Hochschule für angewandte
Wissenschaften München

Fakultät 03 für Maschinenbau,
Flug- und Fahrzeugtechnik

Weiterentwicklung der Steuerung und Regelung zur Steigerung der Reproduzierbarkeit eines Kaffeerösters

STUDIENARBEIT

in Kooperation mit Kaffeewerkstatt München UG

Vorgelegt von:

Andreas Klugbauer (MBB)
Matrikelnummer: 52188414
andiklugbauer@gmx.de

Abderrahim Elaoud (FAD)
Matrikelnummer: 05019911
elaoud_2012@hotmail.de

Betreuer: Dipl.-Ing. (FH) Rohnen Armin

Unternehmensvertreter: Herr Erich Weidler

Ort und Datum: München, den 26.01.2018

Kurzfassung

In der folgenden Arbeit geht es um den Entwurf sowie die Herstellung und die Inbetriebnahme eines Prototyps zum Vollautomatisieren eines Kaffeerösters. Ziel der Arbeit ist hierbei, die vorher ermittelten Einflussparameter wie Temperatur oder Zeitverlauf auf das Rösten einzubeziehen, um eine höchstmögliche Reproduzierbarkeit des Röstvorgangs zu gewährleisten.

Abstract

This thesis is about designing and creating a prototype for implementing a full automatic mode of a coffee-roaster. The intended achievement of this thesis is to include influential parameters calculated beforehand, such as temperature or time response in order to warrant highest possibility of repeatability of the roasting process.

Inhaltsverzeichnis

Abkürzungsverzeichnis

abs	absolut	
Abs. Temp	Abschalttemperatur	[°C]
bt1	Byte Nummer 1	
D	Tag	
datevec	Datumsvektor	
Dp	Differenzdruck	[Pa]
FC	Funktioscode	
GPIO	general purpose input/output	
grd	Gradient	
GUI	GUIDE	
H	Stunde	
I2C	Inetr-Integrated Circuit	
lst	Liste	
M	Monat	
MN	Minuten	
NaN	Not-a-Number	
Op	Hochdruck	[Pa]
P2000	Parameter 2000 des Frequenzumrichters	
RP	Raspberry Pi	
RS	Recommended Standard	
RTU	Remote Terminal Unit	
s	Sekunden	
SCL	Serial Clock	
SDA	Serial Data	
SSH	Secure Shell Anwendung	
T1	Trennen der Verbindung 2	
T2	Trennen der Verbindung 1	
UART	Universal Asynchronous Receiver Transmitter	
v	Strömungsgeschwindigkeit	[m/s]
y	Hauptsollwert	
Y	Jahr	
ρ	Luftdichte	[kg/m^3]

1 Einleitung und Aufgabenstellung

Deutschlandweit gibt es in etwa 600 kleine und traditionelle Kaffeeröstereien, die ihren Kaffee selbst auf die herkömmliche Art und Weise rösten. Das bedeutet zunächst, dass die Kaffeebohnen in 70kg Jutesäcken angeliefert werden und nicht wie bei der industriellen Röstung in Kunststoffsäcken in Containern. Auch der Röstvorgang unterscheidet sich deutlich. In der industriellen Röstung werden die Bohnen bei sehr hoher Temperatur nur wenige Minuten geröstet. Dabei erhält der Kaffee eine bittere und saure Note. Beim hiesigen Röstvorgang hingegen dauert dieser Vorgang je nach Röstprofil 12 bis 20 Minuten. Die Rösttrommel muss zunächst vorgeheizt werden, anschließend werden die Kaffeebohnen eingefüllt und die Temperatur erhöht sich langsam und wird ab einer bestimmten Bohnentemperatur wieder verringert. Auf diese Weise werden die Bohnen schonender geröstet und verlieren so ihre Bitterstoffe. Bei dieser Art des Röstens ist es allerdings schwierig, alle Einflussfaktoren wie optimale Temperatur, Außentemperatur, Qualität der Bohnen etc. zu berücksichtigen und immer zum gleichen Röstergebnis zu gelangen. Meistens orientiert sich der Kaffeerösterei-Meister an seiner eigenen Erfahrung, was dazu führt, dass jeder Röstvorgang zu einem anderen Ergebnis führt.

Deswegen sollte nun ein Prototyp eines vollautomatisierten Reglers entworfen und hergestellt werden, bei dem alle Einflussparameter miteinbezogen werden, um zu einem optimalen Röstvorgang zu gelangen und diesen möglichst immer zu reproduzieren.

Zum einen muss die Temperatur der Luft, die in den Kaffeeröster einströmt, gemessen werden, sowie die Temperatur der Bohnen und die ausströmende Luft. Bei zu geringer oder zu hoher Hitze muss der Brenner durch einen Regler ab- oder eingeschaltet werden, um ein optimales Röstergebnis zu erlangen. Auch die Drehzahl der Trommel muss gemessen und reguliert werden.

In der folgenden Arbeit geht es deswegen, um einen solchen Regler, der diese notwendigen Komponenten misst, auswertet und reguliert.

Die Arbeit lässt sich in drei größere Teile untergliedern. Zuerst wird erläutert, welche Software benötigt wurde sowie die Programmierung dieser, um die benötigten Daten zu messen, weiterzuleiten, auszuwerten und nötigenfalls zu korrigieren. Im Anschluss daran werden die verwendeten Bauteile vorgestellt und erklärt. Im letzten Teil geht es um die Herstellung und die Inbetriebnahme dieses Reglers.

Wichtigstes Ziel des Projekts ist, dass der Vorgang des Kaffeeröstens beliebig oft und zu jedem Zeitpunkt mit dem möglichst gleichen optimalen Ergebnis durchgeführt werden kann.

2 Hinführung und Vorüberlegungen

Um ein besseres Verständnis zu gewährleisten, ist zuerst eine grobe Skizze abgebildet, auf der zu sehen ist, welche Bauteile der Regler beinhaltet, wie die Kommunikation zwischen den einzelnen Elementen aufgebaut ist und wie diese mit dem Kaffeeröster verbunden sind.

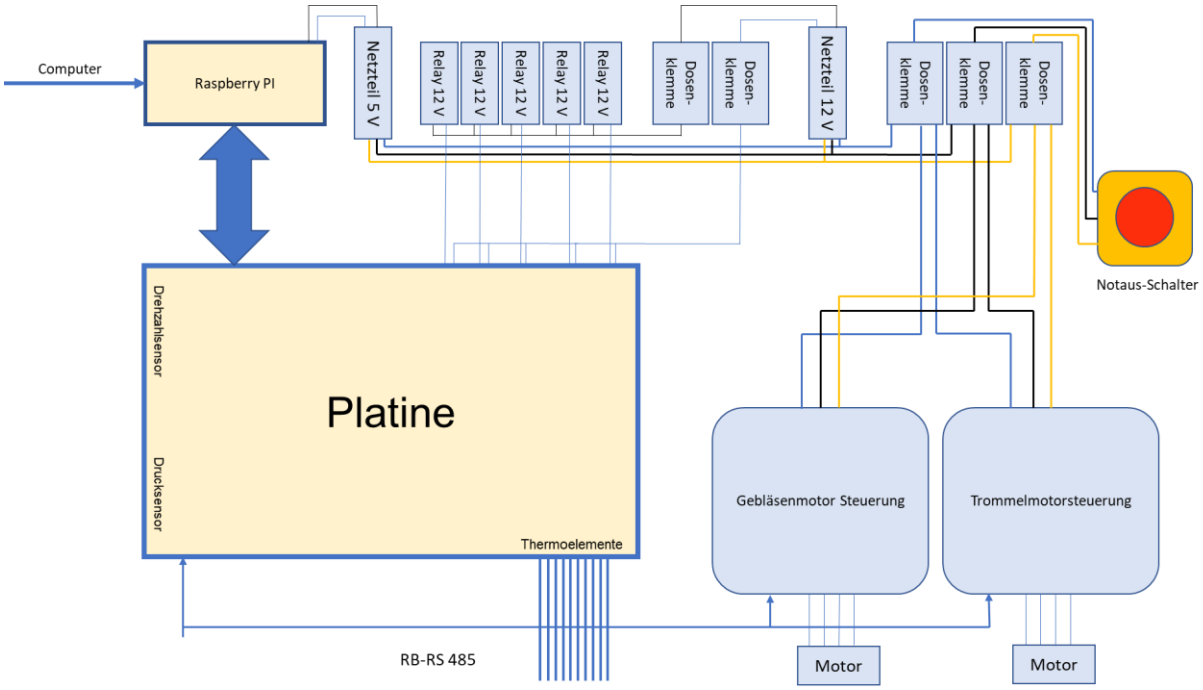

Abbildung 1: Skizze des Gesamtkonzepts

2.1 Eingesetzter Raspberry PI und benötigtes Material

Als Mikrocontroller wurde ein Raspberry Pi 3 Modell B verwendet und zudem eine SD-Karte, um das Betriebssystem Raspbian Jessie für das Raspberry darauf herunterzuladen, da dieses keinen integrierten Speicher besitzt. Zudem wird ein 5V-Netzteil zur Versorgung des Raspberry Pi benötigt sowie ein LAN-Kabel zum Verbinden des Raspberry Pi mit dem Laptop und ein HDMI-Kabel. Um das Raspberry Pi in Betrieb zu nehmen und die Einstellungen durchzuführen, braucht man außerdem eine Tastatur, einen Bildschirm und eine Maus.

2.2 Erststart des Raspberry Pi

Um das Raspberry PI in Betrieb zu nehmen, muss als erstes das Betriebssystem Raspbian Jessie von der frei zugänglichen Homepage:
https://www.raspberrypi.org/downloads/raspbian/
auf einen Computer heruntergeladen werden. Hierbei muss man darauf achten, welche Art Betriebssystem man verwenden möchte. Zum einen gibt es die Version „Desktop“ bei der wie von einem herkömmlichen Computer bekannte Icons angezeigt werden. Zum anderen gibt es die Version „Lite“, bei der man ein schwarzes Terminal erhält, bei dem die Befehle über bestimmte Eingabebefehle erfolgen. In diesem Projekt wurde die „Lite“ Version verwendet.
Nach dem Herunterladen des Betriebssystems kann dieses mit Hilfe eines bestimmten Tools auf die SD-Karte installiert werden. Diese Karte wird nun in die dafür vorgesehene Stelle des Raspberry Pi eingesteckt. Nun kann man den Bildschirm, die Tastatur sowie die Maus an das Raspberry anschließen und dieses nun als Computer bedienen.

2.3 Bedienung des Raspberry Pi über Windows SSH-Anwendung

Um über Windows auf das Raspberry Pi zugreifen und bedienen zu können, muss diesem als erstes eine IP-Adresse zugewiesen werden. Dies erfolgt, indem man zunächst auf folgende Datei zugreift:

```
sudo nano /etc/network/interfaces
```

Jetzt ersetzt beziehungsweise erweitert man die ursprüngliche Datei:

```
iface eth0 inet dhcp
```

durch folgende:

```
iface eth0 inet Static
adress 192.168.178.28
netmask 255.255.255.0
gateway 192.168.178.1
```

Zuletzt muss man mit dem Befehl

```
sudo nano /etc/init.d/networking restart
```

das Netzwerkdienstprogramm neu starten, um die Einstellungen zu übernehmen.

Nun kann man Raspberry Pi mittels der zugewiesenen IP-Adresse mit dem Laptop, der mit dem Betriebssystem Windows ausgestattet ist, über die SSH-Schnittstelle verbinden.

2.4 Verbindung des Raspberry Pi mit Matlab

Zum Schluss muss das Raspberry Pi mit Matlab verbunden werden, um Variablen, Skripte, Programme bzw. Werte etc. auslesen und eingeben zu können. Um dies zu realisieren, muss zuerst über Add-Ons Matlab Support Package für Raspberry Pi Hardware heruntergeladen werden. Nach der Installation dieses Packages kann man mit dem folgenden Matlab Befehl die Verbindung zwischen Raspberry Pi und Matlab herstellen:

```
name = raspi ('IP-Adresse','raspberry pi name','Passwort des Raspberry Pi')
```

`name` wird dem Raspberry Pi zugewiesen, damit man diesen von den anderen, an den selben Computer angeschlossenen Raspberry Pis, unterscheiden kann.

`raspi` ist eine Matlab-Funktion, die die Verbindung zwischen Matlab und Raspberry Pi herstellt.

`IP-Adresse` ist die Adresse, die man vorher dem Raspberry Pi zugewiesen hat.

`raspberry pi name` ist der Name, der beim Herunterladen des Matlab Support Package für Raspberry Pi Hardware ausgewählt wurde.

`Passwort des Raspberry Pi` ist das Passwort, das man mit dem Befehl `raspi-config` dem Raspberry Pi zugewiesen hat.

3 Software

Der Hauptteil der Software wird über einzelne Matlab-Funktionen und eine in Matlab integrierte Grafische Benutzeroberfläche (GUI) bereitgestellt. Bei der Erfassung der Drehzahl und bei der Verwendung der RS-485 Schnittstelle mittels Modbus ist zudem noch ein Python-Programm auf dem Raspberry Pi notwendig.
Mit Hilfe des Matlab Hardware Support Package für den Raspberry Pi werden darüber einzelne Schnittstellen wie I2C und RS-485, einzelne GPIOs oder Python Programme auf dem Raspberry Pi angesprochen und ausgeführt. Das Hardware Support Package stellt dabei die entsprechenden Funktionen zur Verfügung und kompiliert diese in einen C-Code. Durch Matlab-Funktionen generierte Befehle können entweder über eine SSH Schnittstelle an den Raspberry Pi übergeben werden – hierbei sind alle Matlabfunktionen lokal auf einem PC hinterlegt - oder als Stand-Alone-Programm auf dem Raspberry Pi gespeichert werden. Matlab stellt im letzten Fall einen Compiler/Coder zur Verfügung, der aus Matlab m-Files ein eigenständiges Programm generiert.
Für den Benutzer sollte neben einer vor Programmstart zu konfigurierenden Setup-Datei lediglich eine Bedienung über das GUI zur Anwendung kommen. Da das Gesamtsystem modular aufgebaut ist, kann der Benutzer über die Setup-Datei die Anzahl der Sensoren und deren Adresse im dabei verwendeten Kommunikationsprotokoll festlegen. Im begrenzten Rahmen ist es somit möglich, die Anzahl und Art der Sensoren auch nachträglich zu ändern.

Im Folgenden soll nun der grundlegende Aufbau des gesamten Programms dargestellt werden. Die folgende Graphik zeigt lediglich die Verknüpfung der einzelnen Programmfunktionen. Auf den konkreten Inhalt der Programmfunktionen wird dann in den nachfolgenden Punkten detailliert eingegangen.
Es findet eine Unterteilung in Matlabprogramme, globale Variablen, Textdokumente und Excel-Listen, und Programmen, die lokal auf den Raspberry Pi gespeichert sind. Bei den letztgenannten handelt es sich um Programme in der Programmiersprache „Python“.

Struktur der Software

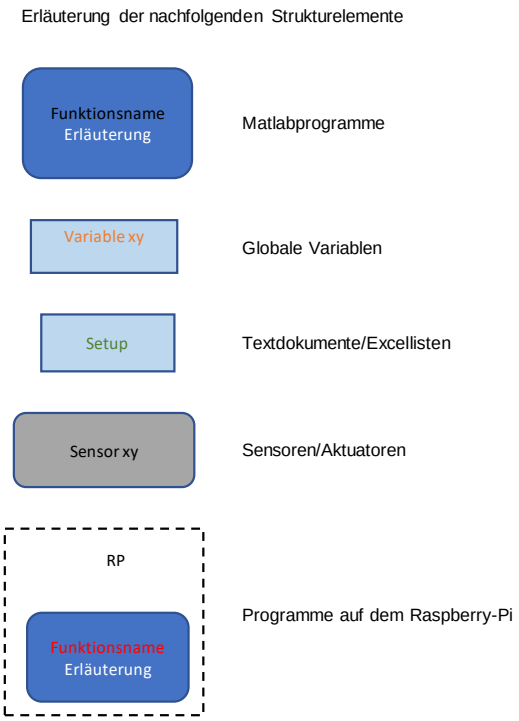

Abbildung 2: Softwarestruktur Teil 1

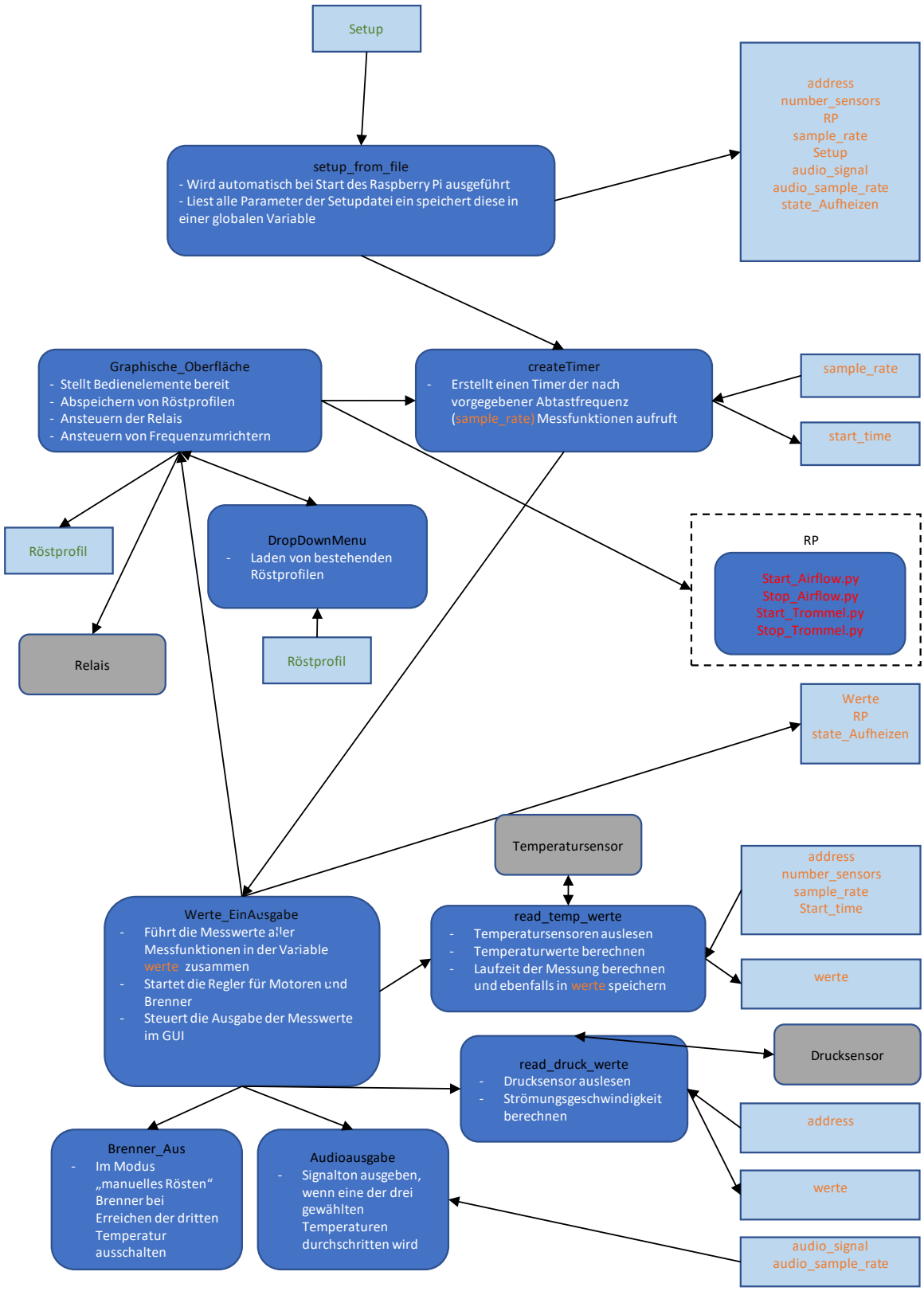

Abbildung 2: Softwarestruktur Teil 2

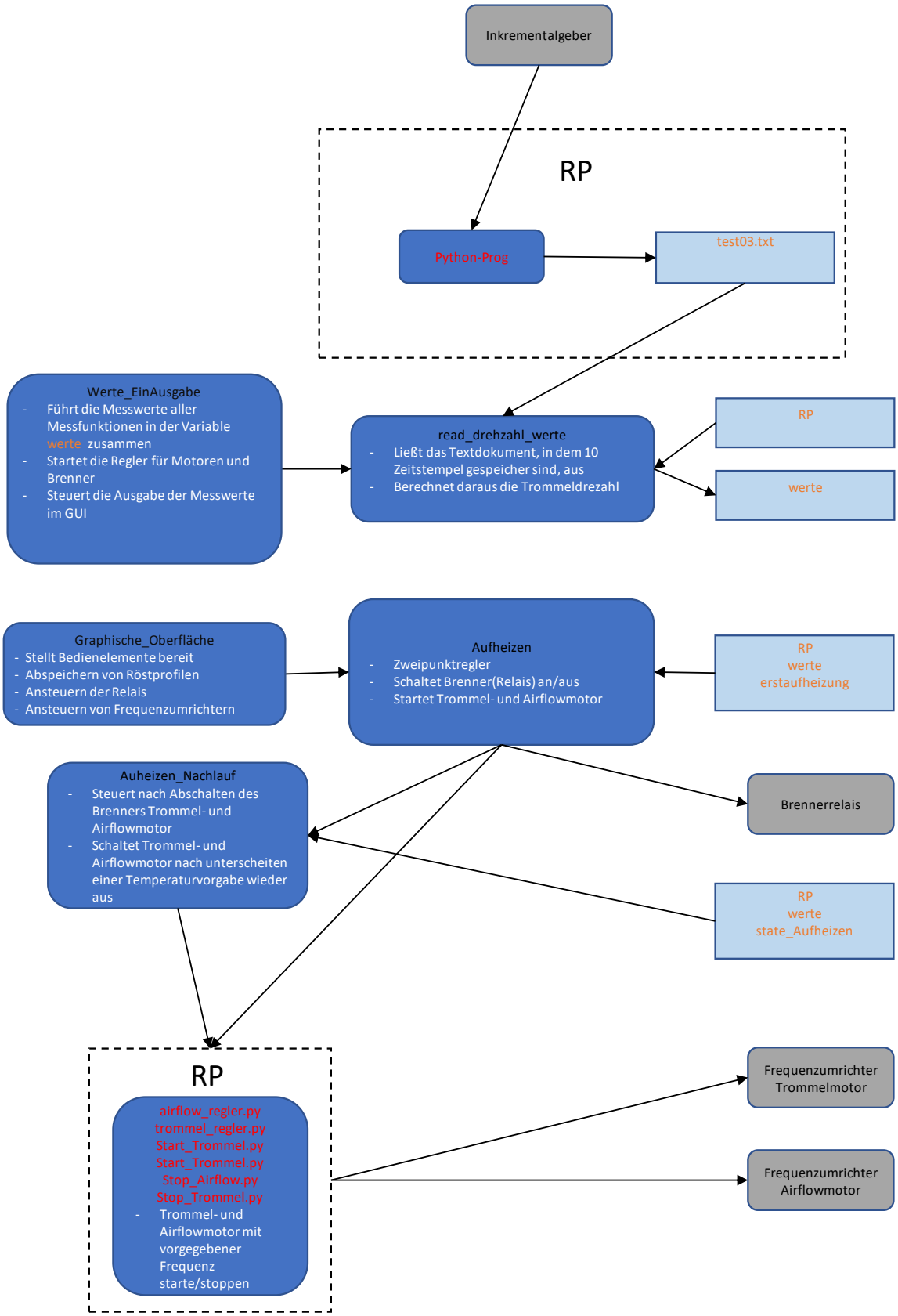

Abbildung 2: Softwarestruktur Teil 3

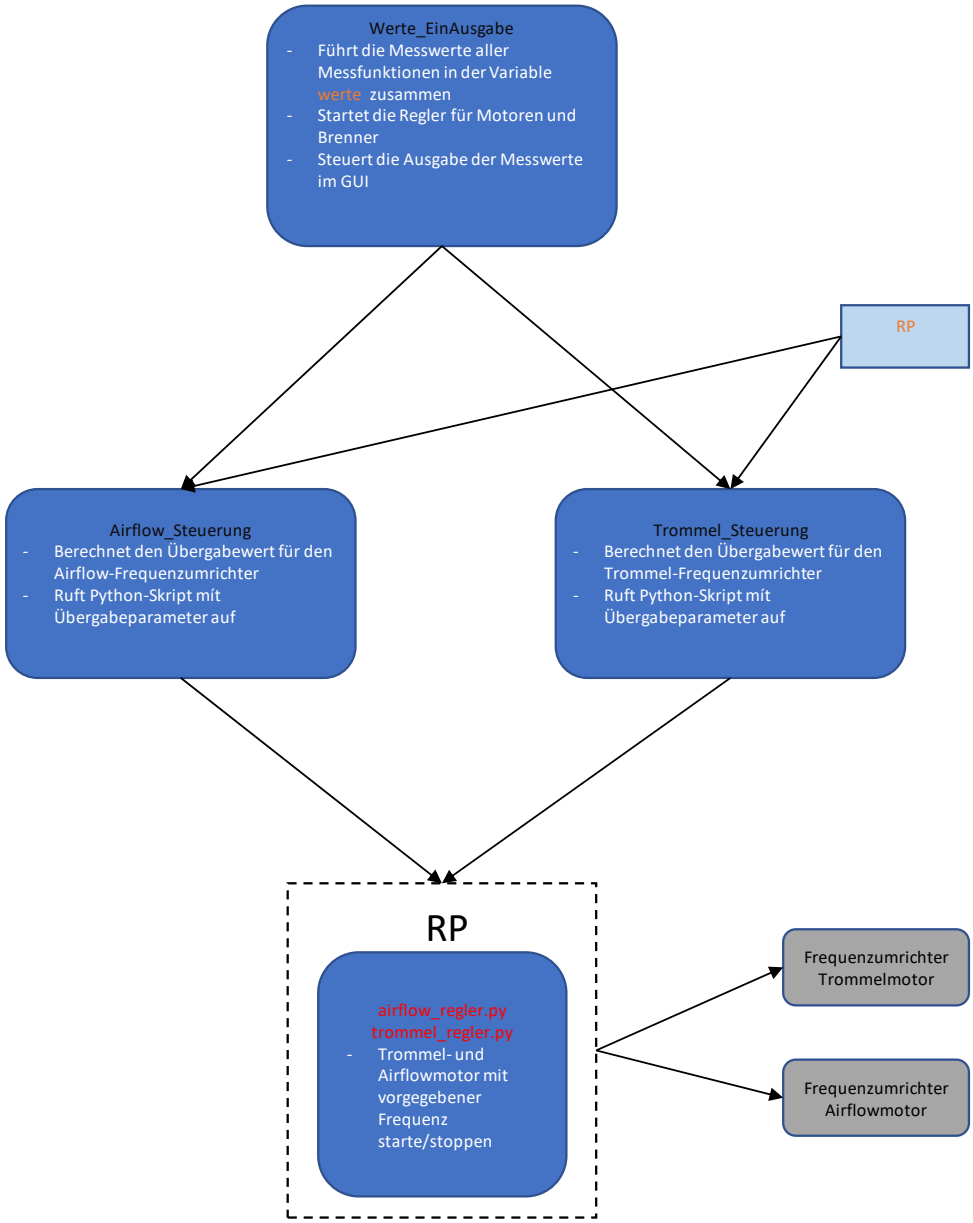

Abbildung 2: Softwarestruktur Teil 4

3.1 Guided User Interface

In Rahmen der Projektarbeit wird dabei auf eine in Matlab integrierte GUI zurückgegriffen. Dieses Tool bildet eine Erweiterung zu den sonst üblichen m-Files und des Command-Windows, welche lediglich in reiner Textform gehalten sind. Durch den Benutzer in das GUI eingegebene Befehle können direkt durch darin hinterlegte Algorithmen verarbeitet werden. Optional startet ein eigenständiges m-File in dem der auszuführende Algorithmus hinterlegt ist.
Durch den Befehl

```
Guide
```

stellt Matlab ein Tool zur Erzeugung der Bedienelemente und des dafür notwendigen Algorithmus bereit. Hiermit können die gewünschten Bedienelemente einfach positioniert und angeordnet werden. Zudem wird nach Fertigstellung des Layouts automatisch ein Grundgerüst eines Algorithmus in Form eines .m-Files erzeugt. Über dieses Grundgerüst können die einzelnen Elemente programmiert werden.
Im Gegensatz zu einem linearen Programmaufbau, kann das GUI direkte Eingaben des Nutzers verarbeiten. Da diese Eingabe auch erfolgen kann, während Matlab eine andere Funktion abarbeitet, muss dafür gesorgt werden, dass alle relevanten Variablen jederzeit und ohne vorherige Rückgabe aus einem anderen Programmteil zur Verfügung stehen. Daher wird im vorliegenden Fall überwiegend auf die Nutzung von lokalen Variablen verzichtet und alle Variablen, die auch von anderen Funktionen genutzt werden, global definiert.
Die Werte von Bedienelementen können dabei mittels „Function Handles“ auch in anderen -Matlab-Funktionen ausgelesen oder manipuliert werden.
Kurze Algorithmen, etwa die Ansteuerung eines Relais nach Betätigung eines Buttons, werden direkt im m-File des GUI abgespeichert. Ausführlichere Algorithmen werden aus Gründen der Übersichtlichkeit als eigenständige Funktionen implementiert.
Bei dem GUI kann zunächst zwischen Eingabeelementen und Ausgabeelementen unterschieden werden.
Die Eingabe erfolgt über Toggle-Buttons, welche eine Zustandsänderung zulassen (hier: Buttonfarbe ändert sich von Rot zu Grün) und durch Push-Buttons, welche bei Betätigung lediglich eine Aktion auslösen und Ihren Zustand dadurch nicht ändern. Numerische Werte werden durch Textfelder eingelesen. Durch ein Drop-Down-Menu erfolgt die Auswahl der Röstprofile.

Als Ausgabeelemente werden ebenfalls Textfelder eingerichtet. Diese stellen die relevanten Messwerte wie Trommeldrehzahl, die Airflowgeschwindigkeit im Abluftrohr und Temperaturen von drei verschiedenen Messstellen dar. Bei Anschluss von mehr als 3 Temperatursensoren erfolgt aus Gründen der Übersichtlichkeit keine weite Anzeige dieser Messwerte. Die Werte werde lediglich für spätere Auswertungen in einer Excel-Datei gespeichert.

Wie in Abbildung 3: Guided User Interface ersichtlich, wurde zudem ein Feld für einen Plot integriert. Hierbei wird der Temperaturverlauf der Bohnentemperatur während des gesamten Röstprozesses dargestellt. Da sich der für den Röstprozess relevante „First Crack“ insbesondere durch eine kurzzeitige Änderung des Temperaturgradienten zeigt, wird zudem auch der Temperaturgradient der Bohnentemperatur graphisch dargestellt.

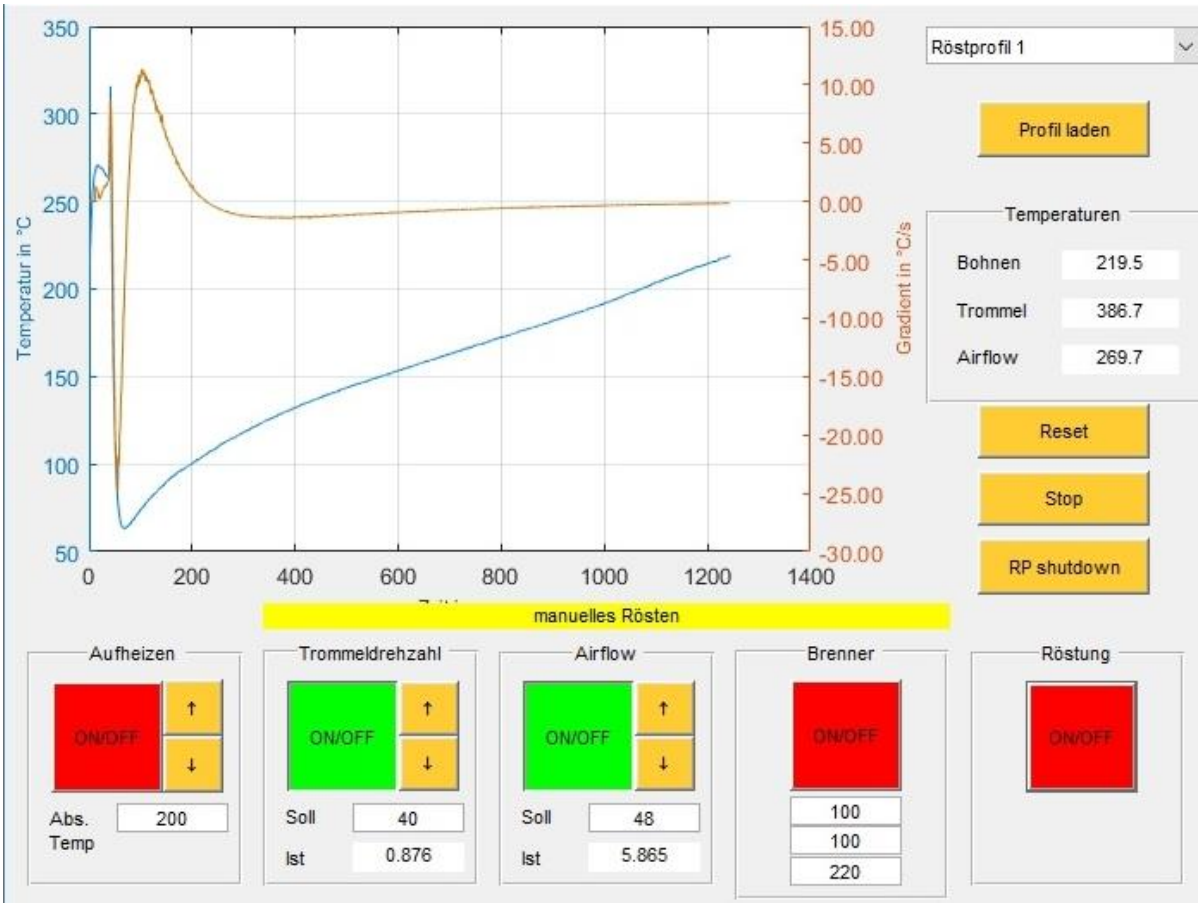

Abbildung 3: Guided User Interface

Im Folgenden sollen nun die einzelnen Funktionen der GUI erläutert werden.

Bei der in Abbildung 3 gezeigten Ansicht handelt es sich bei den Buttons „ON/OFF“ um Toggle-Buttons. Die Buttons „Profil laden“, „Reset“, „Stop“, „RP shutdown“ und Pfeiltasten sind als Push-Buttons definiert. Die Textfelder „Ist“ unter „Trommeldrehzahl“

und unter „Airflow“ und die drei Textfelder unter „Temperaturen“ dienen der Ausgabe von Messwerten.
Die Eingabe von Werten erfolgt entweder durch Eingabe eines konkreten Wertes in den Feldern „Abs. Temp“ und „Soll“ oder durch Nutzung der Pfeiltasten, um Werte in vordefinierten Schritten zu erhöhen oder zu senken. Die drei Felder im Feld „Brenner“ dienen der Eingabe von drei Temperaturen, bei denen bei Durchschreiten die Ausgabe eines Signaltons erfolgt.

Aufheizen

Der ON/OFF-Button unter Aufheizen startet zunächst den Brenner durch Schließen eines Relais. Infolgedessen findet ein Temperaturanstieg statt. Sobald die im Textfeld „Abs. Temp“ (Abschalttemperatur) gewählte Temperatur erreicht wird, öffnet das Relais und der Brenner wird unmittelbar abgestellt. Sobald eine Temperaturdifferenz von 15 °C zur Abschalttemperatur erreicht wird, wird dem Brenner erneut ein Startsignal übermittelt. Über diesen Zweipunktregler wird die gewählte Temperatur in einem vorgesehenen Bereich gehalten, bis der ON/OFF-Button erneut betätigt wird.
Wenn der Button wieder in den Zustand „OFF“ versetzt wird, bleiben der Trommelmotor und der Airflowmotor weiterhin angeschaltet. Zum Schutz des Rösters vor thermischen Schäden werden die Motoren mit einer vordefinierten Frequenz (45 Hz bei dem Trommelmotor, 65 Hz bei dem Airflowmotor) weiterbetrieben, bis die Gehäusetemperatur unter 80 °C fällt. Erst dann werden auch beide Motoren abgeschaltet.
Neben der Ansteuerung des Rösters wird zudem bei „ON“ eine Aufzeichnung aller Messwerte gestartet. Diese dauert so lange an, bis der Button wieder auf OFF gesetzt wird. Anschließend wird mit den Messwerten und den in der Setup-Datei angegebenen Sensorbezeichnungen eine Excel-Liste erstellt. Diese wird dann in einem eigens dafür angelegten Ordner abgespeichert. Die Benennung der Datei folgt immer dem Muster:
„Aufheizen_Datum_Uhrzeit“
Der Prozess gewährleistet, dass der Röster bei der ersten Inbetriebnahme die notwendige Starttemperatur für die Röstung erreicht.

Trommeldrehzahl

In der derzeitigen Version der Software findet lediglich eine Steuerung des Trommelmotors statt. Die unter „Soll“ eingegebene Frequenz wird direkt an den

Frequenzumrichter geschickt. Wenn der Button „ON/OFF“ aktiviert wird, startet der Motor mit der eingegebenen Frequenz.

Airflow

In der derzeitigen Version der Software findet lediglich eine Steuerung des Airflowmotors statt. Die unter „Soll“ eingegebene Frequenz wird direkt an den Frequenzumrichter geschickt. Wenn der Button „ON/OFF“ aktiviert wird, startet der Motor mit der eingegebenen Frequenz.

Brenner

Über dieses Bedienelement kann der Brenner manuell gestartet werden. Zusammen mit den Buttons „Airflow“ und „Trommeldrehzahl“ bilden diese drei Bedienelemente die Grundlage für das manuelle Rösten. Hierbei ist für jeden Zeitpunkt des Röstprozesses manuell eine Trommeldrehzahl, eine Drehzahl des Airflowmotors und ein Start-/Stopbefehl für den Brenner zu definieren.
Damit keine Überhitzung des Brenners erfolgt, ist die Brennerfunktion mit einem Sicherheitsmechanismus ausgestattet, der die Brennerzündung erst dann erlaubt, wenn beide Motoren mit mehr als 30 Hz betrieben werden. Sollten diese Bedingungen nicht erfüllt sein, kann der Brenner nicht gezündet werden.
Für eine Überwachung des Röstprozesses können unter dem Brenner-Button drei Bohnentemperaturen eingegeben werden. Bei Durchschreiten jeder einzelnen Temperatur erfolgt die Ausgabe eines akustischen Signaltons. Die unterste Temperatur stellt zudem die Abschalttemperatur dar. Sobald diese Temperatur erreicht wird, schaltet sich der Brenner aus.
Analog zu dem unter „Aufheizen“ beschriebenen Vorgang erfolgt bei Deaktivierung des Brenners die Erstellung einer Excel-Liste mit den Messdaten. Die Benennung der Datei folgt immer dem Muster:
„Röstung_Datum_Uhrzeit“

Röstung

Hierbei handelt es sich derzeit noch um einen Dummy, welcher noch nicht mit der vorgesehenen Funktion verknüpft ist. Für die Weiterentwicklung der Anwendung ist vorgesehen, dass die Temperaturkurve eines über das Drop-Down-Menu gewählten Röstprofil durch einen geeigneten Regler automatisch nachverfolgt wird. Da der Brenner

nicht stufenlos geregelt werden kann, ist die Temperaturkurve durch Regelung der Größen „Airflowgeschwindigkeit“ und „Trommeldrehzahl“ zu erreichen.

Stop

Grundsätzlich ist vorgesehen, dass die graphische Darstellung von Bohnentemperatur, Temperaturgradient und die Ausgabe aller weiteren Messwerte bereits automatisch nach Start der Anwendung erfolgt. Sollte dies nun nicht weiter gewünscht werden, kann über den Button „Stop“ die Messwerterfassung beendet werden.

Reset

Durch den Reset-Button wird die komplette Anwendung zurückgesetzt. Zunächst wird die Setup-Datei neu eingelesen und der Zustand aller Buttons und Anzeigeelemente zurückgesetzt. Anschließend startet die Messwerterfassung neu. Dies hat auch zur Folge, dass die Ausgabe von Graphen im Plot-Fenster neu beginnt.

RP shutdown

Durch einmalige Betätigung des Buttons wird der Raspberry Pi heruntergefahren. Dies sollte zum Schutz des Raspberry Pis vor jedem Trennen der Spannungsversorgung erfolgen.

Profil laden

Hierrüber lassen sich zuvor aufgezeichnete Röstprofile einlesen. Diese werden zunächst in einen eigenen Ordner gespeichert und können dann über das darüberliegende Drop-Down-Menu ausgewählt werden.
Derzeit werden die eingelesenen Röstprofile nicht weiterverwendet. Die Funktion erhält zukünftig nach Hinzufügen einer Regelung Relevanz.

3.2 Setupprogramm

Um grundlegende Daten auch nachträglich ändern zu können, werden diese vor Programmstart in der Funktion „setup_from_file“ über eine Setupdatei (*Tabelle 1*) im CSV-Format eingelesen. Hierbei werden folgende Daten erfasst:

- *Sample Rate*
- *Bezeichnung*: Benutzername des verwendeten Raspberry Pi
- *IP-Adresse* des Raspberry Pi
- *Passwort* des Raspberry Pi
- *Verwendete Sensoren:* Anzahl aller per I2C angesprochenen Sensoren (hier ein Drucksensor und drei bis zehn Temperatursensoren)
- *Bezeichnung* der Sensorstellen: Frei wählbar, Sensorwerte werden später auch mit dieser Bezeichnung im gespeicherten Röstprofil hinterlegt
- *Adresse:* I2C-Adresse der verwendeten Sensoren

Sample Rate [1/s]	Bezeichnung RP	IP-Adresse	Passwort	verwendete Sensoren (Druck+Temperatur)
1	pi	192.168.137.3	1a2k3h4u	7
Bezeichnung	Adresse			
Druck	0x6C			
Bohnen	0x50			
Trommel	0x51			
Airflow	0x52			
Gehäuse	0x53			
Sensor2	0x54			
Sensor3	0x55			

Tabelle 1: Layout der Setupdatei

In der Auflistung sind lediglich der Drucksensor an erster Stelle und die Temperaturen „Bohnen“, „Trommel“, „Airflow“ und „Gehäuse“ in unveränderter Reihenfolge beizubehalten. Im vorliegenden Programm ist lediglich ein einziger Drucksensor vorgesehen. Eine Erweiterung um weitere Drucksensoren ist in der derzeitigen Konfiguration nicht möglich.
Die anschließend folgende Auflistung der Temperatursensoren ist theoretisch beliebig erweiterbar. Hinzufügen neuer Sensoren erfolgt durch ein Anfügen einer

Sensorbezeichnung und der I2C-Adresse des Sensors unter der bestehenden Liste. Im Programmablauf werden diese dann automatisch eingebunden.
Alle in der Setupdatei erhaltenen Daten werden in globalen Variablen gespeichert und stehen somit anderen Programmteilen dann zur Verfügung.
Darüber hinaus erfolgt im Programmteil „setup_from_file“ der Verbindungsaufbau zu dem Rasperry Pi. Die Verbindungsdaten werden im Objekt

```
RP
```

hinterlegt. I2C-Sensoren werden einem Objekt zugeordnet, über das diese später angesprochen werden können.
Anschließend erfolgt noch eine Konfigurierung der zur Relaissteuerung benutzten GPIOs als Ausgänge:

```
configurePin(RP,17,'DigitalOutput')
```

Im vorgesehenen Hardwareaufbau werden bereits fünf Relais eingebunden, von denen zunächst nicht alle verwendet werden und für zukünftige Erweiterungen der Schaltung genutzt werden können.

Da das GUI bereits während des Programmstarts automatisch gestartet, und auch im Standby-Betrieb die aktuellen Messwerte anzeigen soll, erfolgt zum Schluss noch der Aufruf des Timers über den Befehl

```
createTimer()
```

Der Timer startet zunächst die Funktion „Werte_EinAusgabe“, welche dann das GUI aufruft und auch bereits im Standby-Betrieb alle Messwerte ausgibt.

3.3 Timerfunktion

Über die Erstellung eines Timer-Objekts soll die periodische Ausführung aller Messwertfunktionen erreicht werden. Der Timer „feuert“ dabei in der gewählten Sample-Rate. Durch eine einmalige Initialisierung des Timers, bei der alle relevanten Parameter hinterlegt werden, wird ein Timer-Objekt erstellt, dass durch den Befehl

```
start(t)
```

jederzeit aufgerufen werden kann.
Der Befehl

```
stop(t)
```

beendet den Timer wieder.
Da auch unmittelbar nach Start der gesamten Anwendung bereits eine Datenausgabe erfolgen soll (es ist notwendig bereits nach Programmstart die Messwertfunktionen periodisch abzufragen), wird mit oben genannten Befehl der Timer am Ende des Setup-Skripts bereits gestartet.
Für die Logdateien ist es zudem erforderlich, einen Zeitstempel zu erzeugen, der ab t=0 Sekunden allen Messwerten eine Zeit zuordnet. Dies erfolgt durch:

```
start_time = datetime
```

3.4 Koordinationsprogramm zur Erfassung und Darstellung der Messwerte

Das Programm „Werte_EinAusgabe“ wird in der bei dem Timer hinterlegten Sample-Rate aufgerufen. Es hat einerseits die Aufgabe, die einzelnen Messwertfunktionen auszuführen, andererseits, diese in das GUI einzubinden.
Zudem werden darüber Steuerprogramme der beiden Motoren, im Takt der Sample-Rate angesprochen.
Des Weiteren sind darin noch die Aufheizfunktion, die Funktion zur Audioausgabe und die Brennerabschaltfunktion enthalten.

Alle Messwerte werden in der jeweiligen Messwertfunktion (read_druck_werte.m, read_temp_werte.m, read_drehzahl_werte.m) in die Matrix „werte“ geschrieben.
In der Messwertmatrix „werte“ wird der Zeitstempel (t), die Temperaturwerte (T), der Gradiente der Bohnentemperatur (dT/dt), die Strömungsgeschwindigkeit (v) und die Drehzahl der Röstertrommel (n) hinterlegt. Die Anzahl der Spalten für Temperaturwerte ist von der im Setup angegebenen Sensoranzahl abhängig. Der grundlegende Aufbau dieser Matrix kann *Tabelle 2* entnommen werden.

t_1	$T_{1,1}$	$T_{2,1}$	$T_{n,1}$	$(dT/dt)_1$	V_1	n_1
t_2	$T_{1,2}$	$T_{2,2}$	$T_{n,2}$	$(dT/dt)_2$	V_2	n_2
t_m	$T_{1,m}$	$T_{2,m}$	$T_{n,m}$	$(dT/dt)_m$	V_m	n_m

Tabelle 2: Aufbau der Messwertmatrix

Es gelten folgende Einheiten:

Größe	**Einheit**
Zeit t	s
Temperatur T	°C
Gradient dT/dt	°C/s
Strömungsgeschwindigkeit v	m/s
Drehzahl	min^{-1}

Tabelle 3: Einheiten der Messwerte

Des Weiteren werden die fünf Textfelder mit den jeweiligen Messwerten beschrieben. Die Bohnentemperatur und der Gradient der Bohnentemperatur sollen zudem in einem Plot graphisch dargestellt werden. Es wir je Vorgang (z.B. Röstung, Aufheizen) über die gesamte Dauer des Vorgangs eine Kurve in das Plot-Fenster geschrieben. Die Skalierung der beiden Plot-Achsen erfolgt dynamisch, d.h. diese orientiert sich nach den jeweiligen Maximal- und Minimalwerten von Temperatur und Gradient.

3.5 Messwertprogramm Temperatur

Zunächst wird in der Funktion „read_temp_werte“ ermittelt, wie viele Thermoelemente angeschlossen sind. Der Algorithmus zur Auslesung der Temperaturwerte wird dann so oft ausgeführt, bis alle Sensoren abgefragt wurden. Hierfür wird eine for-Schleife verwendet:

```
for count = 1:(number_sensors-1)
```

Wie dem Datenblatt der Thermoelemente zu entnehmen ist, werden über die I2C-Schnittstelle insgesamt vier Byte je Messung übertragen. Aus den zwei ersten Bytes wird die Thermospannung ermittelt; aus den dritten und vierten Byte wird die Spannung der

Ausgleichsstelle ermittelt. In beiden Fällen lässt sich daraus eine Zahl berechnen, die proportional zu Thermospannung bzw. der Temperatur der Ausgleichsstelle ist. Anhand der im Datenblatt enthaltenen Tabelle kann dann durch Iteration eine Thermospannung und ein Korrekturwert (Ausgleichsstelle) berechnet werden.

Die beiden Werte ergeben sich durch Aneinanderreihung des ersten und zweiten Bytes in hexadezimaler Form. Sollte das Thermoelement z.B. 0x60 0x85 0x3E 0x00 liefern, ist folgendermaßen vorzugehen:

a) Aus ersten beiden Bytes Wert für Thermospannung bilden und in Dezimalzahl umrechnen:
   0x6085 = 24709
b) Aus drittem und vierten Byte Wert für Ausgleichstemperatur bilden und in Dezimalzahl umrechnen:
   0x3E00 = 15872
c) Aus Ausgleichstemperatur über die Korrekturwerttabelle den Korrekturwert ermitteln: -> 1203 digits
d) Den Korrekturwert zu den nummerischen Wert der Thermospannung addieren:
   24709+1203 = 25912
e) Mit diesem Wert nun aus der Thermospannungstabelle die wirkliche Temperatur interpolieren: -> 330 °C

Da Matlab die Werte der einzelnen Bytes als Dezimalzahl ausgibt, erfolgen im Schritt a) und b) noch einige Zwischenschritte.

Umrechnung des Dezimalwerts in einen Hexadezimalwert:

```
bt1 = dec2hex (bt1)
```

Im Fall, dass der Hexadezimalwert eines Bytes einstellig ist, eine „0“ hinzufügen:

```
bt2_h='0'
bt2 = [bt2_h bt2]
```

Zusammenfügen der beiden Bytes:

```
thermospannung = [bt1 bt2]
```

Rücktransformation in Dezimalwert:

```
thermospannung = hex2dec (thermospannung)
```

Für Schritt c) und e) werden im Programm die jeweils zur Interpolation verwendeten Tabellen hinterlegt.

Da durch unsachgemäßen Anschluss der Thermoelemente oder durch Störungen bei der Datenübertragung auch Fehler entstehen können, sollte in diesem Fall die Wertematrix „werte“ mit dem Fehlerwert NaN (Not-a-Number) beschrieben werden:

```
temp = NaN
```

Dadurch wird verhindert, dass bei einer temporären Störung das komplette Programm abbricht und neu gestartet werden muss.

Da read_temp_werte als erstes der Messwertprogramme aufgerufen wird, findet darin auch die Ermittlung der Laufzeit statt:

```
[Y, M, D, H, MN, S] = datevec(datetime - start_time
t=H*3600+MN*60+S
```

Anschließend wird die Wertematrix mit der Laufzeit (erste Spalte) und den Temperaturwerten (alle weiteren Spalten) beschrieben.

Wie sich bei Voruntersuchungen gezeigt hat, kann der first Crack, welcher ein wichtiger Marker im Röstprozesse darstellt, anhand eines temporären Temperaturabfalls erkannt werden. Im Temperaturverlauf selbst ist dieser Abfall nur schwer zu erkennen. Im Temperaturgradient ist dieser Punkt hingegen wesentlich deutlicher ausgeprägt. Daher soll auch noch der Temperaturgradient der Bohnentemperatur berechnet werden. Zunächst wird durch die Matlabfunktionen

```
Polyfit
```

und

```
polyval
```

dem wahren Temperaturverlauf ein Polynom dritten Grades angenähert. Von dieser Kurve wird dann mit

```
grd = gradient (y1/sample_rate)
```

der Gradient zum aktuellen Zeitpunkt berechnet. Dieser wird ebenfalls in der Matrix „werte“ abgespeichert und im Plot des GUIs visualisiert.

3.6 Messwertprogramm Druck

Das Programm „read_druck_werte“ zur Erfassung des Staudrucks im Auslass des Rösters wird ebenfalls von „Werte_EinAusgabe“ gestartet. Durch Messung des Staudrucks soll die Strömungsgeschwindigkeit und somit auch der Volumenstrom durch den Röster ermittelt werden.
Zur Initialisierung des Drucksensors werden über die I2C-Schnittstelle entsprechende Register des Sensors beschrieben. Anschließend können die Druckwerte über

```
wert_dec = read(address(1),8)
```

ausgelesen werden. Aus den dabei erhaltenen 8 Byte sind das 5. und das 7. Byte zur Bildung des Druckwerts relevant. Analog zur Temperaturmessung werden die dezimalen Inhalte der beiden Bytes wieder ins hexadezimale System transformiert, aneinandergereiht, mit fehlenden Nullstellen aufgefüllt und wieder zurück ins dezimale System transformiert. Hieraus ergibt sich die Varialble „Op“
Nach Datenblatt berechnet sich der Differenzdruck nach

$$Dp = (Op - 1024) / 60000 \times 100 – 50$$

Nun kann daraus die Strömungsgeschwindigkeit ermittelt werden:

```
v = (2*abs(Dp)/1.2041^(1/2)
```

Für die Luftdichte ρ wird der Wert der Normatmosphäre auf der geographischen Höhe von München hinterlegt. Sollte die Anwendung in anderen Höhe verwendet werden, ist der Wert im Programm dementsprechend abzuändern.
Alle Druckwerte werden in einer weiteren Spalte am Ende der Matrix „werte“ gespeichert.

3.7 Messwertprogramm Drehzahl

Über den Inkrementalgeber wird je Zahnradzahn ein Impuls ausgelöst. Bei einer vorgegebenen Anzahl an Impulsen kann über den dabei verstrichenen Zeitraum und unter Kenntnis der Zähnezahl die Drehzahl berechnet werden.

Bei der Erfassung dieser Impulse zeigt sich ein Nachteil der Matlab-Architektur, welche durch eine externe Lösung umgangen werden muss. In Matlab ist ein Multitasking nur mit erheblichen Aufwand möglich. D.h. mehrere Tasks können nicht einfach parallel ausgeführt werden. Im Falle der Drehzahlmessung würden daher alle weiteren Programmfunktionen solange angehalten werden, bis eine ausreichende Anzahl an Impulsen erfasst wird. Bei der langsam laufenden Trommelwelle übersteigt dieses unumgängliche Pausieren die Abtastrate.
Um dieses Problem zu umgehen, wird der Prozess der Impulserfassung auf den Raspberry Pi in Form eines eigenständigen, in Schleife ausgeführten Python-Programms ausgelagert.
Im multitaskingfähigen Betriebssystem des Raspberry Pis wird daher bereits bei dem Einschalten den Raspberry Pis ein Python-Programm gestartet.

Anschließend sieht man die Python-Zeilen, die die Befehle zur Erfassung der Impulse anzeigen.

Detektieren der aktuellen Uhrzeit:

```
t = datetime.datetime.now()
```

Umrechnung der detektieren Zeit in Mikrosekunden:

```
t2 = (t.hour*60*60*1000000)+(t.minute*60*1000000)+(t.second*
1000000)+ t.microsecond
```

Darstellung der Werte in Strings und deren Zuweisung in der Variable `lst:`

```
out = str(t2)
lst [count] = out
```

Schleife zum Schreiben der Werte in die Datei „test03.txt“:

```
count = count + 1
if (count == 10):
      target=open('test03.txt', 'w')
      for item in lst:
            target.write ("%s\n" % item)
      target.close()
count = 0
```

Zur Erfassung der Trommeldrehzahl wird ein auf Hall-Effekt Basis funktionierender Zahnraddrehzahlsensor eingesetzt. Um dies erfolgreich und problemlos zu realisieren, ist es sinnvoll, zuerst ein Python Skript auf Raspberry Pi zu erstellen und dann die erfassten Werte der Drehzahl über Matlab aufzurufen und auszugeben. Die untenstehende Skizze stellt eine vereinfachte Erfassung der Trommeldrehzahl mittels Python-Skript und M-File dar.

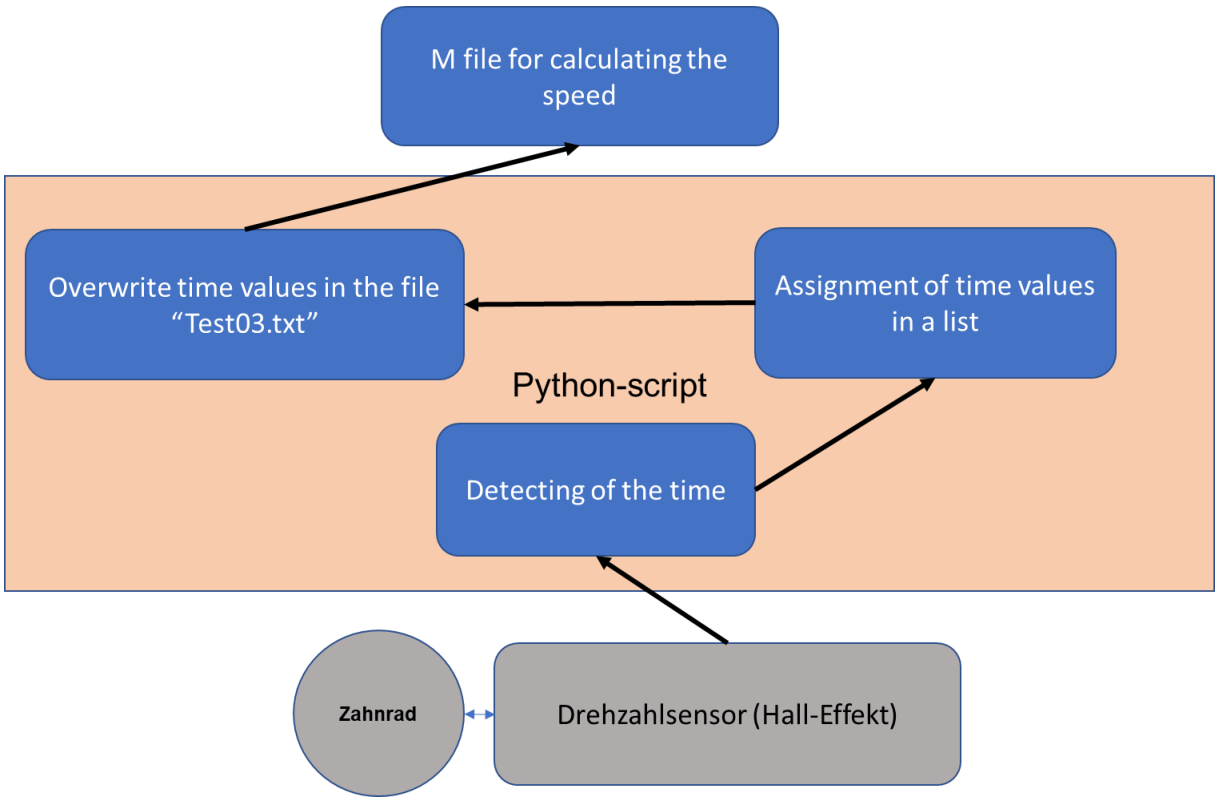

Abbildung 4: Erfassung der Drehzahl anhand Python-Skript und Matlab

Bei Drehen des Zahnrades, das mit Trommelwelle verbunden ist, verändert sich der Abstand zwischen dem Zahnrad und dem Drehzahlsensor. Somit ändert sich das magnetisches Feld permanent. Dies führt anhand der internen Schaltung des Sensors zu Entstehung eines rechteckigen elektrischen Signals. Dieses ist ziemlich schwach und wird über die Versorgungsspannung des Sensors verstärkt. Der Sensor wird mit dem 5V Pin des Raspberry Pis angeschlossen und somit mit 5V versorgt. Die Form des rechteckigen Signals ist ebenfalls auf der vorhergehenden Skizze zu sehen.
Das Python-Programm hat die Aufgabe, die fallende Rampe des Signals zu detektieren und die Zeit des Events auszugeben. Mehr Details zu dem Verlauf des Prozesses sind dem Python-Programm zu entnehmen.
Die durch das Python-Programm ausgegebenen Daten können über Matlab aufgerufen werden und darüber nach einer in die physikalischen Größen kleinen Umrechnung ausgegeben werden. Da es hier in diesem Kapitel nur um das Pythonskript geht, kann die Matlab-Datei bzw. die Matlab-Befehle in dem dafür vorgesehenen Kapitel nachgeschlagen werden.

Da der Start des Python-Programms nicht direkt über Matlab ausgeführt werden kann, soll daher ein Autostart beim Hochfahren des Raspberry Pis eingestellt werden. Diesen kann man je nach dem Gebrauch in mehreren unterschiedlichen Möglichkeiten durchführen.
Mit der .bashrc-Methode lässt sich das Python-Programm beim Booten des Raspberry Pis ausführen, indem der Befehl „ python /home/pi/myprogramm.py“ am Ende des Datei bashrc gesetzt werden muss.

```
sudo nano /home/pi/.bashrc
```

Nach dem Öffnen der Datei bashrc sollen am Ende die zwei folgenden Befehle gesetzt werden:

```
echo running at boot
sudo python /home/pi/myprogramm.py
```

Und somit wird das Python-Programm bei jedem Booten des Mikrocontrollers ausgeführt.

Das Textdokument, in welchem sich die Zeitstempel der einzelnen Inkremente befinden, kann nun von Matlab ausgelesen und in einer Variable abgespeichert werden:

```
werte_drehz_vorlaufig = fscanf(fileID,'%f')
```

Es sollten darin exakt 10 Werte enthalten sein. Bei möglichen Fehlern, etwa wenn das Python-Programm zur selben Zeit einen Schreibzugriff ausführt und somit weniger als 10 Werte hinterlegt sind, wird bis zur nächsten erfolgreichen Abfrage der alte Wert beibehalten.
Bei der Auswertung der Zeitstempel werden die fehleranfälligen ersten und letzten Werte nicht betrachtet. Nur die 7 dazwischenliegenden Zeitstempel werden betrachtet. Aus der Differenz der aufeinanderfolgenden Zeitstempel wird über die Zähnezahl des Zahnrads (hier 24) die Drehzahl berechnet:

```
period(1,inc-2)=60*1000000/(24*(werte_drehz(inc)-
werte_drehz(inc-1)))
```

Nun wird aus diesen Einzelwerte noch der Mittelwert berechnet:

```
drehzahl=mean(period)
```

Dieser Wert wird ebenfalls in einer neuen Spalte am Ende der Matrix „werte“ gespeichert.

Dennoch besitzt auch diese Programmarchitektur zwei nennenswerte Nachteile. Zum einen erlaubt das Programm nicht die Erfassung der Drehzahl ‚0‘. Da immer auf 10 Inkremente gewartet wird bis ein Drehzahlwert errechnet werden kann, kann ein Stillstand der Welle nicht erkannt werden. Mit der Messung der Drehzahl durch einen Inkrementalgeber muss dieses Problem in Kauf genommen werden. Hier könnte man die Überlegung anstellen, auf einen Sensor, welcher auf anderen physikalischen Prinzipien basiert, umzusteigen. Ein Tachogenerator wäre auch in der Lage einen Stillstand der Welle zu erkennen.
Außerdem tritt bei geringen Drehzahlen eine andere Problematik zu Tage. Wenn die zur Berechnung der Drehzahl benötigten 10 Werte in einem Zeitintervall größer der Sample Rate auftreten, wird über zwei oder mehrere Abtastperioden der gleiche Drehzahlwert ausgegeben. Die derzeitige Lösungsvariante könnte entweder mittels einer kleineren Sample-Rate oder mit einer Erhöhung der Zähnezahl des Zahnrads verbessert werden.

3.8 Drop-Down Menu

Da im späteren Entwicklungsstand der Anwendung eine Regelung der Parameter anhand manuell aufgezeichneter Röstprofilen erfolgen soll, muss noch eine Funktion integriert werden, die das Einlesen und die Auswahl dieser Röstprofile ermöglicht. Hierzu wird im GUI ein Button „Profil laden“ und ein Drop-Down Menu integriert. Über den Button werden bereits vorhandene Röstprofile in einen Ordner abgespeichert. Über das Drop-Down Menu erfolgt die Auswahl des Röstprofils, das für den aktuellen Röstvorgang als Referenz verwendet werden soll.
Im derzeitigen Entwicklungsstand der Anwendung ist eine Regelung der Parameter über Röstprofile noch nicht möglich. Diese Funktion steht allerdings für die zukünftige Weiterentwicklung nun bereits zur Verfügung.

Die zugehörige Matlab-Funktion „DropDownMenu.m“ wird nach Betätigung des Buttons „Profil laden“ aufgerufen. Dabei wird zunächst die Programmzeile

```
[FILENAME, PATHNAME] = uigetfile('*.xls')
```

ausgeführt, welche unmittelbar ein Auswahlfenster öffnet. Der Anwender kann darüber, ähnlich wie im Windows-Explorer, die gewünschte Excel-Datei suchen und auswählen. Da der Name der Datei später auch im Drop-Down Menu hinterlegt werden soll, muss dieser noch extrahiert und gespeichert werden:

```
[pathstr, filen, ext] = fileparts(strcat(PATHNAME,FILENAME))
```

Der Inhalt der Datei wird zunächst in der Variable „data“ zwischengespeichert. Hier sollen dann noch die für den Regler relevanten Daten (Zeitstempel und Bohnentemperatur) extrahiert werden.

```
data=data (:,1:2)
```

Die aufbereiteten Daten werden dann in den Ordner „Roestprofile“ geschrieben. Das Verzeichnis des Ordners muss hierbei noch manuell in der Matlab-Funktion hinterlegt werden. Eine Eingabe über die Setup-Datei ist derzeit noch nicht möglich.

Im nächsten Schritt wird das Drop-Down Menu aktualisiert. Alle derzeit im Ordner „Roestprofile“ gespeicherten Excel-Dokumente werden über den GUI-Handle in der Objekteigenschaft „String“ des Menues eingefügt.

```
DropDownString = set(handles.popupmenu1,'String',DropDownNames)
```

3.9 Aufheizfunktion

Die Funktion „Aufheizen“ soll alle relevanten Funktionen des Rösters steuern, die bei dem ersten Aufheizvorgang notwendig sind. Zukünftig soll zudem eine Überwachungsfunktion integriert werden, die eine Abweichung von der idealen Aufheizkurve detektieren und melden.

Wie Testmessungen zeigen (Abbildung 5, gezackter Bereich der Temperaturkurve), findet aktuell eine Zweipunktregelung statt. Der Brenner lässt keine stufenlose Steuerung zu. Er kann nur ein- oder ausgeschaltet werden. Vor dem tatsächlichen Zünden des Brenners muss dieser gespült werden, wodurch eine Zeitverzögerung von etwa 20 Sekunden zwischen Einschaltbefehl und dem tatsächlichen Heizvorgang stattfindet. Zudem müssen das Airflow-Gebläse und die Trommel angeschaltet werden.

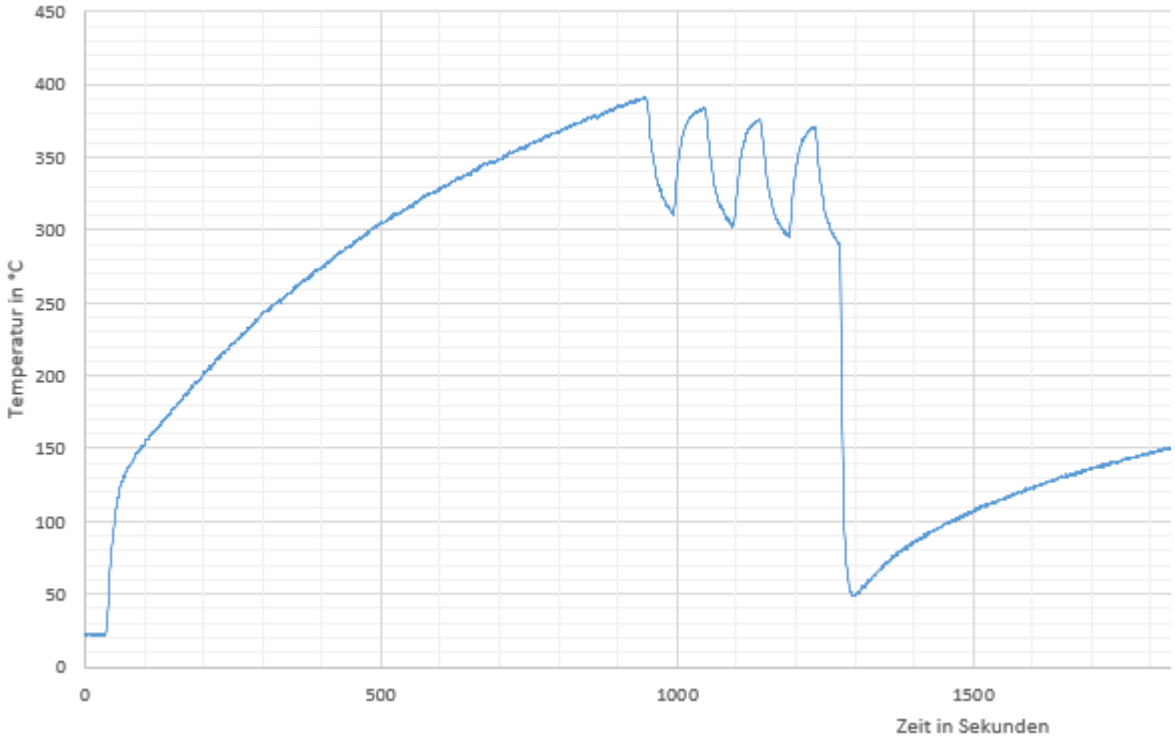

Abbildung 5: Aufheizkurve (Bohnentemperatursensor)

Im Matlab-File soll nun die Zweipunktregelung für einen vordefinierten Bereich umgesetzt werden. Die Abschalttemperatur kann vom Anwender über das Textfeld „Abs. Temp“ das GUIs eingegeben werden. Die Temperaturdifferenz, aus der sich die Anschalttemperatur errechnet, wird im Programm fest hinterlegt. Gestartet wird der gesamte Aufheizprozess über den Button „Aufheizen“. Als Referenz für die Abschalttemperatur wird der Gehäusetemperatursensor verwendet. Wie Versuche gezeigt haben, kann der Brenner durch Trennen der Verbindungen T1 und T2 (Bezeichnung des Brennerherstellers im Schaltplan) abgeschaltet werden. Im Auslieferungszustand ist diese Verbindung gebrückt. Um nun die Schaltung per Raspberry Pi durchführen zu können, wird die Brücke entfernt und an dessen Stelle ein Schütz, welches über den Digitalausgang 17 des Pis angesteuert wird, geschaltet.
Die Funktion „Aufheizen“ wird ebenfalls in der Funktion „Werte_EinAusgabe“ eingefügt und mit der Bedingung:

```
if state_Aufheizen == 1
```

verknüpft. Wenn togglebutton1 (Button „Aufheizen“) betätigt wurde, wird die Variable „state_Aufheizen“ auf 1 gesetzt und damit die Funktion „Aufheizen“ in der festgelegten Sample-Rate regelmäßig aufgerufen bis der Button wieder deaktiviert wird.

Im Programmablauf wird zunächst die Eingabe des Textfeldes ausgelesen.

```
handles = guidata(Grafische_Oberflaeche)
```

Um den gewünschten Ablauf des Aufheizvorgangs zu gewähren, wird der Programmablauf in drei Abschnitte durch „if“ Bedingungen geteilt.
Die erste Bedingung überprüft, ob es sich um die Erstaufheizung handelt.

```
if werte (end,4) < str2num(get(handles.edit9,'String')) &&
erstaufheizung == 1
```

Dieser Fall tritt unmittelbar nach Betätigung des Buttons „Aufheizen“ ein. Hierbei wird zunächst der Pin 17 aktiviert, welcher wiederrum das Schütz schaltet.
Die zweite Bedingung tritt ein, wenn die Abschalttemperatur erreicht wird.

```
if werte (end,4) > str2num(get(handles.edit9,'String'))
```

Daraufhin wird Pin 17 wieder deaktiviert und somit der Brenner unmittelbar abgeschaltet.

Die letzte Bedingung tritt ein, wenn mindestens einmal die Abschalttemperatur erreicht wurde und anschließend die Temperatur den vordefinierten Bereich unterschreitet.

```
if werte (end,4) < (str2num(get(handles.edit9,'String'))-60) &&
erstaufheizung == 0
```

Der Temperaturabfall (hier 15 °C), ab dem der Brenner wieder gestartet werden soll, wird ebenfalls in dieser Programmzeile angegeben. Anschließend wird Pin 17 aktiviert wodurch der Brenner erneut zündet und erst nach erneutem Erreichen der Abschalttemperatur wieder ausgeschaltet wird.

3.10 Schutzfunktion nach dem Aufheizvorgang

Die Funktion „Aufheizen_Nachlauf“ dient dazu, die Brennerelemente vor einer Überhitzung zu schützen. Nachdem der Brenner deaktiviert wurde, soll diese Funktion den Airflowmotor und den Trommelmotor weiterhin in Betrieb halten und somit den Röster mit Frischluft versorgen.

Nachdem der Button „Aufheizen“ wieder auf „OFF“ gesetzt wurde, wird zunächst die Variable „state_Aufheizen“ auf den Wert 2 gesetzt. Durch die Abfrage

```
if state_Aufheizen == 2
```

im Skript „Werte_EinAusgabe“ wird bei Erfüllung der Bedingung dann die Funktion „Aufheizen_Nachlauf“ gestartet.
Beiden Frequenzumrichtern wird zunächst eine Festfrequenz übermittelt.

```
aufruf =['python trommel_regler.py 11342']
```

Diese werden dann mit dieser Frequenz gestartet.

```
system (RP,'python Start_Trommel.py')
```

Der Airflowmotor läuft dabei mit 65 Hz, der Trommelmotor mit 45 Hz.

Erst wenn eine Gehäusetemperatur von 80°C unterschritten wird, schalten sich die Motoren automatisch aus.

```
if werte (end,5) < 80
      system (RP,'python Stop_Airflow.py')
```

```
system (RP,'python Stop_Trommel.py')
```

Die Variable „state_Aufheizen“ wird nun wieder auf 0 gesetzt. Somit ist in allen weiteren Skripten klar, dass der Prozess beendet wurde.

3.11 Airflow- und Trommel-Steuerung

Die Airflow- und Trommelmotor-Steuerung sollen die Frequenzumrichter der beiden Motoren per Modbus ansteuern, diesen Start- und Stop-Befehle erteilen und die Sollfrequenz übermitteln. Da beide Skripte den gleichen Aufbau besitzen, sollen diese nun in einer einzigen Betrachtung zusammengefasst werden.
Die Steuerung soll für die Betriebsart „manuelles Rösten“ verwendet werden. Bei dem automatischen Rösten, bei dem der Airflow und die Trommelgeschwindigkeit per Regler angesteuert werden, ist das Skript dementsprechend zu erweitern. Derzeit wird lediglich die vom Anwender vorgegebene Frequenz in Hertz an die Frequenzumrichter übermittelt.
Im manuellen Betrieb wird der Sollwert über das Eingabefeld „Soll“ im Bereich „Trommeldrehzahl“ und „Airflow“ vom Anwender eingegeben. Im automatischen Betrieb soll der Sollwert aus dem gewählten Referenzröstprofil extrahiert werden.
Die beiden Skripte „Airflow_Steuerung“ und „Trommel_Steuerung“ haben zwei Aufgaben. Einerseits soll das Eingabefeld ausgelesen werden. Andererseits muss die Kommunikation mit dem Frequenzumrichter bewerkstelligt werden. Da der Frequenzumrichter per Modbus-Protokoll angesprochen wird, muss für diese Aufgabe ein zusätzliches Python-Programm auf dem Raspberry Pi hinterlegt werden. Ein direkter Verbindungsaufbau über Matlab per Modbus ist nicht möglich. Im vorliegenden Anwendungsfall übergibt Matlab dem Python-Programm lediglich den ermittelten Stellwert.

Zunächst wird das entsprechende Eingabefeld ausgelesen

```
Sollwert = str2num(get(handles.edit3,'string')
```

Nun muss noch das Python-Skript mit Übergabeparameter aufgerufen werden. Diese sendet dann den Stellwert an das Register 100 des Frequenzumrichters. Register 100 entspricht dem Hauptsollwert (HSW). Daraus errechnet sich dann zusammen mit der im Frequenzumrichter hinterlegten Bezugsfrequenz (Parameter P2000, derzeit hinterlegt: 50 Hz) über die Formel

$$y = \frac{Sollwert\ [Hz]}{P2000\ [Hz]} \cdot 4000\ [Hex]$$

der zu übermittelnde Wert y. Da y als Dezimalzahl übermittelt wird, lautet der Matlab-Befehl:

```
y = Sollwert/50*16384
```

Da Matlab über keine Modbus-Befehle verfügt, muss die Kommunikation zwischen Raspberry Pi und Frequenzumrichter über ein Python-Skripte erfolgen. Für die Steuerung des Gebläsemotors sind vier Python-Programme notwendig. Zwei Programme sind für den Start und den Stop zuständig, die beiden anderen für die Übertragung des Hauptsollwerts. Das Start- und das Stop-Programm wird direkt im GUI aufgerufen. Im Steuerungsskript wird lediglich im Takt der Timers die Sollfrequenz im Frequenzumrichter aktualisiert.
Per Matlab wird nun ein über den Befehl

```
system(RP,aufruf)
```

im Linux-Shell ein Befehl mit Übergabeparameter ausgeführt, der das entsprechende Python-Programm aufruft und diesem den Stellwert der Sollfrequenz übergibt. Die Variable „aufruf“ ist dabei nach dem Schema

„python“ „Name des Programms“ „Übergabewert“

aufgebaut.

Im Folgenden soll nun der grundlegende Aufbau der Python-Programme dargestellt werden.

Motor Starten (Start_Airflow.py und Start_Trommel.py)

Übergabe des Steuerwortes in das Register '99' zum Steuern des Frequenzumrichters mit der Adresse 0x02 und Frequenzumrichter mit der Adresse 0x01 (Gebläsemotor)

```
rr_ready_to_run = client.write_register(99, 0xC7E, unit=0x02)
rr_ready_to_run = client.write_register(99, 0xC7E, unit=0x01)
```

Übergabe des Wertes '1' in das Register '5' zum Starten der Gebläse und Trommel

```
rr_cmd_start = client.write_register(5, 1, unit=0x02)
rr_cmd_start = client.write_register(5, 1, unit=0x01)
```

Übergabe des Sollwerts (airflow_regler.py und trommel_regler.py)

Definition des Übergabewertes als Variable

reg_wert_setValue = int(sys.argv[1])

Drehzahl-Sollwert oder Übergabewert wird über Matlab eingegeben

```
rr_set_value = client.write_register(100, reg_wert_setValue,
unit=0x02)
rr_set_value = client.write_register(100, reg_wert_setValue,
unit=0x01)
```

Motor stoppen (Stop_Airflow.py und Stop_Trommel.py)

Übergabe des Wertes '0' in das Register '5' zum Stoppen der Gebläse

```
rr_cmd_stop = client.write_register(5, 0, unit=0x02)
rr_cmd_stop = client.write_register(5, 0, unit=0x01)
```

3.12 Audioausgabe

Das Skript „Audioausgabe“ liest die drei Werte des GUI-Feldes „Brenner“ aus und gibt bei Durchschreiten der Temperaturen ein akustisches Signal aus.
Sobald der Brennerbutton auf ON gesetzt wird, wird das „Audioausgabe“ im Takt des Timers aufgerufen. Falls eine der Temperaturen das erste Mal durchschritten wird, erfolgt die Ausgabe eines Signaltons

```
sound(audio_signal, audio_sample_rate
```

3.13 Brenner ausschalten

Das Skript „Brenner_Aus“ soll bei Erreichen der dritten, unter dem Feld „Brenner“ hinterlegten Temperaturen den Brenner ausschalten. Das Skript sorgt somit dafür, dass im Modus „manuelles Rösten“ eine Abschalttemperatur definiert werden kann, die den Brenner am Ende des Röstprozesse automatisch deaktiviert.
Hierfür wird das Skript ebenfalls wieder im Takt der Timers aufgerufen, sofern der Brennerbutton aktiviert wurde. Zunächst wird die eingegebene Temperatur ausgelesen

```
Temp_Audio_3 = str2num(get(handles.edit13,'string'))
```

Sobald diese Temperatur kleiner als die aktuelle Bohnentemperatur ist, wird das Brennerrelais geöffnet

```
writeDigitalPin(RP,17,0)
```

und der Brennerbutton wieder in den Zustand OFF zurückgesetzt.
Zudem soll nun ähnlich wie im Modus „Aufheizen“ eine Speicherung aller während des Röstprozesses erfassten Daten in einer Exceldatei erfolgen. Dafür wird die Matrix „werte“ in einer Exceldatei abgespeichert.

```
xlswrite (filename,werte,1,'A2')
```

3.14 Einbindung der Schnittstelle RB-RS485 für Frequenzumrichter

Die RS485-Schnittstelle wird an den GPIOs 14 und 15 verbunden. Da die GPIOs 14 und 15 auch für die serielle Schnittstelle und Bluetooth besonders bei den neuen Versionen von Raspberry Pi verwendet werden, und damit die UART-Schnittstelle stören, soll zuerst die serielle Schnittstelle deaktiviert werden.
Die serielle Schnittstelle kann anhand der Durchführung ein paar Befehle deaktiviert werden:
Zuerst muss die UART-Schnittstelle aktiviert werden. Dafür sollen die folgenden Befehle durchgeführt werden:

Man öffnet die Datei „config.txt“ mit dem Editor „nano“:

```
sudo nano /boot/config.txt
```

Um die UART-Schnittstelle zu aktivieren, muss der Datei folgende Zeile hinzugefügt werden, zum Beispiel am Ende der Datei:

```
enable_uart=1
```

Als nächstes deaktiviert man die serielle Schnittstelle, wobei die Datei cmdline.txt mit dem nächsten Befehl geändert werden muss:

```
sudo nano /boot/cmdline.txt
```

Anschließend entfernt man den Text „console=serial0, 115200“ und speichert die Änderungen.
Zum Schluss muss der Raspberry PI neu gestartet werden, damit die UART-Schnittstelle aktiviert bzw. betriebsbereit ist.

3.15 Modbus RTU Protokoll zur Steuerung des Frequenzumrichters

Zur Steuerung der Motoren, die die Trommel des Kaffeerösters und dessen Gebläse in Drehbewegung setzen, werden zwei Frequenzumrichter von Siemens vom Typ Sinamics V20 eingesetzt. Der Sinamics V20 kann über Modbus RTU-Protocol programmiert bzw. gesteuert werden. Anhand des im Projekt benutzten Mikrocontrollers werden der Trommelmotor und Gebläse über die Frequenz gesteuert und dadurch die Drehzahl ständig überwacht und auf den gewünschten Wert festgelegt.
Um die beiden Motoren zu steuern und das Modbus RTU-Protocol anzusetzen bzw. zu realisieren, wird eine RB-RS485 serielle Schnittstelle benötigt, die die Kommunikation zwischen dem Microkontroller (Raspberry Pi 3) und dem Frequenzumrichter Sinamics V20 unterstützt. Der Sinamics V20 fördert die USS- und Modbus-Kommunikation, daher ist es wichtig über die Parameter festzulegen, dass die RS485-Kommunikation über das Modbus RTU-Protocol verwendet werden soll. Bei der RS485-Kommunikation wird ein abgeschirmtes Twisted-Pair-Kabel benötigt.
Auf dem nachfolgenden Diagramm ist eine Darstellung zum Anschließen des Frequenzumrichters sowie eine detaillierte Erklärung abgebildet.

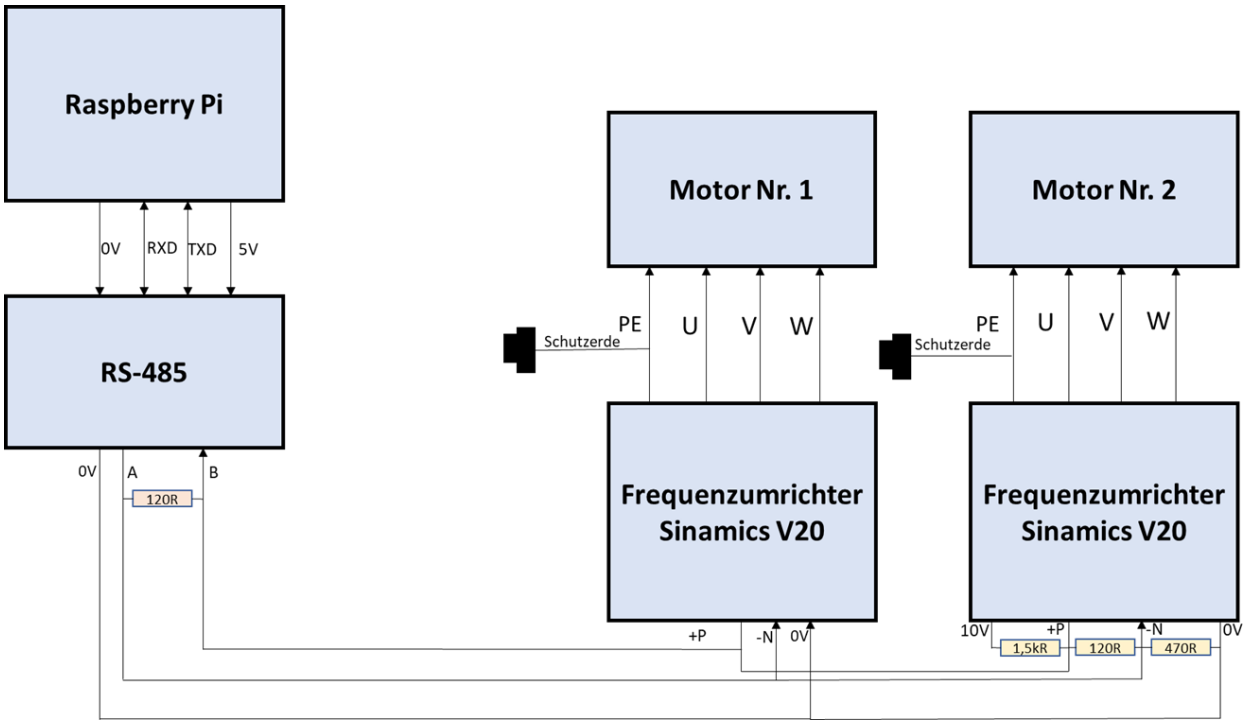

Abbildung 6: Schaltung der RS485-Schnittstelle mit RPi und Frequenzumrichtern

Erstellung des Modbus RTU-Protocol:

Zur Verwendung vom Modbus RTU-Protokoll gibt es in der Literatur bzw. im Internet die libmodbus- und die pymodbus-Bibliothek. Libmodbus ist in der C-Programmiersprache, die pymodbus-Bibliothek ist in Python geschrieben.

Obwohl beide Bibliotheken sich sehr ähneln, und beide keine erkennbaren Nachteile haben, wurde sich bei diesem Projekt für die Pymodbus-Bibliothek entschieden.
Um die Bibliothek erfolgreich verwenden zu können, müssen zunächst einige Schritte befolgt werden:

Zuerst muss die Package „pip“ installiert werden, damit die Bibliothek pymodbus herunterladen werden kann, was mit folgendem Befehl erfolgt:

```
sudo apt-get install python-pip
```

Danach kann die Bibliothek pymodbus installiert werden. Dies geschieht mit diesem Befehl auf dem Raspberry Pi Terminal:

```
sudo pip install pymodbus
```

Nach dem Ausführen des Befehls, steht die Pymodbus-Bibliothek bereit zur Verwendung.

Überblick zur Modbus-Kommunikation des Frequenzumrichters Sinamics V20:

Nur der Master (Raspberry Pi) kann eine Kommunikation beginnen, auf die der Slave (Frequenzumrichter Sinamics V20) antwortet. Es gibt zwei Möglichkeiten dem Sinamics V20 eine Meldung zu übermitteln. Die eine ist die Unicast-Methode, die andere die Broadcast-Methode. Bei dem Projekt wird nur die erste Methode angewandt, d.h. der Slave bekommt die Adresse zwischen 1 und 247, bei der der Raspberry Pi den Umrichter anspricht.

Empfängt Sinamics V20 an ihn gerichtete Meldungen, so erteilt ihm der Funktionscode eine Handlungsanweisung. Für die im Funktionscode festgelegte Aufgabe empfängt der Umrichter einige Daten. Zur Fehlerprüfung wird ein CRC-Code übermittelt.
Nach Empfang und Verarbeitung einer fehlerfreien Unicast-Meldung sendet der Modbus-Umrichter eine Antwort. Tritt ein Verarbeitungsfehler auf, antwortet der Umrichter mit einer Fehlermeldung.

Eine Meldung bzw. Befehlsnachricht muss statische Zeichen gemäß der Skizze enthalten.

Pause am Anfang	Befehlsnachricht oder Meldung					Pause am Ende
>= 3,5 Zeichenlauf-zeit		Protokolldateneinheit		CRC		>= 3,5 Zeichenlauf-zeit
	Slave Adresse	Funktionscode	Daten	2 Byte		
	1 Byte	*1 Byte*	*0 …. 252 Byte*	*CRC niedrig*	*CRC hoch*	

Tabelle 4: Befehlsnachrichten für den Frequenzumrichter

Der Frequenzumrichter Sinamics V20 unterstützt nur drei Funktionscodes. Beim Empfang anderer Funktionscodes wird eine Fehlermeldung gesendet.

Funktionscode FC3: Speicherregister lesen

Beim Empfang einer Befehlsnachricht mit FC3 werden vier Byte an Daten erwartet. Eine Meldung zum Lesen von Registern wird gemäß der Skizze geschrieben.

		Dateneinheit					
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Slave Adresse	FC 0x03	Startadresse (höchstwertiges Byte)	Startadresse (geringstwertiges Byte)	Anzahl der Register (höchstwertiges Byte)	Anzahl der Register (geringstwertiges Byte)	CRC	CRC

Tabelle 5: Meldung zum Lesen der Register mit FC3

Funktionscode FC6: In einzelnes Register schreiben

Beim Empfang einer Befehlsnachricht mit FC6 werden vier Byte an Daten erwartet. Eine Meldung zum Schreiben in einem Register wird gemäß der Skizze geschrieben.

		Dateneinheit					
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Slave Adresse	FC 0x06	Startadresse (höchstwertiges Byte)	Startadresse (geringstwertiges Byte)	Neuer Registerwert (höchstwertiges Byte)	Neuer Registerwert (geringstwertiges Byte)	CRC	CRC

Tabelle 6: Meldung zum Lesen der Register mit FC6

Sinamics V20 unterstützt auch den Funktionscode FC16 zum Schreiben in mehrere Register. In dem Projekt werden nur die zwei ersten Funktionscodes gebraucht, daher wird auf den FC16 nicht eingegangen.

Wie vorher erwähnt wurde, werden zur Steuerung und Regelung des Kaffeerösters der Trommelmotor und Gebläse über die beiden Frequenzumrichter gesteuert. Daher sind die nachfolgenden Python Programme notwendig, um den Trommeldrehzahl und den Airflow zu regulieren.

4 Hardware

4.1 Temperatur-Modul Thermoelement mit Ausgang I2C

Beschreibung

Bei Thermoelementen handelt es sich um ein in der Industrie standardisiertes Verfahrend zur Messung von Temperaturen in einem weiten Bereich. Bei den hier verwendeten Elementen geht dieser von -270 bis 1360 C°. Die Thermoelemente verfügen über eine digitale I2C-Schnittstelle, an der die Temperatur gemessen wird und können einfach an einen Mikrocontroller angebracht werden. Hier werden die Daten gesammelt, gespeichert und umgerechnet und können so bequem abgelesen und überprüft werden. Bei dieser Art der Temperaturerfassung ist es außerdem nötig, eine zusätzliche, absolute Temperaturmessung durchzuführen, da die Elemente relativ zur Temperatur der Ausgleichsstelle messen.

Technische Daten

Messprinzip	Thermoelektrische Spannungsmessung (Seebeck-Effekt)
Signalverarbeitung	Digital im ASIC
Messbereich	Typ -1370 -270…+1370 °C
Auflösung	T3 -1370 ca. 0,5 K (Typ K)
Temperaturmessung	-32…+97°C, Pt1000/B
Ausgleichsstelle	
Ansprechzeit Modul	<30msec
Abmessungen Modul	9,0x46,0x5,0mm
Betriebsspannung	6…24V
Stromaufnahme	<3mA
Ausführung	SMD-Modul
Anschluss	Stiftleiste, 6-polig, RM2,54mm
I2C-Interface	100/400kHz, Adresse 0x78
CE-Konformität	2014/30/EU
EMV-Störaussendung	EN 61000-6-3:2011
EMV-Störfestigkeit	EN 61000-6-1:2007
Artikelnummer	THMOD-I2C-1370

Tabelle 7: Technische Daten der Thermoelemente

4.2 Zahnrad-Drehzahlsensor GS1005 Serie

Beschreibung

Um die Drehzahl der Trommel zu erfassen, wird ein Zahnrad-Drehzahlsensor auf Hall-Effekt Basis eingesetzt. Der Sensor erfasst die Bewegungen der metallischen Zahnräder und sendet ein Rechtecksignal aus. Da sich die Trommel des Kaffeerösters unterschiedlich schnell dreht und mitunter auch sehr langsam, eignet sich der Drehzahlsensor bestens, da er auch extrem langsame Bewegungen erfassen und messen kann.
Dank seines eloxierten Aluminiumgehäuses lässt sich der Sensor im Bereich der Lebensmittelindustrie einsetzen.

Technische Daten

Bestellnummer	GS100502
Betriebsspannung (VDC)	5,0-24
Versorgungs-Strom (mA max.)	6
Ausgang	Kollektor
Ausgangs-Sättigungs-Spannung (mV max.)	400
Ausgangsstrom (mA max.)	20
Einsatztemperaturbereich (°C)	-40 – 125
Lagerungstemperaturbereich (°C)	-40 – 125
Gewinde	M12-1
Zylinderlänge	65mm
Leitung	20AWGx1mBBB
Stecker	-----

Tabelle 8: Technische Daten des Drehzahlsensors

4.3 Zahnrad zur Drehzahlaufnahme

Zum Erfassen der Drehzahl der Trommel mit dem Drehzahlsensor vom Typ GS1005 Serie wird ein Zahnrad gebraucht. Das Zahnrad ist an der Trommelwelle axial angebracht. Es ist ferromagnetisch, aus Stahl und hat 56 Zähne. Die Abbildung 7 zeigt das verwandte Zahnrad.

Abbildung 7: Zahnrad zur Drehzahlaufnahme

4.4 I2C-Repeater V2 für Raspberry PI

Die Problematik des Raspberry Pi besteht darin, dass die I2C-Pins (SDA und SCL) lediglich für eine Spannung von 3,3V ausgelegt ist und die I2C-Schnittstelle der Slaves nur mit einer 3,3V Versorgungsspannung funktionieren. Deswegen ist es nötig, die Spannung vorher herunter zu regeln, um das Gerät nicht zu zerstören. Dafür wurde ein I2C-Repeater verwendet. Dieser ist eine Erweiterungskarte zum Aufstecken auf die GPIOS des Raspberry Pi. Der I2C-Repeater ermöglicht den sogenannten Level-Shift, das heißt die für die I2C-Schnittstellen notwendigen 5V Spannung wird auf 3,3V angepasst. Somit sind die GPIOs geschützt. Mit Hilfe dieses Repeaters sind alle Slaves an dem Raspberry Pi mit der I2C Schnittstelle verbunden.

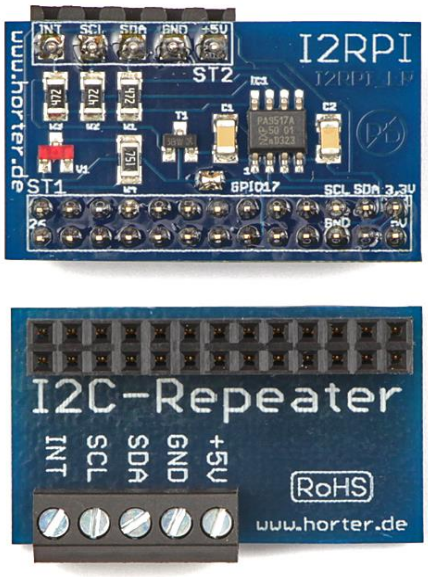

Abbildung 8: I2C-Repeater zur Kommunikation der I2C-Slaves mit Raspberry Pi

4.5 Drucksensor OMRON D6F-PH

Beschreibung

Um den Luftmassenstrom zu messen, der aus der Trommel strömt, benötigt man einen Drucksensor, der die Differenz zwischen dem Druck der ausströmenden Luft und dem im Raum herrschenden Druck misst und die Daten speichert.

Verwendet wurde hierfür ein Drucksensor vom Typ Omron D6F-PH, der die Druckdifferenz zwischen zwei Absolutdrücken misst. Hierbei werden zwei Messkammern verwendet, die durch eine Membran hermetisch voneinander getrennt sind. Je nachdem wieviel Druck herrscht, dehnt sich die Membran unterschiedlich weit aus und dient als Maß für die Größe der Druckdifferenz. Mithilfe einer Auswerteelektronik wird die Bewegung der Membran erfasst und gemessen, verstärkt und in ein elektrisches Signal umgewandelt, das gespeichert und ausgelesen werden kann.
Zum besseren Verständnis der Funktionsweise des Drucksensors folgt ein Bild mit Erläuterungen zu diesem Gerät.

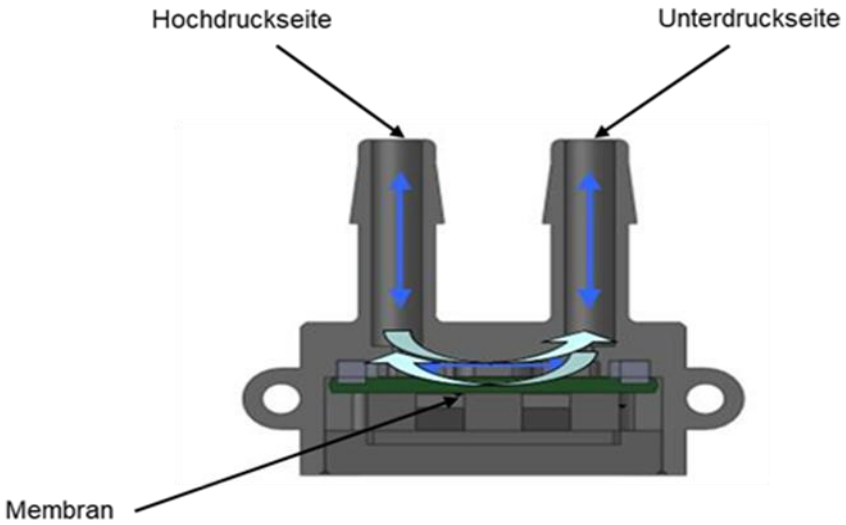

Abbildung 9: Funktionsweise des Drucksensors

Der Drucksensor kommuniziert mithilfe einer ihm vorher vom Werk eingestellten Adresse 0X6C über I2C-Schnittstelle mit dem Raspberry Pi. Dadurch lassen sich die Werte des Druckes mit Matlab auslesen und nach kleinen Umrechnungen in dezimale Werte umwandeln. Daraus lässt sich nun einfach die physikalische Größe des Drucks bestimmen.

Auch hierfür befindet sich nachfolgend eine Skizze, die die Kommunikation zwischen den beiden Geräten vereinfacht darstellt sowie den Zugriff des Raspberry Pi auf den Speicher des Drucksensors.

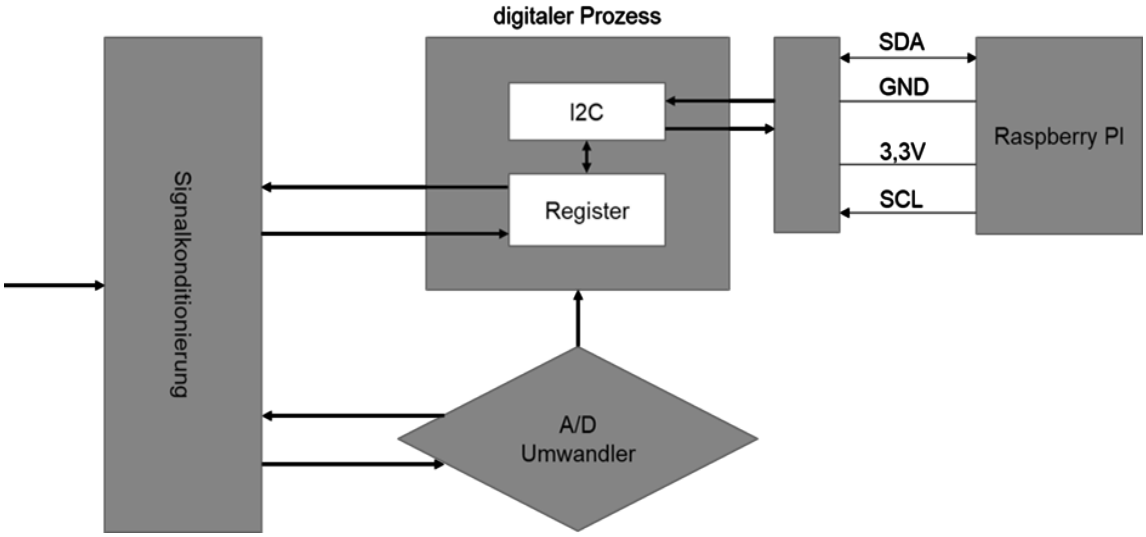

Abbildung 10: Kommunikation zwischen Drucksensor und Raspberry Pi

Technische Daten

	Min	Typ	Max	Einheit
Druckdifferenzbereich	-50	-	50	Pa
Auflösung	-	12	-	Bit
Nullpunktgenauigkeit	-0,2	-	+0,2	Pa
Messgenauigkeit	-3	-	+3	%R.D.
Verschiebungsspanne bei 10°C Temperaturänderung	-0,5	-	+0,5	%R.D.
Reaktionszeit	-	33	50	msek
Umgebungstemperatur	-20	-	80	°C
Lagertemperatur	-40	-	80	°C
Umgebungsluftfeuchtigkeit	35	-	85	%RH
Lagerluftfeuchtigkeit	35	-	85	%RH
Versorgungsspannung	2,3	3,3	3,6	VDC
Stromverbrauch	-	-	6	mA
Geschwindigkeit	-	-	400	kHz

Tabelle 9: Technische Daten des Drucksensors

4.6 Pitot-Rohr zur Druckaufnahme

Zur Messung des Luftmassenstroms, der das Abfuhrrohr durchströmt, muss der Druck an diesem gemessen und anschließend daraus der Luftmassenstrom berechnet werden. Auf der nachfolgenden Skizze sieht man, wo die Pitot-Rohre angebracht sind und wo sich die Druckmesspunkte befinden. Um möglichst genaue Messungen zu erzielen wird der statische Druck an dem Punkt 1 und der Staudruck an dem Punkt 2 gemessen. Wichtig hierbei ist, dass der statische Druckmesspunkt direkt an der Öffnung des Rohres angebracht ist. Der Messpunkt für den Staudruck hingegen befindet sich in der Mitte des Rohres. Die beiden Messpunkte sind über Rohre mit jeweils 4,9 mm Durchmesser mit dem Drucksensor an der Platine verbunden. Die Pitot-Rohre wurden mithilfe von M10 Schrauben am Abfuhrrohr befestigt.

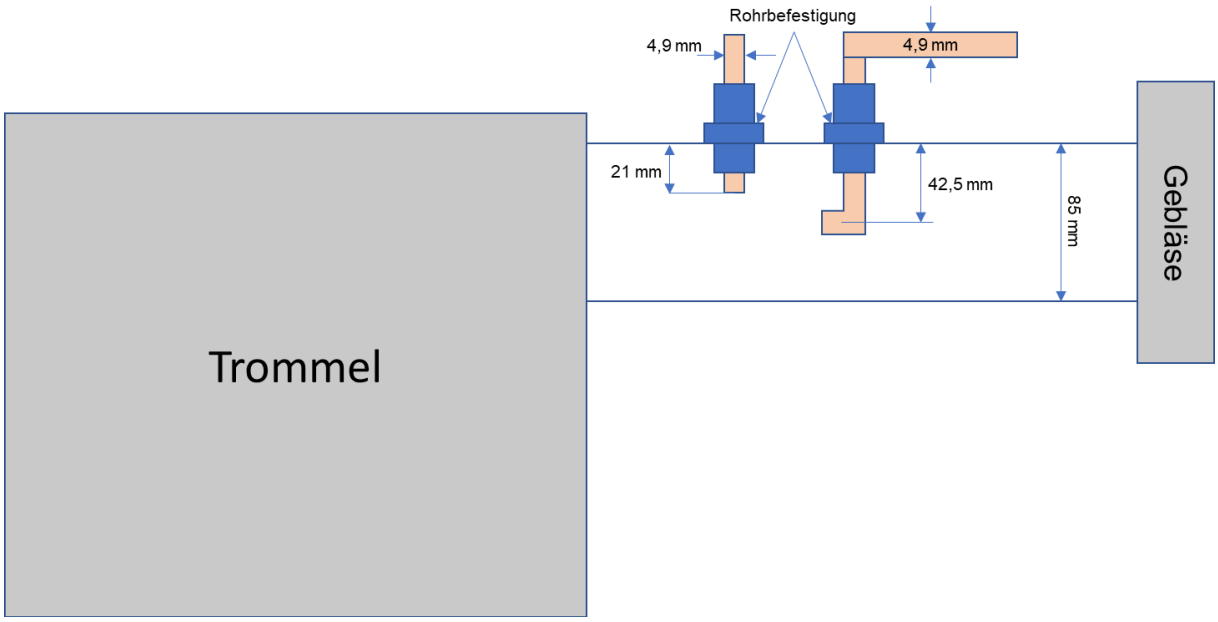

Abbildung 11: Darstellung der Druckmesspunkte an dem Abfuhrrohr der Trommel

Auf den folgenden Bildern sieht man die selbstgefertigten Rohre, die zur Messung des Drucks angefertigt und verwendet werden.

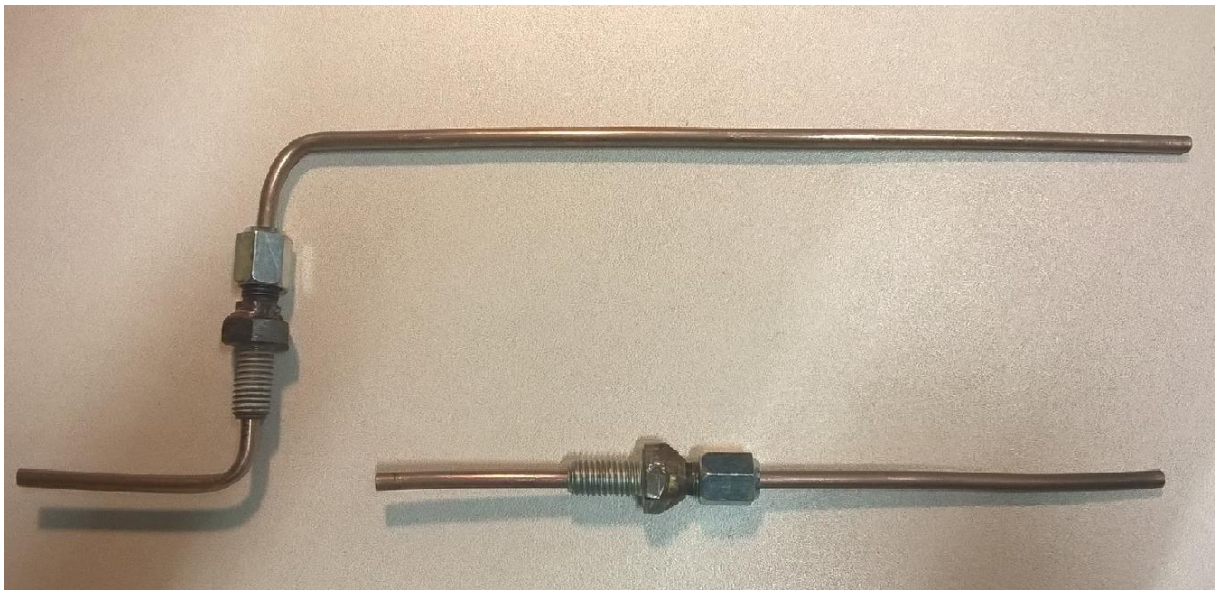

Abbildung 12: Pitot-Rohre

4.7 Frequenzumrichter SINAMICS V20 von Siemens

Zur Steuerung der Motoren, die die Trommel des Kaffeerösters und dessen Gebläse in Drehbewegung setzen, werden zwei Frequenzumrichter von Siemens vom Typ Sinamics V20 eingesetzt. Der Sinamics V20 kann über Modbus RTU-Protocol programmiert bzw. gesteuert werden. Anhand des im Projekt benutzten Microkontrollers werden der Trommelmotor und Gebläse über die Frequenz gesteuert und dadurch die Drehzahl ständig überwacht und auf den gewünschten Wert festgelegt. Die folgende Skizze zeigt den im Bau des Reglers eingesetzten Frequenzumrichter an.

Abbildung 13: Frequenzumrichter Sinamics V20 von Siemens

4.8 Schnittstelle RB-RS485

Für die Kommunikation zwischen dem Frequenzumrichter und dem Raspberry Pi wurde eine Schnittstelle RB-RS485 mit dem Maxim MAX481 DS Controller verwendet. Diese fungiert als Erweiterungskarte und ist kompatibel zum Raspberry Pi 3 und kann hierauf aufgesteckt werden. Mit dieser werden die UART Signale vom GPIO14 und GPIO15 auf RS485 Level übersetzt und auf die DB9 Buchse auf Pin2 (A) und Pin3 (B) ausgegeben. Auf den zusätzlichen Pin Headern ist es außerdem möglich, direkt auf die GPIO Signale zuzugreifen.
Auf dem folgenden Bild ist die RB-RS485 Schnittstelle zu sehen.

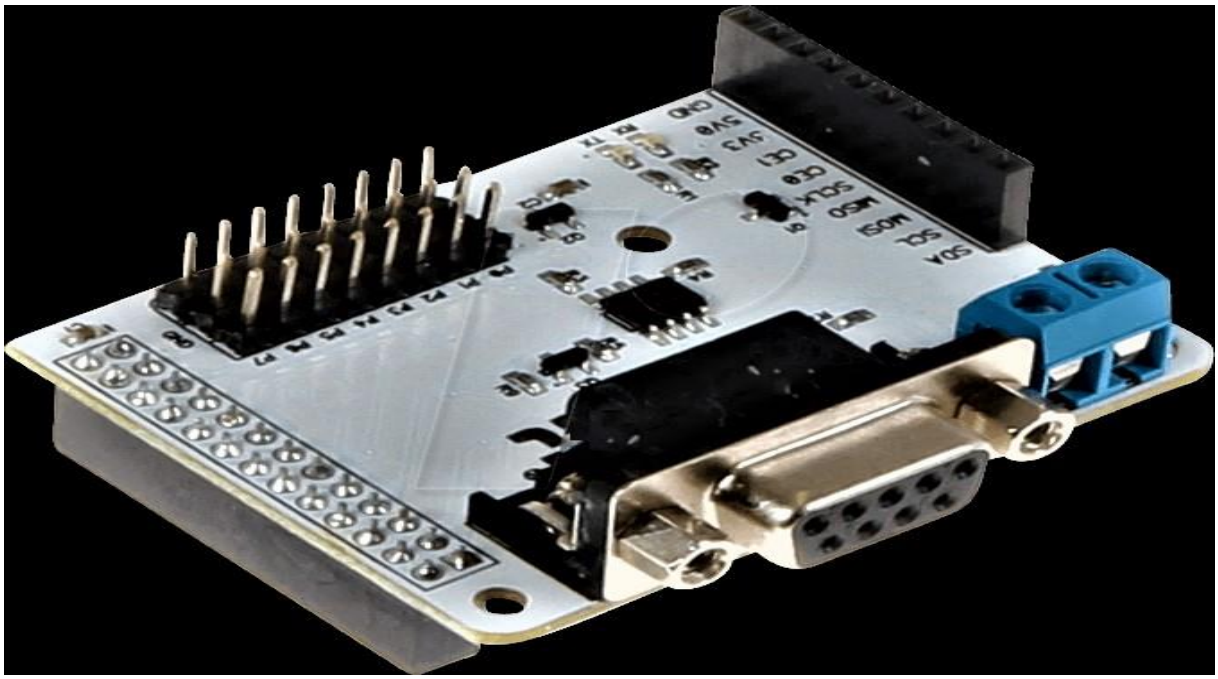

Abbildung 14: RS-RB485 Schnittstelle

4.9 Herstellung des Prototyps

Die Herstellung des Prototyps lässt sich in zwei Phasen gliedern. In der ersten Phase wird die Platine hergestellt, beziehungsweise die einzelnen Bauteile auf diese gelötet und gefädelt. In der zweiten Phase muss die vorher fertig gestellte und getestete Platine mit den übrigen Teilen auf einer Sperrholzplatte befestigt werden, die als Prototyp verwendet wird.

4.9.1 Herstellung der Platine

Zur Herstellung der Platine wird zuerst ein theoretischer Schaltplan erstellt. Dafür lässt sich im Vergleich zu anderen Platinen-Layouts Software wie Kicad, etc. am günstigsten Eagle verwenden.

Als erstes werden alle Schaltungen einzelner Sensoren berechnet und aufgestellt, danach sind sie durch Versuche auf ihre Richtigkeit und ihre physikalische Korrektheit zu testen. Um Fehlfunktionen zu vermeiden, werden alle Sensoren zuerst einzeln auf ihre Funktionsfähigkeit geprüft und erst anschließend in ihrer Gesamtheit.

Zudem ist es sehr wichtig, hohe Spannungen bzw. hohe Ströme auf die Platine zu vermeiden. Da man vom Raspberry Pi nur eine Höchstspannung von 5 Volt abzapfen kann, sollte auch die Platine auf eben diese Spannung beschränkt werden.

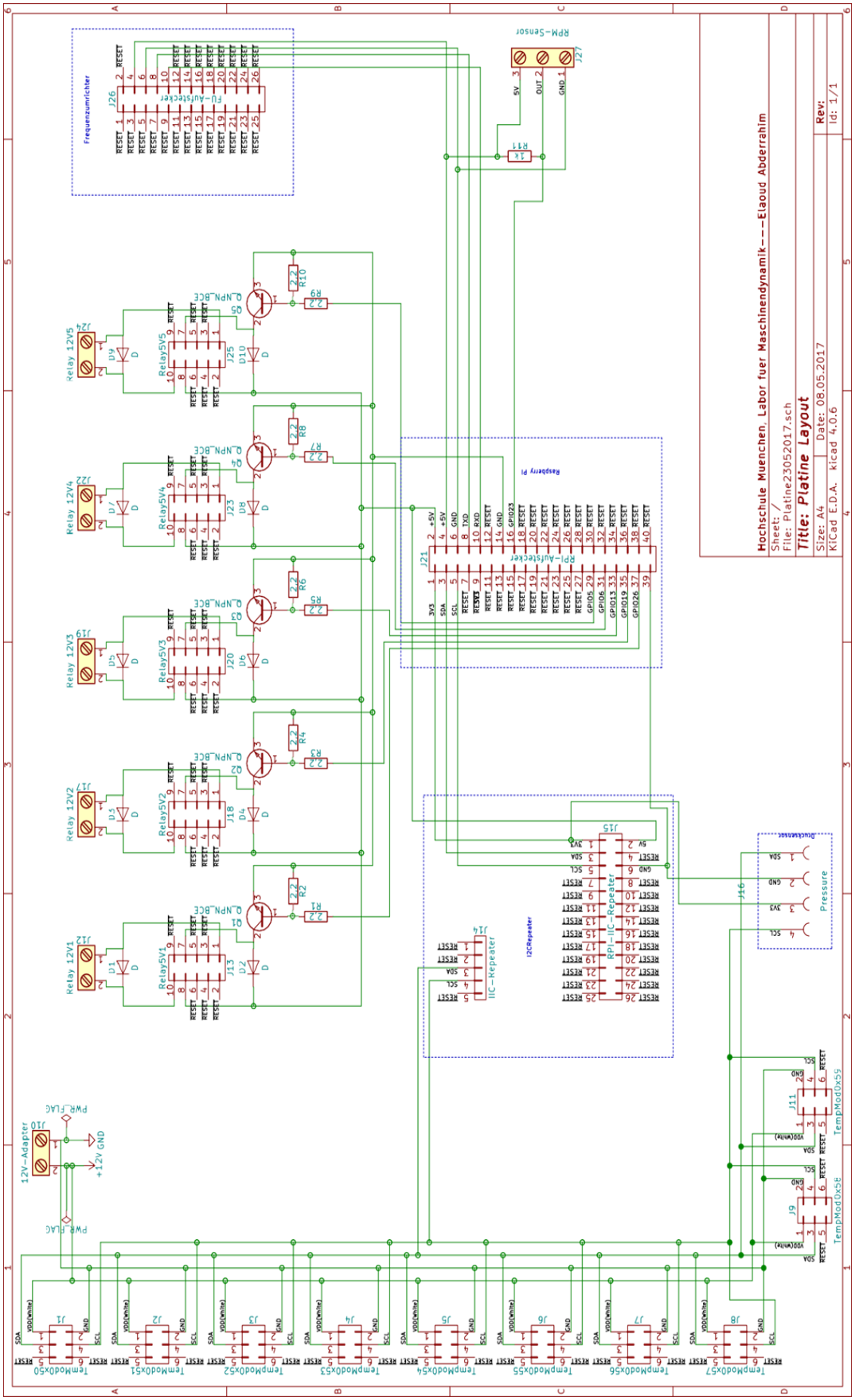

Abbildung 15: Platinen Schaltplan zur Herstellung der Kommunikation zu den Sensoren

Im Anschluss befindet sich eine Tabelle, in der die verwendeten Bauteile sowie deren benötigte Anzahl zur Herstellung der Platine aufgeführt werden.

Bauteilbezeichnung	Anzahl
Diode 1N4007	10
Kohleschicht-Widerstand 2,2kΩ	10
Transistor BC337-25	5
Lötschraubenklemme, 2 polig RM 2,54	6
Lötschraubenklemme, 3 polig RM 2,54	1
Kohleschicht-Widerstand 1kΩ	1
Temperatur-Modul Thermoelement mit Ausgang I^2C	10
Stifteleiste 2x20 polig RM 2,54	1
Stifteleiste 2x13 polig RM 2,54	2
Raspberry Pi, Erweiterungsplatine RB-RS 485 für FU	1
I2C Repeater	1
Omron-Drucksensor	1
Buchsenleiste 2x13 polig RM 2,54	1
Buchsenleiste 2x3 polig RM 2,54	10
Printrelais 5V	5
Doppel-Europlatine	1
Zylinderschraube M2x25	2
Sechskantmutter M2	2
Stifteleiste 1x4 polig RM 2,54	1
Buchsenleiste 1x4 polig RM 2,54	1

Tabelle 10: Bauteilliste für die Platine

Da für die hergestellte Platine viele Bauteile benötigt wurden, ist das Risiko für eventuelle Kurzschlüsse relativ hoch. Deswegen wurden die Bauteile auf die Platine gelötet und anschließend mit einem Kupferdraht verbunden, also gefädelt. Deshalb werden für die Herstellung neben den oben genannten Bauteilen außerdem noch Kupferlackdraht mit einem Durchmesser von 0,35mm für die Verbindung der einzelnen Bauteile benötigt sowie Verdrahtungskämme, um den Draht an den richtigen Positionen zu fixieren.

4.9.2 Erstellung des Platinen Layout

Für die Erstellung des Platinen Layouts ist die Software Eagle eingesetzt. Anhand der Skizze kann man ein anschauliches Bild über die fertiggestellte Platine machen. Weitere Daten bzw. Information über die Platine sind dem beigefügten Programm zu entnehmen.

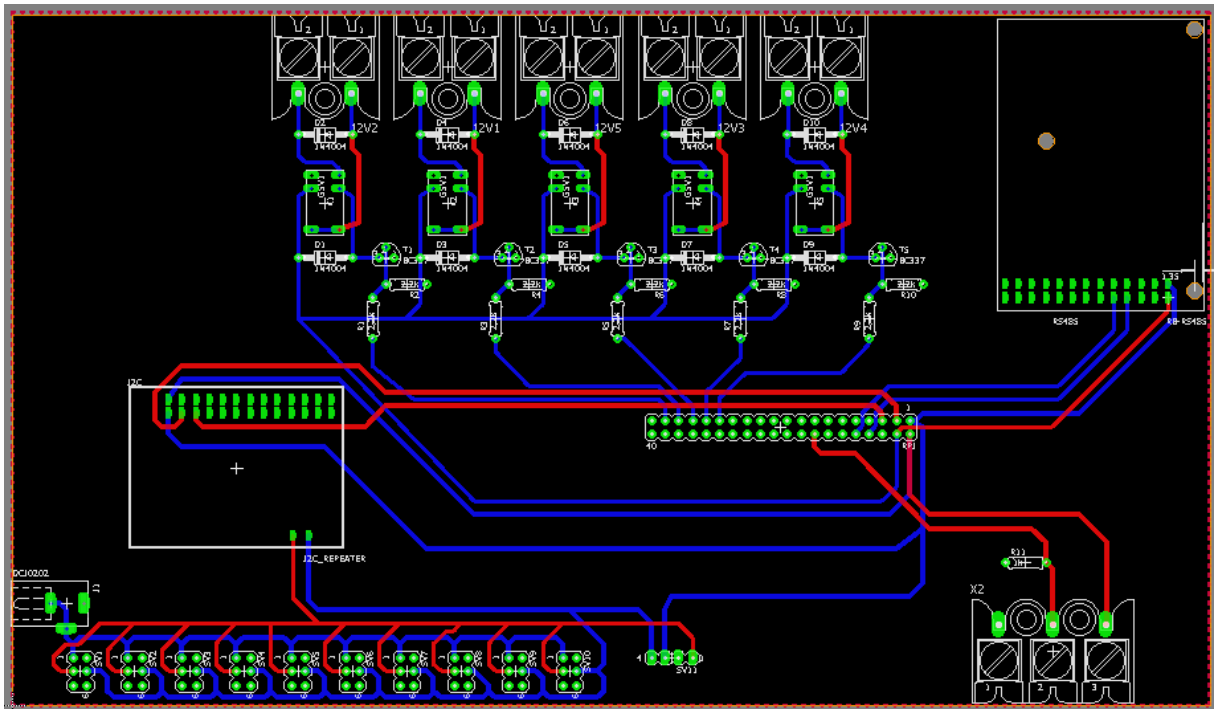

Abbildung 16: Platinen Layout mit Eagle-Software

4.9.3 Montage aller Bauteile auf die Sperrholzplatte

Um nun den Prototyp herzustellen und alle Bauteile zusammen zu befestigen, wird zum Schluss alles auf eine Sperrholzplatte geschraubt. Auch dazu folgt zuerst eine Tabelle mit den hierfür verwendeten und benötigten Teilen:

Bauteilbezeichnung	Anzahl
12 V Netzteil	1
5 V Netzteil	1
Frequenzumrichter Sinamics 20V	2
Relaisbaustein	5
Hutschiene	1
Not-Aus-Schalter	1
RJ45 Netzwerkanschlusskabel	1
Sperrholzplatte 80cmx60cm	1
Raspberry Pi	1
Gelötete Platine	1

Tabelle 11: Bauteilliste für die Montage auf der Sperrholzplatte

Die Bauteile werden nun auf der Sperrholzplatte befestigt. Dabei wird das Raspberry PI mit einem LAN-Kabel direkt an einen Laptop angeschlossen und mit einer Kabelbrücke mit der Platine verbunden.

An den beiden Frequenzumrichter werden zwei Steckdosen angebracht, an die anschließend die Motoren angeschlossen werden, die so frequenzabhängig angetrieben werden können.
Außerdem wird ein Not-Aus- Schalter zwischen dem Regler und dem Stromnetz angeschlossen, um das System im Notfall sofort zu beenden. Dieser wird auch auf die Sperrholzplatte angebracht.
Drehzahlsensor und Thermoelemente sind an den am Röster ausgewählten Messpunkten montiert und mit ungefähr 3 Meter langen Kabel mit der Platine verbunden. Der Drucksensor hingegen ist auf der Platine befestigt und mit etwa 4,9mm Schläuchen mit den Pitot-Rohren an den Messstellen verbunden.
Zur Stromversorgung des Raspberry PI wird außerdem ein Netzteil mit 5 Volt und für die Relais ein Netzteil mit 12 Volt auf der Platte befestigt.
Auf der nachfolgenden Skizze ist zu sehen, wie die einzelnen Bauteile miteinander verbunden und angeschlossen sind. Im Anschluss sieht man ein Foto der fertigen Sperrholzplatte mit den einzelnen Bauteilen.

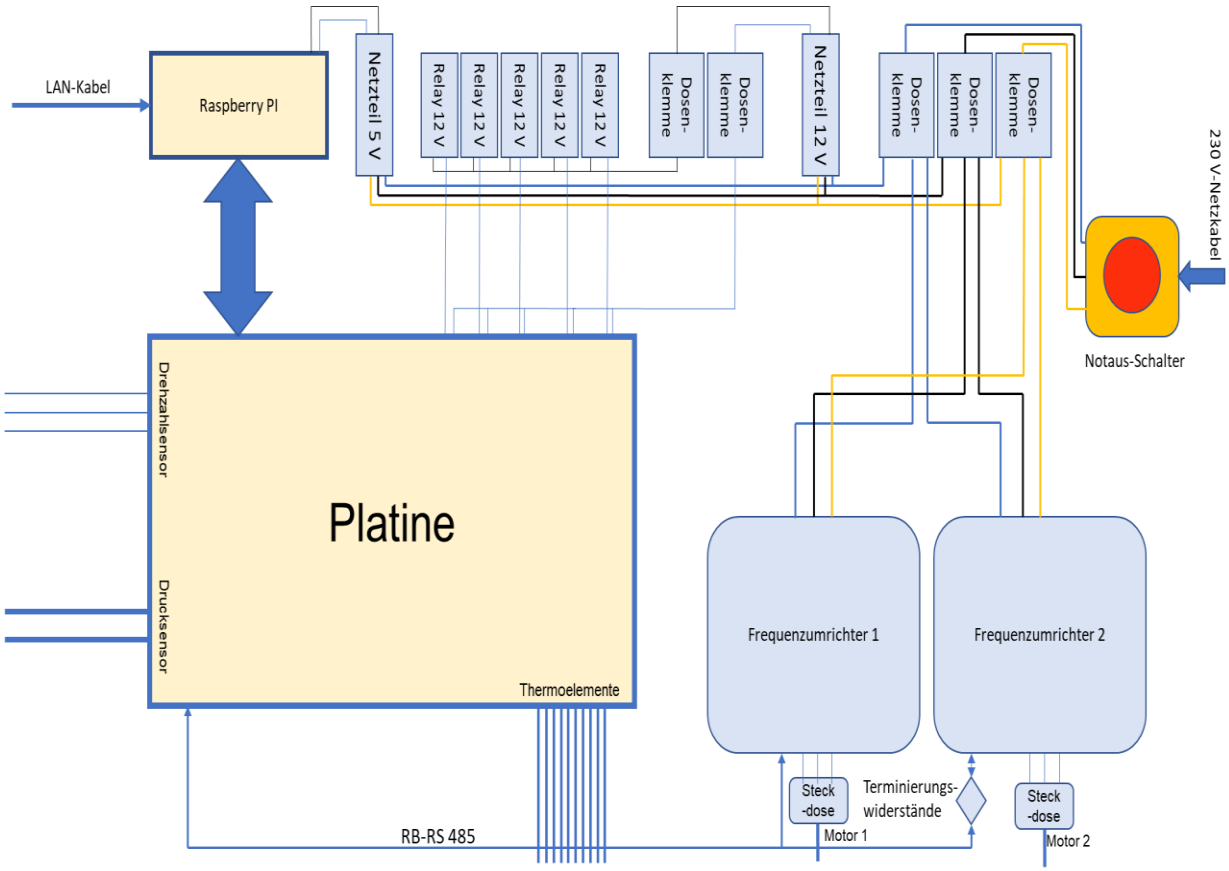

Abbildung 17: Skizze zur Montage der Bauteile auf der Sperrholzplatt

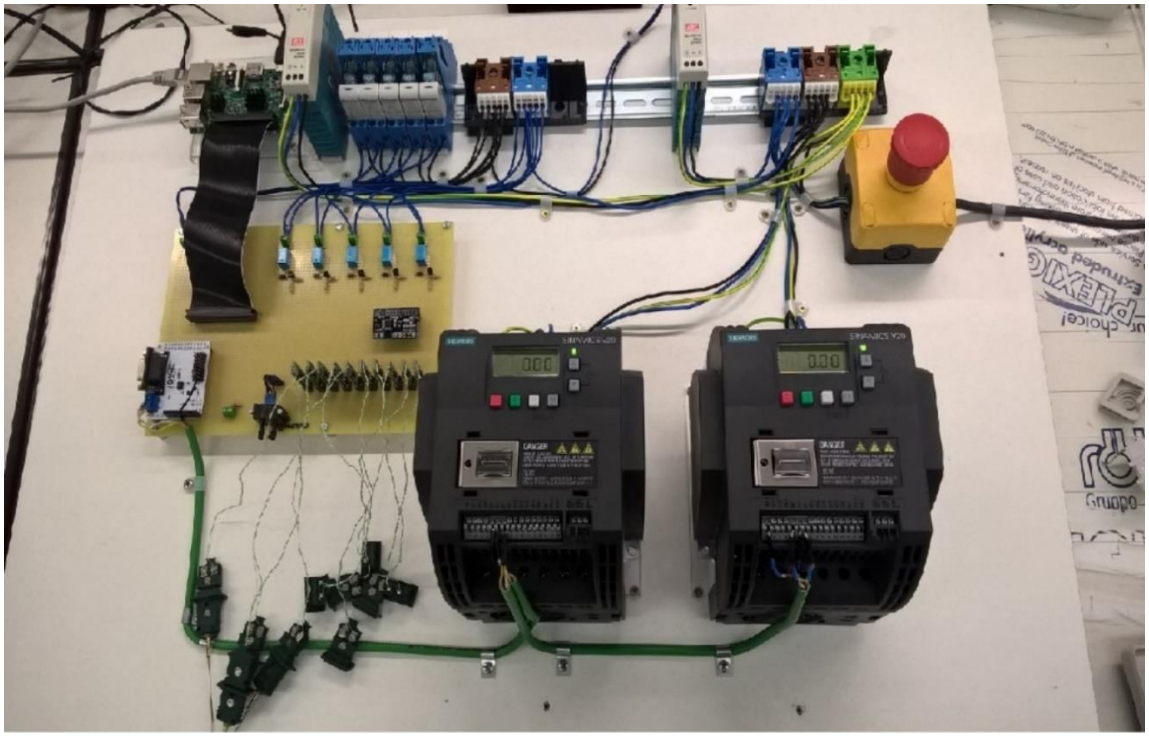

Abbildung 18: Fertig montierte Sperrholzplatte mit den einzelnen Bauteilen

5 Inbetriebnahme

Zur Inbetriebnahmen haben folgende Schritte zu erfolgen:

- Temperatursensoren installieren
- Drucksensor installieren
- Drehzahlsensor installieren
- Motoren anschließen
- PC mit dem Raspberry Pi verbinden

In der aktuellen Konfiguration werden 5 Temperatursensoren verwendet. Die Belegung der Sensoren kann anhand der I2C-Adresse des Thermoelements zugeordnet werden. Jedes Thermoelement besitzt eine individuelle Adresse, welche auf einen Aufkleber auf dem Element vermerkt ist.
Die Thermoelemente müssen daher an nachfolgenden Stellen angebracht werden.

I²C-Adresse	Installationsort
0x50	In der Trommel, innerhalb der eingefüllten Bohnen
0x51	In der Trommel oberhalb der Bohnen
0x52	Mittig im Abluftrohr
0x53	Am Gehäuse
0x54	Frei liegend am auf der Sperrholzplatte für die Umgebungstemperatur

Tabelle 12: Installationsort der Temperatursensoren

Die ersten vier Sensoren können in der Reihenfolge nicht variiert werden. Sofern weitere Sensoren hinzugefügt werden sollen, oder der Sensor für die Umgebungstemperatur entfernt werden soll, ist nach der entsprechenden Logik in der Setup-Datei zu hinterlegen.
Aktuell werden die Sensoren 0x50, 0x51 und 0x53 mittels Durchführungen von außen an den entsprechenden Einsatzort gebracht.
Der Gehäusesensor 0x53 wird in der hinteren Abdeckung des Rösters eingeklemmt.
Der Sensor 0x54 dient lediglich zur Erfassung der Umgebungstemperatur. Dieser Sensor kann überall mit entsprechenden Abstand zum Röster positioniert werden.

Bei dem Drucksensor wird lediglich ein Eingang mittels eines flexiblen Schlauches mit dem unter Kapitel 4.6 beschriebenen Pitot-Rohrs verbunden. Der andere Eingang bleibt frei und

erfasst somit den Umgebungsdruck. Da bei der Ermittlung des Drucks softwareseitig der Betrag des Druckwerts ausgegeben wird, ist es nicht relevant welcher Sensoreingang verwendet wird.

Der Drehzahlsensor wird berührungslos an das Zahnrad der Trommelwelle montiert. Aktuell wird ein Zahnrad mit 55 Zähnen verwendet. Sofern eine andere Zähnezahl vorliegt, muss diese Änderung dementsprechend in der Software berücksichtigt werden.

Die Motoren werden mittels eines dreiphasigen Kabels (U, V und W) mit den Frequenzumrichtern verbunden. Die Belegung der Kabel (Trommel- oder Airflowmotor) ist der Beschriftung auf der Frequenzumrichter-Abschirmung zu entnehmen.

Im letzten Schritt muss nun noch der Raspberry Pi per Ethernetkabel mit dem PC verbunden werden.

6 Betriebsanleitung

Aufruf des GUI

Die Hardware ist zunächst wie in Kapitel 5 beschrieben zu installieren. Durch Anstecken des Hauptsteckers und Entriegeln des Notausschalters werden alle Komponenten mit Spannung versorgt. Auch der Rasperry Pi startet. Sobald dieser hochgefahren ist (ca. nach einer Minute), kann der Röster über das GUI betrieben werden.
Das GUI wird durch Starten der Setupdatei „setup_from_file“ aufgerufen. Nach einigen Sekunden ist der Setupprozess abgeschlossen und das GUI erscheint in einem Fenster. Die Messwerterfassung und Darstellung erfolgt automatisch.

Steuerung der grundlegenden Rösterfunktionen

Über die untere Bedienleiste (Abbildung 19) können die grundlegenden Funktionen des Rösters gesteuert werden.

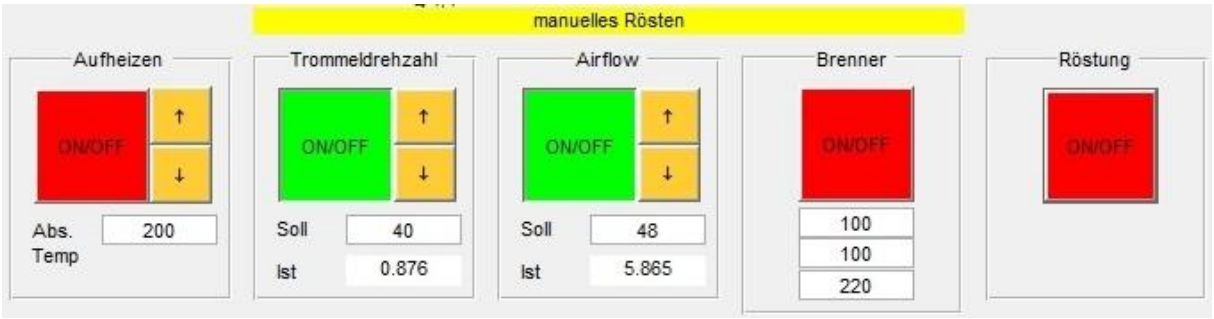

Abbildung 19: Guided User Interface, Element 1

Der Button „Aufheizen“ startet den Aufheizvorgang. Der Röster heizt dann solange auf, bis die im Textfeld hinterlegte Temperatur (Gehäusetemperatur) erreicht wird. Die hinterlegte Temperatur wird durch An- und Abschalten des Brenners konstant gehalten, bis der Button wieder deaktiviert wird. Sofern nun in den Modus „manuelles Rösten“ gewechselt werden soll, muss der Aufheizbutton zu deaktiviert und die Software über den Button „Reset“ (Abbildung 20) zurückgesetzt werden.

Alle drei Buttons des Bereichs „manuelles Rösten“ sind zusammengehörig. Über diesen Bereich können alle notwendigen Eingaben für die Durchführung einer manuellen Röstung vorgenommen werden.

Bevor der Brenner gestartet werden kann, müssen die Trommel und der Airflow aktiviert werden. In den zugehörigen Textfeldern von „Airflow“ und „Trommeldrehzahl“ wird die gewünschte Frequenz der Motoren in Hertz hinterlegt. Damit der Brenner über den ON/OFF-Button gestartet werden kann, müssen folgende Bedingungen erfüllt sein:

- ON/OFF-Button unter „Trommeldrehzahl“ und „Airflow“ aktiviert
- Frequenz in beiden Textfeldern unter „Trommeldrehzahl“ und „Airflow“ ist größer als 30 Hz

Zudem sollten vor dem Start des Brenners alle drei im Bereich „Brenner“ befindlichen Textfelder mit Temperaturen beschrieben werden, die über der maximal erreichbaren Röstertemperatur liegen (z.B. 1000 °C). Erst nachdem durch Einfüllen der Bohnen ein Temperaturabfall und ein anschließender Temperaturanstieg zu beobachten ist, können die Temperaturen auf die gewünschten Werte geändert werden. Bei Durchschreiten der gewählten Temperaturen erfolgt die Ausgabe eines Signaltons.
Sofern die oben angegebenen Bedingungen erfüllt sind, kann nun der Brenner über den ON/OFF-Button gestartet werden. Der Brenner heizt solange auf, bis die im untersten Textfeld hinterlegte Temperatur erreicht wird. Dann schaltet der Brenner ab, der Button nimmt wieder den Zustand „OFF“ ein. Airflow und Trommeldrehzahl bleiben weiterhin aktiv, bis diese über die zugehörigen Buttons deaktiviert werden.
Soll nun eine neue Aktion durchgeführt werden, ist wiederrum der Button „Reset“ zu betätigen.

Der Button „Röstung“ ist für eine automatische Röstung nach vorgegebenen Röstprofil vorgesehen. Derzeit ist der Button noch mit keiner Funktion belegt.

Temperaturanzeige und sonstige Funktionen

Im zweiten Element des GUI (Abbildung 20) werden die Temperaturen der Bohnen, der Trommel und des Airflows ausgegeben.

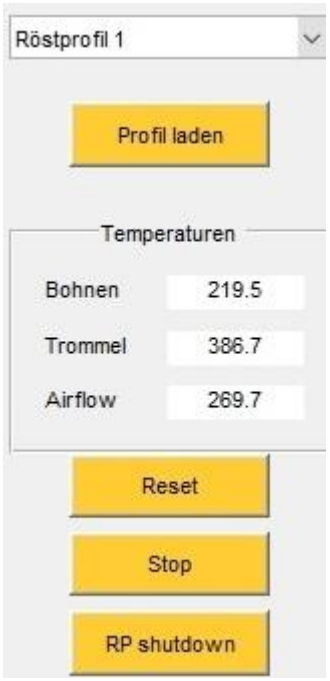

Abbildung 20: Guided User Interface, Element 2

Über den Button „Profil laden“ können vorher aufgezeichnete Röstprofile in einen Zwischenspeicher geladen werden. Diese Profile dienen zukünftig als Referenz für eine automatische Röstung.

Der Button „Reset“ setzt zunächst alle Buttons auf den Zustand „OFF“ zurück und führt einen Reset der Hardware durch (Neuverbindung mit Sensoren und Frequenzumrichter). Zudem wird der Plot (Abbildung 21) zurückgesetzt, womit die graphische Darstellung der Messdaten wieder bei dem Zeitpunkt 0 beginnt.

Durch den Button „Stop“ wird der Timer und die damit verbundene Messwerterfassung gestoppt. Die graphische Darstellung in dem Plot wird eingefroren. Der Zustand kann durch Aktivieren des Reset-Buttons beendet werden

Über den Button „RP shutdown“ wird der Raspberry Pi heruntergefahren. Dies sollte mindestens eine Minute vor dem Trennen der Spannungsversorgung erfolgen, damit der Raspberry Pi keine Schäden davonträgt.

Graphische Darstellung

Zur Darstellung der Bohnentemperatur und des Gradienten der Bohnentemperatur dient der Plot-Bereich (Abbildung 21) des GUI. Über einer Zeitachse (in Sekunden) werden beide Daten

graphisch dargestellt. Die blaue Kurve und die zughörige linke vertikale Achse repräsentieren den Verlauf der Bohnentemperatur. Die rote Kurve und die zugehörige rechte Achse repräsentieren den Gradienten der Bohnentemperatur.
Die Darstellung wird des Graphen wird neu gestartet, wenn eine der folgenden Aktionen durchgeführt wird:

- Brenner-Button auf „ON“
- Aufheiz-Button auf „ON“
- Reset-Button geklickt
- Neustart der Software

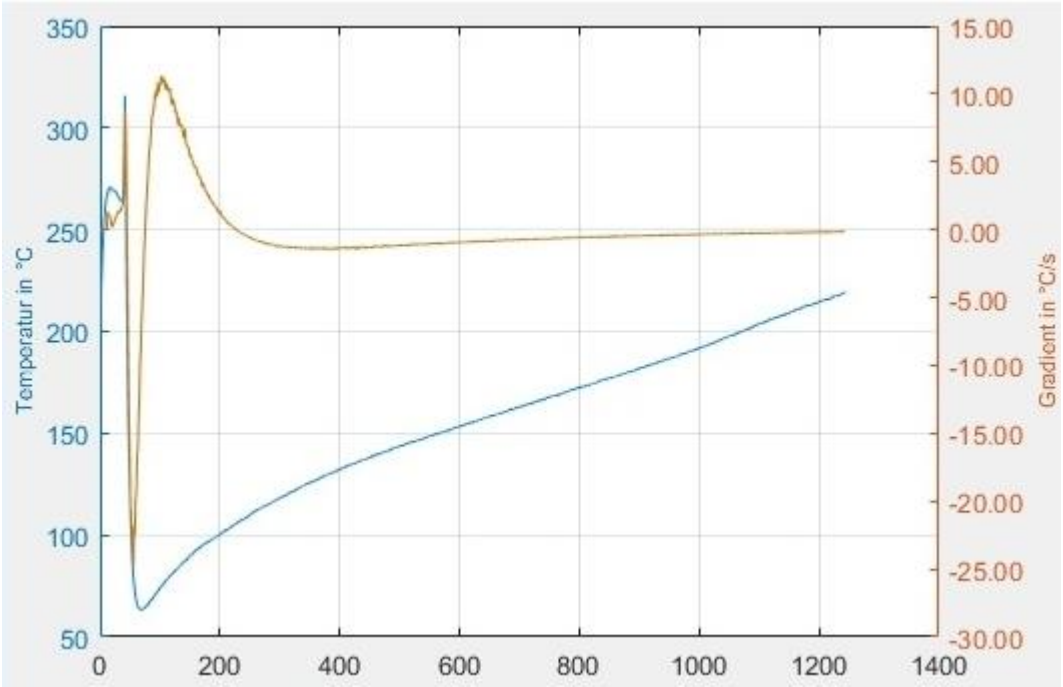

Abbildung 21 Guided User Interface, Plot

7 Ausblick

Das zukünftige Ziel des Projektes ist eine Weiterentwicklung der Software zur vollautomatischen Durchführung des Röstprozesses. Des Weiteren soll eine Überwachung des Aufheizprozesses und eine Regelung hinsichtlich der Bohnentemperatur erfolgen. Die Hardware soll zudem noch kompakter gestaltet werden.

In der aktuellen Version der Software ist es bereits möglich einen automatisierten Aufheizprozess durchzuführen. Es wird allerdings noch nicht überwacht, inwiefern dieser von einer optimalen Aufheizkurve abweicht. Um bereits während des Prozesses Fehlfunktionen zu erkennen, muss noch eine Überwachungsfunktion integriert werden, die Abweichungen von der optimalen Aufheizkurve erkennt und meldet.
Da derzeit nur der Modus „manuelles Rösten“ integriert ist, muss die Software um einen PID-Regler erweitert werden, der anhand noch zu ermittelnder Reglerparameter den Verlauf der Bohnentemperatur beeinflusst. Hierfür soll ein manuell aufgezeichnetes Röstprofil als Referenz dienen. Da die Regelstrecke kaum theoretisch nachgebildet und beschrieben werden kann, sind die Reglerparameter dabei experimentell zu bestimmen. Als Stellgrößen stehen aktuell die Umfangsgeschwindigkeit der Trommel und der Volumenstrom des erhitzten Gasgemisches zur Verfügung. Eine Regelung des Brenners bietet sich derzeit noch nicht an, da dieser nur an- und ausgeschaltet werden kann. Dennoch wäre eine Regelung der Flammgröße wünschenswert, weshalb der Brenner hinsichtlich einer Umrüstung untersucht werden sollte.
Des Weiteren können derzeit über die Setupdatei lediglich grundlegende Daten wie Sensoradressen, Abtastrate und Zugangsdaten des Raspberry Pi eingelesen werden. Damit eine Änderung von einzelnen Parameter nicht aufwendig im Quellcode erfolgen muss, soll die Setupdatei und das zugehörige Skript derart erweitert werden, dass das Einlesen von allen verwendeten Parametern ermöglicht wird.
Die Hardware ist derzeit in anschaulicher Form auf einer Holzplatte montiert. Da diese zukünftig auch an anderen Röstern eingesetzt werden soll, muss diese deutlich kompakter gestaltet werden. In Idealfall soll die komplette Hardware in einem handlichen Koffer untergebracht werden.
Derzeit wird ein Teil der Software (alle Matlab-Skripte) noch auf einen PC ausgeführt. Auch um dem oben genannten Aspekt der kompakten Ausführung gerecht zu werden, ist eine Konvertierung des Matlab-Codes in einen C-Code notwendig. Dadurch kann der komplette Quellcode auf den Raspberry Pi ausgeführt werden womit der PC entfällt.

Im Folgenden soll nun eine Auflistung der zukünftig erforderlichen Schritte erfolgen:

Weiterentwicklung der Software

- PID-Regler entwerfen, welcher die über den Button „Profil laden“ importierten Röstprofile als Referenz nutzt und damit eine Regelung der Bohnentemperatur durchführt.
- Ermittlung der Reglerparameter.
- Reglerfunktionen mit dem Button „Röstung“ verknüpfen.
- Automatische Erkennung des „first cracks“ mit Hilfe des Gradienten der Bohnentemperatur.
- Setup-Skript erweitern, damit alle notwendigen Parameter über eine Setup-Datei eingelesen werden können. Folgende Parameter sollen damit zusätzlich noch eingelesen werden:
    - Trommelumfang zur Ermittlung der Umfangsgeschwindigkeit
    - Zähnezahl des zur Drehzahlberechnung verwendeten Zahnrades
    - Reglerparameter
    - Referenztemperatur, ab deren Unterschreiten durch das Skript „Aufheizen_Nachlauf“ ein Abschalten des Brenners erfolgt
    - Referenztemperatur, bei der im Skript „Aufheizen“ eine erneute Zündung des Brenners erfolgt (unter Temperatur des Zweipunktreglers)
    - Dateipfade für die Speicherung der Röstprofile
    - Die Takt-Geschwindigkeit der I2C-Schnittstelle anpassend erhöhen, um eine höhere Abtastrate zu erreichen
- Für den Raspberry Pi geeignete Stand-Alone-Versionen der Software erzeugen.
- Gehäusetemperatur ebenfalls in einem Textfeld des GUI darstellen.
- Temperaturgradient im Plot nur für kurzem Zeitabschnitt darstellen, da bei einer Darstellung über den gesamten Zeitbereich die Skala so geändert wird, dass der First Crack nicht mehr erkennbar ist.
- Textfeld „manuelles Rösten“ verschieben. Verdeckt aktuell die Beschriftung der Zeitachse im Plot
- Angabe des Datums bei Befehl „xlswrite“ bei Abspeicherung der Röstprofile in normgerechter Form gestalten.

Weiterentwicklung der Hardware

- Koffer entwerfen, der alle Hardwarekomponenten aufnehmen kann
- Platine kompakter gestalten → geätzte Platine anstatt der gefädelten Platine
- Brenner des Rösters hinsichtlich einer Umrüstung prüfen → die Energieabgabe soll durch Änderung der Brennstoffzufuhr stufenlos geregelt werden können
- Display für Betrieb ohne PC (nur per Raspberry Pi) integrieren

Abbildungsverzeichnis und Tabellenverzeichnis

Anhang

A: Programmdateien und Digitale Medien

B: Bauteileliste

C: Projektplan

D: Bearbeiterliste

Anhang A Programmdateien und Digitale Medien

Die diesem Anhang (A) aufgeführten Dateien liegen der Arbeit in digitaler Form bei.

Matlab-Software

- Grafische_Oberflaeche.m
- Grafische_Oberflaeche.fig
- Read_temp_werte
- read_drehzahl_werte.m
- read_druck_werte.m
- setup_from_file.m
- DropDownMenu.m
- Trommel_Steuerung.m
- Airflow_Steuerung.m
- Aufheizen.m
- Aufheizen_Nachlauf.m
- Werte_EinAusgabe.m
- Audioausgabe.m
- createTimer.m
- Brenner_Aus.m
- Setup.csv
- warning_tone.wav

Python Software

- Airflow_regler.py
- Start_Airflow.py
- Stop_Airflow.py
- Trommel_regler.py
- Start_Trommel.py
- Stop_Trommel.py
- DZS_interrupt.py

Schaltpläne

- Platinen-Schaltplan.sch (Eagle 8.4.3)
- Platinen-Layout.brd (Eagle 8.4.3)

Datenblätter

- Drehzahlsensor
- Transistor BC 337
- I2C-Repeater
- Pymodbus-Bibliothek
- Relaisbaustein Typ 48.61
- Frequenzumrichter Sinamics V20
- Thermoelement mit I2C-Ausgang
- D6F-PH_Drucksensor
- Printrelais
- Erweiterungs-Platine RB-RS485
- MW_Hutschienen-Netzteil

Anhang B: Bauteilliste

Bauteilbezeichnung	Anzahl
Diode 1N4007	10
Kohleschicht-Widerstand 2,2kΩ	10
Transistor BC337-25	5
Lötschraubenklemme, 2 polig RM 2,54	6
Lötschraubenklemme, 3 polig RM 2,54	1
Kohleschicht-Widerstand 1kΩ	1
Temperatur-Modul Thermoelement mit Ausgang I^2C	10
Stifteleiste 2x20 polig RM 2,54	1
Stifteleiste 2x13 polig RM 2,54	2
Raspberry Pi, Erweiterungsplatine RB-RS 485 für FU	1
I2C Repeater	1
Omron-Drucksensor	1
Buchsenleiste 2x13 polig RM 2,54	1
Buchsenleiste 2x3 polig RM 2,54	10
Printrelais 5V	5
Doppel-Europlatine	1
Zylinderschraube M2x25	2
Sechskantmutter M2	2
Stifteleiste 1x4 polig RM 2,54	1
Buchsenleiste 1x4 polig RM 2,54	1
Hutschienen-Netzteil Mean Well 5 V/DC 2A 10W	1
Hutschienen-Schaltnetzteil Mean Well 12VDC/1.67A	1
LAN-Kabel	1
Raspberry Pi 3 Modell B	1
Stromverteiler 2x5	2
Hutschiene	1
Notausschalter	1
Frequenzumrichter Sinamics	2
Drehzahlsensor GS1007	1
Zahnrad	1
Schlauch 4,9mm-Durchmesser (4m)	1

		Name	Dauer	Start	Ende	Notizen
1		Besprechung Start	1 tag?	18.10.16 08:00	18.10.16 17:00	Klugbauer/Elaoud
2		Besichtigung Werkstatt	1 tag?	26.10.16 08:00	26.10.16 17:00	Klugbauer/Elaoud
3		**Messtechnik**	**522 tage?**	**28.10.16 08:00**	**30.10.18 08:00**	
4		Hardware in Betrieb nehmen	24 tage?	28.10.16 08:00	30.11.16 17:00	Klugbauer/Elaoud
5		Softwarelösung finden	8,875 tage	01.11.16 09:00	11.11.16 17:00	Klugbauer/Elaoud
6		Messwertprogramm Temperatur erstellen	10,875 tage?	04.11.16 09:00	18.11.16 17:00	Klugbauer
7		Probemessungen Temperatur	37 tage?	06.12.16 08:00	25.01.17 17:00	Klugbauer/Elaoud
8		Drucksensoren bestellen	10,875 tage?	29.12.16 09:00	12.01.17 17:00	Klugbauer
9		Drehzahlsensor bestellen	5,875 tage?	23.02.17 09:00	02.03.17 17:00	Elaoud
10		Messwertprogramm Druck	59,875 tage?	17.02.17 09:00	11.05.17 17:00	Klugbauer
11		Messwertprogramm Drehzahl	59,875 tage?	17.02.17 09:00	11.05.17 17:00	Elaoud
12		Messwertprogramm Temperatur	59,875 tage?	17.02.17 09:00	11.05.17 17:00	Klugbauer
13		graphische Oberfläche	60,875 tage?	16.02.17 09:00	11.05.17 17:00	Klugbauer
14		Timer Funktion	60,875 tage?	16.02.17 09:00	11.05.17 17:00	Klugbauer
15		Pythonskript	5,875 tage?	24.02.17 09:00	03.03.17 17:00	Elaoud
16		Drehzahl	1 tag?	20.12.16 08:00	20.12.16 17:00	Elaoud
17		Ansteuerung Frequenzumrichter	1 tag?	20.12.16 08:00	20.12.16 17:00	Elaoud
18		Probemessung Druck/Drehzahl	56,875 tage?	01.03.17 09:00	18.05.17 17:00	Klugbauer/Elaoud
19		Pitorohr fertigen	58,875 tage?	20.02.17 09:00	11.05.17 17:00	Elaoud
20		Relais bestellen	49,875 tage?	08.03.17 09:00	16.05.17 17:00	Klugbauer
21		Schnittstelle für Frequenzumrichter einbinden	9 tage?	05.05.17 08:00	17.05.17 17:00	Elaoud
22		Modbus Protokoll	33 tage?	01.08.17 08:00	14.09.17 17:00	Elaoud
23		Layout der Platine erstellen	40,875 tage?	22.03.17 09:00	17.05.17 17:00	Elaoud
24		Platine bestellen	6 tage?	17.05.17 08:00	24.05.17 17:00	Elaoud
25		**Versuch**	**290 tage?**	**12.12.16 08:00**	**19.01.18 17:00**	
26		Messpunkte wählen	2 tage?	12.12.16 08:00	13.12.16 17:00	Klugbauer/Elaoud
27		Sensoren anbringen	4 tage?	14.12.16 08:00	19.12.16 17:00	Klugbauer/Elaoud
28		Tests durchführen	284 tage?	20.12.16 08:00	19.01.18 17:00	Klugbauer/Elaoud
29		Datenauswertung	5,875 tage?	02.01.17 08:00	09.01.17 16:00	Klugbauer
30		**Hardwarelösung**	**413,125 ta...**	**29.03.17 16:00**	**30.10.18 08:00**	
31		fehlende Elektronik kaufen	57,125 tage?	29.03.17 16:00	16.06.17 17:00	Klugbauer
32		Sensoren installieren	59,125 tage?	30.10.17 08:00	19.01.18 09:00	Elaoud/Klugbauer
33		Inbetriebnahme	60 tage?	30.10.17 08:00	19.01.18 17:00	Elaoud/Klugbauer
34		**Bericht schreiben**	**0 tage?**	**30.10.18 08:00**	**30.10.18 08:00**	

a - Seite1

Anhang D: Bearbeiterliste

Kapitelnr.	Kapitelbezeichung	Bearbeiter	
		Elaoud	Klugbauer
	Kurzfassung	x	
	Abstract		x
	Inhaltsverzeichnis	x	
	Abkürzungsverzeichnis	x	
1	Einleitung und Aufgabenstellung	x	
2	Hinführung und Vorüberlegungen	x	
2.1	Eingesetzter Raspberry PI und benötigtes Material	x	
2.2	Erststart des Raspberry Pi	x	
2.3	Bedienung des Raspberry Pi über Windows SSH-Anwendung	x	
2.4	Verbindung des Raspberry Pi mit Matlab		x
3	Software		x
3.1	Guided User Interface		x
3.2	Setupprogramm		x
3.3	Timerfunktion		x
3.4	Koordinationsprogramm zur Erfassung und Darstellung der Messwerte		x
3.5	Messwertprogramm Temperatur		x
3.6	Messwertporgramm Druck		x
3.7	Messwertprogramm Drehzahl	x	x
3.8	Drop-Down Menu		x
3.9	Aufheizfunktion		x
3.10	Schutzfunktion nach dem Aufheizvorgang		x
3.11	Airflow- und Trommel-Steuerung	x	x
3.12	Audioausgabe		x
3.13	Brenner ausschalten		x
3.14	Einbindung der Schnittstelle RB-RS485 für Frequenzumrichter	x	
3.15	Modbus RTU Protokoll zur Steuerung des Frequenzumrichters	x	

Kapitelnr.	Kapitelbezeichung	Bearbeiter	
		Elaoud	Klugbauer
4	Hardware		
4.1	Temperatur-Modul Thermoelement mit Ausgang	x	
4.2	Zahnrad-Drehzahlsensor GS1005 Serie	x	
4.3	Zahnrad zur Drehzahlaufnahme	x	
4.4	I2C-Repeater V2 für Raspberry Pi	x	
4.5	Drucksensor OMRON D6F-PH	x	
4.6	Pitot-Rohr zur Druckaufnahme	x	
4.7	Frequenzumrichter SINAMICS V20 von Siemens	x	
4.8	Schnittstelle RB-RS485	x	
4.9	Herstellung des Prototyps	x	
4.9.1	Herstellung der Platine	x	
4.9.2	Erstellung des Platinen Layout	x	
4.9.3	Montage aller Bauteile auf die Sperrholzplatte	x	
5	Inbetriebnahme		x
6	Betriebsanleitung		x
7	Ausblick		x
	Abbildungsverzeichnis und Tabellenverzeichnis	x	
	Bauteilliste	x	
	Projektplan	x	x