Schweizerische
Fachschule
TEKO

METIS

«**M**onitoring of **E**lectrical and **T**hermal **I**nterchange» - **S**olution

Diplomarbeit:	**"Intelligente" Steckverbindung**
Schule:	**TEKO Basel**
Diplomand:	**Lucas Wirz-Vitiuk**
Ausbildung:	**Elektrotechnik**
Jahr:	**2022**

Inhaltsverzeichnis

Management Summary

Im Laufe dieser Diplomarbeit soll die bewährte 16BL-Steckverbindung der Firma Stäubli Electrical Connectors AG um Sensorik und Aktorik erweitert werden, um den elektrischen und thermischen Austausch zu überwachen und auszugeben. Weiter soll ein Demonstrator entstehen, welcher die Funktionen dieser neuen "intelligenten" Steckverbindung aufzeigen kann. Die so gewonnenen Daten sollen den Status der Verbindung anzeigen, für eine zusätzliche Sicherheit sorgen und Schäden oder Ausfälle verhindern. Falls die Sensorik anzeigt, dass bei einer Steckverbindung nicht mehr alles in Ordnung ist, kann diese präventiv ausgetauscht werden, bevor ein Schadenfall eintritt.

Folgende Funktionen wurden im finalen Produkt umgesetzt:

- Steckungen erkennen und das Gegenstück identifizieren
- Seit Beginn getätigte Steckzykeln zählen und abspeichern
- Aktuelle Kern-Temperaturwerte der Verbindung anzeigen
- Bei eingeschaltetem Strom ein Ausstecken verhindern
- Warnungen bei Temperaturlimits ausgeben
- Notabschaltungen bei Grenzwerten einleiten

Die Abbildung 18 sowie die Abbildung 19 auf der Seite 26 zeigen den fertig montierten Demonstrator inklusive zweier verbauten Prototypen.

Die Informationen können simpel anhand von drei LEDs im Ampelprinzip erkannt werden. Grün stellt die Steckbereitschaft dar, Gelb den Steckzustand und Rot den Stromfluss sowie den Temperaturzustand.

Ausserdem können auf dem Dashboard Informationen zum Status, den Steckzyklen, des Verriegelungszustandes des Servos sowie, falls der Stromfluss aktiv ist, die aktuelle Temperatur der Steckverbindung abgelesen werden.

Folgende sechs Status wurden einprogrammiert:

- Offline: METIS nicht verbunden / keine LED leuchtet
- Ready: METIS verbunden und steckbereit / grüne LED leuchtet
- Not Connected: Falsches Gegenstück eingesteckt / gelbe LED blinkt
- Connected: Korrektes Gegenstück eingesteckt / gelbe LED leuchtet
- Current: Stromfluss wurde erkannt / rote LED leuchtet
- Overheat: Temperaturlimit überschritten / rote LED blinkt
- Cutout: Grenzwert überschritten, Notabschaltung / rote LED blinkt

Das Akronym METIS steht, wie auf dem Titelblatt ersichtlich, für «Monitoring of Electrical and Thermal Interchange» - Solution, was zu Deutsch so viel heisst wie Lösung zur Überwachung des elektrischen und thermischen Austauschs.

Metis heisst im altgriechischen ausserdem "kluger Rat" und Metis war in der griechischen Mythologie die Titanin des klugen Ratschlags, der Beratung, der Planung, der Gerissenheit, der List und der Weisheit. Sie war die erste Geliebte von Zeus. Daraus kann man ableiten, dass sie immer den "Strom" im Blick hatte. Aus dieser Herleitung resultieren auch die Namen der beiden Steckverbindungen Athena und Porus, welches die Kinder von Metis und Zeus waren.

Beruflicher Lebenslauf

Meine gesamte berufliche Laufbahn fand bei der Firma Multi-Contact, welche später zu Stäubli Electrical Connectors umbenannt wurde, statt. Dort habe ich meine Lehre absolviert und abgeschlossen sowie berufliche Erfolge gefeiert. Eine entsprechende Übersicht sowie mein beruflicher Werdegang ist in folgender Tabelle aufgelistet:

2008 – 2012	Lehre als Konstrukteur EFZ in Maschinenbau bei der Firma Multi-Contact
2013	Erste Nennung als Erfinder: MULTILAM ML-I Patent: 13159718.9
2013 – 2015	Technische Berufsmatur, berufsbegleitend an der Allgemeinen Gewerbeschule Basel
2016	Nennung als Erfinder: MULTILAM ML-CUX Patente: 16171340.9 / 16171341.7 / 16171346.6
2012 – 2020	Konstrukteur Research & Development bei der Firma Multi-Contact, später Stäubli Electrical Connectors
2019	Nennung als Erfinder: Einstich zum Schutz einer MULTILAM Patent: 19179379.3
2021 – Heute	Konstrukteur Power Transmission & Distribution bei der Firma Stäubli Electrical Connectors
2020 – Heute	Weiterbildung zum Dipl. Techniker HF in Elektrotechnik an der TEKO Basel

Qualifikationsprofil

Lucas Wirz-Vitiuk
Allschwilerstrasse 48
4055 Basel

Qualifikationsprofil
Dipl. Techniker HF in Elektrotechnik

Allgemeine Prozesse		Kompetenzkarte
Menschen führen *Prozess 1*	Projekte geplant; Positiv kommuniziert; Für eine produktive Arbeitsumgebung gesorgt; Lernende bei Fragen und aufgetretenen Problemen unterstützt	1, 2
Entscheidungen fällen *Prozess 2*	Ideen gefunden; Nutzwertanalysen erstellt; Lösungs-Varianten miteinander verglichen; Richtige Entscheidungen getroffen	3, 4
Projekte planen und leiten *Prozess 3*	Programme von der Idee bis zum finalen Produkt erstellt; Sensorgesteuerte Beleuchtung geplant und umgesetzt; Digitale Kunst erstellt und programmiert	5, 6, 7
Sich sprachlich verständigen *Prozess 4*	Konzepte fachmännisch dokumentiert und abgelegt; Schulungen in englischer Sprache durchgeführt	8, 9
Wirkungsvoll präsentieren und kommunizieren *Prozess 5*	Selbst entwickelte Projekte überzeugend präsentiert; Die eigene Meinung sachlich dargelegt; Den eigenen Standpunkt vertreten	10, 11
Unternehmensprozesse verstehen und mitgestalten *Prozess 6*	An firmeninternen Grundsätzen mitgearbeitet; Kennen von Mitarbeitern und Strukturen verschiedener Abteilungen	12, 13
Geschäftsziele erreichen *Prozess 7*	Strategische Ziele der Geschäftsleitung gekannt, verstanden und aktiv unterstützt	14
Umfeld berücksichtigen *Prozess 8*	Gesetzestexte verstanden und angewandt; Normen, Vorschriften, sowie Richtlinien eingehalten	15, 16
Probleme analysieren und lösen *Prozess 9*	Ursachen von Problemen erkannt; Energiebedarf analysiert; Mögliche Lösungen aufgezeigt; Messabweichungen berechnet	17, 18, 19
Sich persönlich weiterentwickeln *Prozess 10*	Sachbücher gelesen; Informationen und Methoden gelernt; Persönliche Ausrichtung angepasst; Persönliches Leitbild erstellt	20, 21

Fachrichtungsspezifische Prozesse		Kompetenzkarte
Produkte entwickeln *Prozess 11*	Funktionen analysiert; Logische Schaltungen aufgezeichnet und simuliert; Neues elektrisches Kontaktelement vom Konzept bis zum fertigen Produkt entwickelt	22, 23
Programme entwickeln *Prozess 12*	Programme erstellt; Mikrocontroller Arduino und Raspberry Pi programmiert; Programmiersprache C++ und Python angewendet; Programme systematisch auf die korrekten Funktionen geprüft; Programme angepasst und korrigiert	24, 25, 26
Anlagen projektieren *Prozess 13*	Elektrische Komponenten berechnet und ausgewählt; Elektrische Schaltpläne aufgezeichnet; Leiterplatinen (PCBs) designt; Programm Autodesk EAGLE verwendet; HF-Steckverbindung konstruiert	27, 28, 29
In Betrieb setzen *Prozess 14*	Programme für Mikrocontroller und SPS erstellt und geladen; Programme systematisch auf die korrekten Funktionen geprüft; Programme angepasst und korrigiert; Steuer- und Regelungen optimiert	30, 31, 32
Elektrotechnische Anlagen unterhalten *Prozess 15*	Automatisierung von Anlagen geplant; Komponenten zur Umsetzung von Automatisierungen vorgesehen; Elektrische Geräte repariert	33, 34
Testeinrichtungen konzipieren und herstellen *Prozess 16*	Elektrische Komponenten evaluiert und ausgemessen; Oszilloskop verwendet; Prüfvorrichtung für Kurzschlussversuche erstellt	35, 36

1 Ausgangslage

1.1 Aufgabenstellung

Die Aufgabe besteht darin, eine Steckverbindung zu entwickeln, die den elektrischen und thermischen Austausch überwachen und ausgeben soll. Dazu soll ein bewährter Steckverbinder aus dem Katalog der Firma Stäubli Electrical Connectors AG um Sensorik und Aktorik erweitert und "intelligent" gemacht werden. Die so gewonnenen Daten sollen den Status der Verbindung anzeigen, für eine zusätzliche Sicherheit sorgen und Schäden oder Ausfälle verhindern. Falls die Sensorik anzeigt, dass bei einer Steckverbindung nicht mehr alles in Ordnung ist, kann diese präventiv ausgetauscht werden, bevor ein Schadenfall eintritt.

1.2 Zieldefinition

Im Laufe der Diplomarbeit soll ein Prototyp einer "intelligenten" Steckverbindung sowie ein Demonstrator, bei welchem man diese Steckverbindung im Einsatz sehen kann, entstehen. Dafür soll die bestehende Steckverbindung 16BL verwendet werden. Die so entstehende "intelligente" Steckverbindung soll eine Steckung erkennen, Steckzyklen zählen und speichern, die Temperatur messen und den Stromfluss erkennen und daraufhin eine Verriegelung aktivieren können.

Die Arbeit soll kein vollends fertigentwickeltes Produkt sein, sondern als Inspiration für zukünftige Entwicklungen dienen. In einem weiteren Schritt ist angedacht, den Prototyp im Prüflabor hinsichtlich erforderlicher Anpassungen für die Serienreife zu untersuchen und zu evaluieren.

1.3 Terminplan

Der Terminplan war eine wichtige Leitplanke bei der Durchführung dieser Diplomarbeit. Zu Beginn lief alles wie geplant. Leider sind im Laufe der Arbeiten unplanbare Probleme und Verzögerungen aufgetreten, welche die eingeplanten Reservezeiten komplett aufgebraucht haben. Glücklicherweise konnten einzelne Schritte, zum Beispiel einige Funktionstests, vorgezogen werden und das Projekt kam nicht komplett zum Stillstand.

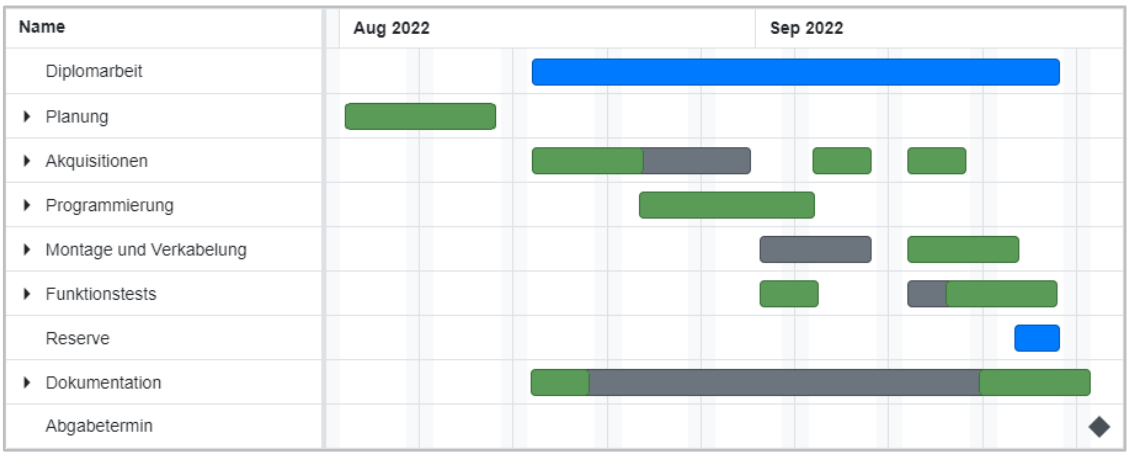

Abbildung 1: Terminplanung mit Ist-Abgleich (grün)

1.4 Steckverbindung 16BL

Die 16BL Steckverbindung ist ein bewährtes Produkt der Firma Stäubli Electrical Connectors AG und kommt zum Beispiel bei mobilen Notstrom-Aggregaten zum Einsatz. Dank der sieben möglichen Kodierungen und etlichen Farbvariationen kann auch schon zum heutigen Zeitpunkt eine grosse Zuverlässigkeit und Sicherheit gewährleistet werden. Dank dem Einsatz einer MULTILAM können hohe Ströme übertragen werden, ohne dass nennenswerte Verlustleistungen oder Erwärmungen auftreten.

Das BL im Produktname steht für "bayonet locking" und beschreibt den Verriegelungsmechanismus. Um Stecker und Buchse miteinander zu verbinden, müssen diese um ungefähr 45° gegeneinander gedreht werden und rasten dann ein. Erst wenn die Isolation zurückgezogen und die beiden Seiten in die andere Richtung gedreht werden, kann die Verbindung wieder getrennt werden.

Zeitlich bedingt liegt der Fokus in dieser Arbeit auf der Einbaudose (16BL-PP), da bei dieser reichlich Platz für mögliche Anbauten vorhanden ist. Vorstellbar ist, dass zukünftig weitere Produkte des Portfolios durch Sensorik erweitert werden könnten.

Abbildung 2: Steckverbindung 16BL

2 Planungsphase

2.1 CAD

Als Vorbereitung wurde die bestehende Isolation des 16BL so angepasst, dass dieser durch einen Mikrokontroller, ein NFC-Board und Sensoren ergänzt werden kann. Dabei wurde darauf geachtet, dass dieser immer noch in eine Bohrung mit einem Durchmesser von 54 Millimetern gemäss Montageanleitung (MA409) passt. Die Platinen werden mit simplen Deckeln abgedeckt, bei welchen darauf geachtet wurde, dass keine Durchdringungen zu den bestehenden Bauteilen auftreten und diese dadurch problemlos montiert werden können. Weiter konnte in diesem Schritt auch die Position des NFC-Stickers auf dem Gegenstück evaluiert werden. Abschliessend wurde der USB-Anschluss freigestellt und ein Anschluss für den Servo mittels dreier Pins eingeplant.

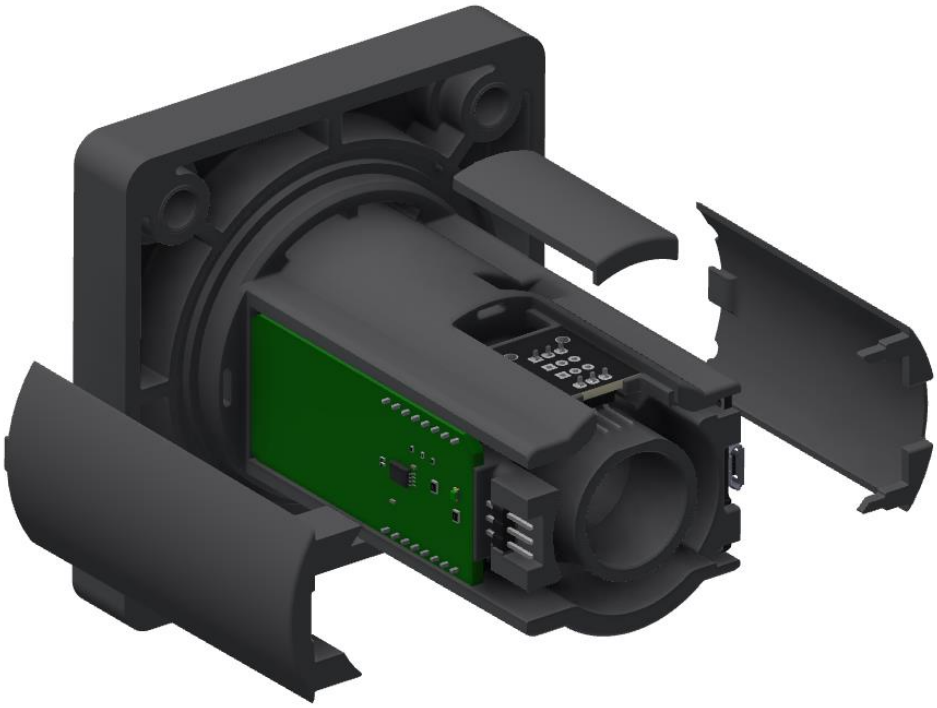

Abbildung 3: Isolation des Prototypes inklusive Platinen und Abdeckungen

Im gleichen Schritt wurde der Demonstrator so weit wie möglich geplant und dimensioniert. Dabei wurden die Farben für die jeweiligen Seiten bestimmt und die Signalisation des Stromflusses mittels entsprechender Lampen eingeplant. Ausserdem wurde ein Bildschirm, LEDs, Schalter und Servos für die Verriegelung positioniert und ins Gehäuse eingepasst.

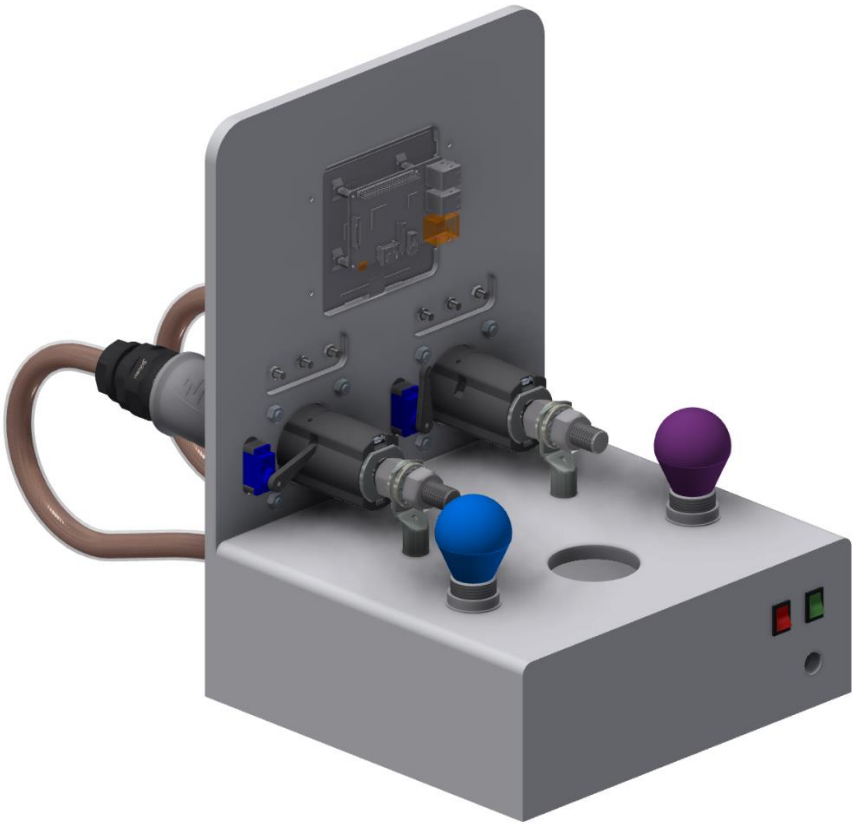

Abbildung 4: Ansicht des Demonstrators von hinten

Abbildung 5: Frontansicht des Demonstrators

2.2 Akquisitionen

Sofort als ich das grüne Licht zur Durchführung dieser Diplomarbeit seitens TEKO bekam, habe ich die ersten Bestellungen mit den geplanten Mikrokontrollern, Sensoren, NFC-Sticker und dem Bildschirm ausgelöst. Glücklicherweise waren alle Lieferanten äusserst schnell und zuverlässig bei ihren Lieferungen und die ersten Pakete trafen rasch ein. Weiter habe ich einen ersten Druckauftrag für die additive Fertigung der neugestalteten Isolationen inklusive der Deckel aufgegeben. Ausserdem wurden auch gleich die Servohalterungen sowie die Verriegelungshaken zum "Ausdrucken" mitgesendet. Leider habe ich mit dem Aufgeben der Werkstatt-Aufträge rückblickend zu lange gewartet. Der ursprüngliche Plan, das Gehäuse aus Holz zu fertigen, wurde nach einem eindringlichen Gespräch durch den Werkstattleiter verworfen und durch einen Aufbau aus Aluminium ersetzt. Der grosse Einsatz der beiden Polymechaniker war beeindruckend mitanzusehen.

3 Programmierung

3.1 Prototyp / ESP32

Das Programm des Mikrokontrollers für die Steuerung dieser "intelligenten" Steckverbindung war das komplexeste Script, welches ich je erstellt habe. Übersichtshalber habe ich dabei zum ersten Mal mit Tabs gearbeitet und so die Programmabschnitte nach Funktion aufgeteilt. Aufgrund der Komplexität werde ich in dieser Dokumentation nur die wichtigsten Programmabschnitte beschreiben.

Hauptteil (Athena.ino und Porus.ino)

Zunächst soll sich der mit WiFi ausgestattete Mikrokontroller ESP32 mit einem Netzwerk verbinden. Dafür werden die SSID und das Passwort hardcoded (zu Deutsch: fest im Code eingetragen), was für ein verkaufsfertiges Produkt keine Option wäre – dieser Punkt wurde schon mit der firmeninternen IT diskutiert, konnte aber in der kurzen zur Verfügung stehenden Zeit nicht besser umgesetzt werden.

Mit dem folgenden Ausschnitt aus dem Code wurde das Netzwerk definiert:

```
6   // Load Wi-Fi library
7   #include <WiFi.h>
8   #include <WebServer.h>
9
10  // Set network credentials
11  const char* ssid = "Edipsos (METIS-Hotspot)";
12  const char* password = "Lucas6.0";
```

Code-Teil 1: Festlegung des Netzwerkes

Im Setup-Teil werden alle Funktionen initialisiert und gestartet:

```
54      // Setup current detection
55      pinMode(Detection_Pin, INPUT_PULLDOWN);
56
57      // Setup and test LED-Outputs
58      setup_LEDs();
59      test_LEDs();
60
61      // Setup temperature measurement
62      setup_temperature_measurement();
63
64      // Setup servo
65      setup_servo();
66
67      // Read mating cycles
68      mating_cycles = readMemory();
69
70      // Connect to network
73      WiFi.begin(ssid, password);
83      server.begin();
84
85      // Setup NFC recognition
86      setup_NFC();
```

Code-Teil 2: Setup

Als nächstes soll der ESP32 die Informationen zum Status mittels einer Homepage ausgeben. Dafür wird die gesamte Formatierung mittels *Cascading Style Sheets* (CSS) und der Inhalt in Form von HTML ausgegeben. Der Code wurde durch das Darstellen mittels C++ sehr unübersichtlich und ich verzichte deshalb darauf, diesen hier abzubilden, aber er ist jedoch im Anhang zu finden.

Da sich die Informationen fortlaufend ändern, musste auch die Homepage so ausgeführt werden, dass sich diese kontinuierlich selbst aktualisiert. Dies wurde mittels dem HTML <meta> http-equiv Attribut umgesetzt und so eine Aktualisierungsrate von einer Sekunde eingestellt. Auch dieser Punkt wurde mit der internen IT diskutiert und würde bei einem realen Produkt nicht so umgesetzt. Idealerweise würden die Informationen in reiner Textform geliefert und an der Endstelle formatiert und so auch fortlaufend aktualisiert. Ich habe dies jedoch für meine Prototypen nicht als erstrebenswerte Lösung erachtete, da dann keine einfache Betrachtung via Computer oder Smartphone mehr möglich gewesen wäre.

Der resultierende Code der Homepage ist aufgrund der Verwendung von *tab stopps* (Tabulatorstopps, '\t') und *new lines* (Zeilenumbrüche, '\n') schön formatiert.

Ein Ausschnitt davon sieht je nach aktuellem Status zum Beispiel so aus:

```
75  <table>
76          <tr>
77                  <td>Status:</td>
78                  <td><p style=color:#32cd32 >Ready</p></td>
79          </tr>
80
81          <tr>
82                  <td>Mating Cycles:</td>
83                  <td id="matingCycles"></td>
84          </tr>
85
86          <tr>
87                  <td>Interlock:</td>
88                  <td>closed</td>
89          </tr>
90
91          <tr>
92                  <td>Temperature:</td>
93                  <td>33.3 &deg;C</td>
94          </tr>
95  </table>
```

Code-Teil 3: Ausschnitt aus der Homepage des Prototypes

Im Hauptteil des Codes läuft ausserdem durchgehend die Abfrage des aktuellen Status ab. Diese besteht zum einen aus der NFC-Erkennung (Status: 'Ready', 'Connected' und 'Not Connected') und zum anderen aus der Stromerkennung (Status: 'Current') und dem Temperaturwert (Status: 'Overheat' und 'Cutout'), welche nur in einem bestimmten NFC-Status aktiviert werden können.

Weiter wird in dieser Abfrage auch der Verriegelungs-Servo und die entsprechenden LEDs angesteuert. Da die Abfrage fortlaufend läuft, reagieren die hier angesteuerten Elemente schneller als die Informationen auf der Homepage (bzw. auf dem Bildschirm, welcher diese Homepage darstellt).

Folgende Abfrage stellt den Status Anhand der NFC-Erkennung fest:

```
93          if (!digitalRead(Detection_Pin)) {
94
95            // Open interlock
96            unlock();
97            interlock = "open";
98
99            // Check NFC state
100           NFCstate = getNFCstate();
101
102           // Actuate LEDs according to state
103           if (NFCstate == "none") {
104             status = "Ready";
105             color = "#32cd32";
106             green_Signal();
107             is_connected = false;
108             wifi_switch_url = wifi_switch + "off";
109           }
110           else if ((NFCstate != "none")) {
111             if (!is_connected) {
112               mating_cycles++;
113               writeMemory("cycles", mating_cycles);
114               is_connected = true;
115             }
116             if ((NFCstate == "right")) {
117               status = "Connected";
118               color = "#ffd700";
119               yellow_Signal();
120               wifi_switch_url = wifi_switch + "on";
121             }
122             else if ((NFCstate == "wrong")) {
123               status = "Not Connected";
124               color = "#ffd700";
125               alarm("yellow");
126               wifi_switch_url = wifi_switch + "off";
127             }
```

Code-Teil 4: Statusabfrage mittels NFC-Erkennung

Folgende fortsetzende Abfrage stellt den Status Anhand der Stromerkennung und dem Temperaturwert fest:

```
129           } else {
130
131             // Close interlock
132             lock();
133             interlock = "closed";
134
135             // Get temperature and check limits
136             temperature = measure_temperature(0);
137             status = "Current";
138             color = "#D61D23";
139             red_Signal();
140             if (temperature >= warning_limit) {
141               status = "Overheat";
142               alarm("red");
143               if (temperature >= maximum_temperature) {
144                 status = "Cutout";
145                 is_cutout = true;
146               }
147             }
148           }
```

Code-Teil 5: Statusabfrage mittels Stromerkennung und Temperaturwert

Auch die Stromerkennung befindet sich noch im Hauptteil, da diese grundsätzlich im Code nur als High-Signal an einem Pin abgebildet werden muss.

Signalisation mittels LEDs (LEDs.ino)

Die unterschiedlichen Funktionen der LEDs sind grundsätzlich sehr simpel. Da immer nur eine LED leuchten soll, müssen beim jeweiligen Aufleuchten einer LED die anderen beiden ausgeschalten werden. Ausserdem wurde eine Funktion eingebaut, welche die gelbe oder die rote LED in einem Zeitintervall von 250 ms blinken lässt.

Um beim Aufstarten sofort zu erkennen, ob alle LEDs wie gewünscht funktionieren wird jeweils beim Starten folgender LED-Test durchlaufen:

```
57      void test_LEDs() {
58        digitalWrite(green_Pin, HIGH);
59        digitalWrite(yellow_Pin, HIGH);
60        digitalWrite(red_Pin, HIGH);
61        delay(10*LED_delay);
62        for (int i = 0; i < 4; i++) {
63          green_Signal();
64          delay(LED_delay);
65          yellow_Signal();
66          delay(LED_delay);
67          red_Signal();
68          delay(LED_delay);
69          yellow_Signal();
70          delay(LED_delay);
71        }
72        green_Signal();
73        delay(LED_delay);
74        digitalWrite(green_Pin, LOW);
75        digitalWrite(yellow_Pin, LOW);
76        digitalWrite(red_Pin, LOW);
77        delay(LED_delay);
78      }
```

Code-Teil 6: Einschalten aller LEDs und Durchlaufen einer Schlaufe zum Testen der LEDs

Messung der Temperatur (Temperature.ino)

Für das Messen und Ausgeben des Temperaturwertes wurde der Empfehlung auf der Homepage des Herstellers des Moduls gefolgt. Grundsätzlich könnten mehrere Module verwendet werden, darum muss der entsprechende Index für die Messung mitgegeben werden. In diesem spezifischen Fall ist der Index immer 0, da nur ein Modul verwendet wird.

Mit folgender Funktion wird die aktuelle Temperatur abgerufen:

```
24      float measure_temperature(int sensor_number) {
25
26        // Measure temperature
27        measure.requestTemperatures();
28
29        // Return temperature in degree Celsius
30        return measure.getTempCByIndex(sensor_number);
31      }
```

Code-Teil 7: Messen und ausgeben der aktuellen Temperatur

Ansteuerung des Verriegelungsmechanismus (Servo.ino)

Auch die Ansteuerung des Servos, welcher für den Verriegelungsmechanismus verwendet wird, ist relativ simpel, da die ESP32 Servo-Library (`Servo.h`) verwendet werden kann. Im Code werden die beiden Zustände verriegelt (`locked`) und geöffnet (`unlocked`) als fixe Werte festgelegt.

Mit folgenden beiden Funktionen können dementsprechend die beiden Zustände des Hebelarmes (`lever`) eingestellt werden:

```
25      void lock() {
26        lever.write(locked);
27      }
28
29      void unlock() {
30        lever.write(unlocked);
31      }
```

Code-Teil 8: Funktionen zum Ansteuern des Servos

Einlesen und Abspeichern der Steckzykeln (Cycles.ino)

Damit die Anzahl der getätigten Steckzykeln nicht bei jedem Neustart zurückgesetzt werden, müssen diese auf dem ESP32 abgespeichert werden. Glücklicherweise bietet dieser eine solche Funktion (`Preferences.h`) an. Nichtsdestotrotz hat die Umsetzung viel Zeit in Anspruch genommen und schlussendlich sind zwei Funktionen entstanden.

Die eine Funktion speichert, sobald eine Steckung erkannt und gezählt wurde, den neuen Wert auf dem ESP32 ab (`writeMemory`). Die Andere liest bei jedem Start (also in der Setup-Phase) den abgespeicherten Wert wieder ein (`readMemory`). So kann der aktuelle Wert anzeigt und im Fall einer Steckung erhöht werden werden:

```
14    unsigned int readMemory() {
15      memory.begin("Storage", false);
16      cycles = memory.getUInt("cycles", 0);
17      memory.end();
18      return cycles;
19    }
20
21    void writeMemory(const char* address, unsigned int content) {
22      memory.begin("Storage", false);
23      memory.putUInt(address, content);
24      memory.end();
25    }
```

Code-Teil 9: Lesen und Schreiben des aktuellen Wertes der Steckzykeln

Erkennen und Einlesen eines NFC-Stickers (NFC.ino)

Das Erkennen und Einlesen der NFC-Sticker hat für grosses Kopfzerbrechen gesorgt, da beim Bestellen des NFC-Boards nicht darauf geachtet wurde, welcher Chip verbaut ist. Für den auf dem Bord verwendeten NFC-Chip PN7150 wurden, im Vergleich zum gängigen Chip PN532, keine vordefinierten Bibliotheken gefunden. Durch einen glücklichen Zufall bin ich bei der Suche auf Pascal Roobrouck alias 'Stroom' gestossen, welcher eine Rohfassung einer Library erstellt und hochgeladen hatte. Damit hatte ich erste Erfolge und konnte Sticker erkennen und später sogar die ID der Sticker einlesen und wie gewünscht mit einem fest einprogrammierten Wert vergleichen. Ich hatte das Gefühl, dass dieses Problem damit erledigt ist – doch ich hatte mich getäuscht. Beim Vereinigen der einzelnen Programmbausteine funktionierte das Einlesen der NFC-Sticker nicht mehr. Zunächst hatte ich das Gefühl, dass Verzögerungen im Programmablauf ein kontinuierliches Einlesen unterbinden. In schierer Verzweiflung habe ich mehrere Dinge ausprobiert, doch nichts veränderte die Situation. Irgendwie kollidierte die NFC-Erkennung mit dem Aufbau der Homepage. Als letzte Option habe ich die Initiierung der NFC-Erkennung an den Schluss des Setups verschoben… und wie durch ein Wunder lief danach alles fehlerfrei durch.

Die eindeutige Identifizierungsnummer eines NFC-Stickers besteht aus vier hexadezimalen Zahlen (z.B.: 33 74 E4 BB, im Code abgebildet als: `0x33, 0x74, 0xE4, 0xBB`). Im folgenden Code-Abschnitt wird eine solche einprogrammierte eindeutige ID mit einer eingelesenen ID verglichen und ausgegeben, ob das Resultat richtig (`right`) oder falsch (`wrong`) ist:

```
59    theReaderWriter.theState = ReaderWriterState::singleTagPresent;
60    for (uint8_t index = 0; index < 4; index++)
61    {
62      theValue[index] = theNCI.rxBuffer[index + 13];
63      if (theValue[index] != theID[index]) {
64        state = "wrong";
65        break;
66      }
67      if (index == 3) {
68        state = "right";
69      }
70    }
```

Code-Teil 10: Einlesen und Abgleichen der ID eines NFC-Stickers

3.2 Demonstrator / Raspberry Pi

Der Raspberry Pi mit angeschlossenem Display übernimmt in diesem Projekt zwei Funktionen. Zum einen dient er als Access Point und stellt ein WiFi-Netzwerk zur Verfügung, zum anderen zeigt er auf dem Display den jeweiligen Status der beiden Steckverbindungen an. Aus ästhetischen Gründen wurden der Mauszeiger sowie die Taskleiste ausgeblendet. Ausserdem wird beim Aufstarten automatisch das Dashboard im Vollbildmodus geöffnet. Zur Erkennung, ob dies bereits geschehen ist oder nicht, ist der Hintergrund des Betriebssystems in Stäubli-Grau gehalten und jener des Dashboards ist weiss.

Für das Einrichten als Access Point wurde das Programm RaspAP installiert und entsprechend konfiguriert, was keine grösseren Schwierigkeiten bereitet hat. Im Konfigurations-Tool war es möglich, den beiden Mikrokontrollern sowie den verwendeten WiFi-Switches statische IPs zuzuweisen.

Raspberry Pi:

- User: pi
- Passwort: DA22

Netzwerk:

- WiFi: Edipsos (METIS-Hotspot)
- Passwort: Lucas6.0
- Config-IP: 10.3.141.1

Athena (lila):

- IP: 10.3.141.201
- WiFi-Switch: 10.3.141.202

Porus (blau):

- IP: 10.3.141.211
- WiFi-Switch: 10.3.141.222

Das Anzeigen der beiden Homepages mit den jeweiligen Informationen der zwei verbauten Prototypen war jedoch ein Punkt, welcher massiv unterschätzt wurde und viel Zeit in Anspruch genommen hat. Das Dashboard musste irgendwie überprüfen können, ob die Prototypen online sind und falls nicht, eine lokale Homepage anzeigen, welche den Offline-Status anzeigt. Erst mit professioneller Unterstützung aus der IT-Abteilung und dem Einsatz einer JavaScript-gestützten Funktion konnte dieses Ziel erreicht werden. Falls ein Steckverbinder (bzw. der entsprechende ESP32) nicht erreichbar sein sollte, wird alle 5 Sekunden überprüft, ob sich die Situation verändert hat.

Folgender Ausschnitt aus der Datei ‘Dashboard.html’ zeigt am Beispiel des linken Fensters, wie die Aktualisierung umgesetzt wurde:

```
34  function left_window() {
35    $.ajax({
36      url: 'http://10.3.141.201',
37        error:  function() {
38            console.log("left window: failed");
39            $('iframe[name="left"]').attr("src", "Offline.html");
40            setTimeout(function() {left_window()}, 5000);
41        },
42        success: function() {
43            console.log("left window: success");
44            $('iframe[name="left"]').attr("src", "10.3.141.201");
45            refresh_left_window();
46        },
47      timeout: 2500
48    });
49  }
50
51  function refresh_left_window() {
52    $.ajax({
53      url: 'http://10.3.141.201',
54        error:  function() {
55            console.log("left refresh: failed");
56            $('iframe[name="left"]').attr("src", "Offline.html");
57           setTimeout(function() {left_window()}, 5000);
58        },
59        success: function() {
60            console.log("left refresh: success");
61            setTimeout(function() {refresh_left_window()}, 5000);
62        },
63      timeout: 2500
64    });
65  }
```

Code-Teil 11: Aktualisierung des linken Fensters des Dashboards

Das Ausschalten des Demonstrators ist aufgrund des Einsatzes des Raspberry Pi etwas schwieriger, denn dieser sollte grundsätzlich nicht einfach vom Strom getrennt, sondern korrekt heruntergefahren werden. Dies ist bei der aktuellen Ausführung nur möglich, wenn man sich mittels SSH-Verbindung via Computer oder Smartphone einloggt und den Befehl `sudo poweroff` übergibt, welcher den Raspberry Pi sofort herunterfährt.

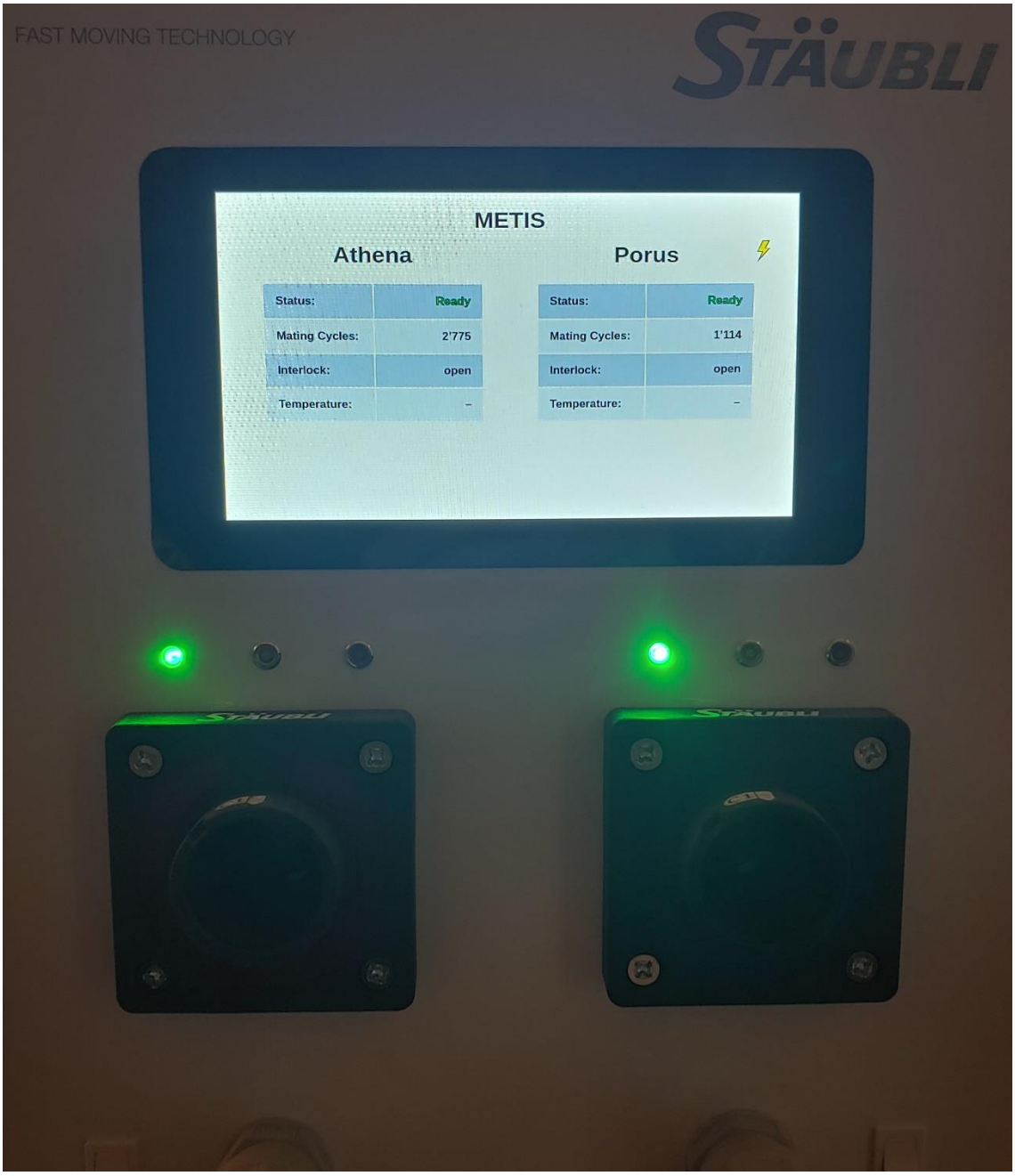

Abbildung 6: Frontansicht des Demonstrators inklusive laufendem Dashboard

4 Funktionstest

4.1 Problem(e): Stromdetektierung

Trotz der langen Vorbereitungsphase konnten einige Dinge nicht abgeklärt und mussten ausprobiert werden. Am meisten Probleme bereitete die Stromerkennung.

Anfangs sollte diese mittels Hall-Sensors umgesetzt werden und dafür wurde auch alles vorbereitet. Leider zeigte dieser erst bei hohen magnetischen Kräften (z.B. ein Küchenmagnet) eine Detektierung an. Hinzu kam, dass die Einstellmöglichkeiten nicht wie gewünscht funktionierten, obwohl ein Potentiometer, welches die Sensitivität des Sensors einstellen sollte, vorhanden war. Denn schon etwa in der Mitte des Einstellungsbereiches gab der Sensor ein High-Signal aus.

In einem nächsten Schritt wurden zwei weitere Sensoren bestellt. Zum einen ein Reed-Kontakt, welcher ebenfalls berührungslos funktionieren könnte und zum anderen ein Strommess-Sensor, welcher bis zu 5 A messen und ausgeben konnte. Letzterer müsste im Stromkreis dazwischengeschaltet werden, um den aktuellen Stromwert messen zu können. Der Reed-Kontakt landete rasch aus dem gleichen Grund wie der Hall-Sensor im Aus. Bei Versuchen mit dem Messen des Stroms mit dem Strommess-Sensor stellte ich fest, dass die 4 Watt der LED-Birne kaum einen messbaren unterschied darstellen. Aus diesem Grund habe ich einen weiteren Verbraucher – konkret einen Staubsauger – parallel zur Lampe geschalten. Sofort war ein deutlicher Ausschlag bei der Strommessung erkennbar. An diesem Punkt habe ich mich gefragt, ob die beiden ausrangierten Sensoren diesen deutlich höheren Strom ebenfalls erkennen könnten. Zwei kurze Tests später wusste ich, dass dies nicht der Fall ist und ich höchstwahrscheinlich keine berührungslose Lösung für die Stromdetektierung finden werde. Auch der Staubsauger war für den finalen Demonstrator keine Lösung.

Die vermeintliche Lösung dafür sollte ein Leistungswiderstand sein. Diesen habe ich mittels der bekannten URI- sowie Leistungs-Formel wie folgt ausgelegt:

Netzspannung:	$U = 230\ V$
Leistung der Lampe:	$P = 4\ W$
Strom der Lampe:	$I = \frac{P}{U} = \frac{4\ W}{230\ V} = 17.4\ mA$
Zielstrom (geschätzt):	$I_{Total} = 1.2\ A$ $\quad I_L = 1.2\ A - 17.4\ mA = 1.18\ A$
Leistungswiderstand:	$R_L = \frac{U}{I} = 194\ \Omega \rightarrow 180\ \Omega$ (zum Kauf gefunden)
Zielstrom (gerechnet):	$I_L = \frac{U}{R} = \frac{230\ V}{180\ \Omega} = 1.28\ A$
Auszuhaltende Leistung:	$P_L = U \times I_L = 230\ V \times 1.28\ A = 294.4\ W$
Reserve:	$1 - \frac{294.4\ W}{600\ W} = 51\%$

Der so ermittelte Leistungswiderstand war nicht billig, trotzdem wollte ich das versuchen und habe mir zwei davon bestellt. Aufgrund der hohen Reserve wollte ich einen davon rasch testen… und er flog mir innerhalb von gefühlten zwei Sekunden um die Ohren.

Abbildung 7: Abbrandspuren eines ungekühlten 600W-Leistungswiderstandes

Die Wärmeentwicklung musste demnach sehr rasch sehr hoch gewesen sein…

Nichtdestotrotz habe mit dem Zweiten weitergetestet. Diesmal habe ich auf der Unterseite Wärmeleitpaste aufgetragen, die Kabel mittels Kabelschuhen korrekt befestigt und den Leistungswiderstand fachmännisch am Aluminiumgehäuse angebracht. Schon nach 10 Sekunden war das Gehäuse so heiss, dass man es nicht mehr berühren konnte. Bei vorsichtigen Tests kam ich mit diesem Aufbau trotzdem auf 30 Sekunden, bevor ich mich nicht mehr traute, das ganze weiterlaufen zu lassen, weil sich das gesamte Gehäuse derart stark erwärmte. Wie bei einer Herdplatte konnte man die Hitze auch auf eine Entfernung spüren. Demnach war auch diese Lösung keine Option für das finale Produkt.

4.2 Lösung: Detektierung simulieren

Die letzte Lösung war also eine Simulation der Stromdetektierung. Dafür muss der Mikrokontroller ein High-Signal erhalten, sobald der Schalter der LED-Lampe betätigt wird. Dies ist einfach umsetzbar, wenn ein Gerät, welches einen 3.3 Volt Strom ausgeben oder schalten kann, parallel zur Lampe dazwischen gehängt wird. Dies könnte entweder ein Netzteil (verbunden mit der Erdung GND und des entsprechenden Signal-Pins des ESP32) oder ein Relais (verbunden mit 3,3 Volt und ebenfalls dem Signal-Pin), welches mit Netzspannung (230 V) geschalten werden kann. Beide Optionen wurden getestet und funktionieren grundsätzlich. Da beim Netzteil eine Ausschaltverzögerung auftritt, habe ich mich schlussendlich für ein Relais entschieden und dieses im finalen Produkt verbaut.

5 Montage

5.1 Prototypen

Die Montage der Prototypen war eine Frickelaufgabe. Eigentlich mussten nur sechs Komponente korrekt miteinander verbunden werden, doch dabei handelte es sich um rund 30 Lötstellen und dies war eine grosse Challenge. Die elektronischen Komponenten des Prototypes sind der ESP32 Mikrokontroller, das NFC-Board, der Temperatursensor, zwei Anschlüsse für die Simulation der Stromerkennung, drei LEDs inklusive Vorwiderstände und drei Pins für den Anschluss des Servos.

Wie diese miteinander verbunden werden mussten, sieht man auf dem folgenden Schaltplan:

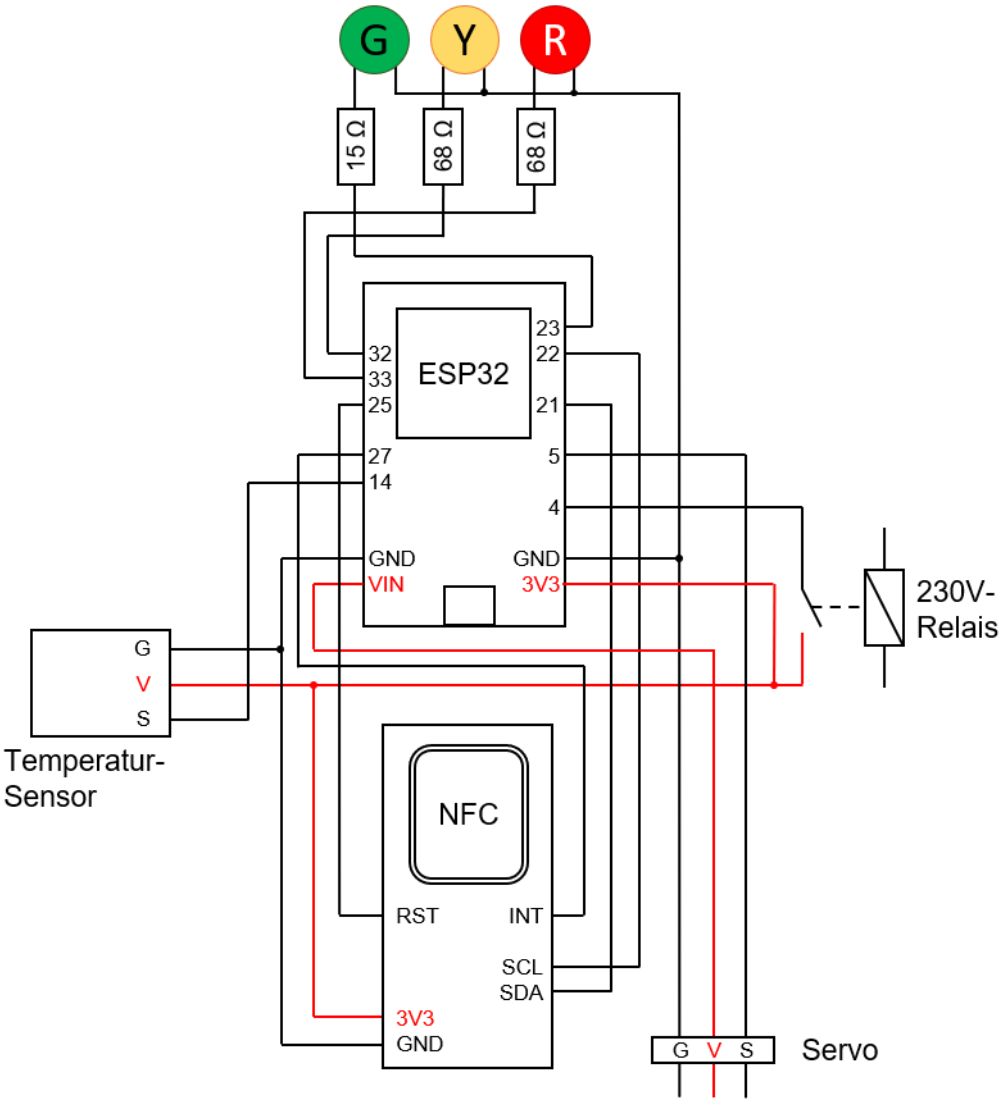

Abbildung 8: Schema zur Verkabelung des Prototypes

Die Vorwiderstände der LEDs berechnen sich wie folgt:

Spannung am ESP32: $U = 3.3\ V$

Strom der LEDs: $I = 20\ mA = 0.02\ A$

Spannung grün: $U_g = 3\ V$

Spannung gelb/rot: $U_{y/r} = 2\ V$

Vorwiderstand grün: $R_g = \frac{U - U_g}{I} = \frac{3.3\ V - 3\ V}{0.02\ A} = \frac{0.3\ V}{0.02\ A} = 15\Omega$

Vorwiderstand gelb/rot: $R_{y/r} = \frac{U - U_{y/r}}{I} = \frac{3.3\ V - 2\ V}{0.02\ A} = \frac{1.3\ V}{0.02\ A} = 65\Omega \approx 68\Omega$

Die Länge der Kabel musste jeweils so gewählt werden, dass alle Komponenten gut einbaubar bleiben und trotzdem alles innerhalb der Isolation Platz hat. Ausserdem mussten die Gabelungen, zum Beispiel bei den Erdungsanschlüssen und der Stromversorgung, gut isoliert werden, damit da keine ungewollten Nebenschlüsse entstehen.

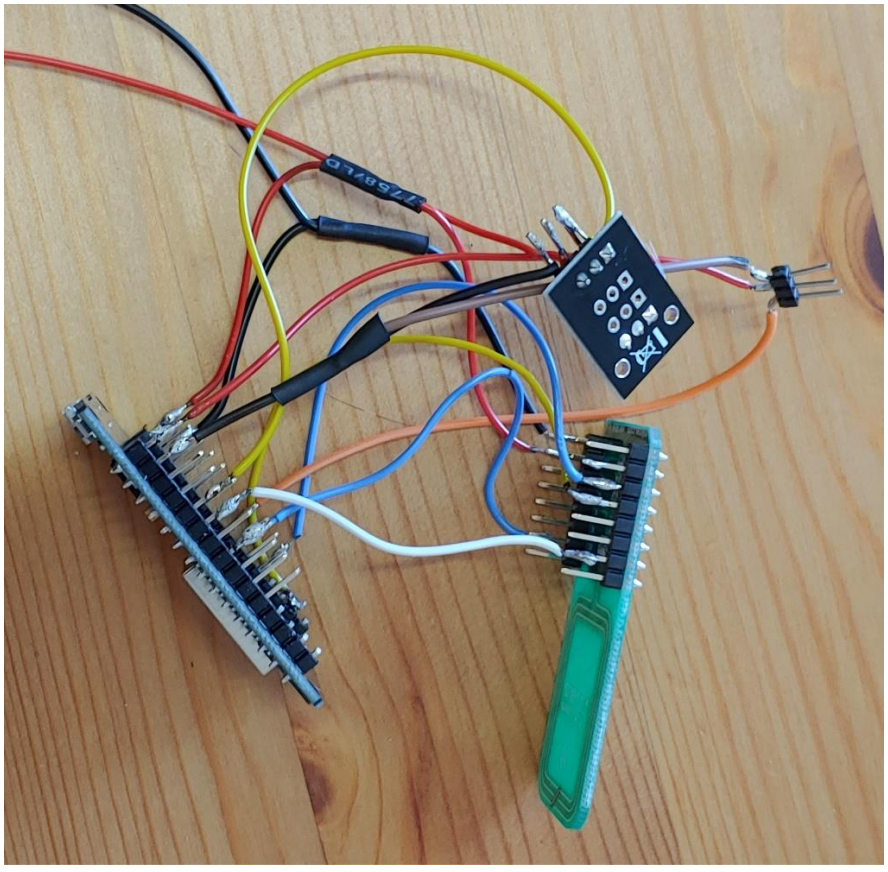

Abbildung 9: Foto der Verkabelung

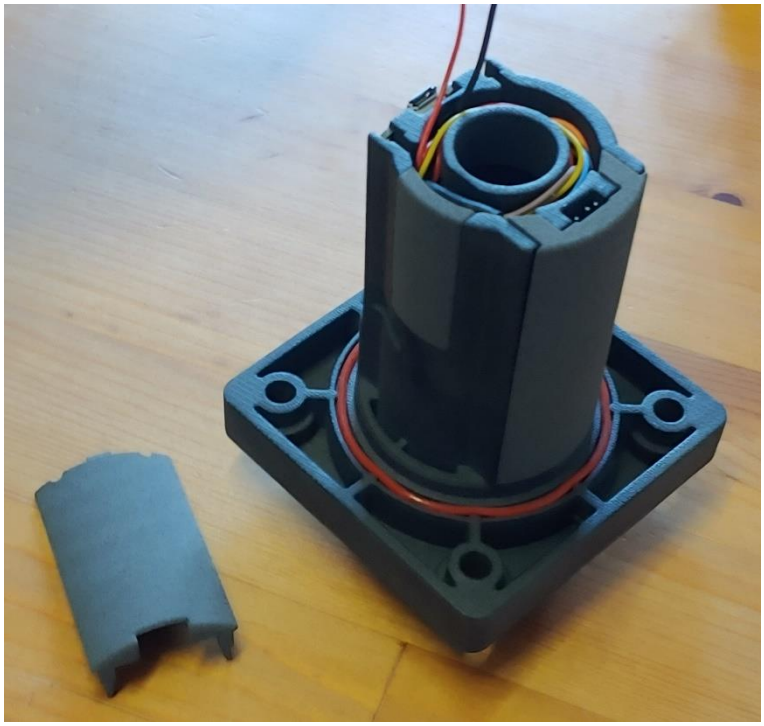

Abbildung 10: Foto der Isolation eines Prototypes inklusive Verkabelung und bereits montierter Abdeckungen

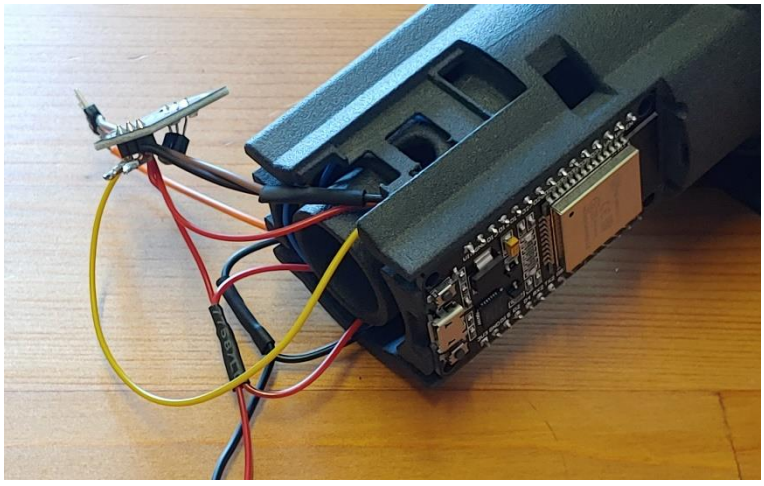

Abbildung 11: Einbau der verlöteten Module in die Isolation

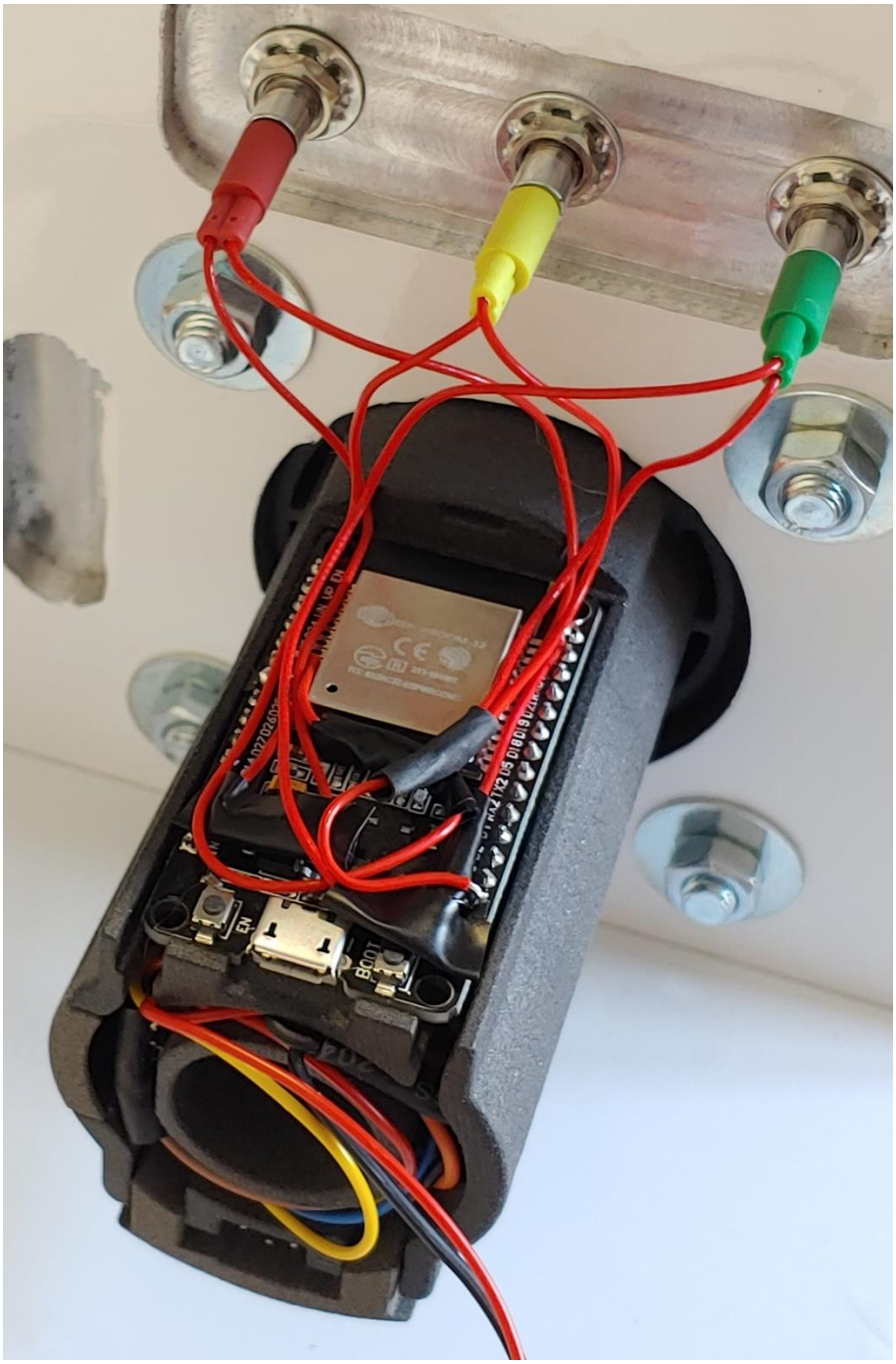

Abbildung 12: Montage der LEDs

5.2 Demonstrator

Unverzüglich als ich die Teile aus Aluminium für das Gehäuse erhalten hatte, begann ich mit der Montage des Demonstrators. Das Gehäuse des Demonstrators besteht aus 5 Platten, bei welchen ich zunächst jeweils die Aussenseite mit 400er-Schleifpapier abgeschliffen habe, damit später keine Beschriftungen oder Bearbeitungsspuren unter der Folierung sichtbar sind.

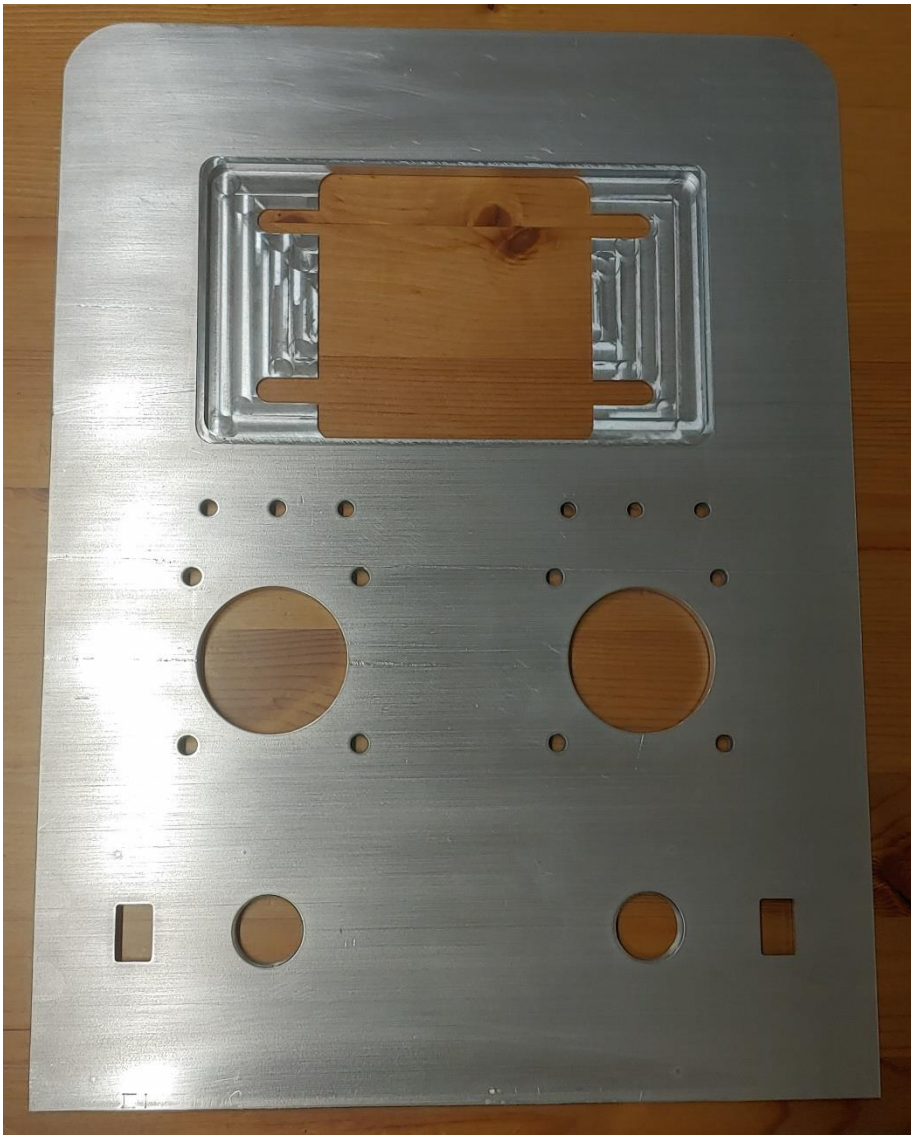

Abbildung 13: Ansicht des nachbearbeiteten Frontpanels

Die Aluminiumplatten konnten, dank präziser Gewindebohrungen, einfach mittels handelsüblicher Metallwinkeln miteinander verschraubt werden. Zunächst habe ich alle Teile mit Aceton gereinigt. Danach habe ich das Frontpanel mit einer Folie, auf welcher das Stäubli-Logo sowie der Slogan FAST MOVING TECHNOLOGY aufgedruckt ist, beklebt. Weiter habe ich die restlichen sichtbaren Seiten des Gehäuses mit einer weissen Folie beklebt.

Abbildung 14: Rückseite des montierten Gehäuses

Abbildung 15: Vorderseite des Gehäuses inklusive Display

Als das Grundgerüst stand, konnte ich mit dem Innenleben und der eigentlichen Funktion des Demonstrators beginnen. Die Stromversorgung geschieht dabei über eine normale Steckdose und ist zusätzlich mit einem Stecker mit eingebautem FI-Schutzschalter abgesichert. Als erstes schloss ich das Stromkabel mit besagtem Stecker an den Hauptschalter, welcher zusätzlich im eingeschalteten Zustand noch grün beleuchtet ist, an. Im gleichen Schritt habe ich das Erdungskabel und eine Verteilerbox am Gehäuse angeschraubt. Danach folgte die 5 Volt Stromversorgung mittels eines normalen USB-Netzteiles und einem USB-Hub in der oberen Platte. Danach folgte der erste Stromkreislauf, welcher durch den Prototyp mittels WiFi-Switch und einem zusätzlichem Handschalter aktiviert werden kann. Daran angeschlossen ist die farblich gekennzeichnete LED-Birne sowie das Relais, welches die Stromerkennung simuliert. Nur bei diesem Kreislauf wurde zusätzlich eine Heizpatrone inklusive eines dazugehörigen Schalters, womit eine mögliche Erwärmung und somit die Notabschaltung durch den Prototyp simuliert werden kann, verbaut. Der zweite Stromkreislauf ist bis auf die Heizpatrone identisch aufgebaut.

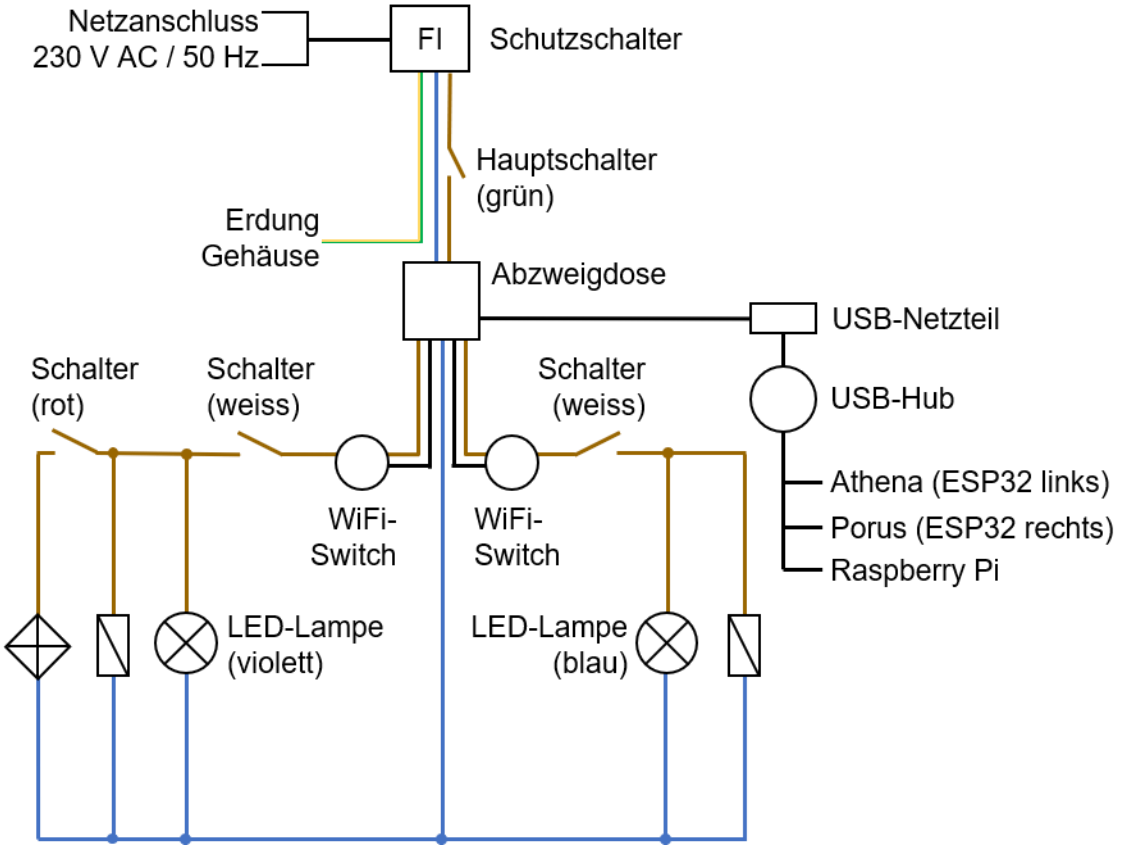

Abbildung 16: Schema zur Verkabelung des Demonstrators

Aus sicherheitstechnischen Gründen wurde schlussendlich kein Stromanschluss über den 16BL angeschlossen. Dank der Lösung mit der Simulation des Stromflusses mittels Relais ist dieser auch nicht mehr zwingend nötig. Ohne Spannung an diesen Teilen, welche potenziell berührt werden könnten, wird nicht mehr zwingend eine Abdeckung bzw. ein Berührschutz benötigt.

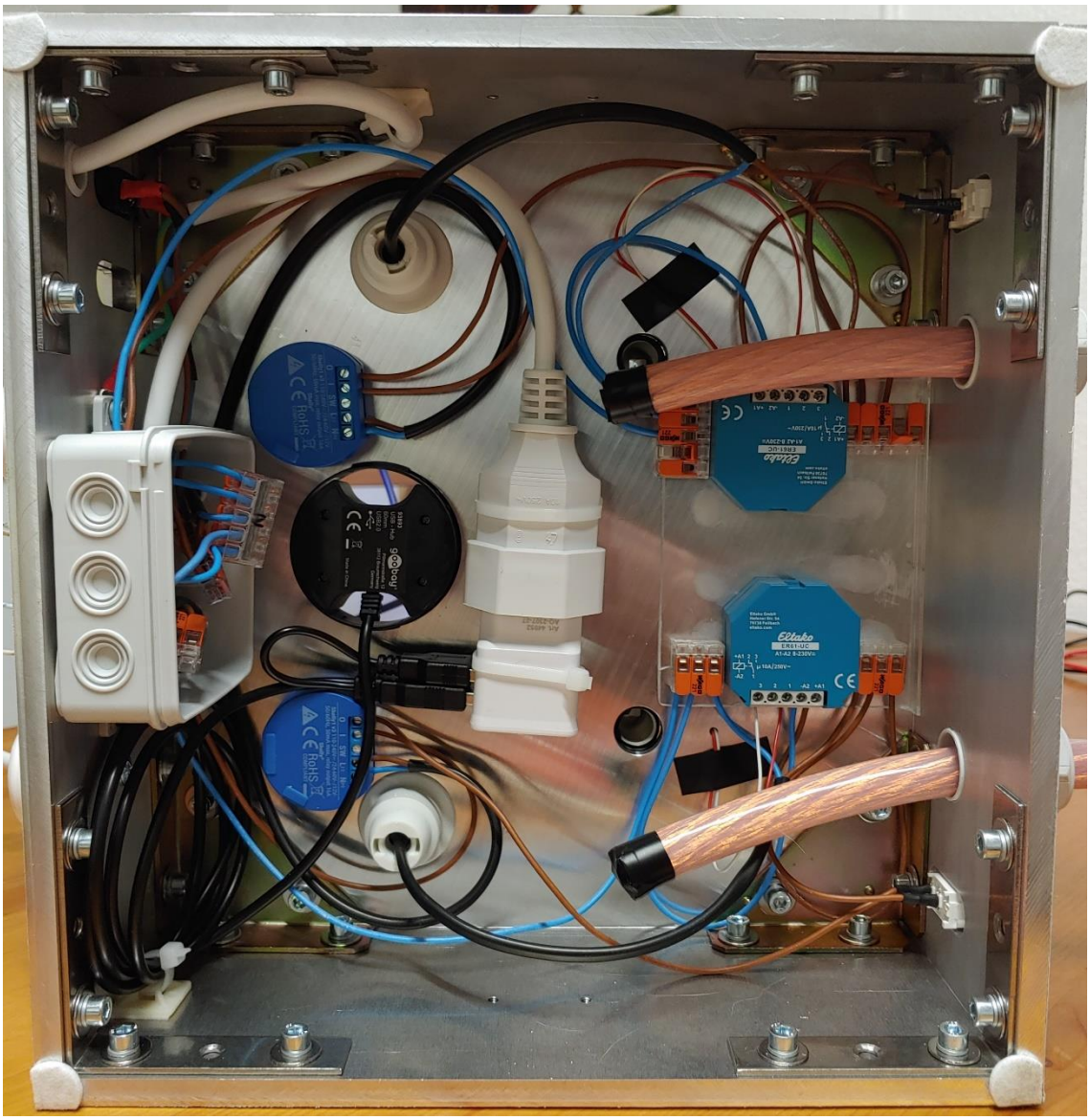

Abbildung 17: Foto der Verkabelung des Demonstrators (noch ohne Heizpatrone)

6 Finales Produkt

Abbildung 18: Foto des eingeschalteten Demonstrators von hinten

Abbildung 19: Foto der Frontansicht des eingeschalteten Demonstrators

6.1 Namensgebung

Für die Vermarktung eines neuen Produktes ist es hilfreich, wenn damit eine Geschichte erzählt werden kann, welche gleichzeitig dem Kunden die Stärken und Vorteile aufzeigt. Dies sollte bei diesem Projekt über den Namen METIS geschehen.

Das Akronym METIS steht, wie auf dem Titelblatt ersichtlich, für *«Monitoring of Electrical and Thermal Interchange» - Solution*, was zu Deutsch so viel heisst wie *Lösung zur Überwachung des elektrischen und thermischen Austauschs*.

Metis heisst im altgriechischen ausserdem *"kluger Rat"* und Metis war in der griechischen Mythologie die Titanin des klugen Ratschlags, der Beratung, der Planung, der Gerissenheit, der List und der Weisheit. Sie war die erste Geliebte von Zeus. Daraus kann man ableiten, dass sie immer den "Strom" im Blick hatte. Aus dieser Herleitung resultieren auch die Namen der beiden Steckverbindungen Athena und Porus, welches die Kinder von Metis und Zeus waren.

Ferner spiegelt sich der Bezug zu Griechenland auch im Namen des WiFi-Hotspots wider. Dieser heisst Edipsos, wobei es sich um eine Gemeinde in Mittelgriechenland handelt, welche berühmt ist für ihre heissen Thermalquellen.

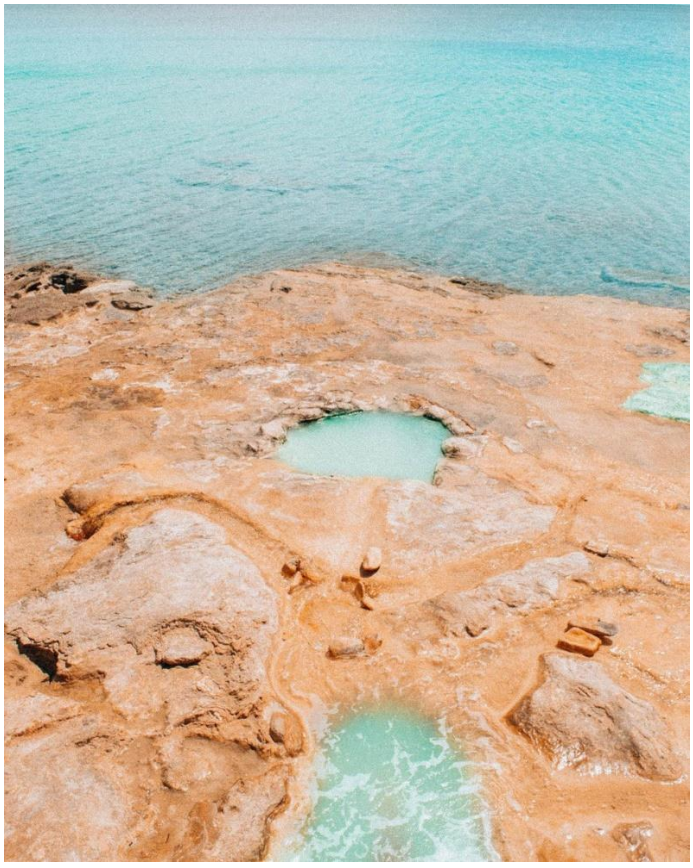

Abbildung 20: Bild der "Hot Spots" in Edipsos

6.2 Funktionsbeschreibung

Zunächst muss man den Demonstrator einschalten. Dazu steckt man den angebrachten Stecker in eine Steckdose und aktiviert durch Drücken auf den grünen RESET-Knopf den Schutzschalter. Falls man sichergehen will, dass dieser korrekt funktioniert, kann der blaue Test-Knopf gedrückt werden. Nur wenn ein Klicken hörbar ist, funktioniert der Schutzschalter. Danach kann der Hauptschalter an der Rückseite betätigt werden, womit das Dashboard sowie die beiden "intelligenten" Steckverbindungen Athena (violett) und Porus (blau) hochfahren. Erkennbar ist dies durch einen farbigen Splashscreen auf dem Display sowie das animierte Aufleuchten der LEDs. Es kann sein, dass zuerst noch der Status ‘Offline’ angezeigt wird, dieser sollte jedoch bald durch ‘Ready’, also das Anzeigen der Steckbereitschaft ersetzt werden. Gleichzeitig wird dieser Zustand durch die grüne LED angezeigt.

Dann kann ein Steckvorgang durchgeführt werden. Unabhängig davon, ob das richtige oder ein falsches Gegenstück verbunden wurde, wird die Anzahl der Steckzykeln um einen Zyklus erhöht. Ob die Verbindung korrekt ist oder nicht, wird durch die gelbe LED angezeigt. Ein Blinken der gelben LED bzw. der Status ‘Not Connected’ signalisiert, dass ein falsches Gegenstück vorhanden ist und das Einschalten des Stromes unterbunden ist. Dies wird zusätzlich als Warnhinweis auf dem Dashboard angezeigt. Leuchtet diese hingegen konstant bzw. wird der Status ‘Connected’ angezeigt, bedeutet dies, dass das richtige Gegenstück verwendet und erkannt wurde. Erst dann kann der Strom eingeschaltet werden.

Wenn der Strom eingeschaltet wird, leuchtet die rote LED auf, der Status ‘Current’ wird auf dem Dashboard angezeigt und der Verriegelungsmechanismus wird geschlossen. Falls sich die Temperatur über den Grenzwert bewegen sollte, beginnt die rote LED zu blinken, der Status wechselt auf ‘Overheat’ und eine entsprechende Warnung wird angezeigt. Falls die Temperatur weiter steigen sollte und das Limit überschreitet, wird der Stromfluss unterbunden und so eine weitere Erwärmung verhindert. Die rote LED blinkt weiterhin und der Status zeigt ‘Cutout’ an. Da dann kein Strom mehr fliesst wird ebenfalls der Verriegelungsmechanismus geöffnet. Erst wenn der festgelegte Schwellenwert unterschritten wurde, wird der Stromfluss wieder aktiviert. Die Temperatur wird in diesem Status die ganze Zeit angezeigt. Wichtig ist, falls die Erwärmung mittels Heizpatrone simuliert wurde, dort auch wieder den Schalter umzuschalten.

Im Normalfall sollten die letzten beiden Szenarien jedoch nicht eintreffen. Dann kann einfach der Strom ausgeschalten werden, wodurch sich der Verriegelungsmechanismus öffnet und erneut die gelbe LED aufleuchtet. Danach kann die Verbindung getrennt werden. Erneut leuchtet die grüne LED und die Anzeige springt wieder auf den Ausgangsstatus, welcher die Steckbereitschaft signalisiert, zurück.

Wenn damit die Demonstration beendet wurde, kann der Raspberry Pi korrekt heruntergefahren, der grüne Hauptschalter ausgeschalten und der Stecker ausgezogen werden. Dabei kann der FI-Schalter klicken, was normal ist.

6.3 Stückliste und Kosten

In folgender Tabelle sind alle im finalen Produkt verwendeten Komponente, ausser Teile, welche intern beschafft werden konnten oder bereits vorhanden waren, inklusive deren Preise festgehalten:

Objekt	Ort	Stück zahl	Kosten Fr.
LED-Leuchte G45 mit E14 Sockel	Tintenmax	2	7.80
MIKROE-2395 NFC click	Distrelec	3	79.65
Pi Foundation Display 7"	Adafruit	1	78.49
13.56MHz RFID/NFC Sticker - Classic 1K	Adafruit	3	6.63
NodeMCU ESP32	reichelt	3	32.63
Temperatursensor DS18B20	reichelt	3	8.74
Shelly 1 WiFi-Schaltaktor	reichelt	2	18.00
Wippschalter, 1x Aus, beleuchtet	reichelt	2	5.44
Wippschalter, 1x Aus, weiss	reichelt	2	2.23
USB 2.0 Kabel, A Stecker, Micro B Stecker	reichelt	2	2.38
Desk HUB 3-Port, 60mm, schwarz	reichelt	1	7.15
Heizpatrone hotrod®, Ø6,5x40mm, 200W	reichelt	1	20.13
Widerstand, Metallschicht 68,0 Ohm	reichelt	4	0.16
Widerstand, Metallschicht 15,0 Ohm	reichelt	2	0.08
LED GRN / RED / YEL 6MM NUT	Digi-Key	6	38.10
Indoor-Aufkleber (rechteckig) mit Motiv	wir-machen-druck	1	10.40
HAMA USB3.2 C/A M/M	Media Markt	1	16.95

Objekt	Ort	Stückzahl	Kosten Fr.
Abzweigdose IP54 85X85X40	Jumbo	1	4.25
WAGO KLEMME	Jumbo	4	6.75
Kabelverschraubung M16	Jumbo	1	1.48
MOEBELKNOPF WEISS/RUND	Jumbo	2	10.50
USB LADEGERAET 2XUSB A	Jumbo	1	15.95
FI-Schutzschalter	OBI	1	27.95
Klebefolie	OBI	1	5.50
Eckplatte verzinkt	OBI	4	15.80
Stuhlwinkel	OBI	4	11.40
Klebesockel	OBI	5	1.15
Kabelbinder weiss	OBI	10	1.60

Total	**CHF 437.29**

7 Fazit

7.1 Reflexion

Im Pflichtenheft wurden folgende 11 Ziele festgelegt, wovon beinahe jedes erfüllt werden könnte. Ersichtlich wird dies in folgender Tabelle, in welcher die erfüllten Ziele grün hinterlegt sind:

Nr.	Ziel
1	Eine Steckung wird erkannt und das Gegenstück mittels NFC identifiziert.
2	Die Steckzyklen werden gezählt und abgespeichert.
3	Im gesteckten Zustand wird die Temperatur der Steckverbindung fortlaufend gemessen.
4	Die Kosten überschreiten das budgetierte obere Kostendach von 800 Fr. nicht.
5	Die Steckverbindung erkennt, ob ein Stromfluss vorhanden ist oder nicht.
6	Auf einem ungebundenen Dashboard (Bildschirm) sind Informationen ersichtlich.
7	Durch Leuchten und Blinken dreier LEDs (grün, gelb, rot) wird der Status signalisiert.
8	Bei einem zu definierenden Temperatur-Limit wird ein Warnhinweis angezeigt.
9	Bei Stromfluss wird die Steckverbindung gegen ungewolltes Ausstecken gesichert.
10	Bei einer zu definierenden Maximal-Temperatur wird der Strom ausgeschaltet.
11	Die Status Informationen der Steckverbindungen sind auf einer Homepage abrufbar.

Lediglich das Ziel Nr. 5, die Stromerkennung, konnte nur teilweise erfüllt werden und ist somit orange markiert. Dass trotzdem eine Simulation des Stromflusses umgesetzt werden konnte, erachte ich als einen Erfolg.

Die praktische Umsetzung kann ebenfalls als Erfolg betrachtet werden. So konnten alle Schritte wie geplant umgesetzt werden und haben am Ende zum gewünschten Ergebnis geführt. Der Weg zu diesem Ziel war jedoch kein leichter. So bin ich zum Beispiel an einigen Stellen angeeckt und wusste zunächst nicht weiter. Durch Ausprobieren, Recherchieren und auch durch firmeninterne Unterstützung fand ich jedoch immer einen Weg oder eine andere Option, die noch ausprobiert werden konnte, was schlussendlich zu einer Lösung geführt hat.

Neben der grossen Herausforderung der eigentlichen Aufgabe spielte auch der Zeitdruck eine grosse Rolle. Durch Verzögerungen bei internen Lieferungen oder aufgrund von Nachbestellungen kam ein enormer Druck auf. Dieser führte vor allem dazu, dass die Zeit für die Dokumentation extrem knapp wurde.

7.2 Lessons learnt

Bei der Umsetzung dieser Diplomarbeit habe ich wieder einmal realisiert, dass eine gute, offene, ehrliche und direkte Kommunikation der Schlüssel zum Erfolg ist. So hätte ich mir zum Beispiel einige Zeit und vor allem Nerven gespart, wenn ich so früh wie möglich die direkte Kommunikation mit dem Leiter der Werkstatt gesucht hätte. Aber auch positive Beispiele traten auf. Intern wurde ich von mehreren Kollegen regelmässig gefragt, wie der Stand ist und mir wurde mehrfach Unterstützung angeboten, welche ich auch in Anspruch genommen habe. Die Ratschläge, Tipps und Lösungsansätze waren wertvolle Leitplanken und haben mich vor unüberlegten Fehlschlüssen bewahrt.

Der technische Lerninhalt war immens. Die Aufgabenstellung hatte konstruktive, IT-spezifische, sowie elektrotechnische Anforderungen – und bei allen stiess ich an meine Grenzen… und darüber hinaus! Im Nachhinein betrachtet, ging der Umfang dieser Diplomarbeit wohl etwas über den Tellerrand des Elektrotechnikers hinaus.

7.3 Schlusswort

Ich möchte mich hiermit bei allen Personen, welche mich in irgendeiner Weise bei der Vollendung dieses Projektes unterstütz haben mit Nachdruck bedanken.

Firmenintern geht grosser Dank an die Abteilung Product Management, die gesamte IT, das Test Department, das Team der Global Communications, die R&D, das Purchasing Department, die gesamte Werkstatt und zu guter Letzt mein Team vom PTD. Besonderer Dank geht dabei an meinen direkten Vorgesetzten Andreas Linder – ohne ihn wäre das Thema dieser Arbeit gar nicht erst zustande gekommen. Durch ihn wurden mir viele Freiheiten bei der Umsetzung des Projektes gewährt. Sein Enthusiasmus dem Thema gegenüber hat mich die ganze Zeit begleitet und ist auch einer der Punkte, die mich zum Erfolg geführt haben.

Weiter bedanke ich mich bei allen Freunden und Familienmitglieder für die Unterstützung während dieser intensiven Zeit. Auch dem Rückhalt durch alle Klassenkammeraden gebührt ein grosses Dankeschön. Insbesondere danke ich meinem Dozenten Silvan Wirth, für die offene Kommunikation sowie die technische Unterstützung.

Danke!

Anhänge

Programmcode

Der ganze Quellcode zu den beiden Scripten, inklusive englischer Kommentare, sowie die beim Dashboard eingesetzten HTML-Dateien sind zu finden unter:

https://github.com/BadPrankster/METIS

Quellenangaben

[1] Titelbild: Rodolf Gorrin, via Pinterest: www.pinterest.com
Abgerufen am 24.09.2022

[2] Stäubli Electrical Connectors AG: www.staubli.com/electrical
Abgerufen im Zeitraum August bis September 2022

[3] Random Nerd Tutorials: randomnerdtutorials.com
Abgerufen im Zeitraum August bis September 2022

[4] Wikipedia: www.wikipedia.org
Abgerufen am 16.08.2022

[5] QR-Code: www.qrcode-generator.ch
Abgerufen am 24.09.2022

Eigenständigkeitserklärung

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und erlaubten Hilfsmittel benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass anderenfalls die Arbeit mit der Note 1.0 bewertet wird.

Ort, Datum:	Signatur des Studierenden:
Basel, 25.09.2022