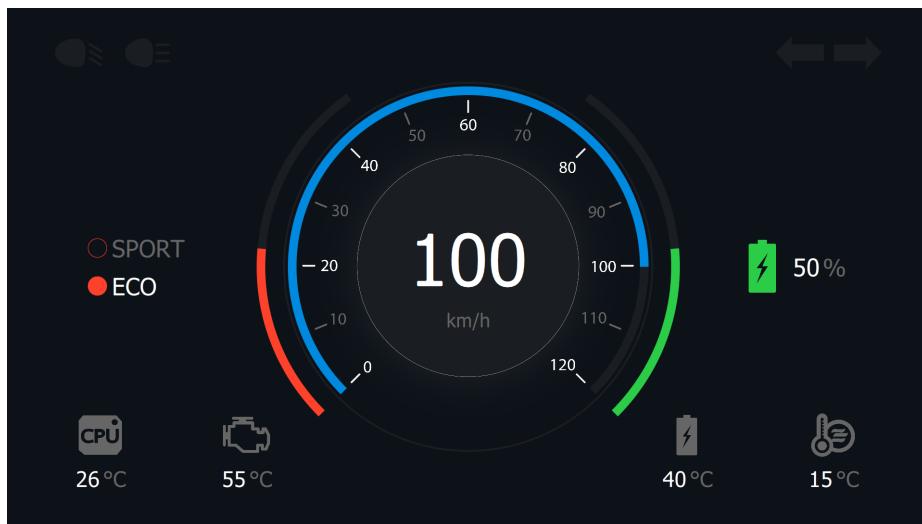


Abbildung V.14: GUI Fenster - Error List

## 4.5 Realisierung der Benutzeroberfläche

### 4.5.1 QML

QML ist eine deklarative Sprache, mit der Benutzeroberflächen anhand ihrer visuellen Komponenten und ihrer Interaktion und Beziehung zueinander beschrieben werden können. Es ist eine gut lesbare Sprache, die entwickelt wurde, um die dynamische Verbindung von Komponenten zu ermöglichen, und die die einfache Wiederverwendung und Anpassung von Komponenten innerhalb einer Benutzeroberfläche ermöglicht. Es bietet eine gut lesbare, deklarative, Syntax mit Unterstützung für JavaScript-Ausdrücke in Kombination mit dynamischen Eigenschaftsverbindungen.

### 4.5.2 Qt-Quick

Das Qt-Quick-Modul ist die Standardbibliothek zum schreiben von QML-Anwendungen. Während das QML-Modul die Engine und die Sprachinfrastruktur bereitstellt, bietet das Qt Quick-Modul alle grundlegenden Typen, die zum Erstellen von Benutzeroberflächen mit QML erforderlich sind. Es bietet eine visuelle Zeichenfläche und Typen zum Erstellen und Animieren visueller Komponenten, zum Empfangen von Benutzereingaben, zum Erstellen von Datenmodellen und Ansichten sowie zum verzögerten Objektinstanziieren. Es können problemlos flüssige, animierte Benutzeroberflächen in QML erstellt werden. Diese Benutzeroberflächen können mit beliebigen Back-End Bibliotheken verbunden werden.

### 4.5.3 Slots und Signals

Slots und Signals werden in Qml zur ereignisgesteuerte Kommunikation zwischen front-end und back-end verwendet. In der folgenden Illustration wird diese anhang eines einfachen Beispiels erklärt.

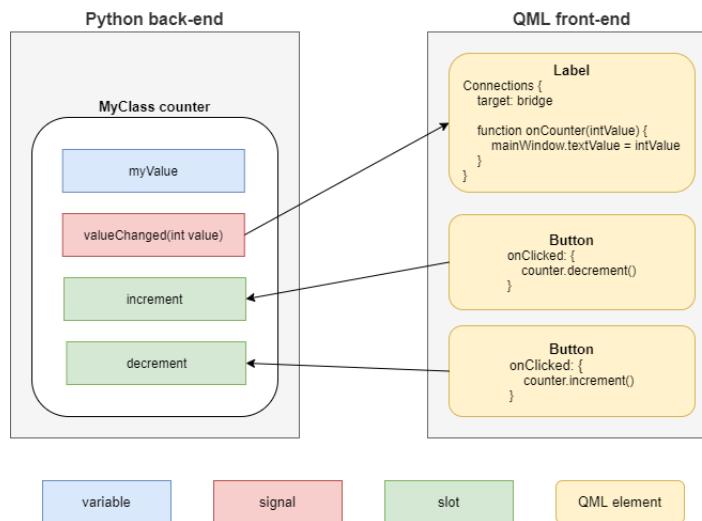


Abbildung V.15: Slots und Signals Konzept

### Signale

Diese können als Mitteilungen angesehen werden welche über das Aufrufen der `signal.emit` funktion vom back-end an das front-end gesehendet wird. Im front-end wird wiederum eine eigens definierte Funktion benötigt um dem Wert einem Property eines QML Elements zuzuweisen.

### Slots

Slots sind Call-Back Funktionen, welche im Back-End definiert werden und sind über die Bridge Class mit dem Front-End verknüpft. Dadurch können diese Funktionen im Front-End aufgerufen und mit Signalen verbunden werden. Sie stellen daher die wichtigste Verbindung zwischen dem Programm und der Benutzeroberfläche dar.

#### 4.5.4 Bridge

## 5 Kommunikation

Um Daten zwischen den mehreren Steuereinheiten des Motorrades zu versenden muss eine echtzeit Kommunikation über ein Bussystem gewährleistet werden. Die Entscheidung ist auf das Controller Area Network Bussystem (CAN-Bus) gefallen. Auschlaggebend für diese Entscheidung war der Curtis Motorcontroller, denn dieser verfügt über eine serielle Schnittstelle (RS-232) sowie ein CAN-System. Für unserer Anwendung bietet das CAN-System eine größere Ausbaufähigkeit sowie größere Übertragungsraten, weshalb wir uns letztendlich auch dafür entschieden haben.

### 5.1 Hardware

#### 5.1.1 CAN-Modul

Da der Raspberry Pi selbst nicht über ein CAN-System verfügt, erfolgt der Anschluss an die Busleitung über ein externes CAN-Modul, welches über das Serial Peripheral Interface (SPI) mit dem Raspberry kommuniziert, welches wie folgt angeschlossen werden muss:

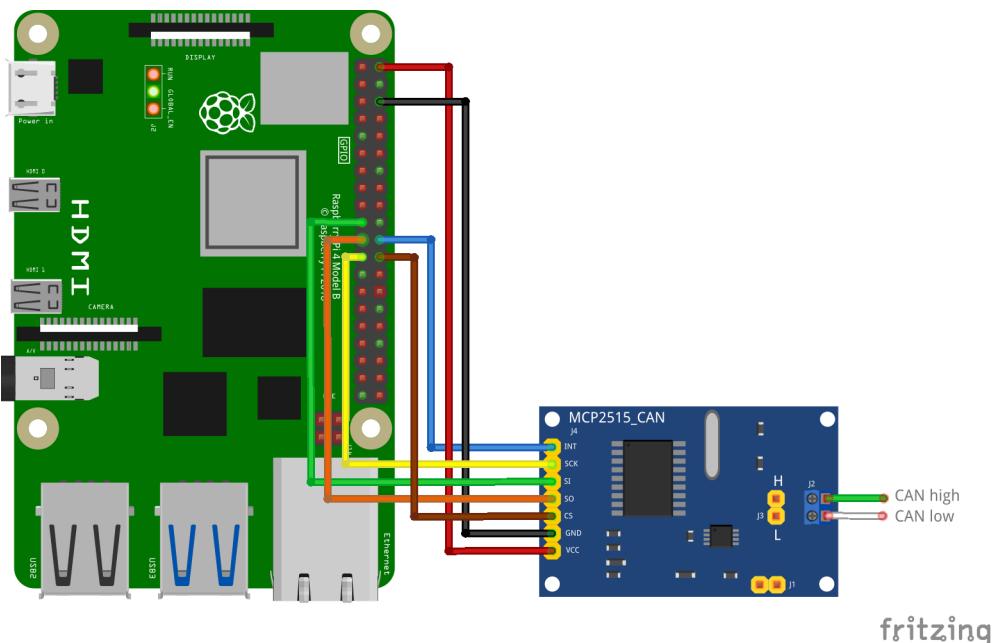


Abbildung V.16: Anschlussplan CAN-Modul

Die Kommunikation wird über zwei Komponenten ermöglicht. Einen MCP2562 Transceiver, welcher für die Verarbeitung der Nachrichten zuständig ist und ein MCP2515 CAN Interface, welches die Daten zwischenspeichert und sich um das Versenden der Nachrichten kümmert. Gemeinsam mit dem Microcomputer ergibt dies nun eine CAN Node, welche fähig ist Nachrichten zu versenden und zu Empfangen.

#### 5.1.2 Netzwerkstruktur

Ein CAN-Netzwerk wird standartmäßig als Linienstruktur oder Sternstruktur aufgebaut. Wir haben uns bewusst für ein Liniensystem entschieden da Sternsysteme nur in bestimmten Anwendungen gebraucht finden und noch dazu markante Nachteile besitzt.

Es müsste zum Beispiel eine Zentrale Steuereinheit den Nachrichtenverkehr steuern, ebenso gibt uns die niedrige Anzahl an Teilnehmern im Netzwerk nicht einmal die Freiheit dazu ein anderes System anzustreben. Erst falls wir in Zukunft das CAN-Netzwerk um Sensoren und Aktoren erweitern wollten, müsste weitere Zeit in die Planung des Netzwerks gesteckt werden.

## 5.2 Listener

### 5.2.1 Receive Data

## 6 Fahrdatenspeicher

### 6.1 Datenbankstruktur

#### 6.1.1 Login System

#### 6.1.2 Motor Daten

#### 6.1.3 Akku Daten

### 6.2 Handler

#### 6.2.1 SELECT Befehl

#### 6.2.2 INSERT Befehl

## Kapitel VI

# Antriebsstrang

## Kapitel VII

# Akku und Ladekonzept

## Kapitel VIII

# Endergebnis

## **Anhang A**

# **Arbeitsnachweis**

**1 Zeitplan**

**2 Kosten**

## Anhang B

# Programmier-Code

## **Anhang C**

# **CAD-Zeichnungen**

## Anhang D

# Schaltpläne

# Abbildungsverzeichnis

V.1	Raspberry Pi - Steuereinheit des HCIS . . . . .	9
V.2	Grundaufbau des Human-Computer Interaction Systems . . . . .	10
V.3	Anschlussplan Inputs . . . . .	12
V.4	Anschlussplan Relai . . . . .	13
V.5	Touch Panel Maße . . . . .	14
V.6	Befestigung des Displays . . . . .	15
V.7	Aufbau der Graphischen Benutzeroberfläche . . . . .	15
V.8	GUI Komponente - Navigation Menu . . . . .	16
V.9	GUI Komponente - Balken Anzeige . . . . .	16
V.10	GUI Fenster - Login Menu . . . . .	17
V.11	GUI Fenster - Fahrdaten . . . . .	17
V.12	GUI Fenster - Akkudaten . . . . .	18
V.13	GUI Fenster - Fahrdaten Diagnose . . . . .	18
V.14	GUI Fenster - Error List . . . . .	19
V.15	Slots und Signals Konzept . . . . .	20
V.16	Anschlussplan CAN-Modul . . . . .	22

# Tabellenverzeichnis

# Listings