



Hochschule Karlsruhe  
Technik und Wirtschaft  
UNIVERSITY OF APPLIED SCIENCES



# Entwicklungsprojekt MYiTOPS

Ansteuerung und Auswertung einer Motorsäge via Remote Control über das Internet

von

Fabian Harlacher	hafa1023@hs-karlsruhe.de	57063
Danial Haris Bin Limi Hawari	lida1017@hs-karlsruhe.de	66089
Nina Kurasch	kuni1018@hs-karlsruhe.de	57727
Aliaa Nabila Abdul Mutaali	abal1015@hs-karlsruhe.de	66091
Munirah Nawi	namu1011@hs-karlsruhe.de	66090
René Schulreich	scre1019@hs-karlsruhe.de	57061

Betreuer: Prof. Dr. Maurice Kettner

Dipl.-Phys. Ferhat Aslan

Koordinator: Prof. Dr. Peter Weber

Zeitraum: Sommersemester 2019

Projekt-Code: 19SS\_KE\_MYiTOPS

---

## Eidesstattliche Erklärung

Wir versichern hiermit, die vorliegende Projektarbeit ohne unzulässige Hilfe selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Karlsruhe, den 29.09.2019



Fabian Harlacher



Danial Haris Bin Limi Hawari



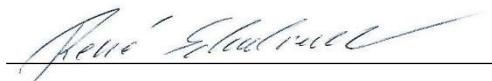
Nina Kurasch



Aliaa Nabila Abdul Mutaali



Munirah Nawi



René Schulreich

---

# Inhaltsverzeichnis

Inhaltsverzeichnis .....	3
1. Einleitung .....	7
2. Projektbeschreibung .....	7
3. Hintergrund .....	7
4. Aufgabenstellung .....	8
5. Projektgruppe .....	8
5.1 Untergruppe 1: „Team mechanical“ .....	9
5.2 Untergruppe 2: „Team LabVIEW“ .....	9
5.3 Untergruppe 3: „Team Python“ .....	9
6. Zeitpläne .....	10
7. System .....	12
7.1 Datenaustausch .....	12
7.2 Mechanischer Aufbau .....	12
8. Hardware .....	13
8.1 Motor .....	13
8.2 Aktorik .....	14
8.3 Relaismodul .....	15
8.4 Not-Aus .....	16
8.5 Befestigung .....	16
9. Drehzahlerfassung .....	17
10. User Interface .....	27
11. LabVIEW .....	28
12. Blockschaltbild .....	30
13. Datentransfer .....	34
14. Datenstruktur .....	34
15. Verwendete Software .....	36
15.1 Datenbank Connector .....	36
15.2 LabVIEW Toolkit .....	38
16. Anforderungen an MYiTOPS-Skript .....	38
17. Microcontroller .....	39
17.1 Betriebssystem .....	40
17.2 Programmiersprache .....	40

---

17.3	Installation.....	42
18.	Programmablauf.....	43
19.	Herausforderungen .....	46
19.1	Datenbankverbindung.....	46
19.2	Frequenzzähler.....	47
19.3	PWM .....	48
20.	Quellen .....	50
20.1	Textquellen.....	50
20.2	Bildquellen.....	51
21.	Anhang .....	52
	Anhang 1 – Programm.....	52
	Anhang 2 - Schaltplan.....	65
	Anhang 3 -Gesprächsprotokolle .....	66

---

## Abbildungsverzeichnis

Abbildung 1: Bisherige Fernsteuerung .....	7
Abbildung 2: Gruppenfoto .....	8
Abbildung 3: Projektzeitplan .....	10
Abbildung 4: Untergruppenzeitplan.....	11
Abbildung 5: Darstellung des Datenaustauschs .....	12
Abbildung 6: Aufbau des Systems .....	12
Abbildung 7: Die Motorsäge.....	13
Abbildung 8: Chokestellungen .....	14
Abbildung 9: Ansteuerung eines Servomotors.....	14
Abbildung 10: Zeichnung des Gehäuses.....	15
Abbildung 11: Alte Platte .....	16
Abbildung 12: Befestigung der Einschlagmuttern.....	17
Abbildung 13: Primärseitiges Zündsignal .....	18
Abbildung 14: Schmitt Trigger .....	20
Abbildung 15: Simulation der Schaltung .....	21
Abbildung 16: Verlauf der Signale .....	22
Abbildung 17: Pinanschlüsse des CMOS 4093.....	23
Abbildung 18: Pinanschlüsse des CMOS 4013.....	23
Abbildung 19: Signal vor dem Schmitt Trigger .....	24
Abbildung 20: Signal nach dem Schmitt Trigger .....	24
Abbildung 21: Frequenzteilung .....	25
Abbildung 22: Aufbereitung des Zündsignals.....	25
Abbildung 23: Funktionalitätsprüfung der Platine .....	26
Abbildung 24: Aktuelle Benutzeroberfläche .....	28
Abbildung 25: Beispielhafte Darstellung der LabVIEW Benutzeroberfläche .....	29
Abbildung 26: Ausschnitt des aktuellen Blockschaltbilds .....	30
Abbildung 27: Beginn des Blockschaltbilds .....	31
Abbildung 28: Verarbeiten der Datenströme.....	32
Abbildung 29: Beispielhafte Darstellung einer Case-Funktion.....	32
Abbildung 30: Ende des Blockschaltbilds .....	33
Abbildung 31: Darstellung des Shift Register .....	33
Abbildung 32: Schematische Darstellung des Datentransfers .....	34
Abbildung 33: Datenbankentwicklung über die Webseite .....	35
Abbildung 34: Beispiel der Datenbankkonventionen.....	35
Abbildung 35: Administratoreinstellungen unter Windows .....	36
Abbildung 36: Auswahl des ODBC Treibers.....	37
Abbildung 37: Parametereinstellungen des Treibers.....	37
Abbildung 38: Funktionspalette mit zusätzlich eingebundenen Funktionsbausteinen .....	38
Abbildung 39: Raspberry Pi 3B+ .....	40
Abbildung 40: Allgemeine Eigenschaften von Python .....	41
Abbildung 41: Ablaufdiagramm .....	44
Abbildung 42: GPIOs.....	45
Abbildung 43: Multithreading in Python.....	46

---

---

Abbildung 44: Beispiel Thread.....	47
Abbildung 45: Frequenzmesssignal.....	47
Abbildung 46: run- und Interrupt-Methode des Frequenzthreads.....	48
Abbildung 47: Pulsweitenmodulation.....	49
Abbildung 48: Klasse Servo mit den zwei wichtigsten Methoden.....	49

## 1. Einleitung

MYiTOPS ist ein bilaterales Forschungsprojekt der Hochschule Karlsruhe – Technik und Wirtschaft und der Universiti Malaysia Pahang. Dieses auf drei Jahre angesetzte Projekt, das jeweils im Sommersemester in Deutschland und in Malaysia stattfindet, wird in gemischten Teams bearbeitet. Die einzelnen Entwicklungsprojekte befassen sich mit physikalischen Systemen, die über das Internet vernetzt sind.

Gefördert wird MYiTOPS von der BWS plus, ein Programm der Baden-Württemberg Stiftung.

## 2. Projektbeschreibung

Dieses Entwicklungsprojekt bildet den Startschuss von MYiTOPS. Es wurde ein System entwickelt, dass die Ansteuerung und Auswertung einer benzinbetriebenen Motorsäge via Fernsteuerung über das Internet ermöglicht. Der Benutzer steuert dazu die Aktorik zum Betrieb der Motorsäge über ein User Interface am Computer, welches über das Internet mit dem Raspberry Pi kommuniziert. Der Raspberry Pi bildet die zentrale Steuereinheit dieses Systems. Über eine selbst entwickelte Sensorik wird die Drehzahl der Säge ermittelt.

## 3. Hintergrund

Grundlage dieses Projektes bildet eine vorhergegangene Bachelorarbeit, bei der ein System zum Steuern verschiedener handgeföhrter Verbrennungsmotoren entwickelt wurde. Dafür wurde unter anderem eine Aktorik entwickelt, die es ermöglicht, den jeweiligen Motor unter Prüfstandbedingungen in der Kälte- und Höhensimulationskammer am Campus Bruchsal zu bedienen und zu testen. Die zentrale Steuereinheit ist ein Arduino Mega 250. Die Schnittstelle zwischen dem Bediener und dem Motor auf dem Prüfstand ist eine physikalische Steuerung in Form eines Bedienpanels, auf dem mehrere Schalter angeordnet sind (s. Abb.1).



Abbildung 1: Bisherige Fernsteuerung

## 4. Aufgabenstellung

Via Remote Control über das Internet soll die vorhandene Aktorik zum Betrieb eines Motors angesteuert werden. Dafür soll die Hardware auf einen Raspberry Pi anstelle des Arduinos umgerüstet werden. Die Aktorik soll über diesen Raspberry Pi angesteuert werden. Außerdem soll für die Bedienung ein User Interface in LabVIEW erstellt werden. Das Gesamtsystem soll auf einer Modellplatte angeordnet und getestet werden. Dazu ist es notwendig, den Motor mit einer Sensorik zu überwachen, um ein regelmäßiges Feedback zu bekommen. Im Laufe des Projektes wurde konkret gefordert, eine Sensorik zu entwickeln, die die aktuelle Drehzahl erfasst. Außerdem soll vom Bediener eine bestimmte Drehzahl vorgegeben werden können.

## 5. Projektgruppe



Abbildung 2: Gruppenfoto

Vlnr: Aliaa Nabila Abdul Mutaali, Prof. Dr.-Ing. Maurice Kettner, Nina Kurasch, Fabian Harlacher, René Schulreich, Munirah Nawi, Dipl.-Phys. Ferhat Aslan, Danial Haris Bin Limi Hawari

---

Für eine koordinierte und effektive Vorgehensweise wurde die Projektgruppe in drei Untergruppen aufgeteilt.

### **5.1 Untergruppe 1: „Team mechanical“**

Aufgabe von Aliaa Nabila Abdul Mutaali und Nina Kurasch war es, eine Sensorik zur Überwachung des Motors zu entwickeln, die es gleichzeitig ermöglicht, die aktuelle Motordrehzahl zu erfassen. Der Aufbau des Systems auf der Modellplatte, sowie die Auswahl der dafür benötigten Hardware und deren Bestellung war ebenfalls Aufgabe dieser Untergruppe.

### **5.2 Untergruppe 2: „Team LabVIEW“**

Für Munirah Nawi und René Schulreich galt es, das User Interface zu entwickeln, mit dem die Aktorik gesteuert und Feedbackinformationen empfangen werden können. Außerdem bestand ihre Aufgabe darin, ein Blockschaltbild zu erstellen, ein Lösungskonzept für den Datenaustausch zwischen dem Raspberry Pi und der LabVIEW Bedienoberfläche auszuarbeiten sowie eine Datenbankstruktur zu entwerfen.

### **5.3 Untergruppe 3: „Team Python“**

Fabian Harlacher und Danial Haris Bin Limi Hawari erstellten eine Ablaufsteuerung für die Aktoren sowie ein Automatisierungssystem und eine Datenaustauschlösung über die Cloud. Außerdem erstellten sie ein Skript, das eine Motorfrequenzmessung durchführen kann und welches effizient mit weniger Debugging arbeitet, benutzerfreundlich, einfach und schnell ist.

## 6. Zeitpläne

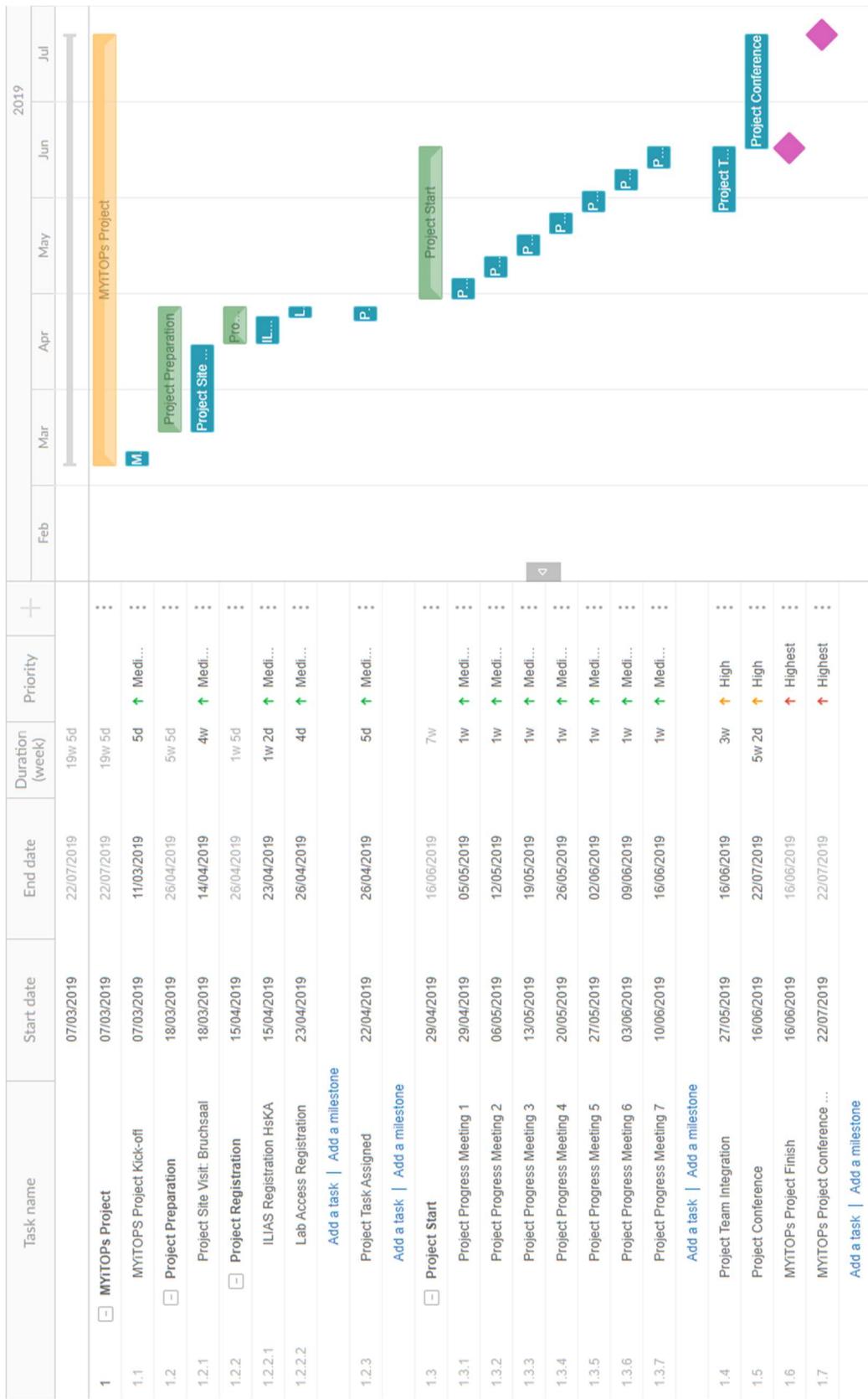


Abbildung 3: Projektzeitplan



Abbildung 4: Untergruppenzeitplan

## 7. System

### 7.1 Datenaustausch

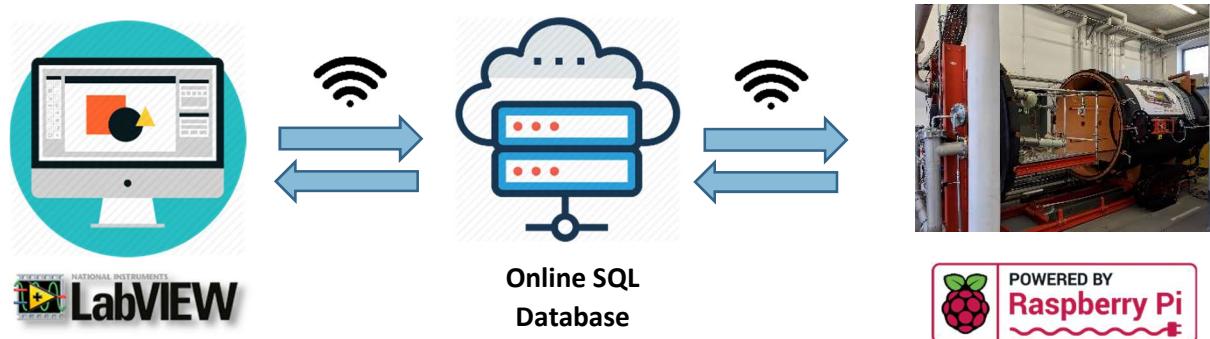


Abbildung 5: Darstellung des Datenaustauschs

In der Grafik von Abbildung 5 ist der Datenaustausch dargestellt. Dieser wird mittels einer externen Datenbank realisiert. Der Benutzer sendet alle relevanten Informationen in definierten Tabellen in dieser Datenbank. Feedbackdaten werden vom Steuergerät aus an den Server gesendet. Durch regelmäßige Anfragen an die Datenbank entsteht zwischen dem Steuergerät und der Benutzeroberfläche ein konstanter Datenaustausch zwischen Input- und Feedbackinformationen.

### 7.2 Mechanischer Aufbau

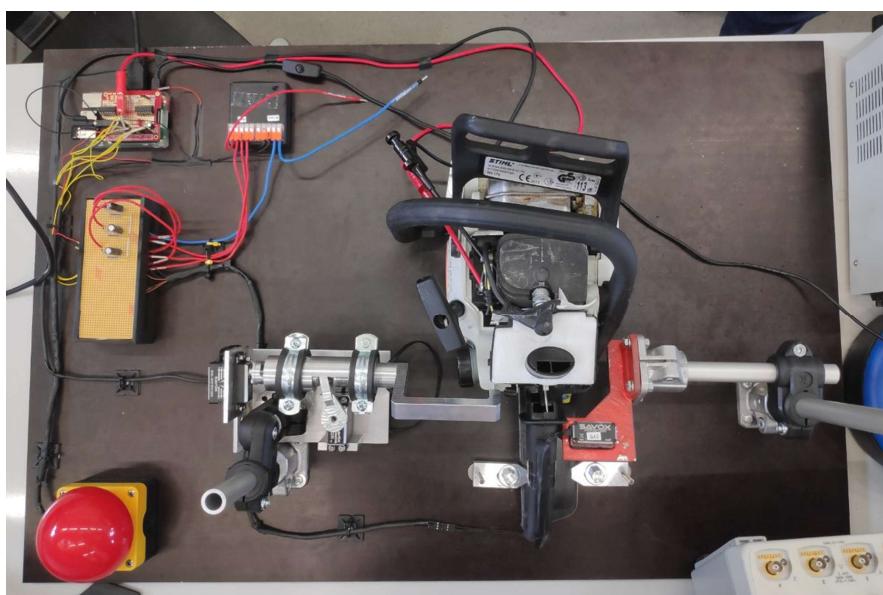


Abbildung 6: Aufbau des Systems

---

Abbildung 6 zeigt die Anordnung des mechanischen Systems auf der Modellplatte. Zentral die Motorsäge, rechts davon der Aktuator zur Ansteuerung der Gasstellung, links davon die Aktorik zum Auswählen der Chokehebelstellung. Am linken unteren Rand befindet sich der Not-Aus Buzzer, darüber das Relaismodul in einem 3D gedruckten Gehäuse und drei Elektrolytkondensatoren zur Stabilisierung der Versorgungsspannung der Servomotoren. Am linken oberen Rand ist der Raspberry Pi platziert, daneben befindet sich die zentrale Spannungsversorgung.

## 8. Hardware

### 8.1 Motor

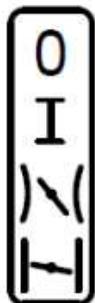
Zu Beginn der Projektarbeit einigen wir uns darauf, die Fernsteuerung auf einen Motor zu beschränken. Wir entscheiden uns für die Motorsäge. Zur Verfügung stand uns die Stihl Motorsäge MS170 (s. Abb. 7). Sie ist circa 4 kg schwer. Mit ihrem Einzylinder – Zweitaktmotor und einen Hubraum von  $30,1 \text{ cm}^3$  beträgt die Leistung 1,2 kW bzw. 1,6 PS. Die Leerlaufdrehzahl ist  $2800 \frac{1}{\text{min}}$ . Als Zündanlage wird ein elektronisch gesteuerten Magnetzünder verwendet. <sup>[1]</sup>

Die zur Durchführung unserer Projektarbeit wichtigen Komponenten sind Folgende: das Starterseil, der Chokehebel, der Gashebel bzw. die Gashebelsperre und die Kettenbremse.



Das Starterseil befindet sich auf der linken Seite der Säge. In dem bestehenden System wird der Motor mithilfe eines Pneumatikzylinders gestartet, der über ein Seilzug mit dem Starterseil verbunden ist. In dieser Projektarbeit wurde auf die Einbindung des Pneumatikzylinders verzichtet, die Motorsäge wird händisch gestartet.

Am linken Rücken ist der Chokehebel. Je nach Stellung wird die in den Vergaser einströmende Luft begrenzt. Es können vier verschiedene Stellungen (s. Abb. 8) ausgewählt werden (von oben):



Stop: Der Motor ist aus, die Zündung ist ausgeschaltet

Betrieb: Der Motor läuft oder kann starten

Warmstart: In dieser Stellung wird ein warmer Motor gestartet, der Motor sollte mindestens eine Minute gelaufen sein

Kaltstart: So wird der kalte Motor gestartet

Abbildung 8: Chokestellungen

Am hinteren Handgriff befindet sich auf der Unterseite der Gashebel, auf der Oberseite die Gashebelsperre. Gashebel und Gashebelsperre müssen aus Sicherheitsgründen gleichzeitig betätigt werden. Da der Bereich zu Beginn der Betätigung des Gashebels zu keiner Veränderung der Drehzahl führt, ist die Spannweite von unterschiedlichen Gasstellungen nicht sehr groß. Eine minimale Veränderung der Drehzahl ist daher nicht möglich.

Die Kettenbremse sollte im Stillstand, zum Transport und im Leerlauf betätigt sein. Sie sorgt dafür, dass die Sägekette blockiert wird. Bei der für die Projektarbeit zur Verfügung gestellte Motorsäge wurde die Führungsschiene, auf der die Sägekette läuft, demontiert. Trotzdem ist es wichtig, die Kettenbremse bei Motordrehzahl zu lösen, indem man den Handschutz zum Griffrohr zieht, da sonst Schäden am Triebwert und am Kettenantrieb entstehen.

## 8.2 Aktorik

Die bei der vorhergegangenen Bachelorarbeit entwickelte Aktorik kam bei dieser Projektarbeit zum Einsatz. Das ist zum einen ein Aktuator zum Verstellen des Gashebels, zum anderen ein Aktuator zur Bedienung des Chokehebels.

Die Bewegungen der beiden Aktuatoren werden mit Servomotoren aus dem Modellbau realisiert. Diese sind kompakt und leicht. Sie besitzen drei Anschlüsse: Versorgungsspannung  $V_{CC}$ , Masse GND und Signal. Die Steckerbelegung ist auf der Steckerverbindung abgebildet. An die Signalleitung wird ein pulsweitenmoduliertes (PWM) angeschlossen. Die Periode ist 20 ms.

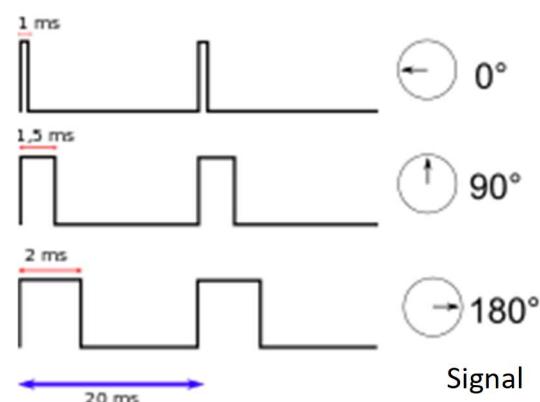


Abbildung 9: Ansteuerung eines Servomotors

Innerhalb dieser 20 ms wird ein Puls erwartet. Die Breite der Pulse beschreibt den Winkel, auf den der Servoarm gestellt werden soll (siehe Abb. 9). Mithilfe eines an der Drehachse befestigten Potentiometers wird intern die aktuelle Position bestimmt. [2]

### 8.3 Relaismodul

Der Raspberry Pi kann nur eine begrenzte Menge an Strom über seine GPIOs zur Verfügung stellen. Die Relais ermöglichen mithilfe eines kleinen Stroms vom Raspberry Pi (Steuerstromkreis) Stromkreise mit stärkerem Strom zu schalten (Arbeitsstromkreise). In dieser Projektarbeit werden damit die Servomotoren mit Spannung versorgt.

Aus optischen Gründen wurde für die Anordnung des Relaismoduls auf der Modellplatte ein Gehäuse entworfen (s. Abb. 10), welches dann 3D gedruckt wurde.

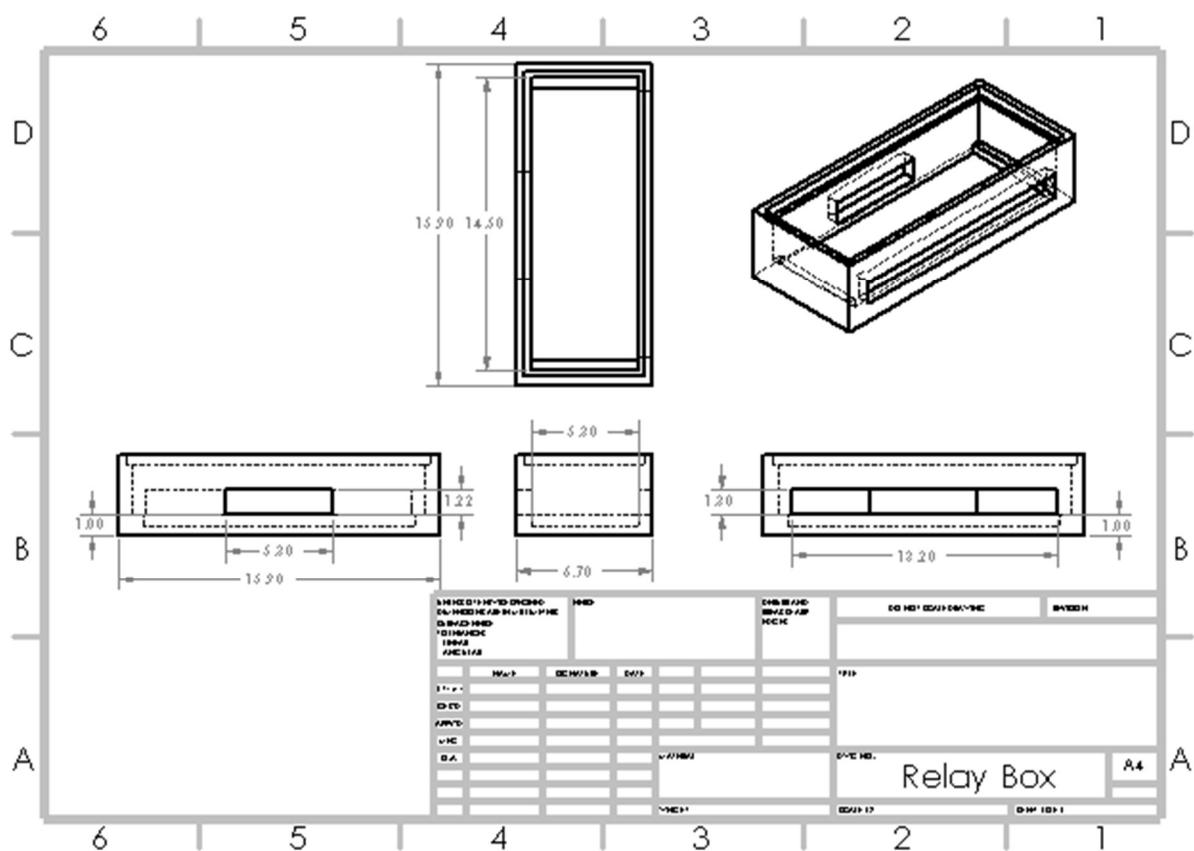


Abbildung 10: Zeichnung des Gehäuses

---

## 8.4 Not-Aus

Der verbaute Not-Aus Buzzer hat drei Kontaktelemente – ein Schließerkontakt (normally open) (NO) und zwei Öffnerkontakte (normally closed) (NC).

Mithilfe des Schließerkontakte wird die Zündung der Motorsäge kurzgeschlossen. Das bedeutet, dass bei Betätigung des Buzzers die Säge sofort ausgeht.

Ein Öffnerkontakt schaltet die Spannungsversorgung des Relaismoduls ab. Dadurch werden die Servomotoren spannungsfrei geschalten. Die Aktuatoren können nun händisch bewegt werden.

Der zweite Öffnerkontakt unterbricht bei Betätigung das Signal zum Raspberry Pi. Man kann es wie eine Drahtbruchsicherheit verstehen - der Raspberry Pi bekommt dauerhaft ein Signal an seinen Not-Aus GPIO, bei Betätigung wird das Signal unterbrochen, er weiß somit, dass der Not-Aus betätigt wurde und kann dementsprechend darauf reagieren.

## 8.5 Befestigung

Die Hardware wurde auf eine Pressspanplatte montiert. Die ursprüngliche Platte, auf der das alte System zusammengebaut wurde (s. Abb 11), war zu klein für das neue Arrangement. Wir besorgten deshalb eine Platte mit größeren Abmessungen.



Abbildung 11: Alte Platte

---

In diese Platte wurden drei Löcher gebohrt und Schrauben durchgeführt. Mit Einschlagmuttern wurden die Schrauben auf der Rückseite der Platte befestigt (s. Abb. 12). Diese Schrauben werden benötigt, um die Motorsäge mit Spannpratzen halten bzw. fixieren zu können. Die Spannkraft wird durch Anziehen der Muttern über der Pratze aufgebracht.



Abbildung 12: Befestigung der Einschlagmuttern

Im bisherigen System wurden die Aktoren mittels Rohrverbindungen auf der Platte befestigt. Da sowohl die Rohre als auch die Verbindungsstücke aus Stahl gewählt wurden, haben sich die einzelnen Klemmverbindungen „festgefressen“. Sie waren für uns nicht mehr verwendbar, da keine Veränderung der einzelnen Positionen möglich war. Wir bestellten daher Aluminiumrohre und Kunststoff-Verbindungsstücke und bauten die Verbindungssysteme neu auf.

## 9. Drehzahlerfassung

Es galt eine Sensorik zu entwickeln, die möglichst einfach und flexibel die aktuelle Motordrehzahl erfassen kann.

Ein Ansatz war, mithilfe eines vorhandenen kapazitiv gekoppelten sekundären Zündimpulsauflnehmer der Firma Fluke das Sekundärsignal an der Zündkerzenleitung zu erfassen. Aufnahmen mit dem Oszilloskop zeigten eine periodische Schwingung mit starken Ober- und Unterschwingern. Der Raspberry Pi ist nicht im Stande, dieses Signal zu verarbeiten. Die Idee war dann, die entstehenden Zündimpulse mit Hilfe eines Eventzähler zu zählen. Diese Information sollte dann an den Microcontroller übergeben werden. Dieser Ansatz scheiterte

---

jedoch an der Auswahl eines Eventzählers, der zuverlässig jeden einzelnen Impuls erfassen kann.

Schließlich verfolgten wir den Ansatz, das Signal an der Primärseite zu nutzen. Im Folgenden wird die Entwicklung der Sensorik zur Drehzahlerfassung Schritt für Schritt beschrieben.

- Schritt 1: Primärsignal aufnehmen

Zuallererst wurde das Signal mit Hilfe eines Oszilloskops aufgenommen, um dessen Verlauf zu beobachten. Dafür musste die Haube der Motorsäge angenommen werden, um Zugang zu dem Zündmodul zu erhalten. Nachdem die Maschine gestartet wurde, griffen wir das Signal ab: der Tastkopf wurde an den Kabelschuh der Unterbrecherleitung der Zündung geklemmt, die Erdung an die Masseleitung.

Der nachfolgende Screenshot (s. Abb. 13) zeigt das Zündsignal im Leerlauf.

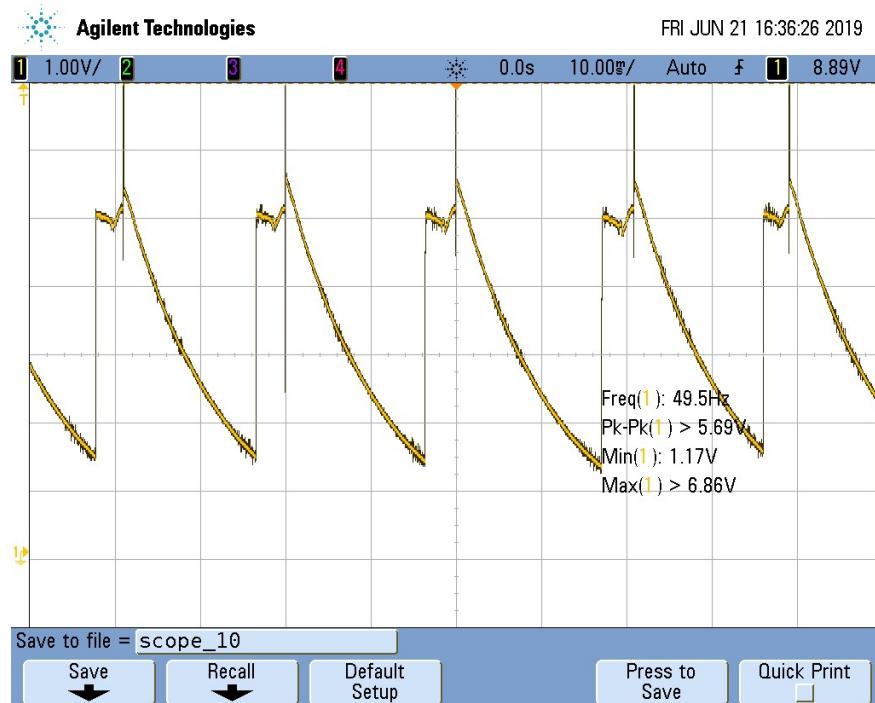


Abbildung 13: Primärseitiges Zündsignal

Es lässt sich wie folgt beschreiben: es ist eine periodische Schwingung, bei der die einzelnen Zündimpulse deutlich zu erkennen sind. Die steigende Flanke eines Impulses verläuft fast senkrecht, dann entstehen Oberschwinger, am Ende eine signifikante Spitze. In diesem Bild wird eine Maximalspannung von 6,9 V gemessen, da wegen des gewählten Rasters nur dieser

---

Bereich angezeigt wird. Diese können somit durchaus höher sein. Die fallende Flanke läuft im Gegensatz zur steigenden Flanke deutlich langsamer abwärts bis der nächste Zündimpuls entsteht. Die Frequenz beträgt 50 Hz, das entspricht einer Drehzahl von  $3000 \frac{1}{min}$  im Leerlauf.

- Schritt 2: Signalaufbereitung

Unser Ziel war, das Signal so aufzubereiten, dass dieses von dem Raspberry Pi verarbeitet werden kann. Da der Raspberry Pi an seinen GPIOs nur eine Eingangsspannung von maximal 3,3 V verträgt, müssen die Spannungsspitzen geglättet bzw. die Spannung begrenzt werden. Wünschenswert wäre außerdem, dem Microcontroller ein digitales Signal mit entweder High- oder Low-Pegel zu liefern.

- Schritt 3: Schaltung konzipieren

Die Spannungsbegrenzung soll mittels Z-Diode realisiert werden. Diese soll den Raspberry Pi vor Überspannungen bzw. Spannungsspitzen schützen.

Diese Z-Diode wird in Sperrrichtung betrieben, die Anode wird auf Masse geschalten. Die Charakteristik dieser Art von Diode ist, dass sie bei einer bestimmten Spannung, der Z-Spannung, nicht mehr sperrt, sondern Strom fließen lässt. Dieser Strom ist so hoch, dass die Versorgungsspannung bis auf den gewünschten Wert zusammenbricht. Der Strom muss mittels eines Widerstands begrenzt werden. Bei einer Reihenschaltung von Widerstand und Z-Diode fällt am Widerstand bei schwankender Eingangsspannung eine unterschiedliche Teilspannung ab, die Spannung über der Z-Diode bleibt nahezu konstant. Diese Spannung wird dann als (begrenzte) Versorgungsspannung genutzt. <sup>[3]</sup>

Um die einzelnen Impulse sicher erfassen zu können, sollen diese länger aufrecht gehalten werden. Ein Kondensator, der elektrische Ladung speichert, soll diese Funktion übernehmen. Da Kondensatoren immer über einen Widerstand geladen und entladen werden sollten, wird dieser in Reihe zu dem Eingangswiderstand geschalten. Die Aufladung erfolgt umso schneller, je kleiner die Kapazität des Kondensators und je kleiner der Widerstand ist. Die angelegte Spannung hat dabei keinen Einfluss auf die Ladezeit. <sup>[4]</sup>

Durch die Reihenschaltung des Widerstands und des Kondensators entsteht ein RC-Tiefpass. Dieser besitzt eine Filterwirkung – er lässt niedrige Frequenzen durch und dämpft bzw. sperrt

---

---

hohe. Die Spannungsspitzen werden dadurch gedämpft. Die vollständige Eliminierung dieser Spitzen übernimmt aber letztendlich die Z-Diode.

Das geglättete Signal soll dann von einem Schmitt-Trigger erkannt und in ein digitales Signal umgewandelt werden. Es handelt sich hierbei um einen Schwellwertschalter, der eine analoge Spannung in eine Schaltimpuls umwandelt. Er hat einen Eingang und einen Ausgang und liefert abhängig vom Eingangspegel immer einen definierten Ausgangspegel High oder Low (s. Abb. 14).

- Am Ausgang liegt HIGH an, wenn der Pegel am Eingang eine Spannung  $U_H$  überschreitet.
- Am Ausgang liegt LOW an, wenn der Pegel am Eingang eine Spannung  $U_L$  unterschreitet.
- Dabei wird der bisherige Ausgangspegel aufrechterhalten, wenn sich der Eingangspegel zwischen  $U_L$  und  $U_H$  befindet.
- Der Übergang von LOW auf HIGH bzw. von HIGH auf LOW erfolgt stets mit steiler Flanke. [5]

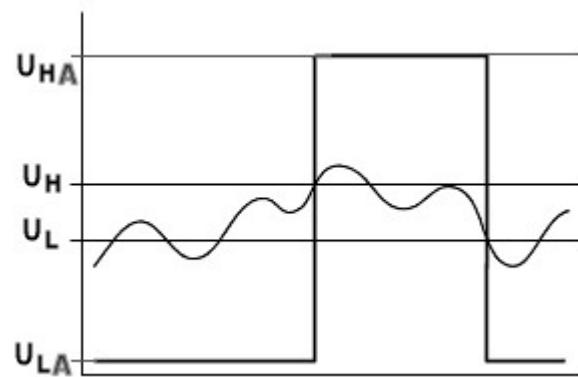


Abbildung 14: Schmitt Trigger

Damit der Microcontroller die Flanken des nun entstandenen Rechtecksignals vor allem bei Vollgas zuverlässig erfassen und zählen zu können, soll die Frequenz mit Hilfe von D-Flipflops geteilt werden. Es werden Frequenzen im Bereich von 46,67 Hz bis 166,67 Hz erwartet. Die Leerlaufdrehzahl der Motorsäge beträgt laut Datenblatt  $2800 \frac{1}{min}$ , das entspricht einer Frequenz von 46,67 Hz. Aufnahmen mit dem Oszilloskop zeigen, dass die Motordrehzahl bei Vollgas circa  $10000 \frac{1}{min}$  beträgt, was wiederum einer Frequenz von 166,67 Hz entspricht. Der Raspberry Pi ist im Gegensatz zum Arduino ein „kleiner“ Computer mit eigenem Betriebssystem. Im Hintergrund laufen viele Prozesse ab. Wohingegen ein Arduino nur genau das macht, was man ihm „sagt“. Der Raspberry Pi kann daher nicht mit der gleichen Geschwindigkeit Daten erfassen. Die D-Flipflops werden als Sicherheit in die Schaltung

integriert, um jeden Impuls zu erfassen indem ihre Frequenz verringert wird. Das erste Flipflop bricht die Frequenz auf die Hälfte, das zweite auf ein Viertel der eigentlichen Frequenz.

- Schritt 4: Schaltung simulieren

Im nächsten Schritt wurde die Schaltung mit Hilfe des Programmes LTSPICE XVII simuliert.

Da die Spannung auf 3,3 V begrenzt werden muss, wird eine Z-Diode benötigt, deren Sperr- bzw. Z-Spannung 3,3 V beträgt. Für den Eingangswiderstand wird ein Widerstandswert von  $R = 1 \text{ k}\Omega$  gewählt, für den Kondensator  $C = 47 \text{ nF}$ . Wichtig ist, den Eingangswiderstand möglichst gering zu wählen. Man könnte auch einen  $33 \text{ k}\Omega$  Widerstand und entsprechend einen kleineren Kondensator wählen, sodass die Zeitkonstante ( $\tau = R \times C$ ) gleichbleibt. Allerdings wird die Schaltung dann, je höher der Widerstand ist, umso empfindlicher gegenüber Außeneinwirkungen. Der Mensch hat beispielsweise einen Widerstand von  $33 \text{ k}\Omega$ . Würde man diesen Widerstandswert wählen, wird jede Berührung wahrgenommen und hindert die Signalaufbereitung. Wählt man stattdessen einen geringen Widerstandswert, haben Störungen von außen einen sehr geringen Einfluss.

Die Schaltung wurde folgendermaßen in LTSPICE aufgebaut (s. Abb. 15):

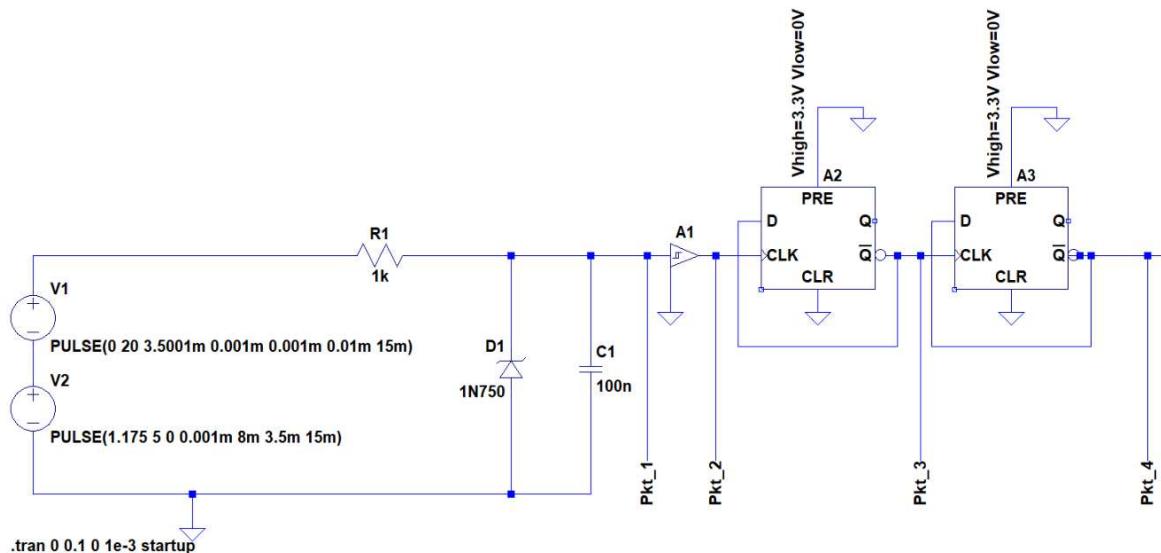


Abbildung 15: Simulation der Schaltung

Das Zündsignal wird mit zwei Spannungsquellen simuliert. Da keine Z-Diode mit 3,3 V Sperrspannung verfügbar ist, wurde die nächste größere Sperrspannung gewählt, 4,7 V. Das

Prinzip ist aber das gleiche. In der Simulation wird also die Spannung anstatt auf die gewünschten 3,3 V auf 4,7 V begrenzt. Für den Schmitt Trigger sowie die beiden D-Flipflops wird jeweils ein Digitalbaustein ausgewählt.

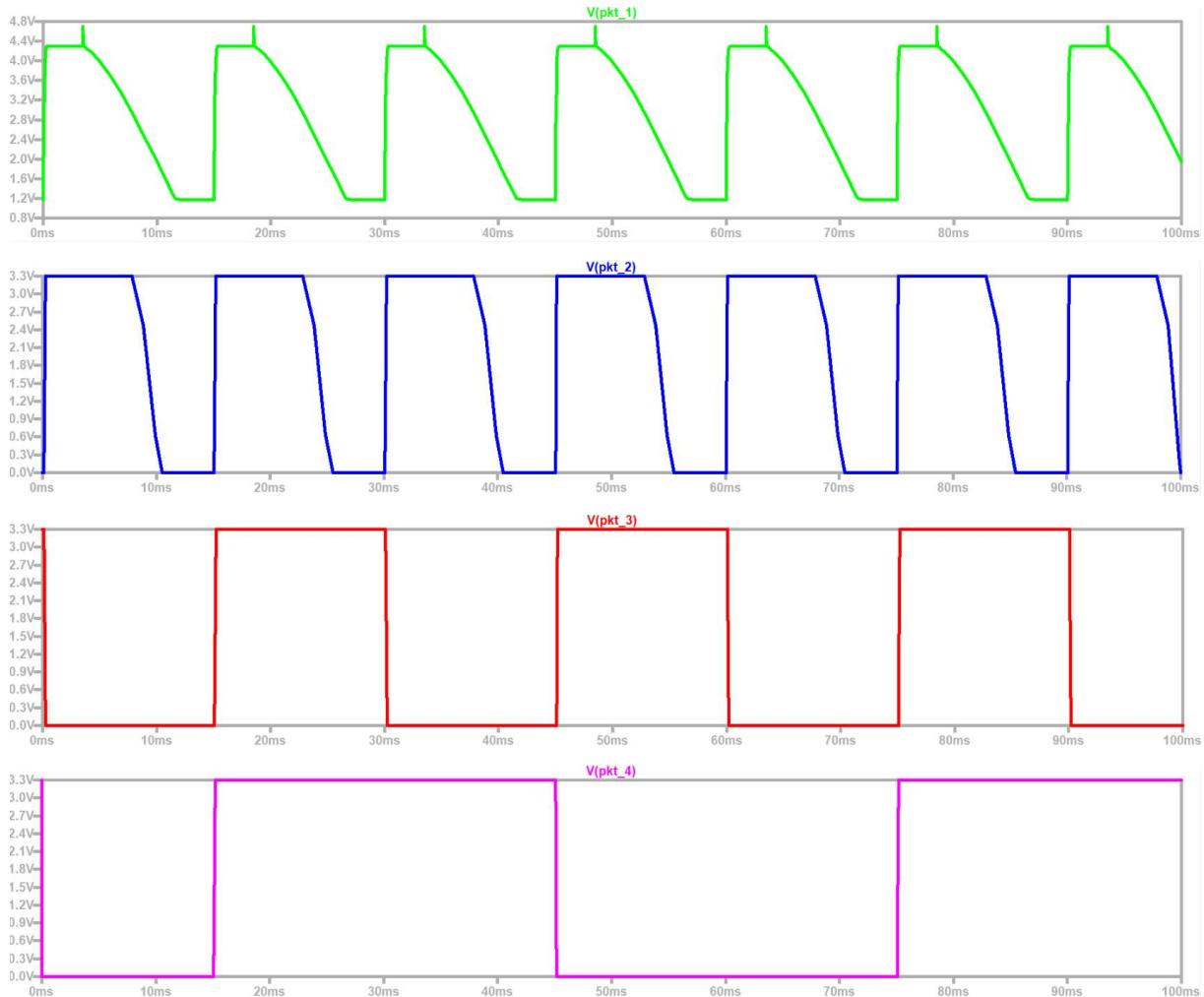


Abbildung 16: Verlauf der Signale

- Schritt 5: Schaltung auf Steckplatine testen

Nachdem alle Bauteile besorgt wurden, konnte die konzipierte Schaltung getestet werden. Als Kondensator wird ein Polyester-Folienkondensator mit einer Kapazität von 47 nF und einer Bemessungsspannung von 63 V verwendet. Es wird keine Spannung größer als 63 V erwartet. Als Widerstand wird ein Schichtwiderstand gewählt, ein Festwiderstand mit dem

---

gewünschten Wert von 1 kΩ. Die Funktion des Schmitt Triggers übernimmt der CMOS 4093 und die der D-Flipflops ein CMOS 4013, jeweils als dual in-line package.

Die Beschaltung der Logikbausteine wurden den jeweiligen Datenblättern entnommen. Der CMOS 4093 (s. Abb. 17) beinhaltet vier Schmitt-Trigger, wobei nur einer für unsere Anwendung benötigt wird. Das Signal wird an die Eingänge A und B gebrückt. Der Ausgang J wird an den ersten D-Flipflop angeschlossen. Zur Spannungsversorgung wird 3,3 V an  $V_{DD}$  sowie Masse an  $V_{SS}$  gelegt. Es handelt sich hierbei um einen NAND-Baustein, was bedeutet, dass das Ausgangssignal invertiert ist. Das ist bei der Drehzahlerfassung allerdings nicht relevant, da die Anzahl der Impulse gleichbleibt.

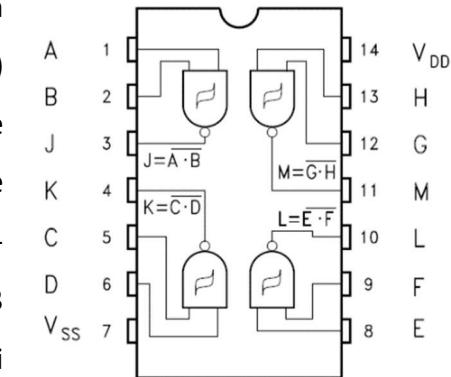


Abbildung 17: Pinanschlüsse des CMOS 4093

Auch der CMOS 4013 (s. Abb. 18) wird zur Spannungsversorgung 3,3 V an  $V_{DD}$  und Masse an  $V_{SS}$  geschlossen. Dieser Baustein enthält zwei getrennte D-Flipflops. Es gibt zwei Betriebsarten, direkt und getaktet.<sup>[6]</sup> Da die Flipflops auf die Impulse des Schmitt Triggers reagieren sollen, wird die getaktete Betriebsart gewählt, bei der die direkten Set- und Reset-Eingänge an Masse liegen. An den Clock1-Eingang wird das Ausgangssignal des Schmitt-Triggers geschalten. Indem man den Dateneingang D des D-Flipflop mit seinem invertierten Ausgang  $\bar{Q}$  verbindet bzw. rückkoppelt, toggelt es immer wenn der Clock-Eingang von Null nach Eins wechselt. Angenommen es ist beim ersten Mal nicht gesetzt, dann ist  $\bar{Q}$  auf High. Kommt der erste Clock-Impuls, dann schiebt das D-Flipflop die „1“ in sein Register und der Q-Ausgang wird zur „1“. Damit wird aber der  $\bar{Q}$ -Ausgang und auch der Dateneingang zu „0“. Beim nächsten Clock-Impuls schiebt es diese „0“ am Dateneingang in das Register und kehrt damit wieder zum vorherigen Zustand zurück. Da es immer zwei Clock-Impulse braucht, um wieder den alten Zustand zu erreichen, teilt das D-Flipflop den Clock-Eingang durch Zwei. Da das zweite D-Flipflop auf den Takt des Ersten reagieren soll, wird der Clock2-Eingang mit dem  $\bar{Q}_1$  - Ausgang verbunden. Auch das zweite Flipflop teilt die Frequenz des

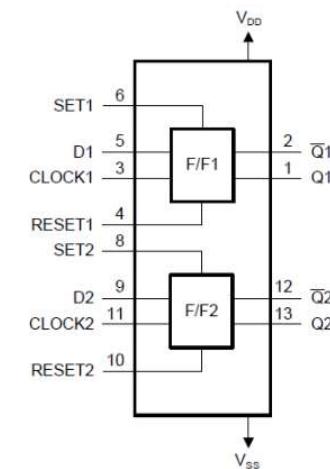


Abbildung 18: Pinanschlüsse des CMOS 4013

---

Ersten um die Hälfte. Somit wird die Frequenz des Schmitt Trigger-Ausgangs um ein Viertel geteilt.

Nachdem die Schaltung mit den ausgewählten Bauteilen auf einer Steckplatine aufgebaut wurde, konnte sie nun auf ihre Funktionalität getestet werden. Das Zündsignal wurde mittels Hirschmannklemmen vom Zündmodul abgegriffen und an das Steckbrett angeschlossen. Die Spannungsversorgung der IC's wurde mit einem Netzteil realisiert. Die verschiedenen Signale wurden mit dem Oszilloskop aufgenommen.

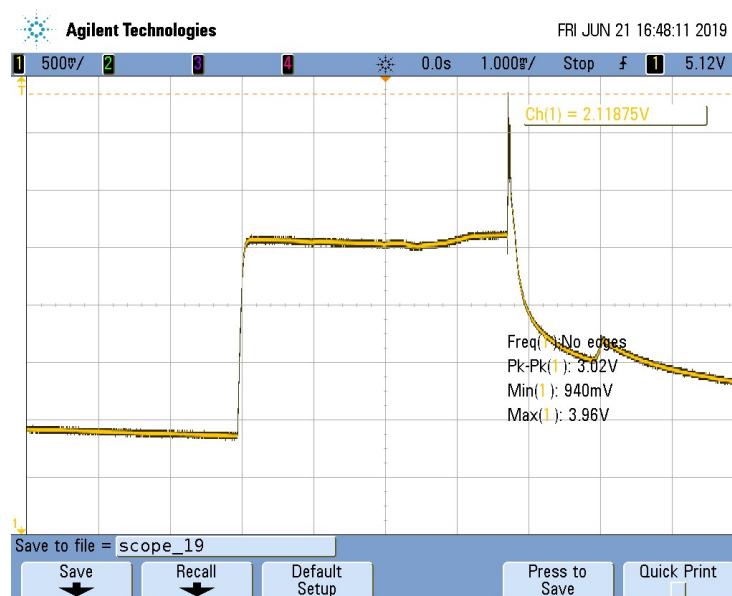


Abbildung 19: Signal vor dem Schmitt Trigger

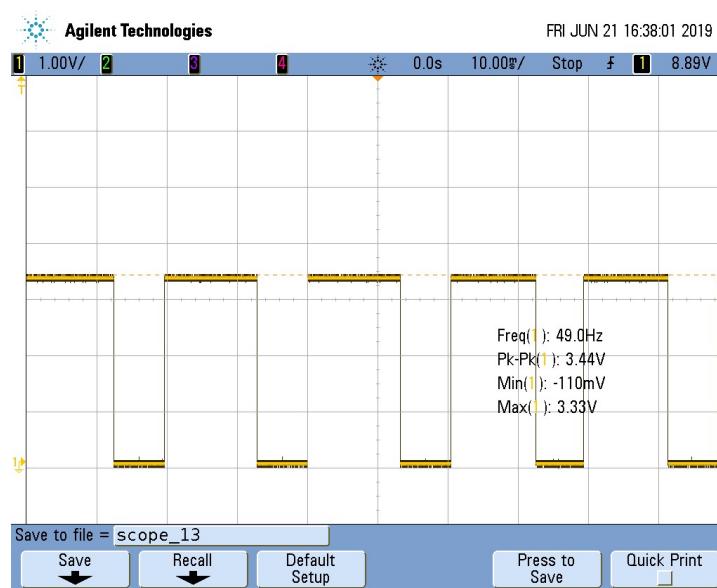


Abbildung 20: Signal nach dem Schmitt Trigger

Abbildung 19 zeigt einen einzelnen Zündimpuls am Eingang des Schmitt Triggers. Abbildung 20 entstand am Ausgang des Schmitt Triggers. Man sieht, dass dieser im Stande ist eine Rechteckspannung zu formen.

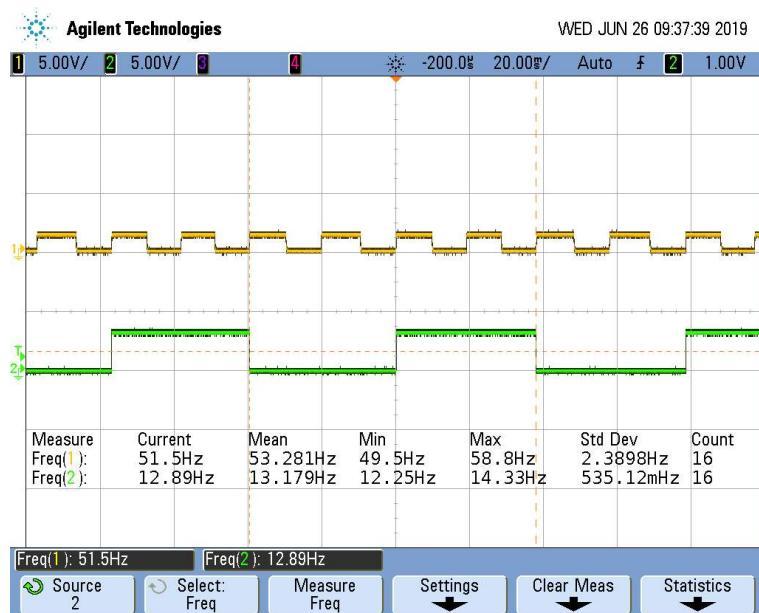


Abbildung 21: Frequenzteilung

In Abbildung 21 ist das Rechtecksignal nach dem Schmitt Trigger zu sehen (in Gelb). Die erfolgreiche Frequenzteilung nach den beiden D-Flipflops ist in grün zu sehen. Die Frequenz wurde von 51,5 Hz auf 13 Hz geteilt, was einer Teilung auf ein Viertel entspricht.

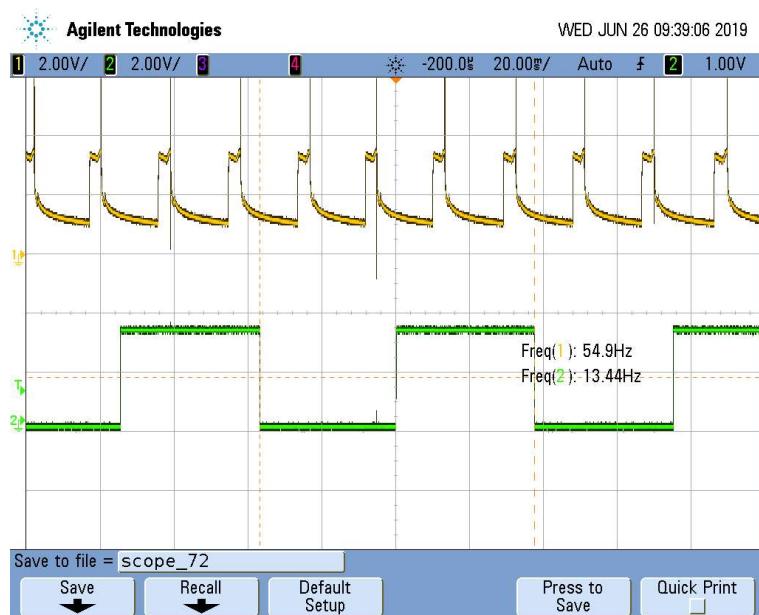


Abbildung 22: Aufbereitung des Zündsignals

---

In Abbildung 22 sind die Signale am Ein- und am Ausgang der konzipierten Schaltung dargestellt: in Gelb die einzelnen Zündimpulse mit einer Frequenz von 55 Hz, in Grün das Signal nach dem zweiten D-Flipflop mit einer Frequenz von 13,4 Hz.

Dieser Testdurchlauf bzw. diese Aufnahmen bestätigten die Funktionalität der Schaltung.

- Schritt 6: Schaltung mit Raspberry Pi testen

Nachdem sich das Python-Team Gedanken über die Erfassung des nun aufbereiteten Signals gemacht und diese programmiert hat, konnte die Schaltung mit dem Raspberry Pi getestet werden. Eine wichtige Information dafür ist, dass eine Zündung einer Umdrehung entspricht, da die Motorsäge einen Zweitaktmotor verwendet. Das Signal am Ausgang des zweiten D-Flipflops wurde an den dafür definierten GPIO gesteckt, die Spannungsversorgung der IC's erfolgte über den Raspberry Pi.

- Schritt 9: Schaltung auf Platine löten

Nachdem die Drehzahl erfolgreich mit dem Microcontroller erfasst werden konnte, wurde die Schaltung auf eine Platine gelötet. Wir entschieden uns für ein „Prototyping HAT“ (Hardware attached on top). Der Vorteile dieser Platine ist, dass man diese direkt auf die GPIO-Pins des Raspberry Pis stecken kann, indem man eine Buchsenleiste auf die Platine lötet. Es wurden außerdem zwei Buchsen auf die Platine gelötet, um die beiden Bananenstecker der Zündsignalleitung darin zu befestigen.

- Schritt 10: Platine testen

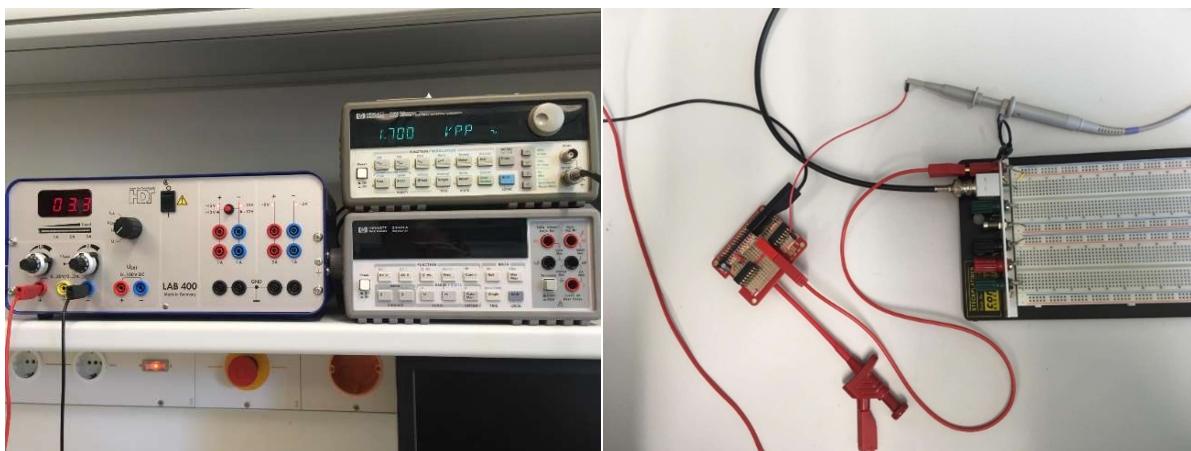


Abbildung 23: Funktionalitätsprüfung der Platine

---

Bevor die Platine auf den Raspberry Pi gesteckt wurde, haben wir sie auf ihre Funktionalität getestet. Das Zündsignal wurde mittels Frequenzgenerator simuliert. Das Signal wurde mit dem Oszilloskop beobachtet. Es wurde überprüft, ob bei der Verlötung der Schaltung ein Kurzschluss entstanden ist, ob ein Bauteil durch Überhitzung beschädigt wurde oder ob eine unzureichende bzw. fehlerhafte Kontaktierung vorhanden ist.

- Schritt 11: Platine auf Raspberry Pi testen

Nachdem eine fehlerhafte Verlötung der Schaltung auf der Platine ausgeschlossen werden konnte, wurde die Platine auf den Raspberry Pi gesteckt und getestet. Das Zündsignal wurde mittels Hirschmannklemmen vom Zündmodul abgegriffen und auf die Platine geleitet. Nachdem der Motor gestartet wurde, konnte die Drehzahl mit dem Python Skript erfasst werden.

## 10. User Interface

In Kommunikation mit der gesamten Projektgruppe wurden alle anzusteuernenden Aktoren identifiziert. Des Weiteren wurden alle benötigten Feedbackdaten, die der Benutzer empfangen soll, erarbeitet. Das Bedienpanel konnte somit in zwei Bereiche eingeteilt werden. Die Oberfläche musste Bedienmöglichkeiten für die verschiedenste Funktionen bieten. Für das Starten und Stoppen des Motorgerätes, sowie die Auswahl der Betriebsmodi sollten Schaltflächen vorhanden sein. Ebenso sollte der Bediener zwischen drei ausgewählten Motorgeräten umschalten können. Durch ein Bedienelement sollte außerdem die Drehzahl des jeweiligen Gerätes eingestellt werden. Im Verlauf der Projektarbeit wurde noch eine Schaltfläche zum Einstellen des Timeoutwertes implementiert.

Auf dem Bedienelement sollte der Benutzer alle relevanten Daten, die das Steuergerät sendet, auslesen können. Hierzu wurden LED-Leuchten und Fenster für eine intuitive Benutzung verwendet. So wird beispielsweise die Information, ob der Motor an ist, mittels eines Leuchtsignales dargestellt. Drehzahl und aktueller Betriebsmodus sind aus entsprechenden Fenstern abzulesen.

Die Benutzeroberfläche (s. Abb. 24) wurde mit LabVIEW entwickelt. Das Programm bietet viele verschiedene Softwareschnittstellen an. Das Programm erleichtert das Erstellen einer

---

---

Grafischen Benutzer Oberflächen (GUI), da hierbei kein Programmcode geschrieben werden muss. Es ermöglicht eine übersichtliche und intuitive Bedienung sowie eine gute Erweiterbarkeit für zukünftige Funktionen.

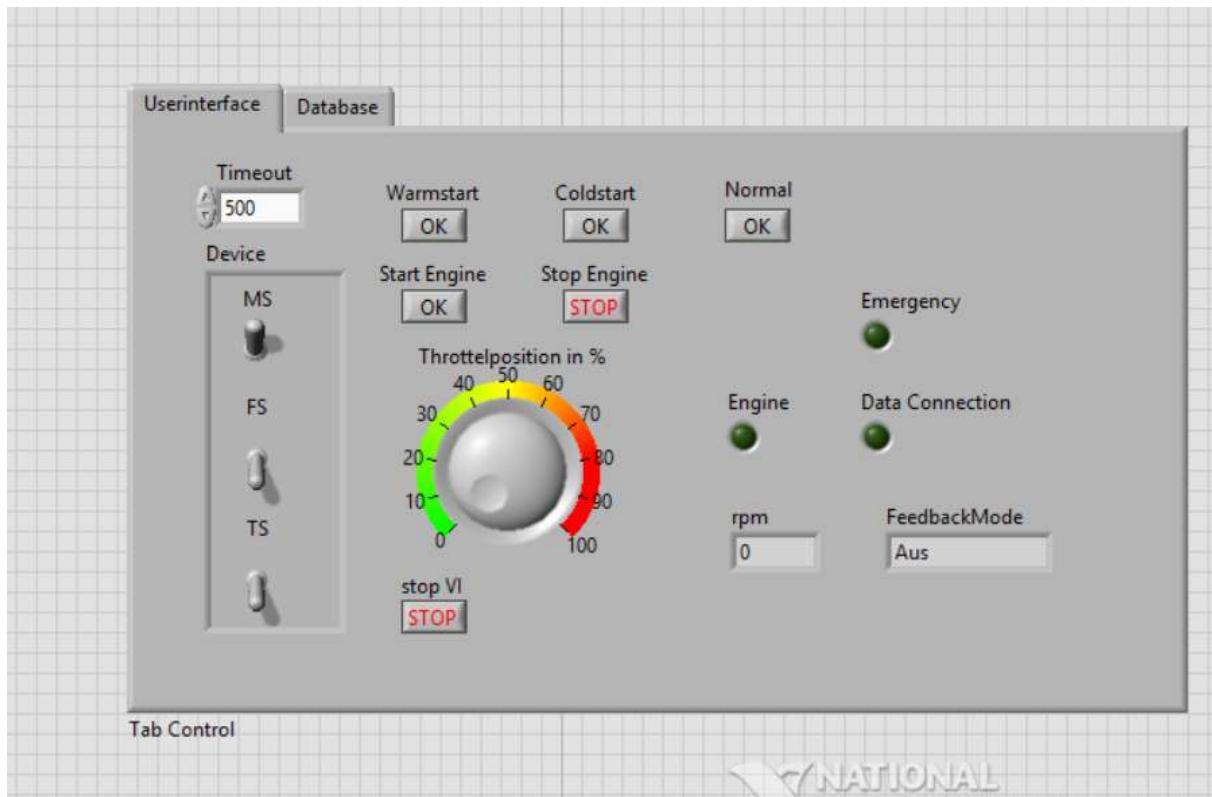


Abbildung 24: Aktuelle Benutzeroberfläche

## 11. LabVIEW

Labor Virtual Instrument Engineering Workbench (LabVIEW) ist eine Software von National Instruments, die als System-design-Plattform in einer visuellen Programmiersprache namens "G" dient.

Eine Systemfunktion kann mit den Unterprogrammen im LabVIEW-Programm Virtual Instruments (VIs) erstellt werden. Es besteht aus drei Komponenten: Frontplatte, Blockdiagramm und Anschlussbereich. Die folgende Abbildung (s. Abb. 25) zeigt ein Frontpanel und ein Blockdiagramm von LabVIEW.

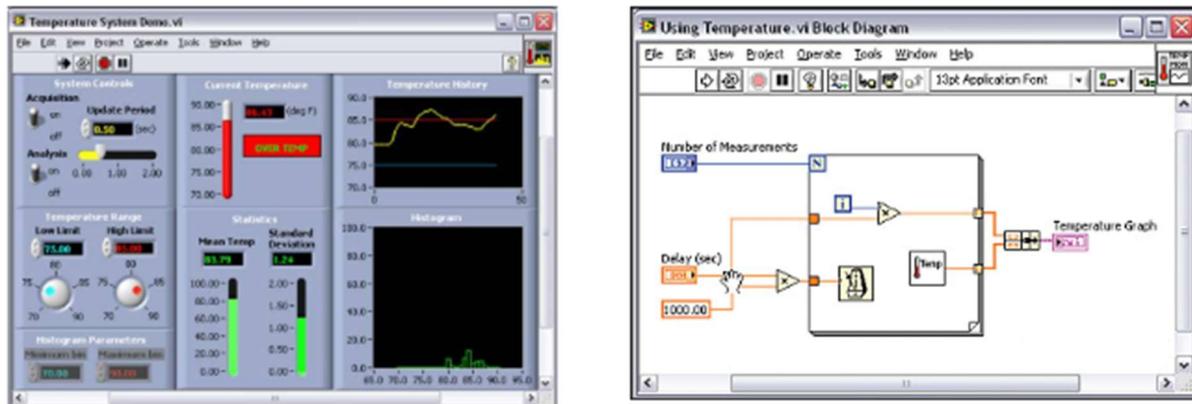


Abbildung 25: Beispielhafte Darstellung der LabVIEW Benutzeroberfläche

Als grafische Benutzeroberfläche dienen Frontpanel-Funktionen, bei denen der Benutzer Ein- und Ausgabefunktionen für das Programm platziert. Das Blockdiagramm enthält den grafischen Quellcode, in dem der Benutzer Strukturen, Indikatoren oder Funktionen aus der Funktionspalette auf dem Panel einfügen kann. Diese Strukturen, Indikatoren und Funktionen sind vom Benutzer in einem gewünschten Fluss angeordnet und werden mit Drähten verbunden. Alle Objekte auf der Vorderseite haben ihr Terminal auf der Rückseite.

LabVIEW wird in der Datenerfassung, industriellen Automatisierung und zur Ansteuerung verschiedener Messinstrumente verwendet. Dies macht es zu einer der Software, welche eine große Benutzergemeinschaft hat und als ein weithin akzeptiertes Designmuster angesehen werden kann. Die Software bietet eine große Anzahl an Toolkits und Bibliotheken, um zusätzliche Funktionen wie die Signalgenerierung und statische Auswertung einbinden zu können.

Da die Software graphische Programmieransätze verwendet, werde alle Funktionsdarstellungen per Drag and Drop auf das Frontpanel gezogen, was ein Vorteil für die einfache Benutzung ist. Zusätzlich erlaubt LabVIEW parallele Programme. Diese Funktion erleichtert das Entwickeln eines großen Programmes, bei dem in der Regel viele Aufgaben gleichzeitig ausgeführt werden müssen.

## 12. Blockschaltbild

Beim Entwickeln der internen Datenstruktur innerhalb der LabVIEW Software mussten verschiedene Funktionen berücksichtigt werden:

- Empfangen der Feedbackdaten
- Übermitteln der Benutzerinformationen
- Kontrolle der Verbindung

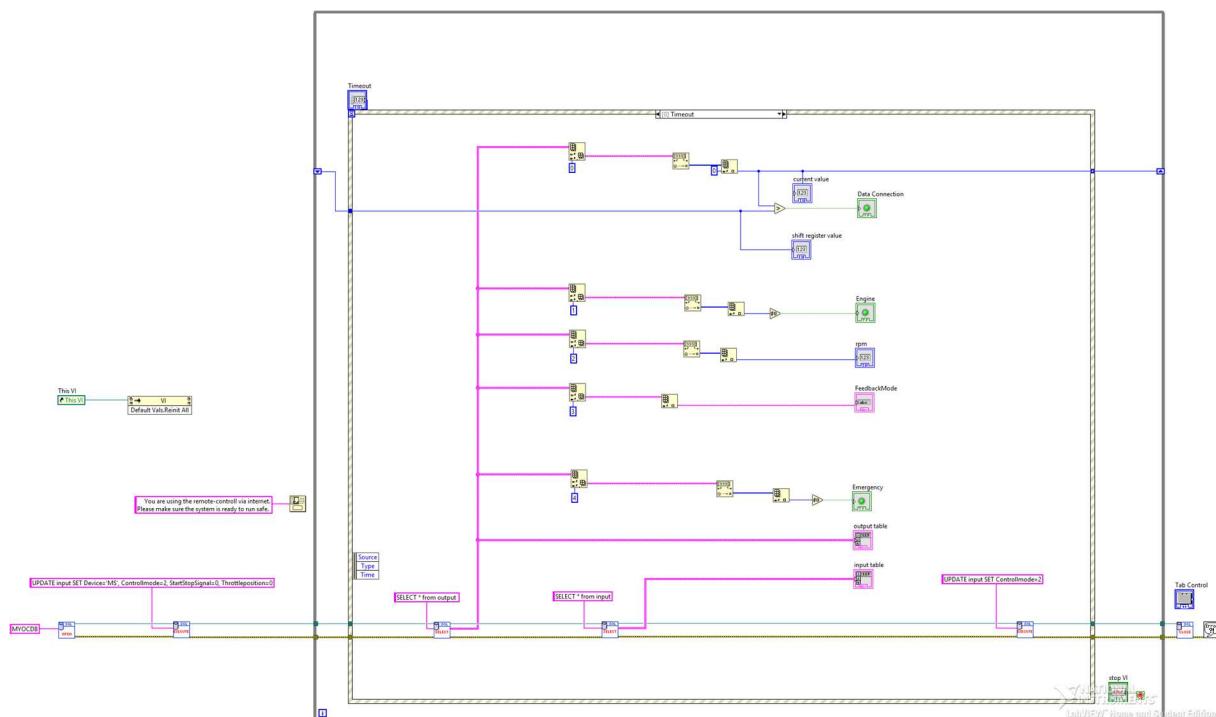


Abbildung 26: Ausschnitt des aktuellen Blockschaltbilds

Das Blockschaltbild (s. Abb. 26) besteht im Wesentlichen aus einer Case-Schleife, die auf die unterschiedlichen Funktionen der Benutzeroberfläche reagiert (grau/weiß dargestellt). Des Weiteren sind unterschiedliche Funktionsbausteine im Schaltbild implementiert. Um mit der SQL-Datenbank online zu kommunizieren, wurde eine zusätzliche Bibliothek in die Software eingebunden. Der Datenfluss der Informationen ist über verschiedenen Farben gekennzeichnet. Die Farbe beschreibt den Datentyp der Information. Die Übersicht über das Schaltbild ist folgend stückweise erklärt.

Der erste verwendete Baustein ist der „Open“-Funktionsbaustein (s. Abb. 27). Dieser Baustein erstellt die Verbindung zum Server mit der entsprechenden Datenbank. Der Baustein

---

bekommt eine Zeichenkette mit dem Namen des hinterlegten Connector als Eingabewert. Das Zugreifen auf den Connector erfolgt automatisch, es sind keine weiteren Adressparameter notwendig.

Der „Execute“-Baustein (s. Abb. 27) bekommt ebenfalls ein String als Eingabe. Dieser wird in der Syntax in der SQL Datenbanksprache eingegeben. Hierzu ist die genaue Deklaration der Variablen notwendig, sowie der eingehaltene Datentyp mit dem korrekten Wert. Der Wert wird mittels des „Execute“-Bausteins an die Datenbank übermittelt.

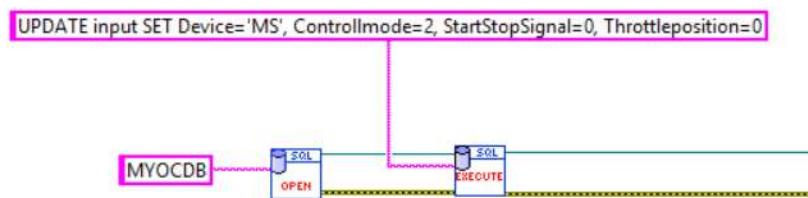


Abbildung 27: Beginn des Blockschaltbilds

Die Feedbackinformationen werden in einem regelmäßigen Zeitabstand vom Server abgefragt. Dieser Zeitwert kann der Benutzer im Controlpanel einstellen. Mittels des „Select“-Funktionsbausteins (s. Abb. 28) werden alle Informationen aus der Datenbank-Tabelle extrahiert um auf die jeweilige deklarierte Zeile zurückzuführen. Das Extrahieren wurde mit unterschiedlichen Funktionsbausteine umgesetzt, abhängig vom jeweiligen Datentyp. Die Informationen werden an das Bedienpanel weitergegeben. Die Schaltfläche der Benutzeroberfläche gibt somit ein Signal entsprechend des Datentyps aus. Ein Sonderfall hierbei ist die Not-Aus-Information: diese gibt eine Fehlermeldung an den Benutzer aus.

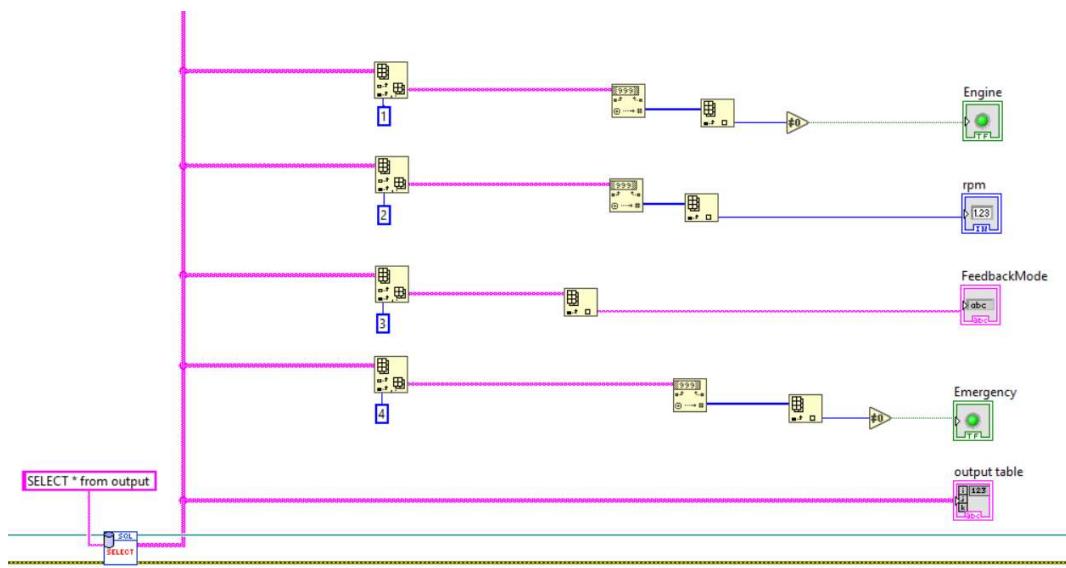


Abbildung 28: Verarbeiten der Datenströme

Das Senden der Benutzereingabe wird mit Hilfe des „Execute“-Bausteins umgesetzt (s. Abb. 29). Dieser Baustein wird aktiviert, sobald die Schaltfläche des Bedienpanels eine gültige Eingabe erfasst. Bei der Eingabe wird die entsprechende Case-Struktur im Blockschaltbild aufgerufen.

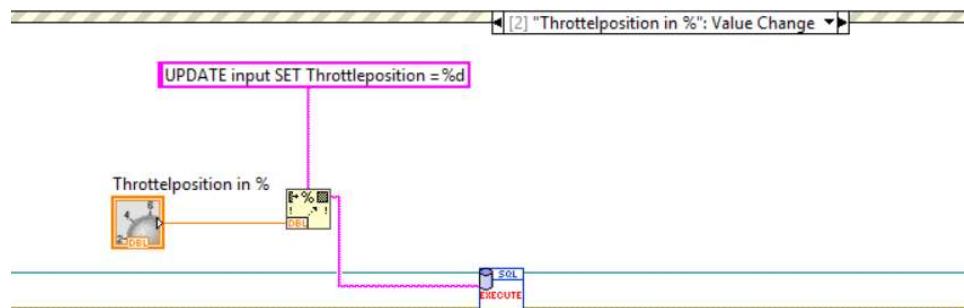


Abbildung 29: Beispielhafte Darstellung einer Case-Funktion

Um das System ausfallsicher zu gestalten, wurden sich auf Default-Werte geeinigt, welche bei jedem Start der Software erneut hochgeladen werden. Somit wird gewährleistet, dass vor jedem Start der Software die Datenbankinformationen mit den Informationen des Bedienpanels übereinstimmen. Ebenfalls wurde innerhalb des regelmäßigen Timeout-Zyklus

---

der Wert des Betriebsmodus aktualisiert, um die zugehörige Aktorik in eine dauerhafte definierte Stellung zu bringen.

Nach dem Stoppen des Programms wird der Funktionsbaustein „Close“ benutzt (s.Abb. 30). Hier wird die Verbindung zur Datenbank beendet. Falls es zu einen Fehlerfall kommt, gibt der hintere Ausgabeblock eine Error Meldung an den Benutzer aus.

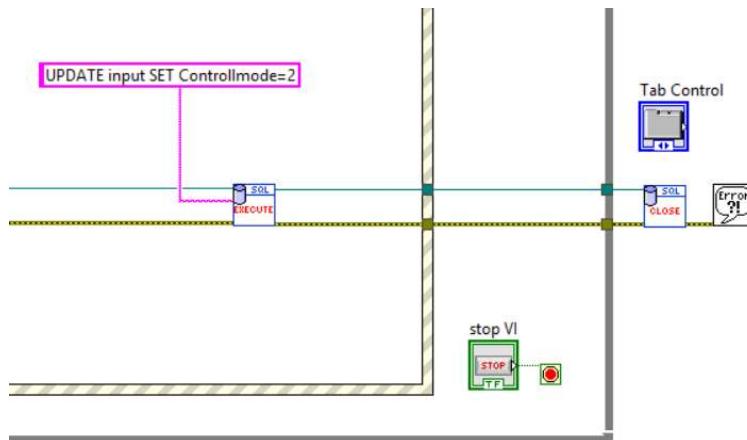


Abbildung 30: Ende des Blockschaltbilds

Die Kontrolle einer dauerhaften Verbindung wurde mittels eines „Shift Register“ realisiert (s. Abb. 31). Diese Struktur dient dem kontinuierlichen Vergleich eines sich ändernden Zahlenwertes. Dieser Wert wird in einem regelmäßigen Zeitabstand vom Steuergerät erhöht. Wenn sich dieser Wert innerhalb des Timeout-Zyklus nicht ändert, sind die Systeme nicht mehr miteinander verbunden. Ist das der Fall, wird eine Fehlermeldung an den Benutzer ausgegeben.

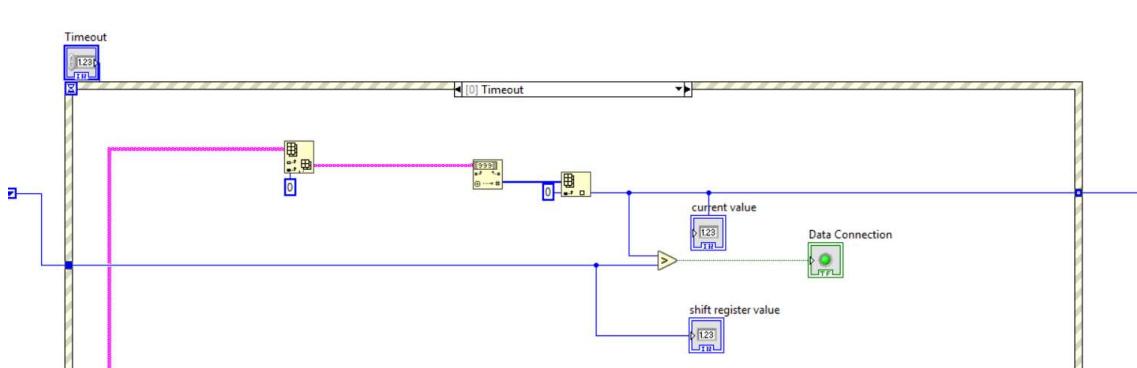


Abbildung 31: Darstellung des Shift Register

## 13. Datentransfer

Die günstigste Lösung für unsere Rahmenbedingungen war es, den Datenaustausch mittels eines Servers zu realisieren. Dabei entwickelte sich die Idee, eine Datenbank zu nutzen, um die Eingabeinformationen des Benutzers speichern zu können. In dieser werden Informationen in einem regelmäßigen Zyklus abgefragt. Der Vorteil eines Servers ist, dass dieser geräteunabhängig genutzt werden kann und dass ein Gesamtsystem beliebig erweitert werden kann.

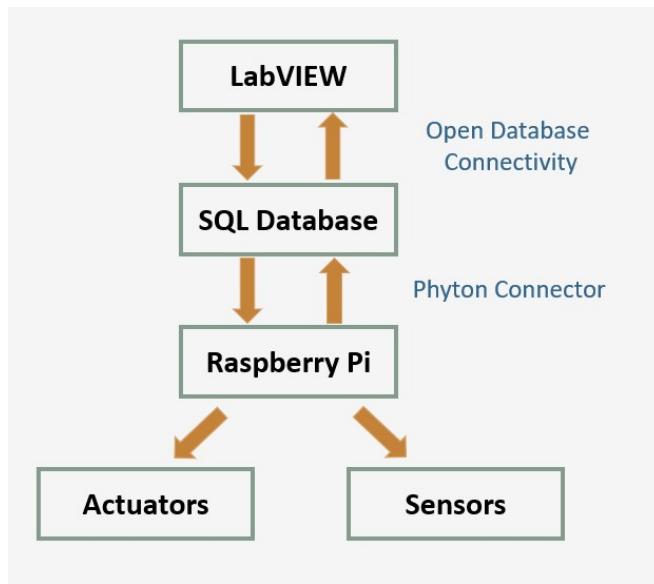


Abbildung 32: Schematische Darstellung des Datentransfers

kann und dass ein Gesamtsystem beliebig erweitert werden kann.

Eine Datenbank ist eine organisierte Sammlung von digitalen Daten. In der Regel werden diese gespeichert und können von einem Computersystem abgerufen, verwaltet und aktualisiert werden. Die Daten werden in Tabellen, Spalten und Zeilen eingeteilt. Jede Information muss eindeutig durch einen Datentyp und einen Namen deklariert werden. Solange die Verbindung zur Datenbank hergestellt ist, können die Daten in Echtzeit mit dem neuen Informationszusatz aktualisiert, erweitert und gelöscht werden.

## 14. Datenstruktur

In Zusammenarbeit mit der gesamten Projektgruppe wurde die folgende Struktur der Datenbank ausgearbeitet. Hierzu wurden alle benötigten Funktionen für das System deklariert und dokumentiert. Die Datenbank befindet sich auf einem externen Server und wird mit dem Datenbanksystem MySQL verwaltet. Das Erzeugen der Datenbanktabellen erfolgte mit der SQL-Datenbanksprache (s. Abb. 33). Die Datenbank wurde in zwei Tabellen gegliedert. In der Input-Tabelle (s. Abb. 34) wurden alle Informationen, die der Benutzer an das Steuergerät sendet, gespeichert, in der Output-Tabelle wurden alle Feedbackdaten des Raspberry Pis gespeichert.

**phpMyAdmin**

Server: remotemysql.com » Datenbank: VisiuLdKXI » Tabelle: output

Anzeigen Struktur SQL Suche Einfügen Exportieren Importie

Letzte Favoriten

information\_schema  
VisiuLdKXI  
  Neu  
  input  
  Interface  
  LED  
  output

⚠ Die aktuelle Markierung enthält keine eindeutige („unique“) Spalte. Gitter-Bearbeitungsfunktion, Kontrollkästchen

✓ Zeige Datensätze 0 - 0 (1 insgesamt, Die Abfrage dauerte 0.1416 Sekunden.)

SELECT \* FROM `output`

Alles anzeigen Anzahl der Datensätze: 25 Zeilen filtern: Diese Tabelle durchsuchen

+ Optionen

Statusconnection	Statusengine	rpm	StatusMode	lokaleEmergency
98	0	0	Aus	0

Alles anzeigen Anzahl der Datensätze: 25 Zeilen filtern: Diese Tabelle durchsuchen

Operationen für das Abfrageergebnis

Drucken In Zwischenablage kopieren Exportieren Diagramm anzeigen Erzeuge View

*Abbildung 33: Datenbankentwicklung über die Webseite*

*Abbildung 34: Beispiel der Datenbankkonventionen*

## 15. Verwendete Software

### 15.1 Datenbank Connector

Ein Datenbank Connector bieten die Möglichkeit eine direkte Verbindung zu einem Server aufzubauen. Hierzu wurde eine Treiber Software installiert. In dem Connector sind alle relevanten Daten des Servers hinterlegt. Der Connector wird benötigt, damit das LabVIEW Programm selbstständig mit dem Server kommunizieren kann. Das Erstellen des Connector ist in Folgendem dargestellt.

In diesem Projekt wurde eine Software namens ODBC Database Driver von MySQL verwendet. Sie diente als Verbindungsstück zwischen der MySQL Database und dem LabVIEW database connectivity toolkit.

Im Folgenden werden die Schritte zum Installieren des ODBC database driver von MySQL beschrieben:

1. Laden und Installieren des Treibers von MySQL  
(<https://dev.mysql.com/downloads/connector/odbc/>)
2. Öffnen der Datenquellenliste des Computers, um den Treiber hinzuzufügen  
(Systemsteuerung -> System und Sicherheit -> Verwaltung -> Datenquellen (ODBC))
3. Hinzufügen einer ODBC Verbindung (s. Abb. 35)

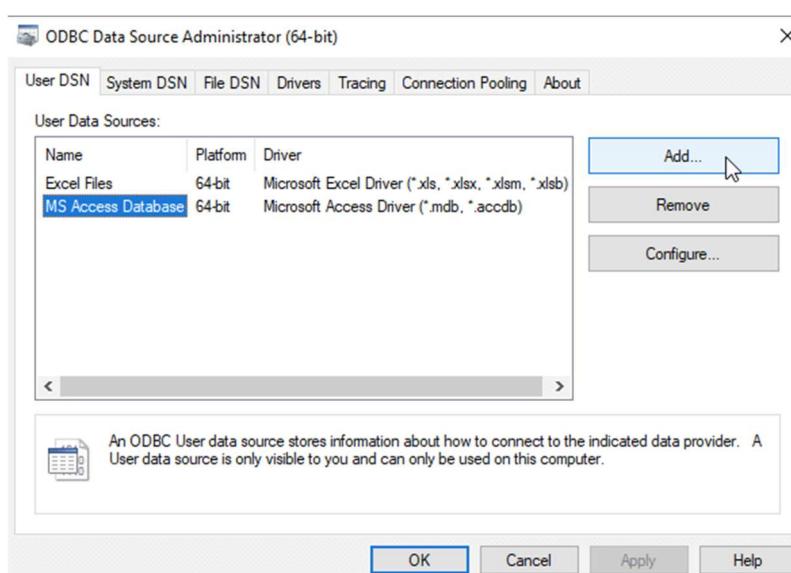


Abbildung 35: Administratoreinstellungen unter Windows

4. Verwenden des MySQL ODBC-Treiber, entweder ANSI oder Unicode (s. Abb. 36)  
<https://dev.mysql.com/doc/connector-odbc/en/connector-odbc-installation.html>

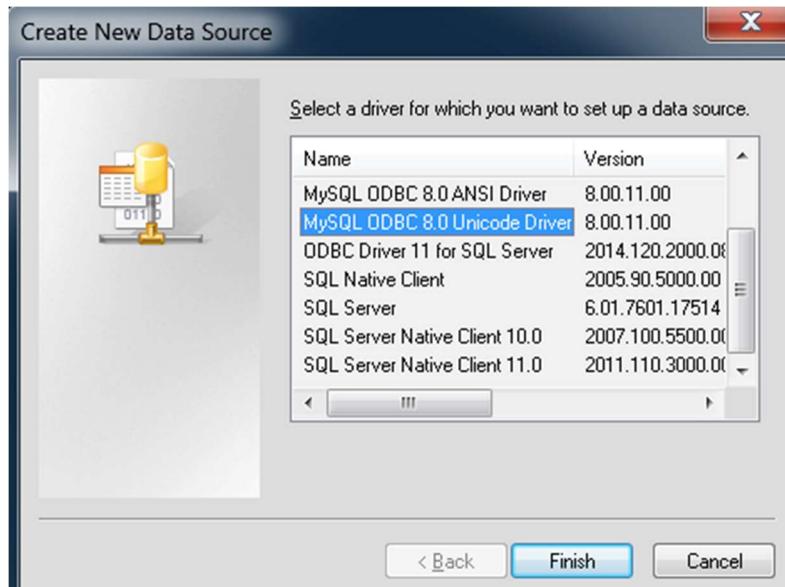


Abbildung 36: Auswahl des ODBC Treibers

5. Einstellen der Verbindungsparameter für den MySQL-Server (s. Abb. 37)



Abbildung 37: Parametereinstellungen des Treibers

6. Anschließend kann der Quellename der Verbindung in der LabVIEW Software verwendet werden.

## 15.2 LabVIEW Toolkit

Für die Kommunikation zwischen LabVIEW und der Datenbank müssen neben dem OCDB weitere Funktionen in die Bibliothek eingebunden werden. Das verwendete Toolkit ermöglicht das Verwenden verschiedener Bausteine in der LabVIEW Benutzeroberfläche. Im Folgendem Link befinden sich die verwendeten Bibliotheken:

<https://www.halvorsen.blog/documents/programming/labview/resources/code/database/SQLOpenSource.zip>

Im Download wird die Einbindung der Software in einer Textdatei beschrieben. Dieser Download ist für die Version LabVIEW 2017 32/64Bit anwendbar. Nach der Installation sind die nötigen Funktionsbausteine in der Funktionspalette zu finden (s. Abb. 38).

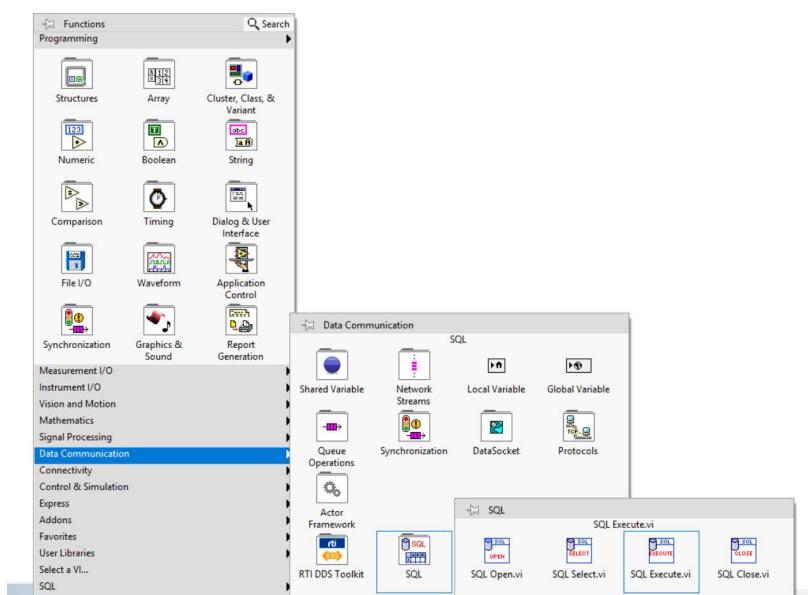


Abbildung 38: Funktionspalette mit zusätzlich eingebundenen Funktionsbausteinen

## 16. Anforderungen an MYiTOPS-Skript

Es galt ein System zu etablieren, dass sich zwischen Mechanik und Elektronik integrieren lässt. Daher wurden Skripte und Software als Kern benötigt, um es entsprechend den Projektanforderungen programmieren zu können.

---

Das MYiTOPS-Skript ist ein Skript, dass die Bewegungen der Aktoren steuert, Daten vom Sensor liest und sich mit der Cloud-Datenbank verbindet, die LabVIEW- und Feedbackdaten enthält. Dieses Skript sollte in der Lage sein, ein integriertes Echtzeitsystem über das Internet zu bedienen, einen geschlossenen Regelkreis mit Rückmeldungen bzw. ein Not-Aus-System zu besitzen und alle möglichen Ereignisse abzudecken, die während des Prüfstandbetriebs auftreten können.

Das Programmierskript sollte effizient laufen, damit der Mikrocontroller nicht viel Kapazität für die Berechnungszeit aufwenden muss. Dies ist wichtig, da Daten mit höchster Genauigkeit generiert werden sollten.

Um Daten über die Cloud oder das Internet zu beziehen, wurde ein stabiler Connector oder ein Protokoll zur Datenbank für das MYiTOPS-Skript verwendet. Zur Standardisierung des Gesamtsystems wurde das gleiche Protokoll auch für die LabVIEW Graphical User Interface (GUI)-Seite verwendet. Durch die Verbindung zur Cloud kann das System von überall und jederzeit bedient werden.

## 17. Microcontroller

Es wurde ein Raspberry Pi 3B+, wie in Abb. 39 dargestellt, als rechnergestützten Mikrocontroller gewählt, da er alle Projektanforderungen erfüllt. Es ist ein erschwinglicher, schneller und stabiler Mikrocontroller, der Peripheriegeräte wie USB-Ports, HDMI-Ports, Ethernet-Port, drahtlose Verbindungen wie WLAN- und Bluetooth-Verbindungen und GPIOs bereitstellt. Darüber hinaus wird der Raspberry Pi Microcontroller in verschiedenen Bereichen, wie z.B. in der Bildung, der Haus- und Industrieautomation sowie in kommerziellen Produkten, eingesetzt.



Abbildung 39: Raspberry Pi 3B+

## 17.1 Betriebssystem

Raspbian Buster, die neueste Version von Debian Linux, wurde als Betriebssystem (OS) für den Raspberry Pi 3B+ installiert. Raspbian ist ein Debian-basiertes Computer-Betriebssystem für Raspberry Pi. Seit 2015 wird es von der Raspberry Pi Foundation offiziell als primäres Betriebssystem für die Familie der Raspberry Pi Single-Board-Computer bereitgestellt.

Raspbian wurde von Mike Thompson und Peter Green als unabhängiges Projekt entwickelt. Der erste Bauabschnitt wurde im Juni 2012 abgeschlossen. Das Betriebssystem befindet sich noch in der aktiven Entwicklung. Raspbian ist hochgradig optimiert für die leistungsstarken ARM-CPUs der Raspberry Pi-Linie. [7]

Raspbian verwendet PIXEL, Pi Improved X-Window Environment, Lightweight als Hauptdesktop-Umgebung ab dem letzten Update. Es besteht aus einer modifizierten LXDE-Desktop-Umgebung und dem Openbox Stacking Window Manager mit einem neuen Design und einigen anderen Änderungen.

## 17.2 Programmiersprache

Der Pi (Raspbian) ist mit zwei Versionen von Python vorinstalliert, nämlich Version 3 und Version 2. Um diesen Vorteil voll auszuschöpfen, wurde Python 3 als Programmiersprache für das MYiTOPS-Skript verwendet.



Abbildung 40: Allgemeine Eigenschaften von Python

Python ist eine interpretierte, hochrangige, universell einsetzbare Programmiersprache. Python wurde in den späten 1980er Jahren als Nachfolger der ABC-Sprache konzipiert, von Guido van Rossum entwickelt und 1991 erstmals veröffentlicht. Pythons Designphilosophie betont die Code-Lesbarkeit durch die Verwendung von signifikantem Whitespace. Seine Sprachkonstrukte und sein objektorientierter Ansatz sollen Programmierern helfen, einen klaren und logischen Code für kleine und große Projekte zu schreiben.

Python ist dynamisch typisiert mit garbage-collector. Es unterstützt mehrere Programmierparadigmen, einschließlich prozeduraler, objektorientierter und funktionaler Programmierung. Python wird aufgrund seiner umfangreichen Standardbibliothek oft als "batteriebetriebene" Sprache bezeichnet. [8]

In Abbildung 40 sind die Vorteile von Python gegenüber anderen Programmiersprachen dargestellt.

Insgesamt ist Python eine robuste Programmiersprache, bietet eine einfache Bedienung der Code-Zeilen und auch das Debugging kann einfach durchgeführt werden.

---

## 17.3 Installation

Für dieses Projekt wird eine MicroSD-Karte (ab 16 GB), ein Computer mit einem MicroSD-Kartenadapter, einen Raspberry Pi und Peripheriegeräte wie Maus, Tastatur, Monitor und Stromquelle benötigt. Um das MYiTOPS-Skript ausführen zu können, müssen zuvor ein paar Schritte erledigt werden. Die Schritte sind wie folgt:

1. Raspbian Buster-Betriebssystem (OS) auf die Speicherkarte von Raspberry Pi installieren oder flashen<sup>[9]</sup>:

Zuallererst wird das Betriebssystem für den Raspberry Pi installiert. NOOBS (New Out Of Box Software) ist ein einfacher Betriebssystem-Installationsmanager für den Raspberry Pi. Nachdem Rasbian heruntergeladen und das Disc-Image auf eine MicroSD-Karte geschrieben wurde, wird der Raspberry Pi auf diese MicroSD-Karte gebootet.

2. Firmware von Raspberry Pi aktualisieren<sup>[10]</sup>:

Als nächstes müssen Software-Updates auf dem Pi bereitgestellt werden, um auf dem neuesten Stand zu bleiben. Grund dafür ist die Sicherheit. Ein Gerät, auf dem Raspbian läuft, enthält Millionen von Codezeilen. Im Laufe der Zeit werden bei diesen Millionen von Codezeilen bekannte Schwachstellen aufgedeckt. Diese sind als Common Vulnerabilities and Exposures (CVE) bekannt und in öffentlich zugänglichen Datenbanken dokumentiert, sodass sie leicht zu nutzen sind.

3. Bibliotheken installieren: MySQL Konnektoren<sup>[11]</sup> und pigpio<sup>[12]</sup>:

Wie bereits erwähnt, ist die Programmiersprache Python eine erweiterbare Sprache und verfügt über große Standardbibliotheken. Diese Bibliotheken müssen installiert werden, damit das Skript sie richtig verwenden kann. MySQL-Konnektoren oder die mysql.connectors-Bibliothek bieten standardbasierte Treiber für JDBC, ODBC und .Net. Sie ermöglichen Datenbankanwendungen in der jeweiligen Sprache, hier in Python, zu erstellen. Darüber hinaus ermöglicht eine native C-Bibliothek, MySQL direkt in die Anwendungen einzubetten. Die pigpio-Bibliothek wird benötigt, um die GPIOs auf dem Pi zu nutzen. Pigpio ist eine stabilere Bibliothek als die Standard-Pi-Bibliothek, wenn es

---

um die Handhabung der GPIOs geht. Es kann ein genaues PWM-Signal für die Stellglieder und Schalter erzeugen und ein präzises Signal von den Sensoren lesen.

4. pigpio Daemon-Tool starten
5. MYiTOPS-Skript mit Thonny IDE oder einer anderen IDE, oder mit Debian Linux Terminal ausführen:  
Das pigpio Daemon Tool muss im Terminal ausgeführt werden, damit das pigpio im Hintergrund läuft, während das MYiTOPS-Skript interpretiert wird.
6. Syntaxfehler oder andere Fehler überprüfen und bei Bedarf reparieren

## 18. Programmablauf

Der Prozess des MYiTOPS-Skripts (s. Abb. 41) beginnt, wenn der Benutzer das Skript im LINUX-Terminal ausführt. Der Raspberry Pi ist dann bereits mit dem Internet verbunden. Von der Hauptfunktion des Skripts aus werden Datenbankverbindungen über das Internet eingerichtet, sodass eine Verbindung mit der Datenbank hergestellt werden kann, in der die LabVIEW-Benutzereingabedaten gespeichert sind.

Wenn die Verbindung hergestellt wird, versucht das Skript immer, die Daten aus der Datenbank abzufragen und zu lesen, bis der Benutzer es stoppt. Wenn die Verbindung fehlschlägt oder nicht hergestellt wird, läuft sie so lange, bis die Verbindung hergestellt ist oder die Aufbauzeit abgelaufen ist.

Nachdem die Verbindung hergestellt wurde, wartet das Skript auf alle Änderungen in der Datenbank. Die Änderungen werden vom Benutzer auf der LabVIEW-GUI-Seite vorgenommen. Während das Skript die Daten abfragt, richtet es auch den General Pin Input Output (GPIO) ein, die für die Aktoren, Relais und Sensoren verwendet werden.

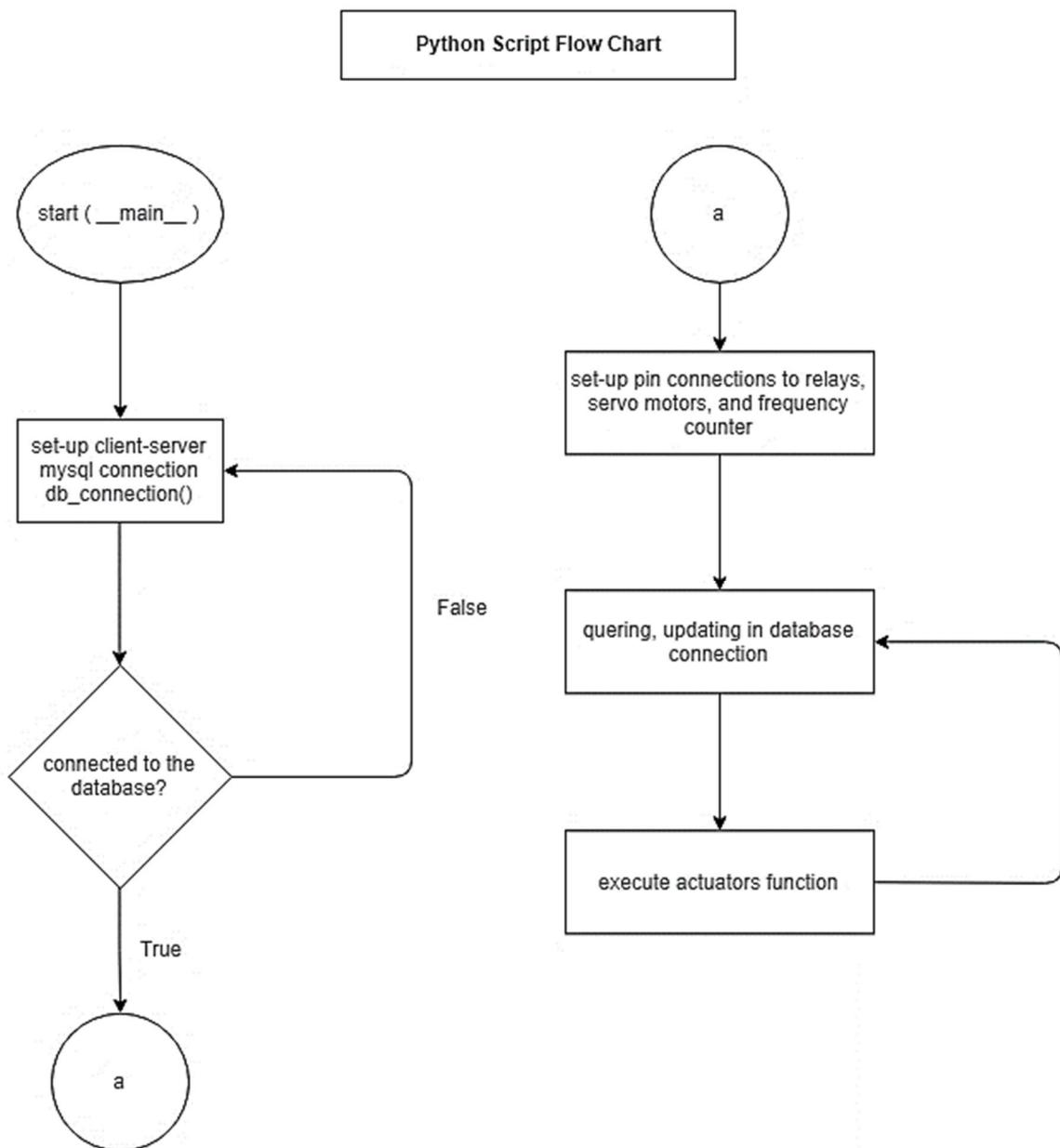


Abbildung 41: Ablaufdiagramm

Wenn es Änderungen in der Datenbank gibt, fährt das Skript mit dem nächsten Prozess fort, bei dem die Akteure in ihren eigenen spezifischen Sequenzen basierend auf dem Modus, den der Benutzer zuvor gewählt hat, bewegen werden. Das Skript beginnt die Frequenz der Motorzündung über die entwickelte Sensorik zu lesen. Die Daten der Zündfrequenz des Motors werden mit dem Pi berechnet und in der Datenbank aktualisiert, die dann in der LabVIEW-GUI angezeigt wird.

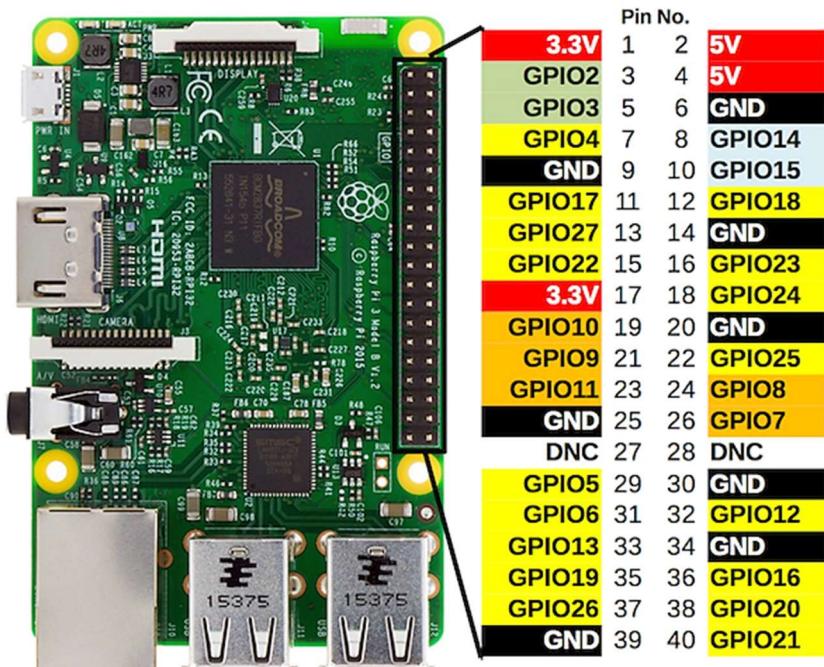


Abbildung 42: GPIOs

In Abbildung 42 ist die Zuordnung der GPIOs des Raspberry Pis dargestellt. Alle Ein- und Ausgänge wurden im Skript als Objekte erstellt und einem GPIO zugeordnet. Da verschiedene Erweiterungsbibliotheken unterschiedliche Syntaxen handhaben, musste bei der Verkabelung darauf geachtet werden, ob das Objekt über die GPIO-Nummer definiert oder einer entsprechenden Pinnummer zugewiesen wurde.

Zur Realisierung des Programms wurden mehrere Bibliotheken benutzt. Die Wichtigsten sind:

- **pigpio:** pigpio erlaubt mehr und direktere Kontrolle über die programmierbaren GPIOs (General Purpose Input Outputs) des Raspberry Pi als die Standardbibliothek RPi.GPIO, was diese Bibliothek besser für zeitkritische Anwendung macht.
- **mysql.connector:** Zur Kommunikation mit der Datenbank wurde der MySQL-Konnektor für Python genutzt. Dieser Treiber ermöglicht den Datenaustausch mit einem MySQL-Server unter Python mittels Funktionen und Befehlen in der Datenbanksprache SQL.
- **Threading:** Da das System erfordert, mehrere Aufgaben zur gleichen Zeit bearbeiten zu können, kam das Modul threading zum Einsatz, das die Erstellung von Threads unter Python erlaubt.

## 19. Herausforderungen

### 19.1 Datenbankverbindung

Die erste Herausforderung war die Aufrechterhaltung der Datenbankverbindung während der Ausführung des Skripts. Der Fehler trat auf, weil das Skript in einer einzigen Sequenz lief. Das bedeutet, dass nur ein einzelner Prozess ausgeführt werden konnte. Das Problem wurde behoben, indem Threads erstellt wurden. Damit ist es möglich, mehrere Prozesse gleichzeitig ausführen zu können (s. Abb. 43).

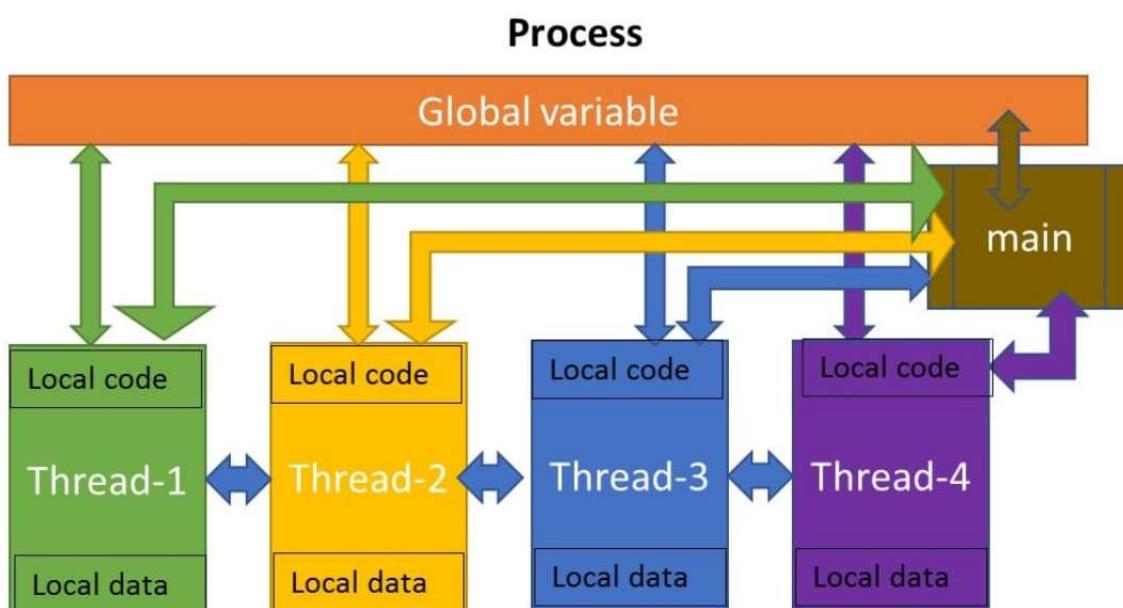


Abbildung 43: Multithreading in Python

Entscheidend war hierbei das Threading der Ablaufsteuerungen der Servomotoren. Diese würden ohne Threading aufgrund der Wartezeiten für die Servobewegungen die Datenbankverbindung blockieren.

In Abbildung 44 ist ein Code-Snippet eines Threads dargestellt. Sobald eine Ablaufsteuerung gefordert ist, wird ein Objekt erstellt. Dieses initialisiert einen Thread und startet sofort. In der Methode run() steht die eigentliche Ablaufsteuerung. Nachdem diese Methode durchlaufen wurde, löscht sich das Objekt selbst. Um mehrere gleichzeitige Ablaufsteuerungs-Threads zu verhindern, wird beim Starten und Beenden ein globales Flag gesetzt.

---

```

43  class Kaltstart_Thread(threading.Thread):
44      def __init__(self):
45          threading.Thread.__init__(self)
46          print("Kaltstart Thread wird gestartet")
47
48      def run(self):

```

Abbildung 44: Beispiel Thread

## 19.2 Frequenzzähler

Die nächste Herausforderung war ein inkonsistenter Frequenzzähler. Das Problem trat auf, weil anfangs jede steigende Flanke über eine bestimmte Zeit gezählt und daraus die Frequenz berechneten wurde. Das Problem wurde gelöst, indem die Zeit zwischen zwei steigenden Flanken, die Periode (s. Abb. 45), gemessen und daraus die Frequenz berechnet wurde.

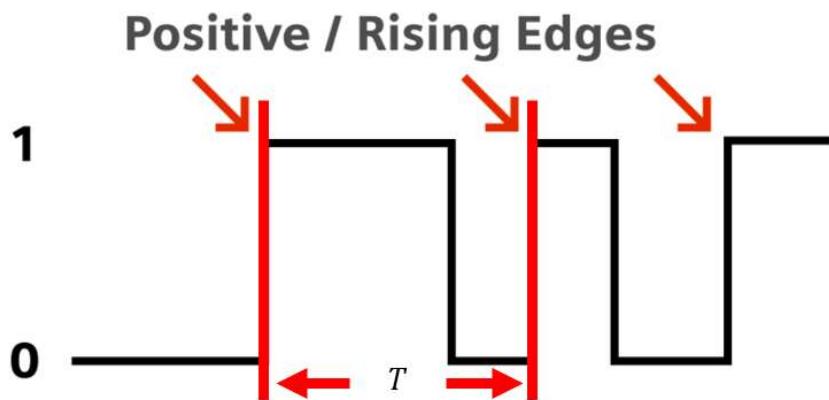


Abbildung 45: Frequenzmesssignal

Die pigpio-Bibliothek bietet hier eine genaue Lösung an (s. Abb. 46). Mittels „tick“ kann die Zeit ab dem Starten des Systems in Mikrosekunden bei Interruptaufruf gemessen werden. Durch Berechnung der Differenz aus zwei „ticks“ in Kombination mit dem Threaden der Frequenzberechnung kann hiermit eine ausreichende Genauigkeit erreicht werden. Um zusätzliche Schwankungen zu kompensieren, wird der Mittelwert aus mehreren Messungen mittels Liste berechnet.

```

337     def run(self):
338         while True:
339             self.diff = time() - self.Timer
340             if self.last_tick + 0.15 < time():
341                 self.frequenceclist.insert(0, 0)
342                 self.frequenceclist.pop()
343             sum = 0
344             l = self.frequenceclist
345             for i in range(0, self.frequenceclistlength):
346                 sum += l[i]
347
348             self.averagefrequence = sum / self.frequenceclistlength
349             self.RPM = self.averagefrequence * 60 * self.frequencedivider
350
351     def _cbf(self, gpio, level, tick):
352         self.last_tick = time()
353         t = tick - self._tick
354         self.frequence = (1000000.00 / t)
355         self.frequenceclist.insert(0, self.frequence)
356         self.frequenceclist.pop()
357         self._tick = tick

```

Abbildung 46: run- und Interrupt-Methode des Frequenzthreads

Allerdings bestand dann das Problem, dass der Interrupt nur bei einem positiven Flankenwechsel ausgelöst wird. Dies bedeutet, dass wenn keine Frequenz mehr vorhanden ist, keine neue Messung durchgeführt wird und somit der letzte Wert gespeichert bleibt. Um dies zu umgehen, wurde ein Timer programmiert, der eine 0 in die Liste schiebt, wenn eine bestimmte Zeit kein positiver Flankenwechsel auftritt.

19.3 PWM

Eine weitere Herausforderung war, ein konsistentes PWM-Signal zu erhalten. Es kam vor, dass die Stellglieder nicht wie gewünscht reagierten und sich unregelmäßig verhielten. Hervorgerufen wurde dieses Problem hauptsächlich durch das vom Microcontroller inkonsistent erzeugte PWM-Signal. Der Raspberry Pi erzeugt softwarebasierte PWM-Signale, die nur dann auftreten, wenn normale GPIO-Pins anstelle von PWM-Pins auf dem Pi verwendet werden.

Abbildung 47 zeigt verschiedene PWM-Signale, die ihren Modulationen entsprechen. Diese Modulationen können entsprechend den Wünschen des Benutzers eingestellt werden.

Das Problem wurde behoben, indem eine richtige Bibliothek verwendet wurde, die pigpio-Bibliothek anstelle der Standard-GPIO-Bibliothek, die RPi.GPIO-Bibliothek. Dies ermöglichte, ein korrekt generiertes PWM vom Mikrocontroller über die Standard-GPIO-Pins zu erhalten.

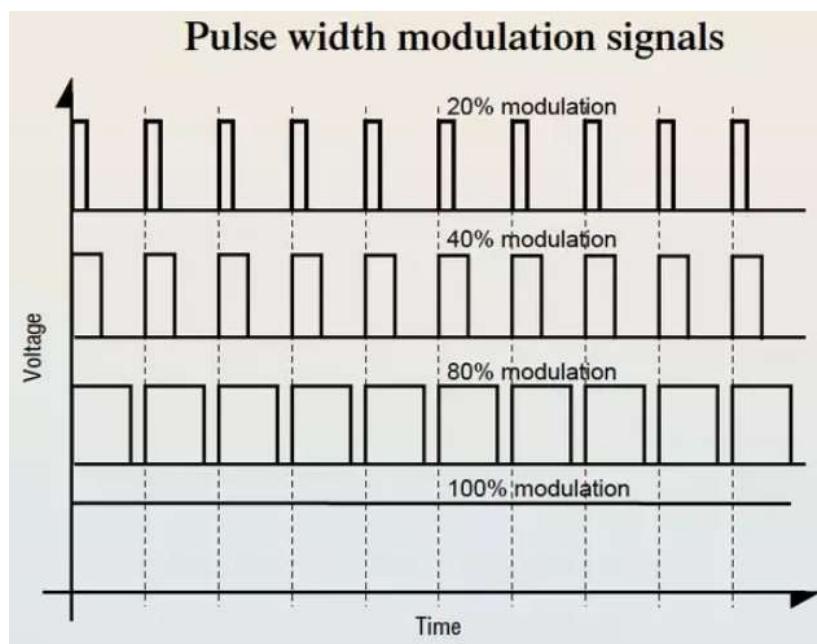


Abbildung 47: Pulsweitenmodulation

Pigpio bietet bereits eine Funktion zur Ansteuerung von Servomotoren an. Mit `set_servo_pulsewidth()` kann man ein PWM-Signal an einem beliebigen GPIO konfigurieren. Als Argumente werden der GPIO sowie die Pulsweite übergeben. Allerdings ist die Programmierung mittels Pulsweite speziell für mehrere Servomotoren umständlich, weswegen die Klasse Servo erstellt wurde, die die Funktion `set_servo_pulsewidth()` benutzt (s. Abb. 48).

```

20  class Servo:
21      def __init__(self, pi, GPIO, Nullposition = 1500, min_Winkel = -20, anschlag_Winkel
22          = 0, max_Winkel = 20): # Nullposition gibt die Mittelstellung in Pulsweite an,
23
24      def move(self, Winkel): # Bewegen mit Winkel, Umrechnung auf Pulsweite
25          self.Servo_Lib.set_servo_pulsewidth(self.GPIO, self.Nullposition + (Winkel * 10))
26          self.Position = Winkel
27
28      def gas_prozent(self, prozent): # Gassteuerung
29          self.Prozent = prozent
30          self.Position = self.anschlag_Winkel +
31              (prozent*(self.max_Winkel-self.anschlag_Winkel))/100
32          self.Servo_Lib.set_servo_pulsewidth(self.GPIO, self.Nullposition +
33              self.Position * 10)

```

Abbildung 48: Klasse Servo mit den zwei wichtigsten Methoden

Die Methode `move()` berechnet aus einem gegebenen Winkel mit der Nullposition und der Pulsweite für die Mittelstellung die benötigte Pulsweite. Hierbei sind Winkel im Uhrzeigersinn als positiv und Winkel gegen den Uhrzeigersinn als negativ definiert.

---

Zusätzlich war für die Gashebelsteuerung eine Methode nötig, um den Gashebel über LabVIEW prozentual steuern zu können. Die Methode `gas_prozent()` berechnet dafür aus den zuvor gemessenen Winkeln, der aus der Datenbank erhaltenen Prozentzahl und der Nullposition die benötigte Pulsweite.

Wenn sich die Stellglieder auch nach Verwendung der pigpio-Bibliothek noch zufällig verhalten, muss die Verdrahtung und die Schaltung überprüft werden. Es gilt die Masse oder den negativen Pin des Microcontrollers zusammen mit dem negativen Port des Netzteils zu verbinden. Das einfache Anschließen der Aktoren an die Stromversorgung würde nicht funktionieren, da der positive Signalpin zwischen den Aktoren und dem Microcontroller die Erdung vom Microcontroller selbst benötigt.

## 20. Quellen

### 20.1 Textquellen

- [1] <https://www.stihl.de/STIHL-Produkte/Motors%C3%A4gen-und-Kettens%C3%A4gen/S%C3%A4gen-f%C3%BCr-den-Privatanwender/22216-110/MS-170.aspx#>
- [2] <https://rn-wissen.de/wiki/index.php/Servos>
- [3] <https://www.electronics-tutorials.ws/de/dioden/zenerdiode.html>
- [4] <https://www.elektronik-kompendium.de/sites/grd/0205301.htm>
- [5] <https://www.mikrocontroller.net/articles/Schmitt-Trigger>
- [6] <http://www.cmos4000.de/cmos/4013.html>
- [7] <https://en.wikipedia.org/wiki/Raspbian>
- [8] [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
- [9] <https://www.raspberrypi.org/downloads/>
- [10] <https://www.raspberrypi.org/documentation/raspbian/updating.md>
- [11] <https://dev.mysql.com/doc/connector-python/en/connector-python-installation-binary.html>
- [12] <http://abyz.me.uk/rpi/pigpio/download.html>

---

## 20.2 Bildquellen

- Abb. 1: Dennis Weierter, „Entwicklung und Aufbau eines Systems zur Fernsteuerung von handgeführten Verbrennungsmotoren“, März 2017, S. 52
- Abb. 7: [https://static.stihl.com/upload/assetmanager/modell\\_imagefilename/scaled/zoom/02ec9102da0f4293aa89c76afa24ad91.jpg](https://static.stihl.com/upload/assetmanager/modell_imagefilename/scaled/zoom/02ec9102da0f4293aa89c76afa24ad91.jpg)
- Abb. 8: <https://www.stihl.de/STIHL-Produkte/Motors%C3%A4gen-und-Kettens%C3%A4gen/S%C3%A4gen-f%C3%BCr-den-Privatanwender/22216-110/MS-170.aspx>
- Abb. 9: [https://upload.wikimedia.org/wikipedia/commons/thumb/f/f6/Tiempos\\_Servo.svg/220px-Tiempos\\_Servo.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/f/f6/Tiempos_Servo.svg/220px-Tiempos_Servo.svg.png)
- Abb. 14: <https://www.mikrocontroller.net/wikifiles/5/57/Schmitt-trigger-diagramm.png>
- Abb. 17: <http://www.cmos4000.de/media/cmos/ic-cmos-4093.pdf>
- Abb. 18: <http://www.ti.com/lit/ds/symlink/cd4013b.pdf>
- Abb. 39: [https://cdn.shopify.com/s/file/1/0176/3274/products/rpi3p\\_angle\\_1024x1024.jpg?v=1542643374](https://cdn.shopify.com/s/file/1/0176/3274/products/rpi3p_angle_1024x1024.jpg?v=1542643374)
- Abb. 40: <https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2017/12/Features-of-python-01.jpg>
- Abb. 42: <https://www.bigmessowires.com/wp-content/uploads/2018/05/Raspberry-GPIO.jpg>
- Abb. 43: <http://www.webdevelopmenthelp.net/2017/08/python-interview-questions.html/multithreading-in-python>
- Abb. 45: <https://www.plcacademy.com/ladder-logic-tutorial-part-2/positive-rising-edge/>
- Abb. 47: <https://www.quora.com/What-is-use-of-pulse-width-modulation>

## 21. Anhang

### Anhang 1 – Programm

```

1  """
2  MYiTOSGermany
3  Python Subteam
4  Autoren : Danial Haris Bin Limi Hawari, Fabian Harlacher
5
6  Script zur Steuerung der Aktorik der Höhen- und Klimakammer des IKKU in Bruchsal per
7  Datenbank
8  """
9
9  import pigpio
10 from enum import Enum
11 from time import sleep
12 from time import time
13 import mysql.connector
14 from mysql.connector import errorcode
15 import threading
16
17
18
19 # Klasse Sevo zum einbinden von pigpio und Fahren mit integrierter Winkel zu
# Pulswerteumrechnung / Gassteuerung
20 class Servo:
21     def __init__(self, pi, GPIO, Nullposition = 1500, min_Winkel = -20, anschlag_Winkel
= 0, max_Winkel = 20): # Nullposition gibt die Mittelstellung in Pulswerte an,
durch Tests bestimmt
22         self.GPIO = GPIO
23         self.Nullposition = Nullposition
24         self.min_Winkel = min_Winkel
25         self.anschlag_Winkel = anschlag_Winkel
26         self.max_Winkel = max_Winkel
27         self.Prozent = 0
28
29         self.Servo_Lib = pi
30
31     def move(self, Winkel): # Bewegen mit Winkel, Umrechnung auf Pulswerte
32         self.Servo_Lib.set_servo_pulsewidth(self.GPIO, self.Nullposition + (Winkel * 10))
33         self.Position = Winkel
34
35     def gas_prozent(self, prozent): # Gassteuerung
36         self.Prozent = prozent
37         self.Position = self.anschlag_Winkel +
            (prozent*(self.max_Winkel-self.anschlag_Winkel))/100
38         self.Servo_Lib.set_servo_pulsewidth(self.GPIO, self.Nullposition +
            self.Position * 10)
39
40     def ausschalten(self):
41         self.Servo_Lib.set_servo_pulsewidth(self.GPIO, 0)
42
43 class Kaltstart_Thread(threading.Thread):
44     def __init__(self):
45         threading.Thread.__init__(self)
46         print("Kaltstart Thread wird gestartet")
47
48     def run(self):
49         global Kaltstart_Flag
50         global Warmstart_Flag
51         global Betrieb_Flag
52         global ChangeMode_Flag
53
54         if Warmstart_Flag:
55             print("Warmstart eingestellt, umstellen auf Kaltstart")
56             Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Warmstart)
57             sleep(0.5)
58             Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
59             sleep(0.5)
60             Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Kaltstart)
61

```

```

122         elif Betrieb_Flag:
123             print("Betrieb eingestellt, umstellen auf Kaltstart")
124             Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Betrieb)
125             sleep(0.5)
126             Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
127             sleep(0.5)
128             Gas_Servo.move(Winkel_Gas_max)
129             print("Gas drücken")
130             sleep(0.5)
131             Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Kaltstart)
132             print("Hebel auf Kaltstart")
133             sleep(0.5)
134             Gas_Servo.move(Winkel_Gas_min)
135             print("Gas auf min")
136
137     else:
138         print("Kaltstart einstellen")
139         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_aus)
140         sleep(0.5)
141         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
142         print("Hebel ausfahren")
143         sleep(0.5)
144         Gas_Servo.move(Winkel_Gas_max)
145         print("Gas drücken")
146         sleep(0.5)
147         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Kaltstart)
148         print("Hebel auf Kaltstart")
149         sleep(0.5)
150         Gas_Servo.move(Winkel_Gas_min)
151         print("Gas auf min")
152
153         sleep(0.5)
154         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren)
155         print("Hebel einfahren")
156         sleep(0.1)
157         #Ein wenig zurückfahren weil Einfahren hängt
158         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren+1)
159         print("Kaltstart eingestellt")
160
161         Warmstart_Flag = False
162         Betrieb_Flag = False
163         Kaltstart_Flag = True
164         ChangeMode_Flag = False
165
166         print("Kaltstart Thread wird geschlossen")
167         return
168
169
170
171 class Warmstart_Thread(threading.Thread):
172     def __init__(self):
173         threading.Thread.__init__(self)
174         print("Warmstart Thread wird gestartet")
175
176     def run(self):
177         global Kaltstart_Flag
178         global Warmstart_Flag
179         global Betrieb_Flag
180         global ChangeMode_Flag
181
182         if Kaltstart_Flag:
183             print("Kaltstart eingestellt, umstellen auf Warmstart")
184             Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Kaltstart)
185             sleep(0.5)
186             Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
187             sleep(0.5)
188             Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Warmstart)

```

```

129
130     elif Betrieb_Flag:
131         print("Betrieb eingestellt, umstellen auf Warmstart")
132         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Betrieb)
133         sleep(0.5)
134         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
135         sleep(0.5)
136         Gas_Servo.move(Winkel_Gas_max)
137         print("Gas drücken")
138         sleep(0.5)
139         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Kaltstart)
140         print("Hebel auf Kaltstart")
141         sleep(0.5)
142         Gas_Servo.move(Winkel_Gas_min)
143         print("Gas auf min")
144         sleep(0.5)
145         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Warmstart)
146         print("Hebel auf Warmstart")
147
148
149     else:
150         print("Warmstart einstellen")
151         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_aus)
152         sleep(0.5)
153         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
154         print("Hebel ausfahren")
155         sleep(0.5)
156         Gas_Servo.move(Winkel_Gas_max)
157         print("Gas drücken")
158         sleep(0.5)
159         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Kaltstart)
160         print("Hebel auf Kaltstart")
161         sleep(0.5)
162         Gas_Servo.move(Winkel_Gas_min)
163         print("Gas auf min")
164         sleep(0.5)
165         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Warmstart)
166         print("Hebel auf Warmstart")
167
168         sleep(0.5)
169         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren)
170         print("Hebel einfahren")
171         sleep(0.1)
172         #Ein wenig zurückfahren weil Einfahren hängt
173         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren+1)
174         print("Warmstart eingestellt")
175
176         Kaltstart_Flag = False
177         Betrieb_Flag = False
178         Warmstart_Flag = True
179
180         ChangeMode_Flag = False
181
182         print("Warmstart Thread wird geschlossen")
183         return
184
185
186     class Betrieb_Thread(threading.Thread):
187         def __init__(self):
188             threading.Thread.__init__(self)
189             print("Betrieb Thread wird gestartet")
190
191         def run(self):
192             global Kaltstart_Flag
193             global Warmstart_Flag
194             global Betrieb_Flag
195             global ChangeMode_Flag

```

```

196
197     if Kaltstart_Flag:
198         print("Kaltstart eingestellt, umstellen auf Betrieb")
199         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Kaltstart)
200         sleep(0.5)
201         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
202         sleep(0.5)
203         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Warmstart)
204         sleep(0.5)
205         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren)
206         print("Hebel einfahren")
207         sleep(0.1)
208         #Ein wenig zurückfahren weil Einfahren hängt
209         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren+1)
210         Gas_Servo.move(Winkel_Gas_max)
211         sleep(0.5)
212         Gas_Servo.move(Winkel_Gas_min)
213         print("Betrieb eingestellt")
214
215 elif Warmstart_Flag:
216     print("Warmstart eingestellt, umstellen auf Betrieb")
217     Gas_Servo.move(Winkel_Gas_max)
218     sleep(0.5)
219     Gas_Servo.move(Winkel_Gas_min)
220     print("Betrieb eingestellt")
221
222 else:
223     print("Betrieb einstellen")
224     Chokehebel_drehen_Servo.move(Winkel_Chokehebel_aus)
225     sleep(0.5)
226     Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
227     print("Hebel ausfahren")
228     sleep(0.5)
229     Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Betrieb)
230     print("Hebel auf Betrieb")
231     sleep(0.5)
232     Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren)
233     print("Hebel einfahren")
234     sleep(0.1)
235     #Ein wenig zurückfahren weil Einfahren hängt
236     Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren+1)
237     print("Betrieb eingestellt")
238
239 Kaltstart_Flag = False
240 Warmstart_Flag = False
241 Betrieb_Flag = True
242
243 ChangeMode_Flag = False
244 print("Betrieb Thread wird geschlossen")
245 return
246
247 class Motor_stopp_Thread(threading.Thread):
248     def __init__(self):
249         threading.Thread.__init__(self)
250         print("Betrieb Thread wird gestartet")
251
252     def run(self):
253         global Motor_an_Flag
254         global Kaltstart_Flag
255         global Warmstart_Flag
256         global Betrieb_Flag
257         global ChangeMode_Flag
258
259
260
261
262

```

---

```

263     if Betrieb_Flag:
264         Gas_Servo.move(Winkel_Gas_min)
265         sleep(0.5)
266         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Betrieb)
267         print("Drehen Betrieb")
268         sleep(0.5)
269         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
270         print("Ausfahren")
271         sleep(0.5)
272         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_aus)
273         print("Drehen Aus")
274         sleep(0.5)
275         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren)
276         print("Einfahren")
277
278     elif Warmstart_Flag:
279         Gas_Servo.move(Winkel_Gas_max)
280         sleep(0.5)
281         Gas_Servo.move(Winkel_Gas_min)
282         sleep(0.5)
283         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Betrieb)
284         print("Drehen Betrieb")
285         sleep(0.5)
286         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
287         print("Ausfahren")
288         sleep(0.5)
289         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_aus)
290         print("Drehen Aus")
291         sleep(0.5)
292         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren)
293         print("Einfahren")
294
295     elif Kaltstart_Flag:
296         Gas_Servo.move(Winkel_Gas_min)
297         sleep(0.5)
298         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_Kaltstart)
299         print("Drehen Betrieb")
300         sleep(0.5)
301         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_ausfahren)
302         print("Ausfahren")
303         sleep(0.5)
304         Chokehebel_drehen_Servo.move(Winkel_Chokehebel_aus)
305         print("Drehen Aus")
306         sleep(0.5)
307         Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren)
308         print("Einfahren")
309
310
311     Kaltstart_Flag = False
312     Warmstart_Flag = False
313     Betrieb_Flag = False
314     ChangeMode_Flag = False
315     print("Motor gestoppt")
316
317 class frequence_Thread(threading.Thread):
318     def __init__(self, pi, gpio, frequencedivider = 1):
319         threading.Thread.__init__(self)
320         print("Frequenz Thread wird gestartet")
321
322         self.pi = pi
323         self._tick = 0
324
325         self.frequencelistlength = 10
326         self.frequencelist = [0]*self.frequencelistlength
327         self.frequencedivider = frequencedivider
328         self.frequence = 0
329         self.averagefrequence = 0

```

---

```

330         self.RPM = 0
331         self.Timer = time()
332         self.last_tick = 0
333         self.pi.set_mode(gpio, pigpio.INPUT)
334
335         self._cb = self.pi.callback(gpio, pigpio.RISING_EDGE, self._cbf)
336
337     def run(self):
338         while True:
339             self.diff = time() - self.Timer
340             if self.last_tick + 0.15 < time():
341                 self.frequencelist.insert(0, 0)
342                 self.frequencelist.pop()
343             sum = 0
344             l = self.frequencelist
345             for i in range(0, self.frequencelistlength):
346                 sum += l[i]
347
348             self.averagefrequence = sum / self.frequencelistlength
349             self.RPM = self.averagefrequence * 60 * self.frequencedivider
350     def _cbf(self, gpio, level, tick):
351         self.last_tick = time()
352         t = tick - self._tick
353         self.frequence = (1000000.00 / t)
354         self.frequencelist.insert(0, self.frequence)
355         self.frequencelist.pop()
356         self._tick = tick
357
358     def cancel(self):
359         self._cb.cancel()
360
361 class Not_Aus_class:
362     def __init__(self, pi, gpio):
363         self.pi = pi
364         self.pi.set_mode(gpio, pigpio.INPUT)
365         self.pi.set_glitch_filter(gpio, 10000)
366         self._cb = self.pi.callback(gpio, pigpio.EITHER_EDGE, self._cbf)
367
368     def _cbf(self, gpio, level, tick):
369         global Motor_an_Flag
370         global Not_Aus_Flag
371         global Kaltstart_Flag
372         global Warmstart_Flag
373         global Betrieb_Flag
374         global Gas_Relais_GPIO
375         global Chokehebel_fahren_Relais_GPIO
376         global Chokehebel_drehen_Relais_GPIO
377         global Zustand
378
379         #Not aus betätigt
380         if level == 0:
381             Not_Aus_Flag = True
382             Motor_an_Flag = False
383             print("Notaus betätigt\n-----WICHTIG-----\nVor Lösen
des Notaus Chokehebel auf 0 stellen!")
384             Gas_Servo.ausschalten()
385             Chokehebel_fahren_Servo.ausschalten()
386             Chokehebel_drehen_Servo.ausschalten()
387             pi.write(Gas_Relais_GPIO, 0)
388             pi.write(Chokehebel_fahren_Relais_GPIO, 0)
389             pi.write(Chokehebel_drehen_Relais_GPIO, 0)
390             LED_off()
391             Zustand = Zustende.Aus
392             Kaltstart_Flag = False
393             Warmstart_Flag = False
394             Betrieb_Flag = False
395

```

---

```

396         if level == 1:
397             sleep(2) #Sichergehen dass die BewegungsThreads geschlossen sind
398
399             Relais_Setup()
400             Grundstellung()
401
402             Not_Aus_Flag = False
403             print("Notaus gelöst")
404
405     def cancel(self):
406         self._cb.cancel()
407
408 class Zustände(Enum):
409
410     Aus = "'Aus'"
411     Betrieb = "'Betrieb'"
412     Warmstart = "'Warmstart'"
413     Kaltstart = "'Kaltstart'"
414
415 def Relais_Setup():
416     global Gas_Relais_GPIO
417     global Chokehebel_fahren_Relais_GPIO
418     global Chokehebel_drehen_Relais_GPIO
419
420     pi.write(Gas_Relais_GPIO, 1)
421     print("Gas Relais zugeschalten")
422     sleep(0.5)
423     pi.write(Chokehebel_fahren_Relais_GPIO, 1)
424     print("Chokehebel fahren Relais zugeschalten")
425     sleep(0.5)
426     pi.write(Chokehebel_drehen_Relais_GPIO, 1)
427     print("Chokehebel drehen Relais zugeschalten")
428
429 # Grundstellung anfahren
430 def Grundstellung():
431     print("Grundstellung wird angefahren")
432     sleep(0.5)
433     Gas_Servo.move(Winkel_Gas_min)
434     Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren)
435     sleep(0.2)
436     Chokehebel_fahren_Servo.move(Winkel_Chokehebel_einfahren+1) # Rohr hängt, ein
437     bisschen zurückfahren
438     sleep(0.5) # Warten bis Chokehebel eingefahren ist bevor gedreht wird
439     Chokehebel_drehen_Servo.move(Winkel_Chokehebel_aus)
440     print("Grundstellung angefahren")
441     sleep(2)
442
443 # Datenbank verbindung
444 def db_connection():
445     global connection_counter
446     connection_counter = 0
447     while True:
448         try:
449             global cnx
450             global cursor
451
452             cnx = mysql.connector.connect(**db_config3)
453             cursor = cnx.cursor()
454             return
455         except mysql.connector.Error as err:
456
457             print("Database-Connection failed, Error: ", err)
458
459             print("Datenbank Verbindungsversuche : {}".format(connection_counter))
460             connection_counter += 1
461             sleep(5)

```

---

---

```

462
463 # Datenbankabfrage
464 def get_input():
465     global query_input
466     global Input
467     global Device
468     global Controlmode
469     global StartStopSignal
470     global Throttleposition
471     global cnx
472     global cursor
473
474     if cnx.is_connected():
475         try:
476             cursor.execute(query_input)
477             Input = cursor.fetchall()
478             Input = Input[0]
479             Device = Input[0]
480             Controlmode = Input[1]
481             StartStopSignal = Input[2]
482             Throttleposition = Input[3]
483         except mysql.connector.Error as err:
484             print("Connection lost")
485             print(err)
486             cursor.close()
487             cnx.close()
488             db_connection()
489     else:
490         print("Connection lost")
491         cursor.close()
492         cnx.close()
493         db_connection()
494
495
496
497 #Datenbank aktualisieren
498 def update_output(update, value):
499     global cnx
500     global cursor
501
502     if cnx.is_connected():
503         try:
504             cursor.execute(update % value)
505             cnx.commit()
506         except:
507             print("Connection lost")
508             cursor.close()
509             cnx.close()
510             db_connection()
511             update_output(update, value)
512     else:
513         print("Connection lost")
514         cursor.close()
515         cnx.close()
516         db_connection()
517         update_output(update, value)
518
519
520 def update_output_all():
521     global cnx
522     global cursor
523     global update_output_table
524     global Zustand
525     global Connection_Counter
526     global Motor_an_Flag
527     global Not_Aus_Flag
528

```

---

---

```

529     if cnx.is_connected():
530         try:
531             cursor.execute(update_output_table.format(Connection_Counter,
532                                         Motor_an_Flag, Zustand.value, frequence_Reader.RPM, Not_Aus_Flag))
533             cnx.commit()
534         except:
535             print("Connection lost")
536             cursor.close()
537             cnx.close()
538             db_connection()
539
540     else:
541         print("Connection lost")
542         cursor.close()
543         cnx.close()
544         db_connection()
545
546 def Motor_start():
547     #Led anschalten oder so
548     sleep(2)
549
550
551
552 def LED_off():
553     global LED_blue_GPIO
554
555
556     # Nullsetzen der LED Ausgänge
557     pi.write(LED_blue_GPIO, 0)
558
559
560
561
562 def LED_starting():
563     global LED_blue_GPIO
564
565     pi.write(LED_blue_GPIO, 1)
566
567
568 def LED_motor_an():
569     global LED_blue_GPIO
570
571     pi.write(LED_blue_GPIO, 0)
572
573
574 #gpio Zuweisung
575 pi = pigpio.pi()
576
577 # Pinzuweisung(in GPIO)
578 # Ausgänge
579 # PWM
580 Gas_GPIO = 14
581 Chokehebel_fahren_GPIO = 15
582 Chokehebel_drehen_GPIO = 18
583
584 #LED
585 LED_blue_GPIO = 8
586
587
588 # Relais
589 Gas_Relais_GPIO = 23
590 Chokehebel_fahren_Relais_GPIO = 24
591 Chokehebel_drehen_Relais_GPIO = 25
592
593
594

```

---

---

```

595 #Eingänge
596 Not_Aus_GPIO = 17
597 Frequenz_GPIO = 20 #normal 27, umgesteckt für Q1 Ausgang
598
599 # Angefahrenen Winkel der Servos deklarieren, bei Werkzeugwechsel anpassen
600
601 Winkel_Chokehebel_einfahren = 2
602 Winkel_Chokehebel_ausfahren = 30
603 Winkel_Chokehebel_aus = 58
604 Winkel_Chokehebel_Betrieb = 19
605 Winkel_Chokehebel_Warmstart = -8
606 Winkel_Chokehebel_Kaltstart = -28
607 Winkel_Gas_min = -26
608 Winkel_Gas_Anschlag = -19
609 Winkel_Gas_max = 5
610
611 # Servos deklarieren (Pin, Nullstellung, min Winkel, max Winkel)
612
613 Gas_Servo = Servo(pi, Gas_GPIO,1500, Winkel_Gas_min, Winkel_Gas_Anschlag, Winkel_Gas_max)
614 Chokehebel_fahren_Servo = Servo(pi, Chokehebel_fahren_GPIO, 1500)
615 Chokehebel_drehen_Servo = Servo(pi, Chokehebel_drehen_GPIO, 1500)
616
617 # Frequenz Thread definieren
618
619 frequence_Reader = frequence_Thread(pi, Frequenz_GPIO, 2)
620
621 # Not Aus Klasse deklarieren
622
623 Not_Aus = Not_Aus_class(pi, Not_Aus_GPIO)
624
625 # Zustand deklarieren
626
627 Zustand = Zustaende.Aus
628
629 #Ausgänge deklarieren
630 #Relais
631 pi.set_mode(Gas_Relais_GPIO, pigpio.OUTPUT)
632 pi.set_mode(Chokehebel_fahren_Relais_GPIO, pigpio.OUTPUT)
633 pi.set_mode(Chokehebel_drehen_Relais_GPIO, pigpio.OUTPUT)
634 # Pull Down der Relaisausgänge
635 pi.set_pull_up_down(Gas_Relais_GPIO, pigpio.PUD_DOWN)
636 pi.set_pull_up_down(Chokehebel_fahren_Relais_GPIO, pigpio.PUD_DOWN)
637 pi.set_pull_up_down(Chokehebel_drehen_Relais_GPIO, pigpio.PUD_DOWN)
638
639 #Nullsetzen der Relaisausgänge
640 pi.write(Gas_Relais_GPIO, 0)
641 pi.write(Chokehebel_fahren_Relais_GPIO, 0)
642 pi.write(Chokehebel_drehen_Relais_GPIO, 0)
643
644 #LEDs
645 pi.set_mode(LED_blue_GPIO, pigpio.OUTPUT)
646
647 # Pull Down der LEDausgänge
648 pi.set_pull_up_down(LED_blue_GPIO, pigpio.PUD_DOWN)
649
650 # Nullsetzen der LED Ausgnänge
651 pi.write(LED_blue_GPIO, 0)
652
653
654 # Flags
655
656 Kaltstart_Flag = False
657 Warmstart_Flag = False
658 Betrieb_Flag = False
659 ChangeMode_Flag = False
660 Motor_an_Flag = False
661 Not_Aus_Flag = False

```

---

---

```

662
663 # Datenbankkonfiguration
664
665 # Munirahs DB
666
667 db_config1 = { 'user': 'sql7291661',
668     'password' : '5NMuDuvLTZ',
669     'host' : 'sql7.freemysqlhosting.net',
670     'port' : '3306',
671     'database' : 'sql7291661',
672     'connect_timeout' : 5}
673
674 # Renes DB--> mit endgültiger Struktur
675
676 db_config2 = { 'user': 'sql2295094',
677     'password' : 'jJ8!xI1%',
678     'host' : 'sql2.freemysqlhosting.net',
679     'port' : '3306',
680     'database' : 'sql2295094'}
681
682 #Daniels DB
683 db_config3 = { 'user': 'VisiuLdKXI',
684     'password' : 'fdakQ9qTvL',
685     'host' : 'remotemysql.com',
686     'port' : '3306',
687     'database' : 'VisiuLdKXI',
688     'connect_timeout' : 5}
689
690 # Befehle für Datenbank request und write
691
692 query_input = "SELECT * from input"
693 update_Connection = "UPDATE output SET Statusconnection = %s"
694 update_Statusengine = "UPDATE output SET Statusengine = %s"
695 update_Startcounter = "UPDATE output SET Startcounter = %s"
696 update_mode = "UPDATE output SET StatusMode = %s"
697 update_RPM = "UPDATE output SET rpm = %s"
698 update_Not_Aus = "UPDATE output SET lokalEmergency = %s"
699
700 update_output_table = "UPDATE output SET Statusconnection = {}, Statusengine = {},
StatusMode = {}, rpm ={}, lokalEmergency = {}"
701
702 # Variablen für Datenbankabfragen
703 #Input
704 Input = None
705 Device = None
706 Controlmode = None
707 StartStopSignal = None
708 Throttleposition = None
709 connection_counter = 0
710
711
712 #Output
713
714 Connection_Counter = 0
715 Counter = 0
716
717
718 # Zeit
719 Motor_Timer = time()
720 Connection_Timer = 0
721
722
723
724
725
726
727

```

---

---

```

728 # Sonstige Variablen
729
730 if __name__ == "__main__":
731     frequencereader.start()
732     db_connection()
733     Relais_Setup()
734     Grundstellung()
735
736
737     try:
738         while True:
739             get_input()
740
741             # Connection Scheduler
742             if Connection_Timer + 0.2 < time():
743                 Connection_Counter += 1
744                 update_output_all()
745                 #print("Connection updated")
746                 print(frequencereader.averagefrequency, "\t", frequencereader.RPM)
747                 Connection_Timer = time()
748
749
750             if Device == "MS" and not Not_Aus_Flag:
751                 # Warm oder Kaltstart
752                 if not ChangeMode_Flag and Controlmode == 0 and not Kaltstart_Flag and
753                     StartStopSignal == 0 and not Not_Aus_Flag and not Motor_an_Flag:
754                     ChangeMode_Flag = True
755                     Kaltstart = Kaltstart_Thread()
756                     Kaltstart.start()
757                     Zustand = Zustaende.Kaltstart
758
759             elif not ChangeMode_Flag and Controlmode == 1 and not Warmstart_Flag:
760                 StartStopSignal == 0 and not Not_Aus_Flag and not Motor_an_Flag:
761                     ChangeMode_Flag = True
762                     Warmstart = Warmstart_Thread()
763                     Warmstart.start()
764                     Zustand = Zustaende.Warmstart
765
766             elif not ChangeMode_Flag and Controlmode == 3 and not Betrieb_Flag and
767                 StartStopSignal == 0 and not Not_Aus_Flag and not Motor_an_Flag:
768                     ChangeMode_Flag = True
769                     Betrieb = Betrieb_Thread()
770                     Betrieb.start()
771                     Zustand = Zustaende.Betrieb
772
773
774             # Motor starten
775             if (Kaltstart_Flag or Warmstart_Flag or Betrieb_Flag) and
776                 StartStopSignal == 1 and not Motor_an_Flag and not Not_Aus_Flag and not
777                 ChangeMode_Flag:
778                 if Counter == 0:
779                     LED_starting()
780
781                     if Motor_Timer + 10 < time():
782                         LED_off()
783                         if frequencereader.RPM > 1500:
784                             print("Motor an erkannt")
785                             Motor_an_Flag = True
786                             LED_motor_an()
787                             if Warmstart_Flag or Kaltstart_Flag:
788                                 Betrieb = Betrieb_Thread()
789                                 Betrieb.start()
790                                 Zustand = Zustaende.Betrieb
791
792             continue
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809

```

---

---

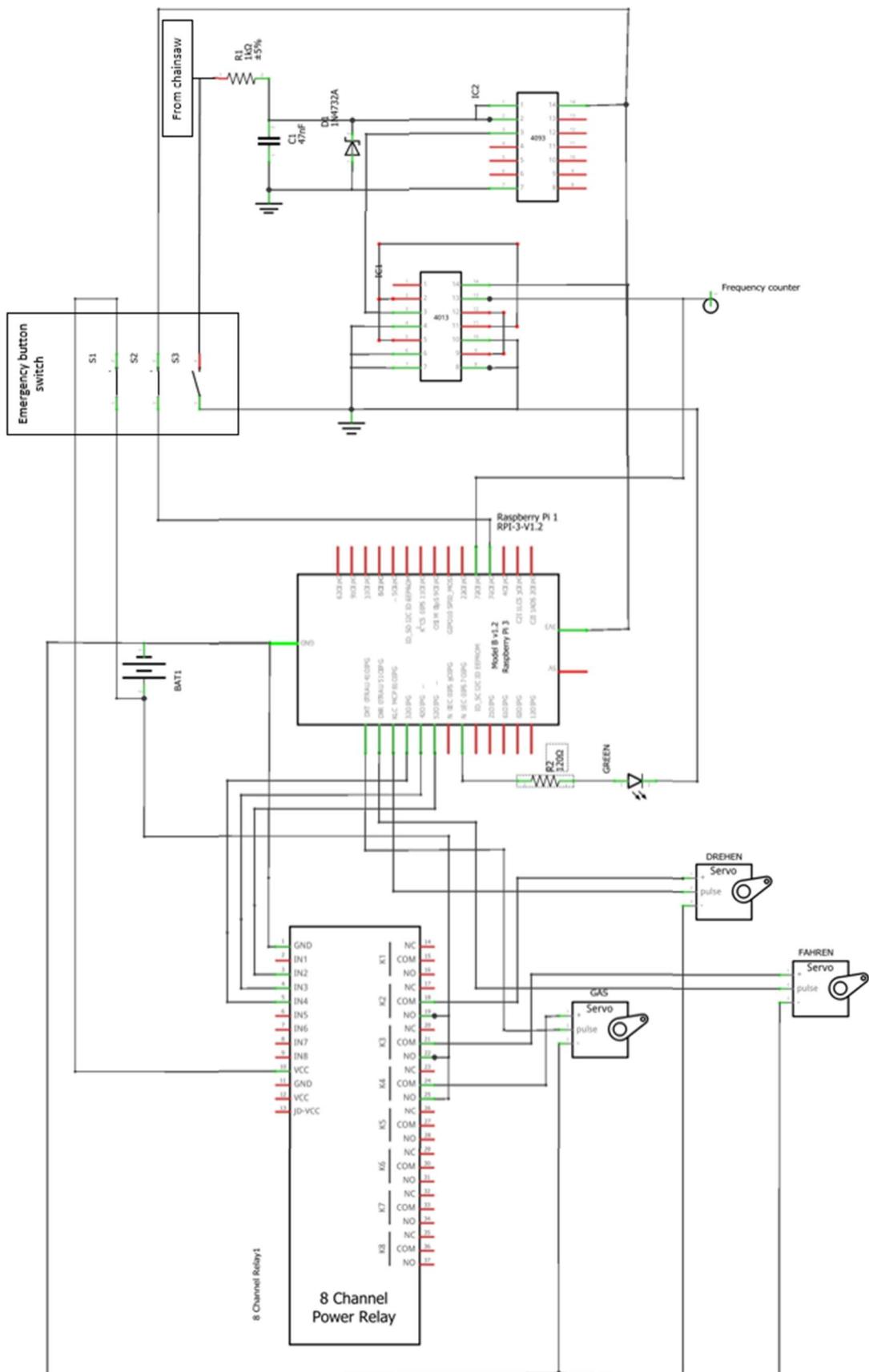
```

790             # Bei 5 Startversuchen in Warm oder Kaltstart auf Betrieb
791             umschalten
792             if Counter == 5 and (Kaltstart_Flag or Warmstart_Flag) and
793                 StartStopSignal == 1 and not Motor_an_Flag and not Not_Aus_Flag
794                 and not ChangeMode_Flag:
795                     ChangeMode_Flag = True
796                     print("Fünf Startversuche in Warm/Kaltstart, umstellen auf
797                         Betrieb")
798                     Betrieb = Betrieb_Thread()
799                     Betrieb.start()
800                     Zustand = Zustaende.Betrieb
801                     continue
802
803             if Motor_Timer + 15 < time():
804                 LED_starting()
805                 Counter += 1
806                 Motor_Timer = time()
807                 print("Motor Start\nAnzahl Startversuche: {}".format(Counter))
808
809             # Gassteuerung aktivieren
810             if (Kaltstart_Flag or Warmstart_Flag or Betrieb_Flag) and Motor_an_Flag
811                 and Gas_Servo.Prozent != Throttleposition and not Not_Aus_Flag:
812                     print("aktuelles Gas in Prozent: ", Throttleposition)
813                     Gas_Servo.gas_prozent(Throttleposition)
814
815             #Motor ausschalten
816             if Motor_an_Flag and (StartStopSignal == 0 or frequence_Reader.RPM ==
817                 0) and not Not_Aus_Flag:
818                 print("Motor war an, aussstellen")
819                 ChangeMode_Flag = True
820                 Motor_an_Flag = False
821                 LED_off()
822                 Counter = 0
823                 Motor_stopp = Motor_stopp_Thread()
824                 Motor_stopp.start()
825                 Zustand = Zustaende.Aus
826
827
828         except KeyboardInterrupt:
829             print("Anlage aus")
830             Gas_Servo.ausschalten()
831             Chokehebel_fahren_Servo.ausschalten()
832             Chokehebel_drehen_Servo.ausschalten()
833             pi.write(Gas_Relais_GPIO, 0)
834             pi.write(Chokehebel_fahren_Relais_GPIO, 0)
835             pi.write(Chokehebel_drehen_Relais_GPIO, 0)
836             pi.stop()
837             Not_Aus.cancel()
838             if cnx.is_connected():
839                 cursor.close()
840                 cnx.close()
841
842
843
844

```

---

## Anhang 2 - Schaltplan



## Anhang 3 -Gesprächsprotokolle

### Sitzungsprotokoll 1

<b>Teilnehmer:</b>  Dipl.-Phys. Ferhat Aslan(as) Prof. Dr.-Ing. Maurice Kettner(ke)	<b>Projektteam:</b>  Fabian Harlacher(fh) Danial Haris Bin Limi Hawari(ha) Nina Kurasch(ku) Aliaa Nabila Abdul Mutaali(mu) Munirah Nawi(na) René Schulreich(sc)  Protokollführer: Fabian Harlacher	<b>Datum:</b> 01.04.2019 <b>Beginn:</b> 09:00 <b>Ende:</b> 10:30 <b>Raum:</b> F101	
<b>Tagesordnungspunkte, Montag 01.04.2019</b>			
<b>Top 0:</b> <b>Top 1:</b> <b>Top 2:</b> <b>Top 3:</b>	Erstmaliges Treffen, Organisation Aufgabenstellung vordefinieren Einrichtung eines GitHub Ordners Kommunikation mit der in Malaysia arbeitenden Gruppe		
<b>Protokoll vom 01.04.19</b>			
		<b>Name</b>	<b>Termin</b>
<b>Top 0:</b>	Organisatorisches: - Abklärung der Vorgehensweise - Termin ausmachen mit Herrn M.Sc. Artur Martel bezüglich genauer Anforderungen und vorhandenen Dokumentationen	ke sc	01.04.2019 02.04.2019
<b>Top 1:</b>	Aufgabenstellung vordefinieren: - Schnittstellen Sensorik/Aktorik - Benutzeroberfläche - Labview, TeamViewer - Bedienung über App - Programmiersprachen - Python, C++ - Entwicklungsumgebung - QT - FailSafety - Webserver - Start der Werkzeuge - Umgang mit Verzögerungen – Vorgegebener Prüfablauf	ke, as	01.04.2019
<b>Top 2:</b>	Zur Kommunikation wurde vereinbart, Dokumentationen, Dokumente und Codes über GitHub im bereits existierenden Ordner für MYiTOPS abzulegen.	alle	01.04.2019
<b>Top 3:</b>	Zum Austausch mit der in Malaysia arbeitenden Gruppe wurde vereinbart eine WhatsApp Gruppe zum schnellen Austausch zu erstellen, und nach Bedarf über Skype zu kommunizieren.	mu	01.04.2019

## Sitzungsprotokoll 2

<b>Teilnehmer:</b>  M.Sc. Artur Martel(ma)	<b>Projektteam:</b>  Fabian Harlacher(fh) Danial Haris Bin Limi Hawari(ha) Nina Kurasch(ku) Aliaa Nabila Abdul Mutaali(mu) Munirah Nawi(na) René Schulreich(sc)  Protokollführer: Fabian Harlacher	<b>Datum:</b> 05.04.2019 <b>Beginn:</b> 14:30 <b>Ende:</b> 16:00 <b>Raum:</b> IKKU BR	
<b>Tagesordnungspunkte, Freitag 05.04.2019</b>			
<b>Top 0:</b> <b>Top 1:</b>	Treffen mit Herrn M.Sc. Artur Martel Weiteres Vorgehen		
<b>Protokoll vom 05.04.19</b>	<b>Name</b>	<b>Termin</b>	
<b>Top 0:</b>	Herr Martel erklärte uns detailliert den Aufbau der Höhen- und Klimakammer im IKKU Bruchsal. Außerdem baute er das bisherige System auf, sodass dieses getestet werden konnte. Im Hinblick auf unsere Projektarbeit wurden alle benötigten Prozesse und Anforderungen besprochen.	ma	05.04.2019
<b>Top 1:</b>	Die nächste Besprechung wurde auf Dienstag, den 09.04.2019, 15:30 Uhr gelegt. Bis dahin sollte sich jeder mit der Aufgabenstellung auf Basis des Gesprächs mit Herrn Martel befassen.	alle	09.04.2019

## Sitzungsprotokoll 3

<b>Teilnehmer:</b>	<b>Projektteam:</b> Fabian Harlacher(fh) Danial Haris Bin Limi Hawari(ha) Nina Kurasch(ku) Aliaa Nabila Abdul Mutaali(mu) Munirah Nawi(na) René Schulreich(sc)	<b>Datum:</b> 09.04.2019 <b>Beginn:</b> 15:30 <b>Ende:</b> 16:45 <b>Raum:</b> MU07
Protokollführer: Fabian Harlacher		
<b>Tagesordnungspunkte, Dienstag 09.04.2019</b>		
<b>Top 0:</b> Verlesen des Protokolls vom 05.04.19 <b>Top 1:</b> Organisation <b>Top 2:</b> Aufgabenstellung definieren <b>Top 3:</b> Bestellliste <b>Top 4:</b> Dropbox Ordner <b>Top 5:</b> nächste Besprechung		
<b>Protokoll vom 09.04.19</b>	<b>Name</b>	<b>Termin</b>
<b>Top 0:</b> Protokoll vom 05.04.19 wurde verlesen	fh	09.04.2019
<b>Top 1:</b> Der Antrag für den Raumzugang wurde gemeinsam ausgefüllt. Für die Sicherheitsunterweisung und der Unterschrift für die Aufgabenstellung wird ein Termin mit Herrn Kettner ausgemacht. Geeinigt wurde sich auf die kommende Woche am Mittwoch 4. Block, Donnerstag 1. Block oder freitags ab 13:00.	fh	09.04.2019
<b>Top 2:</b> Auf Basis des Gesprächs mit Herrn Martel am 05.04.19 wurde die Aufgabenstellung für das Projekt definiert. Nina Kurasch erklärte sich bereit, die Vorlage der Projektbeschreibung auszufüllen und mit Bildern zu versehen.	ku	10.04.2019
<b>Top 3:</b> Bezuglich der Bestellliste wurde sich geeinigt, dass sich jeder bis zum Termin mit Herrn Kettner Gedanken über für das Projekt nötige Bestellungen macht.	alle	16.04.2019
<b>Top 4:</b> Zusätzlich zu GitHub wurde sich geeinigt einen Dropbox Ordner anzulegen. Codes werden künftig in GitHub abgelegt, Dokumente etc. in Dropbox.	ha	10.04.2019
<b>Top 5:</b> Es wurde beschlossen, nach dem Treffen mit Herrn Kettner einen weiteren Termin für die gemeinsame Iliasanmeldung des Projekts und das weitere Vorgehen auszumachen.	alle	10.04.2019

## Sitzungsprotokoll 4

<b>Teilnehmer:</b>	<b>Projektteam:</b>	<b>Datum:</b>	16.04.2019
Prof. Dr.-Ing. Maurice Kettner(ke)	Fabian Harlacher(fh) Danial Haris Bin Limi Hawari(ha) Nina Kurasch(ku) Aliaa Nabila Abdul Mutaali(mu) Munirah Nawi(na) René Schulreich(sc)	<b>Beginn:</b>	15:40
Protokollführer: Fabian Harlacher			
<b>Tagesordnungspunkte, Dienstag 16.04.2019</b>			
<b>Top 0:</b> Besprechung der Projektbeschreibung <b>Top 1:</b> Sicherheitsunterweisung für das Labor <b>Top 2:</b> Rücksprache über benötigtes Material/Bestellliste <b>Top 3:</b> Freischaltung für Raumzugang <b>Top 4:</b> nächste Besprechung			
Protokoll vom 16.04.19	Name	Termin	
<b>Top 0:</b> Unsere vordefinierte Projektbeschreibung wurde Herrn Kettner vorgelegt, mit ihm besprochen und korrigiert. Die geänderte Fassung soll für die Unterschrift die kommenden Tage bei Herrn Kettner vorgelegt werden. Nina Kurasch erklärte sich bereit, die Änderungen vorzunehmen und die Projektbeschreibung vorzulegen.	ku	17.04.2019	
<b>Top 1:</b> Die Sicherheitsunterweisung für das Labor wurde von Herrn Kettner erteilt und von allen Anwesenden unterschrieben.	ke	16.04.2019	
<b>Top 2:</b> Mit Herrn Kettner wurde Rücksprache über die Bestelliste gehalten. Der Kostenrahmen beträgt 1500 €. Zusätzlich wurde eine Halterung aus Kunststoff für das Gestänge vorgeschlagen und in die Bestellliste aufgenommen. Die Liste soll zeitnah vervollständigt und bestellt werden.	ku, mu	01.05.2019	
<b>Top 3:</b> Es wurde sich geeinigt, im Anschluss an das Treffen mit Herrn Kettner die nun vollständigen Anträge für den Raumzugang zu Herrn Huber zu bringen.	ku, fh	16.04.2019	
<b>Top 4:</b> Die nächste Besprechung für die gemeinsame Iliasmeldung und das weiter Vorgehen wurde auf Donnerstag, den 18.04.19, 11:30 Uhr gelegt.	alle	16.04.2019	

## Sitzungsprotokoll 5

<b>Teilnehmer:</b>	<b>Projektteam:</b> Fabian Harlacher(fh) Danial Haris Bin Limi Hawari(ha) Nina Kurasch(ku) Aliaa Nabila Abdul Mutaali(mu) Munirah Nawi(na) René Schulreich(sc)	<b>Datum:</b> 18.04.2019 <b>Beginn:</b> 11:30 <b>Ende:</b> 13:00 <b>Raum:</b> F101
Protokollführer: Fabian Harlacher		
<b>Tagesordnungspunkte, Donnerstag 18.04.2019</b>		
<b>Top 0:</b> <b>Top 1:</b> <b>Top 2:</b> <b>Top 3:</b>	Verlesen des Protokolls vom 16.04.19 Iliasanmeldung des Projekts Untergliedern der Projektaufgabe nächste Besprechung	
<b>Protokoll vom 18.04.19</b>	<b>Name</b>	<b>Termin</b>
<b>Top 0:</b>	Protokoll vom 16.04.19 wurde verlesen.	fh 18.04.2019
<b>Top 1:</b>	Gemeinsam wurde die Projektbeschreibung in Ilias hochgeladen. Anschließend meldete sich jeder einzelne für das Projekt im entsprechenden Ilias Kurs an.	alle 18.04.2019
<b>Top 2:</b>	Eine Untergliederung der Projektaufgabe wurde festgelegt: - 1. Team: Einrichtung der Verbindung zu Raspberry Pi mittels Labview, User Interface erstellen - 2. Team: Arduino Code in Python übersetzen und ergänzen (Ablaufsteuerung MS, FS, Zeiten, Zähler für Starten) - 3. Team: Drehzahlsteuerung, Überwachen des Motors, Sicherheitsaspekte Bis zur nächsten Besprechung soll sich jeder für eine Untergruppe entscheiden.	alle 24.04.2019
<b>Top 3:</b>	Die nächste Besprechung wurde auf Mittwoch, den 24.04.19, 11:30 Uhr gelegt.	alle 24.04.2019

## Sitzungsprotokoll 6

<b>Teilnehmer:</b>  <b>Projektteam:</b> Fabian Harlacher(fh) Danial Haris Bin Limi Hawari(ha) Nina Kurasch(ku) Aliaa Nabila Abdul Mutaali(mu) Munirah Nawi(na) René Schulreich(sc)		<b>Datum:</b> 24.04.2019 <b>Beginn:</b> 11:30 <b>Ende:</b> 13:30 <b>Raum:</b> MU07	
Protokollführer: Fabian Harlacher			
<b>Tagesordnungspunkte, Mittwoch 24.04.2019</b>			
<b>Top 0:</b> <b>Top 1:</b> <b>Top 2:</b> <b>Top 3:</b>	Verlesen des Protokolls vom 16.04.19 Zuweisung der Unterteams Skype Anruf mit Projektteam in Malaysia Weiteres Vorgehen		
<b>Protokoll vom 24.04.19</b>		<b>Name</b>	
<b>Top 0:</b>	Protokoll vom 18.04.19 wurde verlesen.	fh	24.04.2019
<b>Top 1:</b>	Die Unterteams wurden folgendermaßen zugewiesen: - 1. Team: Einrichtung der Verbindung zu Raspberry Pi mittels Labview, Userinterface erstellen Munirah Nawi, René Schulreich - 2. Team: Arduino Code in Python übersetzen und ergänzen (Ablaufsteuerung MS, FS, Zeiten, Zähler für Starten) Danial Haris Bin Limi Hawari, Fabian Harlacher - 3. Team: Drehzahlsteuerung, Überwachen des Motors, Sicherheitsaspekte Aliaa Nabila Abdul Mutaali, Nina Kurasch	na, sc	24.04.2019
<b>Top 2:</b>	Zum Austausch über aktuellen Stand und Tipps wurde ein Skype Anruf mit dem Team in Malaysia getätigt. Inhalte des Workshops in Malaysia, das Steuern einer Platine über Raspberry und Internet wurde gemeinsam wiederholt.	ha, fh	24.04.2019
<b>Top 3:</b>	Um schnellstmöglich anfangen zu können, werden zwei zusätzliche Raspberry Pi 3b+ auf eigene Kosten bestellt. Die Erstattung folgt später mit der Bestellliste.  Vorerst wurde kein weiterer Termin für die nächste Besprechung vereinbart. Die Untergruppen organisieren sich untereinander.	mu, ku alle	01.05.2019 24.04.2019

## Scrum Meeting 1

Datum: 07.05.2019

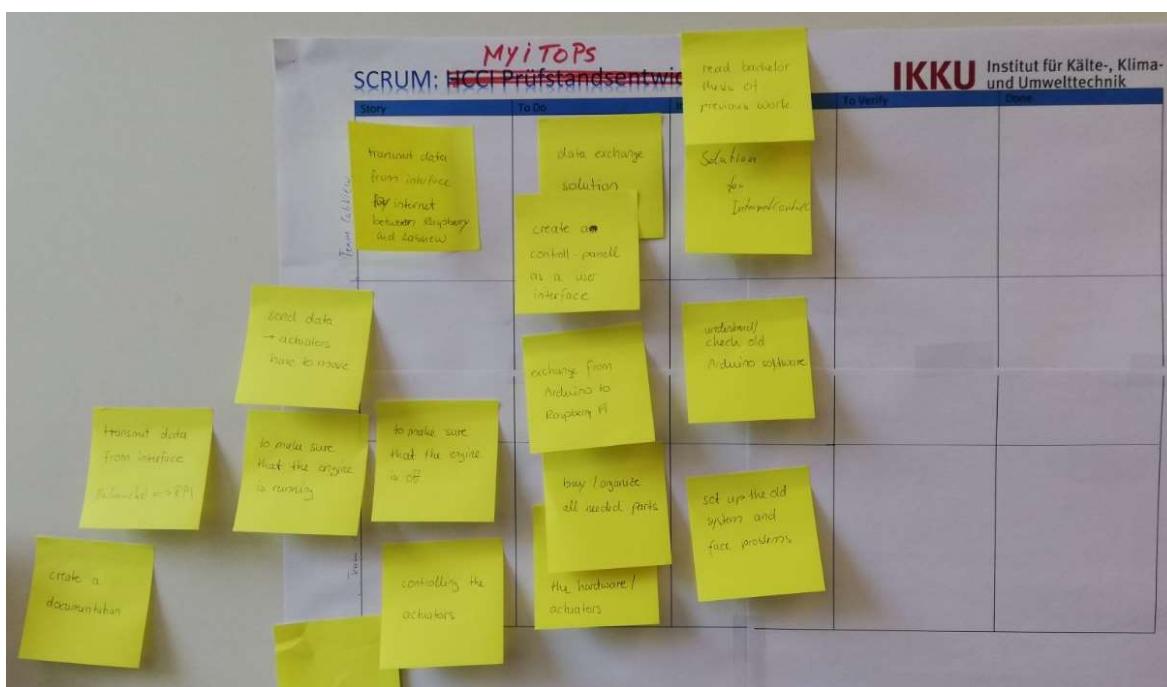
Zeitraum: 10:00 – 12:00 Uhr

Raum: LI043

Teilnehmer: Dipl.-Phys. Ferhat Aslan, Projektgruppe

Inhalt:

- Ferhat Aslan stellte der Projektgruppe die SCRUM-Methode vor und empfiehlt diese bis zur Fertigstellung des Projektes zu nutzen. Das SCRUM Board ist wie folgt aufgebaut: es gibt 3 Zeilen, je Untergruppe eine, und 5 Spalten: „Story“, „to do“, „in progress“, „to verify“, „done“. Bei der SCRUM-Methode werden alle Anforderungen als Story aufgeschrieben und derjenigen Untergruppe zugeteilt, deren Aufgabe es ist, sich um die Umsetzung zu kümmern. Es können im Laufe des Projekts Anforderungen hinzugefügt werden.
- Die Projektgruppe einigte sich auf die Nutzung der SCRUM-Methode. Es wurde direkt umgesetzt. Jedes Projektmitglied machte einen oder mehrere Story- Vorschläge. Im Anschluss wurde darüber diskutiert - wenn die Story angenommen wurde, wurde sie auf ein Post-it notiert und darüber entschieden, welcher Untergruppe sie zugewiesen wird. Die teamübergreifenden Storys wurden zwischen den Zeilen angeordnet.
- Die Projektgruppe beschloss, sich immer montags im 3. Block um 11:30 Uhr in LI043 zu treffen. Jede Untergruppe soll dann von seinen Fortschritten berichten, das SCRUM-Board wird dementsprechend aktualisiert.
- Es wurden 2 Meilensteine gesetzt: das Projekt soll bis zum 26. Juli fertiggestellt werden. Bis Ende Mai muss jede Untergruppe ein Konzept erarbeitet haben.



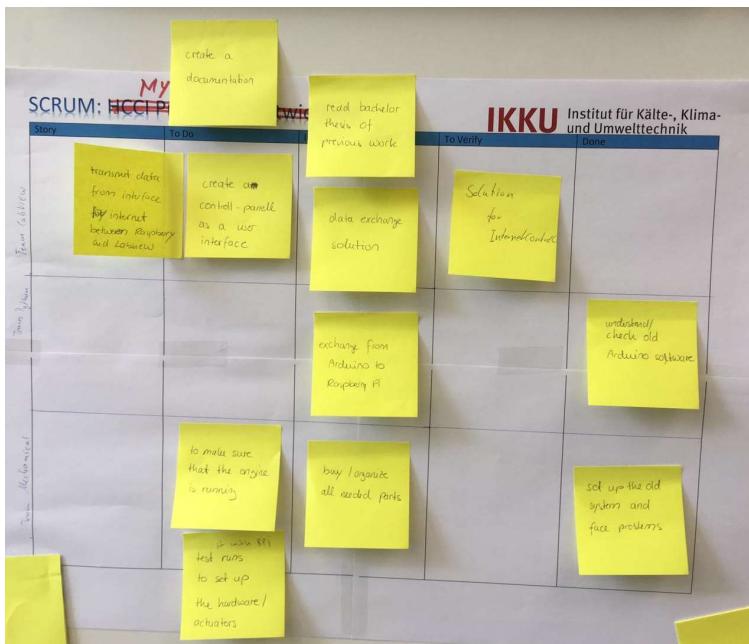
## Scrum Meeting 2

Datum: 13.05.2019

Zeitraum: 11:30 – 12:00 Uhr

Raum: LI043

Teilnehmer: Dipl.-Phys. Ferhat Aslan, Projektgruppe



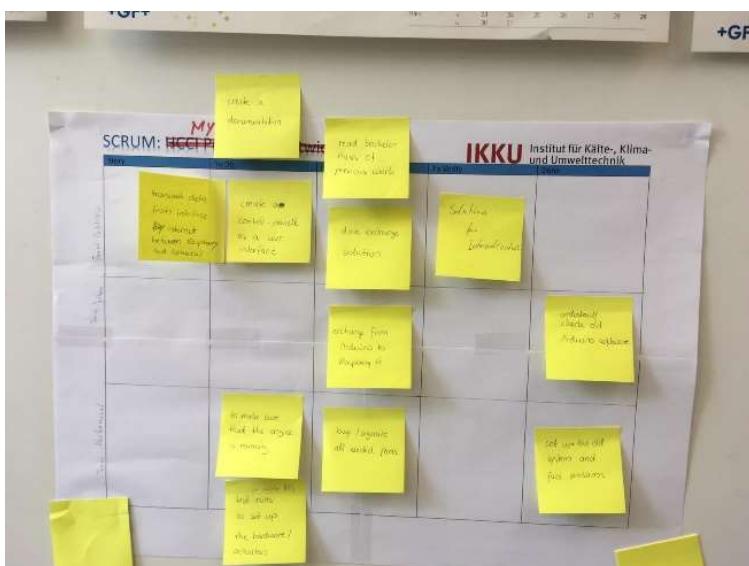
## Scrum Meeting 3

Datum: 20.05.2019

Zeitraum: 11:30 – 12:10 Uhr

Raum: LI043

Teilnehmer: Dipl.-Phys. Ferhat Aslan, Projektgruppe



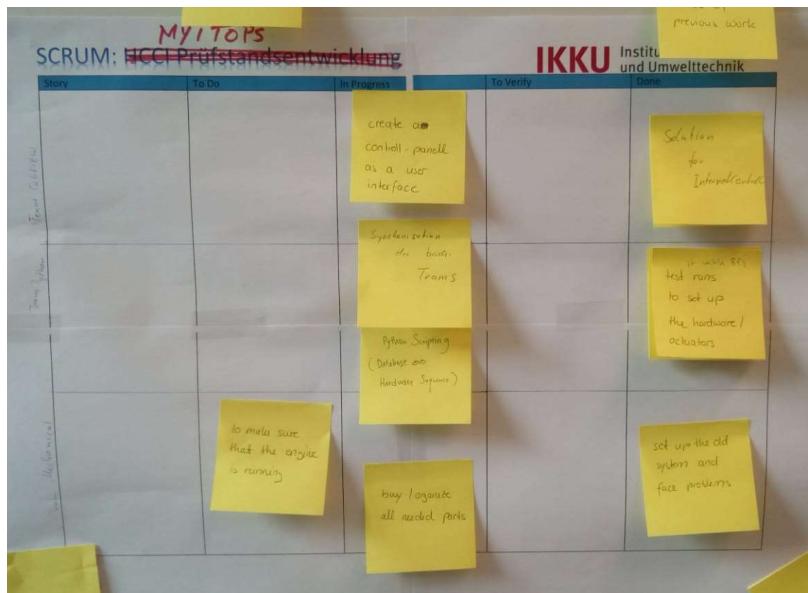
## Scrum Meeting 4

Datum: 27.05.2019

Zeitraum: 11:30 – 12:15 Uhr

Raum: LI043

Teilnehmer: Dipl.-Phys. Ferhat Aslan, Projektgruppe



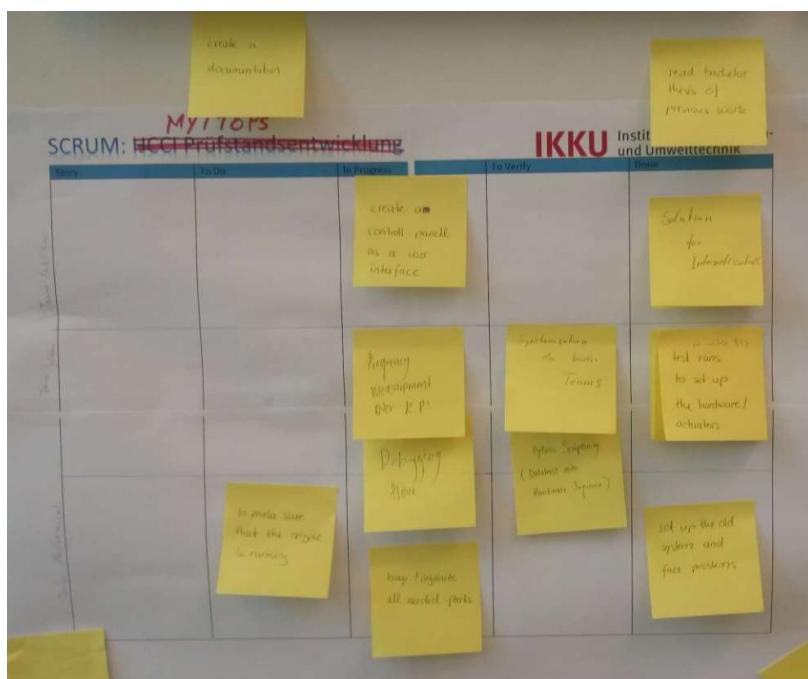
## Scrum Meeting 5

Datum: 03.06.2019

Zeitraum: 11:30 – 12:00 Uhr

Raum: LI043

Teilnehmer: Dipl.-Phys. Ferhat Aslan, Projektgruppe



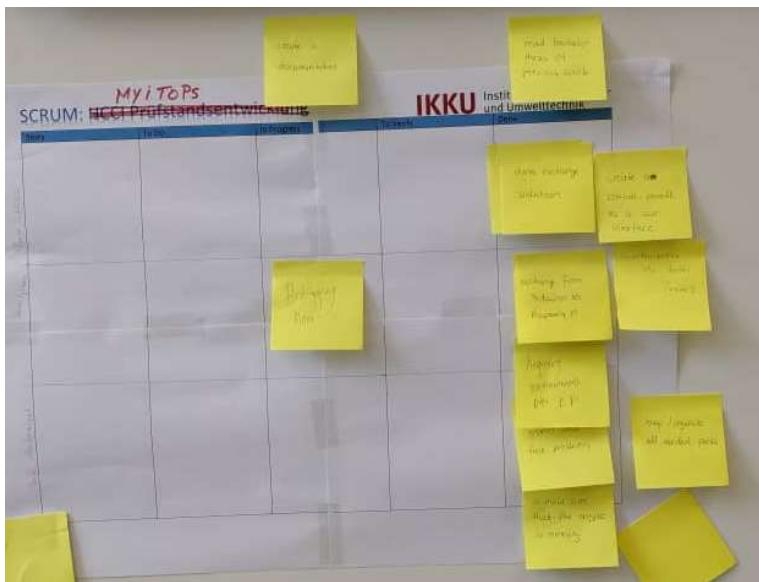
## Scrum Meeting 6

Datum: 17.06.2019

Zeitraum: 11:30 – 12:30 Uhr

Raum: LI043

Teilnehmer: Dipl.-Phys. Ferhat Aslan, Projektgruppe



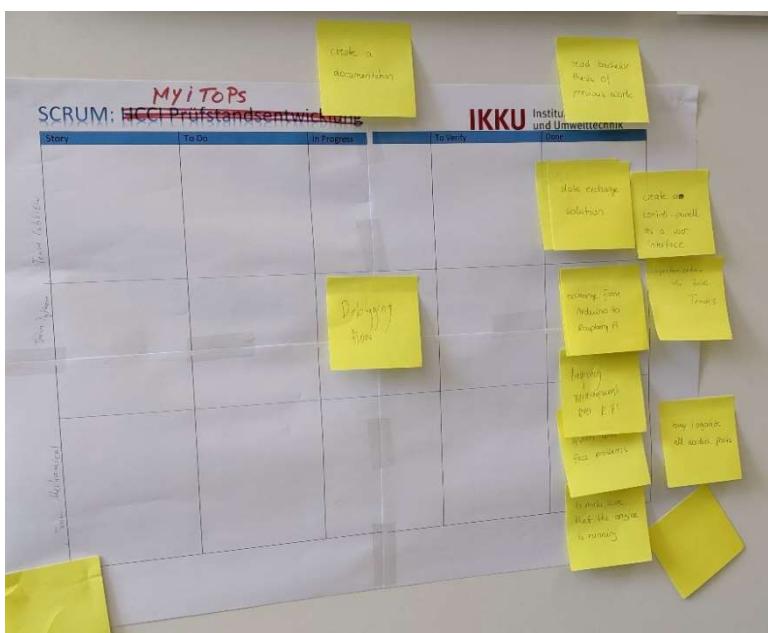
## Scrum Meeting 7

Datum: 24.06.2019

Zeitraum: 11:30 – 12:00 Uhr

Raum: LI043

Teilnehmer: Dipl.-Phys. Ferhat Aslan, Projektgruppe



## Scrum Meeting 8

Datum: 01.07.2019

Zeitraum: 11:30 – 12:0 Uhr

Raum: LI043

Teilnehmer: Dipl.-Phys. Ferhat Aslan, Projektgruppe

