Automatisierung einer Modellanlage für Kristallzüchtung mit Induktionsheizung basierend auf Python und Raspberry Pi

Masterarbeit

Elektrotechnik (M)

Fachbereich 1 - Ingenieurwissenschaften – Energie und Information

vorgelegt von

Vincent Funke 570994

Datum:

Berlin, 03.06.2024

Erstgutachter: Prof. Dr.-Ing. Steffen Borchers-Tigasson
Zweitgutachter: Dr. Kaspars Dadzis

Inhaltsverzeichnis

1 Einleitung

Wenn das Wort Kristall fällt, wird zunächst meist an die Kristalle gedacht, die in der Natur vorkommen, wie Schnee und Minerale. Ein Kristall wird jedoch durch dessen spezielle Struktur gekennzeichnet, weshalb auch Kristalle aus Metallen entstehen können. Die Kristallographie ist die dazugehörige Wissenschaft, in der Kristalle auf Eigenschaft und Form untersucht werden. (Quelle: [Krib]) Im Leibniz-Institut für Kristallzüchtung (IKZ) werden genau solche Kristalle aus Metall, aber auch aus anderen Materialien angefertigt. So können Kristalle z.B. aus Zinn, Silizium, Fluor-Verbindungen wie Yttrium-Lithiumfluorid ($LiYF_4$) und Calciumfluorid (CaF_2) oder auch Germanium gezüchtet werden. Diese Kristalle finden besonders in der Elektrotechnik und Halbleitertechnik z.B. als Wafer Anwendung. Um bestimmte Kristalle herzustellen werden Anlagen für deren Züchtung benötigt. Um die Prozesse der Züchtung noch zu verbessern, werden diese Anlagen automatisiert. Somit können sogenannte Rezepte (Abläufe) zum Beispiel von einer Steuerung übernommen werden, sodass z.B. schon funktionale Rezepte vollkommen von der Anlage durchgeführt werden. In Zusammenarbeit mit dem Leibniz-Institut für Kristallzüchtung wird eine solche Steuerung entworfen. Dabei wird die Gruppe für Modellexperimente in der Abteilung für Materialwissenschaften unterstützt. In dieser Gruppe werden Modellversuche für die Kristallzüchtung erstellt und untersucht.

1.1 Kristalle und Kristallzüchtung

Zunächst sollen die Themen Kristall und Kristallzüchtung im Allgemeinen geklärt werden. Für diesen Zweck werden für die Begriffe Kristall und Kristallzüchtung zwei Definitionen herangezogen. Diese sind im Folgenden zu sehen:

„**Kristạll** [grch.] *der*, homogener fester Körper, dessen Bausteine (Atome, Ionen, Moleküle) sich während seines Wachstums, der →Kristallisation, aus Schmelzen, Lösungen, Dämpfen oder anderen festen Körpern räumlich-periodisch anlagern, und zwar nach ebenen Flächen, wenn das Wachstum nicht durch die Umgebung behindert wird. Der regelmäßige Aufbau unterscheidet K. von quasikristallinen oder amorphen Substanzen wie Gläsern, Flüssigkeiten oder Gasen. Alle Individuen einer K.-Art haben die gleiche Anordnung der Bausteine, das gleiche →Kristallgitter.“ Zitat: [wis10a] S. 137

„**Kristạllzüchtung,** die synthet. Herstellung von Einkristallen für wiss. und techn. Zwecke, z.B. von Siliciumkristallen für Transistoren und integrierte Schaltkreise, von Industriediamanten für die Materialbearbeitung, von Quarz für die Frequenzstabilisierung von elektr. Schwingkreisen oder von opt. Kristallen für die Lasertechnik und nichtlineare Optik.“ Zitat: [wis10a] S. 139

Aus den Definitionen ergibt sich, dass bei der Kristallzüchtung die Vorbereitung und Herstellung von sogenannten Einkristallen im Vordergrund steht. Unter einem Einkristall wird dabei ein einzelnes Kristallindividuum verstanden, dessen Bausteine (siehe Definition Kristall) ein Kristallgitter erzeugen, das durchgehend gleich strukturiert ist. Ein solcher Kristall ist in Abbildung 1 zu sehen und wird für die Wafer-Herstellung genutzt. Somit unterscheidet sich der Einkristall von den polykristallinen Körpern, die dementsprechend aus mehreren Einzelkristallen bestehen, die durch die sogenannten Korngrenzen getrennt sind. Somit kann sich die Kristallzüchtung z.B. von Methoden wie Massenkristallisation und Produktion von Metallen oder Keramiken abgrenzen. Die Kristallzüchtung ist im Sinne ihrer wissenschaftlichen Grundlagen der Kristallographie und der Physikalischen Chemie zuzuordnen. (Quellen: [Wil88] S. 13, [Kria], [Kric])

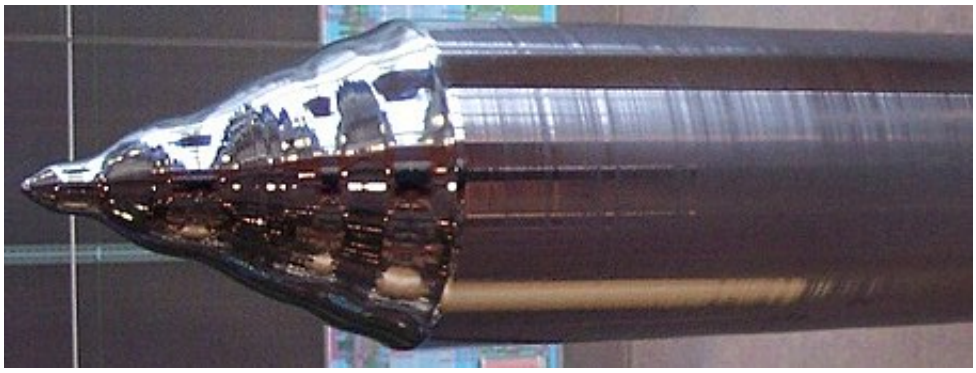


Abbildung 1: Einkristall (Silizium) (Quelle: [Kria])

Um die Kristallzüchtung noch weiter zu vertiefen, muss dazu gesagt werden, dass es verschiedene Verfahren gibt. Am IKZ werden die Verfahren Czochralski und Float-Zone angewendet. Die optimale Methode hängt dabei von verschiedensten Punkten ab. Neben den Eigenschaften des Materials/der Substanz im Sinne der Physik und Chemie müssen auch die Eigenschaften des Endproduktes wie Abmessungen, Reinheit und Homogenität bei der Wahl beachtet werden. (Quellen: [Wil88] S. 13)

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist die Erstellung einer neuen Steuerung, die von den Mitgliedern der Modellexperimente-Gruppe am IKZ genutzt werden kann. Dabei soll diese Steuerung, sowie ihre GUI, bestimmte Kriterien vorweisen.

Folgende Kriterien soll die Steuerung vorweisen:

1. Die Steuerung soll **modular aufgebaut** sein, in Bezug auf die Auswahl der steuerbaren Geräte.
2. Die Steuerung soll für jedes Experiment (**verschiedene Anlagen und Tisch-Experimente**) **anders konfiguriert** werden können.
3. Bei der Steuerung sollen Geräte über **dieselbe ober verschiedene Schnittstellen** ansprechbar sein.
4. Die GUI soll **überschaubar** und **benutzerfreundlich** sein, sodass sie möglichst kompakt und mit wenigen Handgriffen bedienbar ist.
5. Die Kristallzüchtung soll mithilfe von Rezepten von der Steuerung übernommen werden.
6. Die Steuerung soll mit **Raspberry Pi kompatibel** sein und auch auf **PCs mit Linux und Windows funktionieren**.

Zusammenfassend soll die Steuerung nicht nur speziell für einen Aufbau erstellt werden, sondern für mehrere unterschiedliche Anlagen, Tisch-Aufbauten und Experimente nutzbar sein. Für diesen Zweck muss die Steuerung konfigurierbar und modular sein. Mit modular ist gemeint, dass die Steuerung bzw. das Programm und die GUI beliebig mit Geräten erweiterbar sind. Weiterhin soll es möglich sein, dasselbe Gerät bzw. dieselbe Geräteart mehrfach einzufügen. Im Sinne der Geräte soll es auch möglich sein, die Schnittstellen ohne Fehlermeldungen mehrfach zu verwenden.

Für diesen Zweck müssen auch der Stand der Technik und die nutzbaren Geräte bekannt sein. Zu diesem wird jedoch erst in Kapitel 2.1 (Stand der Technik) und 3 (Geräte) etwas erzählt. Im Laufe dieser Arbeit wird dem Leser die Steuerung VIFCON vorgestellt, auf die speziell in den Kapiteln 4 und 5 eingegangen wird. Mit Experimenten (Kapitel 6) soll VIFCON dann auf seine Funktionalität und Einsetzbarkeit getestet werden.

Aus den Zielen wird deutlich, dass in dieser Arbeit eine Steuerung entworfen wird. Der Name dieser Steuerung ist VIFCON. Die Abkürzung VIFCON steht für **Visual Furnace Control**. Mit dieser Steuerung sollen Kristallzüchtungsanlagen angesteuert werden. Das englische Wort Furnace bedeutet im Deutschen Ofen und spielt auf die Züchtungskammer an.

2 Grundlagen und Theorie

In dem Kapitel sollen verschiedene Grundlagen für die Arbeit erläutert werden.So wird in Kapitel 2.2 eine SPS erläutert, da in den Anlagen am IKZ solche verwendet werden. Weiterhin wird auch eine Kommunikation mit der SPS und anderen Geräten benötigt, weshalb eine Erläuterung der Schnittstellen und Kommunikationsprotokolle (Kapitel 2.3) auch Teil dieses Kapitels ist. Zunächst wird jedoch der Stand der Technik (Kapitel 2.1) am IKZ erläutert.

2.1 Anlagentechnik

Der Inhalt dieses Kapitels beinhaltet den Stand der Technik am IKZ. Für diesen Zweck wurden sich die folgenden drei Anlagen angesehen. Diese sind:

1. die Nemo-1-Anlage (Czochralski-Verfahren (CZ)) (3 m hoch) der Modellexperimente-Gruppe,
2. eine Floating Zone (FZ) Anlage (5 m hoch) der Gruppe FZ-IV und,
3. eine CZ Anlage (3 m hoch) der Gruppe Fluoride.

Die drei genannten Anlagen können in Abbildung 2 gesehen werden. Bei der Nemo-1-Anlage handelt es sich um eine der Anlagen, die von VIFCON gesteuert werden soll. Aus dem Grund wird diese Anlage in Kapitel 3.4 näher beschrieben.

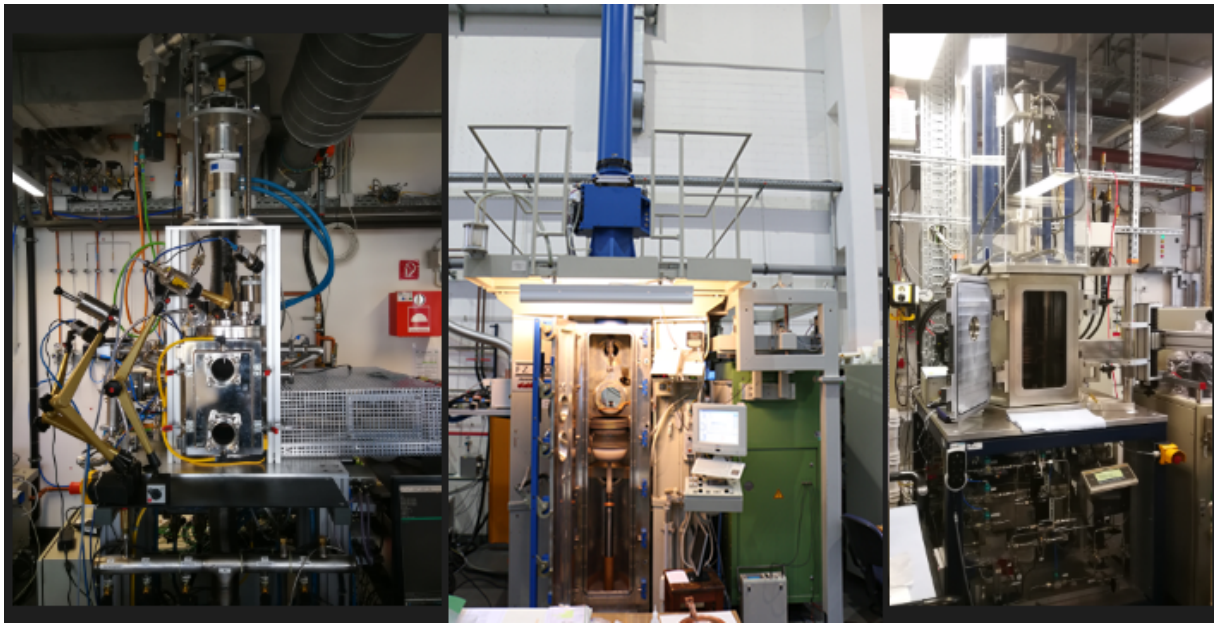


Abbildung 2: Anlagen für die Kristallzüchtung - l. Nemo-1-Anlage (Modellexperimente) (Quelle: eigene Darstellung) | m. FZ-Anlage (FZ-IV) (Quelle: [Perc]) | r. CZ Anlage (Fluorid-Gruppe) (Quelle: eigene Darstellung)

Neben dem Stand der Technik am IKZ sollen in Kapitel 2.1.1 auch der Züchtungsprozess und die Züchtungsverfahren erwähnt werden. Die Erläuterungen zu dem Stand der Technik am IKZ entstanden aus den Gesprächen und Anlagen-Vorführungen mit den verschiedenen Mitarbeitern am IKZ. Die Modellexperimente-Anlage wurde von Dr. Sepehr Foroushani [Perd], die FZ-Anlage von Max Scheffler [Perg] und die CZ-Anlage von Stefan Pueschel [Perh] vorgestellt. Diese Beschreibungen fließen dann in den folgenden Kapiteln mit ein und sind somit die Grundlage für diese.

2.1.1 Züchtungsprozess

Am IKZ gibt es zwei wesentliche Züchtungsverfahren. Diese sind das Floating-Zone-Verfahren (FZ) und das Czochralski-Verfahren (CZ). Einer der wesentlichen Unterschiede ist die Position des Impfkristalls und die Richtung der Züchtung.

Beim FZ-Verfahren wird durch eine einwindige Spule das Material geschmolzen bzw. eine Tropfenbildung erzielt. Im Folgenden wird der Impfkristall von unten in einen Vorratsstab (Schmelze) geimpft. Bei dem Verfahren wird somit die Induktion und das magnetische Feld verwendet. Dieses Verfahren kann auch als Pedestal-Verfahren durchgeführt werden. In dem Fall wird der Impfkristall von oben in den Vorratsstab geimpft.

Der Ablauf des CZ-Verfahrens sieht wie folgt aus. Zunächst wird das Material in einem Tiegel eingeschmolzen. Das Einschmelzen wird mit einer Temperatur knapp über der Schmelztemperatur und einer Induktionsheizung oder einem Widerstandsheizer durchgeführt. Wenn das Material so weit ist, wird der Keimkristall eingeführt und anschließend hochgezogen, wodurch das Material am Keim anwächst. Dabei wird auf Ziehgeschwindigkeit (typisch: 100 mm/h bis 0,1 mm/h) und eventuelle Rotation (typisch: wenige min^{-1} bis über 100 min^{-1}) des Keims geachtet. Beim Wachsen muss darauf geachtet werden, dass der Kristall nicht in die Schmelze wächst, sondern ein stabförmiger Kristall entsteht. Dies wird erreicht, wenn die Wachstumsfront in der Höhe ungefähr bei der Schmelzoberfläche liegt. Somit liefert das CZ-Verfahren eine gute Voraussetzung, um den Wachstumsvorgang zu beobachten und zu kontrollieren. (Quelle: [Wil88] S. 680)

Somit ergibt sich auch der andere wesentliche Unterschied der Verfahren. Dies ist die Methode zum Heizen und Schmelzen des Rohstoffs. Entweder wird ein Tiegel mit Material oder ein Vorratsstab verwendet. Beim Tiegel wird das gesamte Material geschmolzen, während beim Vorratsstab nur das Material flüssig ist, welches an der Spule ist. Der grundlegende Ablauf der beiden Verfahren ist jedoch identisch. Die Abbildung 3 zeigt diesen grundlegenden Ablauf eines Züchtungsprozesses von Kristallen am IKZ. Im Nachfolgenden werden diese Schritte kurz beschrieben.

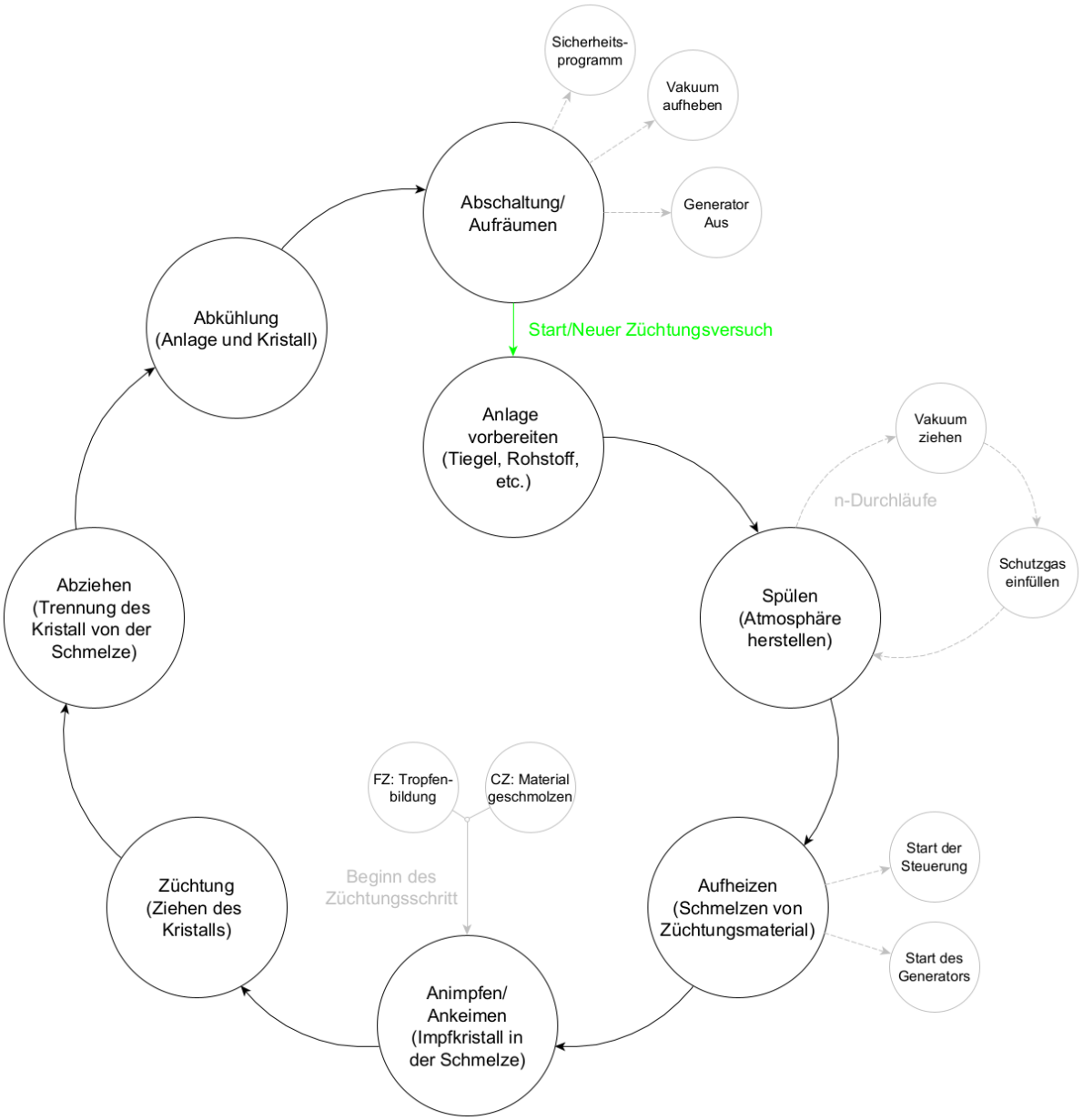


Abbildung 3: Ablauf einer Kristallzüchtung (Quelle: eigene Darstellung)

Der Start der Züchtung besteht immer aus der **Anlagenvorbereitung**. In diesem Schritt entscheidet der Züchter, was in der Züchtung verwendet wird. Während des Schrittes werden Punkte wie Rohstoff (Silizium, Zinn, etc.) und Tiegel (Aluminium, Grafit) bzw. Vorratsstäbe ausgewählt. Somit wird der Rezipient für die Züchtung vorbereitet. Auch die Vorbereitung des Impfkristalls und dessen Anbringung in der Halterung sind Teil der Vorbereitung.

Der zweite Punkt in der Züchtung ist das **Spülen**. Damit ist die Vorbereitung der Atmosphäre im Züchtungsofen gemeint. Dabei wird die Anlage mit z.B. Luft, Vakuum, Argon oder Stickstoff gefüllt. Argon und Stickstoff werden dabei als Schutzgas bezeichnet. In Abbildung 3 kann an dem Punkt ein weiterer Ablauf beobachtet werden. Bei der Fluorid-Gruppe wird mehrfach Vakuum gezogen und dann ein Schutzgas eingefüllt. Durch diesen Prozess soll der Sauerstoff aus den kleinsten Räumen entfernt werden. Fluoride sind anfällig für Sauerstoff, weswegen lieber das Schutzgas in den Ritzen vorhanden sein soll. Je nach Anlage und Material muss die Atmosphäre also anders behandelt werden. Die genannten Atmosphären werden am IKZ genutzt.

Nachdem die Anlage vorbereitet ist, wird der Schritt **Aufheizen** ausgeführt. Zum Aufheizen des Materials werden nun auch die Steuerung und der Generator gestartet. Über eine GUI können nun, auch schon beim Spülen, bestimmte Geräte angesteuert und die Werte beobachtet werden. Beim CZ-Verfahren wird hierbei das Material in dem Tiegel aufgeschmolzen. Meist werden Grafit-Tiegel, auch als Suszeptor genutzt, verwendet. Der Suszeptor spielt bei der Nutzung einer Induktionsheizung eine Rolle, da hier das Magnetische Feld und die Induktion genutzt werden. Bei dem FZ-Verfahren wird eine Tropfenbildung ausgelöst. Dabei wird auch wieder die Induktion verwendet, wodurch nur ein Teil des Vorratsstabs schmilzt, der Tropfen. Bei Silizium werden noch sogenannte Tantalschalen verwendet. Der Grund dafür ist, dass Silizium erst bei höheren Temperaturen leitfähig ist, wodurch die Induktion nicht funktioniert. Die Tantalschalen sind leitfähig und heizen somit den Vorratsstab so lange, bis das Material leitfähig wird. Bevor die Züchtung dann beginnt, werden die Schalen entfernt.

Sobald das Material aufgeschmolzen wurde bzw. sich der Tropfen gebildet hat, kann die Züchtung eines Kristalls beginnen. Nun beginnt der Schritt **Animpfen/Ankeimen**, bei dem der Impfkristall in das flüssige Material eingeführt wird. Hier müssen sich Schmelze und Keim verbinden. Je nach Gruppe müssen auch bestimmte Werte betrachtet werden. Bei der Fluorid-Gruppe wird eine Tiegelwaage zur Steuerung der Temperatur genutzt.

Nachdem der Kristall und die Flüssigkeit verbunden sind, kann der wesentlichste Schritt, die **Züchtung**, beginnen. Dieser Schritt kann dabei mehrere Tage dauern. Bei der Züchtung sind nun Werte wie Ziehgeschwindigkeit und Rotationsgeschwindigkeit relevant. Mit diesen kann der Durchmesser des Kristalls verändert werden. Diese Züchtung kann nun automatisch durch ein Rezept oder manuell durch den Bediener durchgeführt werden. Bei der Fluorid-Gruppe wird z.B. in einem Programm ein Rezept mit Intervalllängen abgearbeitet. Bei der Züchtung wird eine Reglung der Temperatur verwendet. Diese Regelung kann über das Kristallgewicht oder eine Temperaturmessung erfolgen. Die Heizer werden dabei über Generatoren gespeist. Diese können über Spannung, Strom oder Leistung betrieben werden.

Wenn der Kristall gezüchtet wurde, muss dieser von der Schmelze abgezogen werden. Nun befindet sich der Prozess im Zustand **Abziehen**. Bei allen drei Gruppen wird dies durch ein kurzzeitiges schnelleres Ziehen des Kristalls ermöglicht.

Nachdem der Kristall gezogen wurde, folgt die kontrollierte **Abkühlung** von Anlage und Kristall. Neben dem Abkühlen am Ende der Züchtung muss die Kühlung auch während der Züchtung beachtet werden. Zum Beispiel kann ein Nachheizer benutzt werden, um die Kühlung des Kristalls zu verlangsamen. Beim Abkühlen des Kristalls wird eine umgekehrte Aufheiztabelle verwendet.

Als letztes wird die Anlage abgeschaltet. In dem Prozess werden die **Abschaltung** und das **Aufräumen** betrachtet. Zunächst werden die Generatoren (Leistung runter) und Antriebe abgeschaltet. Danach wird das Vakuum (wenn vorhanden) im Züchtungsofen aufgehoben. Dies geschieht durch Einfüllen von Schutzgas. Sobald der normale Druck hergestellt ist, kann die Tür geöffnet und der Kristall entnommen werden. Auch die restlichen Ventile (z.B. Gas und Wasser) werden geschlossen. Des Weiteren kann auch ein Sicherheitsprogramm ablaufen, das prüft, ob der Heizer abgeschaltet und die Gaszufuhr geschlossen ist.

Bei jeder Anlage und Gruppe können Einzelheiten in den genannten Phasen der Züchtung unterschiedlich sein. So kann z.B. das Spülen bei jeder Gruppe anders durchgeführt werden. Es können andere Gase benutzt werden oder auch die Länge des Schrittes kann unterschiedlich sein. Somit zeigt Abbildung 3 die allgemeinen Schritte des Züchtungsablaufes eines Kristalls.

2.1.2 Hardware am IKZ

Aus der Besichtigung der Anlagen (Abbildung 2) ergibt sich, dass es 7 Gruppen von Geräten gibt, die für das Betreiben einer Kristallzüchtungsanlage wesentlich sind. Diese Gruppen sind:

1. **Antriebe** zur Bewegung und Ziehung des Kristalls,
2. **Generator** zum Betreiben der Heizer,
3. **Regler** um gegebenenfalls die Generatoren anzusteuern,
4. **Messgeräte** zur Messung verschiedenster Messgrößen (z.B. Temperatur, Gewicht),
5. **SPS** zur Steuerung der Geräte und,
6. **Pumpen** für die Kühlung und die Herstellung der Atmosphäre.

Wenn ein Programm genutzt wird, gibt es immer noch einen Rechner, auf dem das Programm zur Steuerung und Überwachung der Anlage läuft. Somit verfügen alle drei Anlagen auch über eine GUI, über die die Anlage gesteuert und ausgelesen wird. Diese GUI wird in Kapitel 2.1.3 näher erläutert. Neben all dem sollte auch der Heizer erwähnt werden. Der Heizer wird zwar über z.B. einen Generator betrieben, ist jedoch auch eine wichtige Komponente der Anlage, da dieser den Rohstoff zum schmelzen bringt und die Züchtung so ermöglicht. Der Heizer kann entweder induktiv oder resistiv sein.

2.1.3 Prozesssteuerung am IKZ

In den Gruppen am IKZ werden zur Steuerung der Anlagen SPS genutzt. Neben dieser gibt es immer noch ein Programm zur Steuerung, welches eine GUI besitzt. Auch die Nemo-1-Anlage besitzt eine Steuerung mit GUI in dem Schaltschrank der Anlage. VIFCON soll die Steuerung der Nemo-1-Anlage übernehmen, wodurch dieses mehr Funktionalität und Flexibilität anbietet, wie es in den Zielen (Kapitel 1.2) beschreiben wurde. Im Folgenden werden die Steuerungen und ihre GUI vorgestellt.

Den Anfang soll die Nemo-1-Anlage machen. In Kapitel 3.4 wird die Anlage näher beschrieben. Hier wird es nun um den Schaltschrank und dessen GUI gehen. Wie erwähnt steht im Hintergrund eine SPS, die mit den Geräten an der Anlage kommuniziert. Die GUI bzw. Steuerung hat in dem Fall vier Seiten, die in Abbildung 4 zu sehen sind. VIFCON wird dann mit den Seiten 1 und 2 aus der Abbildung arbeiten. Um dies zu erreichen wird der sogenannte Modbus verwendet, der in Kapitel 2.3.2 erläutert wird. Die spezifischen Punkte dazu sind in Kapitel 3.4 zu finden.

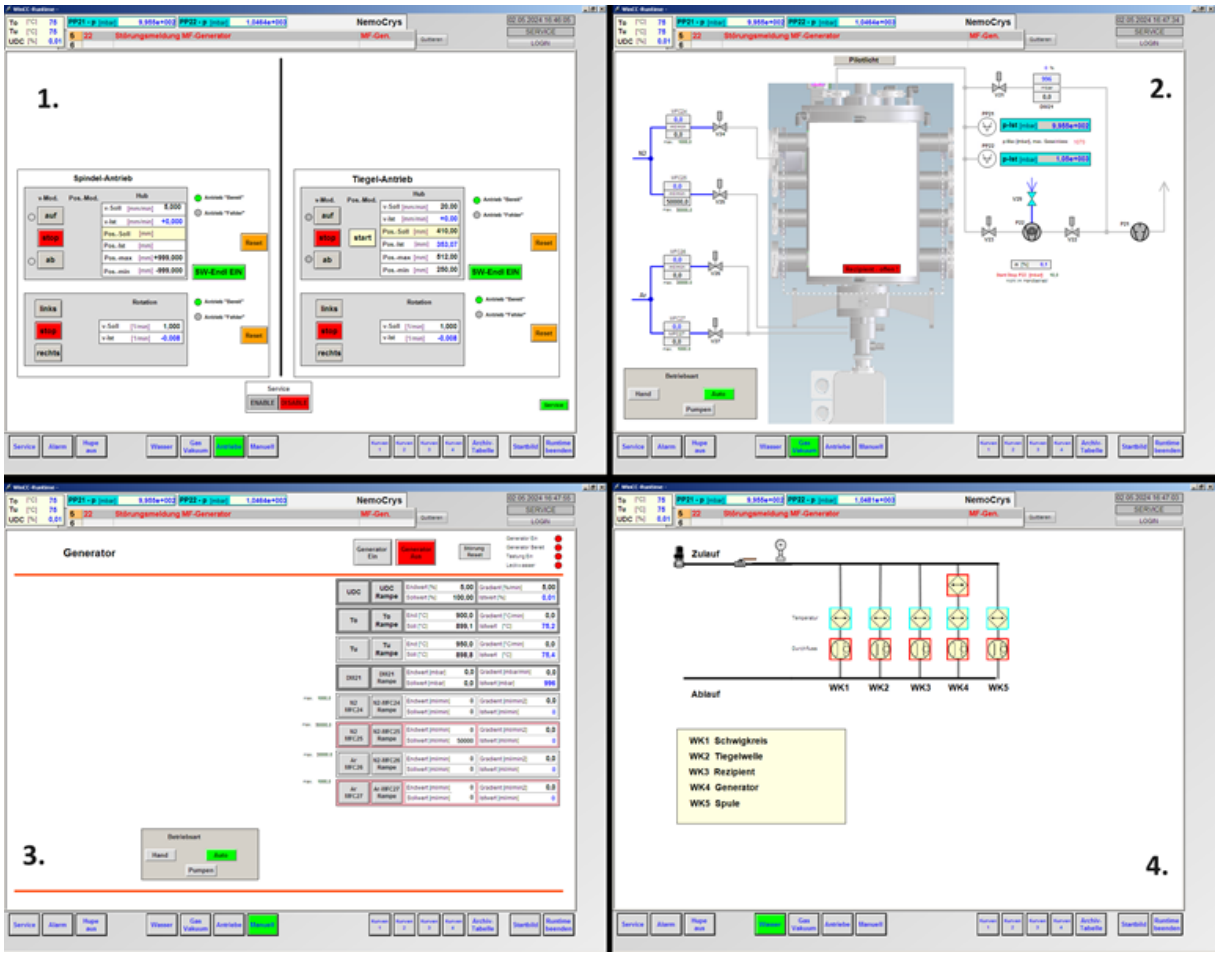


Abbildung 4: GUI des Nemo-1-Schaltschrankes - 1. Antriebe | 2. Gase, Druck | 3. Generator | 4. Wasser/Kühlung (Quelle: eigene Darstellung)

Nun soll eine kurze Erläuterung der in Abbildung 4 zu sehenden Teile erfolgen. Die Nemo-1-Anlage verfügt über 4 Antriebe, zwei für den Hub und zwei für die Rotation. Dabei werden sowohl der Tiegel als auch die Spindel (z.B. Keimhalterung) verwendet. Diese Antriebe sind auf Seite 1 in Abbildung 4 zu finden. Die Antriebe können dabei in ihren zugehörigen Richtungen bewegt werden. Die schwarzen Größen sind dabei über die Tastatur an der GUI einstellbar. Die Seite 2 zeigt die Ventile und Geräte für die Gase, also dem Druckaufbau und der Herstellung einer Atmosphäre. VIFCON wird die dort zu sehenden Werte in einem Monitoring-Tab anzeigen. Mit Seite 3 kann der angeschlossene Generator ein- und ausgeschaltet werden. Die letzte Seite zeigt den Zustand der Kühlung an. Für die Regelung der Anlage wird entweder ein Soll-Istwert-Vergleich der Temperatur mit einem PID-Regler oder die Regelung von Strom, Spannung oder Leistung verwendet. Bei beiden Varianten wird ein Eurotherm-Regler genutzt. Geregelt wird hier dann der angeschlossene Generator, der die Heizer versorgt.

Neben dieser Steuerung wird bei der Nemo-1-Anlage auch das Python-Programm Multilog [Mul] verwendet, welches eine reine Logging-Software ist. Das heißt dieses nimmt Messdaten auf und speichert diese. Für VIFCON hat dieses Programm eine große Bedeutung, was in den Kapiteln 4 und 5 näher erläutert wird. Die GUI von Multilog ist in Abbildung 5 zu sehen.

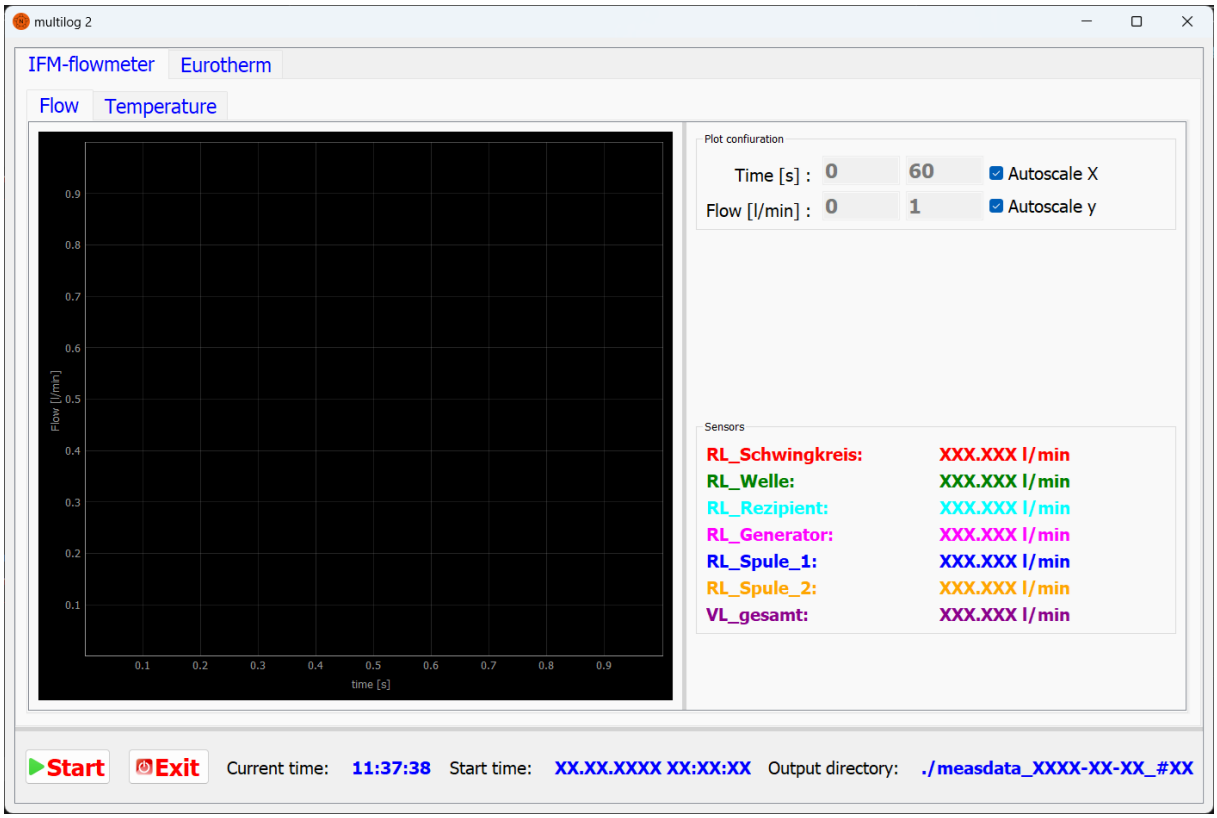


Abbildung 5: GUI von Multilog (Quelle: eigene Darstellung)

Die Anlage der Fluorid-Gruppe (Sektion Oxide & Fluoride) des IKZ besitzt auch eine Steuerung. Von der GUI dieser Steuerung werden in den Abbildungen 6 bis 8 drei Seiten dieser gezeigt. Das Programm an sich stammt nicht direkt vom IKZ und wurde von einer anderen Firma erstellt. Weiterhin verwendet es SQL-Server. Das besondere an der GUI ist, dass es sehr viele Tabs gibt. So wird jeder Prozessschritt der Kristallzüchtung (siehe z.B. Abbildung 3) durch einen Tab dargestellt. Auch beinhaltet die GUI ein Hauptbild (Abbildung 6) und eine Übersicht für die Medien (Abbildung 8). Neben den drei gezeigten gibt es noch viele weitere Seiten.

Die Abbildung 6 zeigt das Hauptbild der Steuerung. In diesem finden sich alle Informationen zur Anlage kompakt wieder. So kann der Nutzer hier auch Werte schreiben (blau) und Werte anzeigen lassen (grau). Die Steuerung kann manuell oder auch automatisch arbeiten. So kann der Nutzer eine Kristallzüchtung automatisch laufen lassen oder manche Schritte auch manuell selbst durchführen. In dem Hauptbild werden der Prozess und die Fehlermeldungen gezeigt.

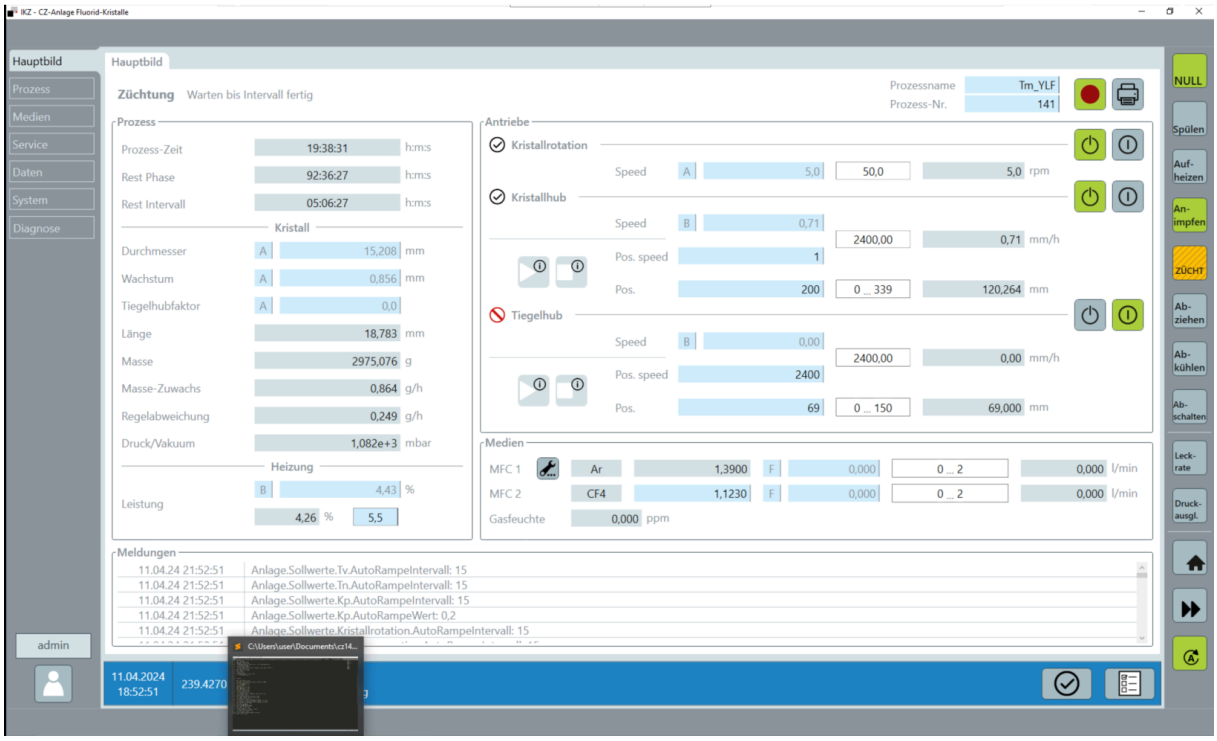


Abbildung 6: GUI der Anlage der Fluorid-Gruppe - Hauptbild (Quelle: [Perh])

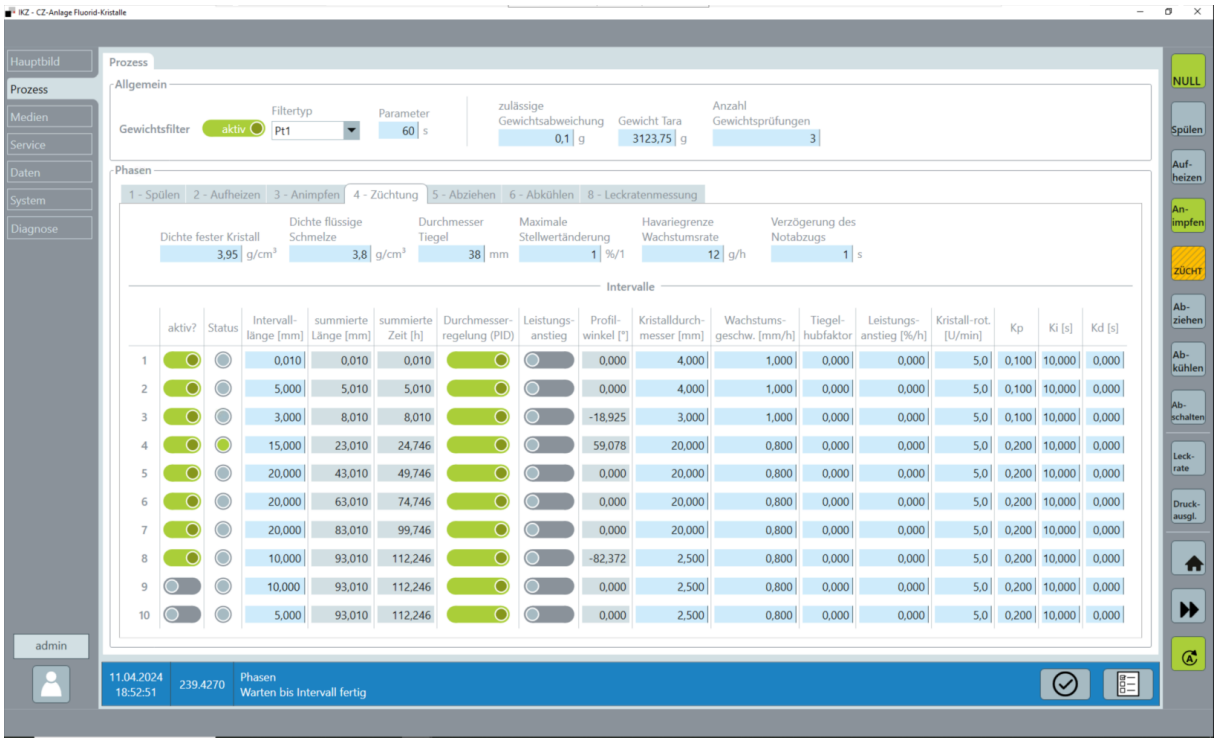


Abbildung 7: GUI der Anlage der Fluorid-Gruppe - Prozess Züchtung (Quelle: [Perh])

In der Abbildung 7 ist der Prozess-Schritt „Züchtung“ zu sehen. Hier kann die ganze Zucht des Kristalls in Teilschritten angegeben werden. Dies geschieht durch die Intervalllänge. Das bedeutet z.B. für den Schritt 2, dass hier eine Kristalllänge von 5 mm abgewartet wird, bevor in den Schritt 3 gesprungen wird. Neben diesem Umstand wird auch immer noch die aktuelle Gesamtlänge des Kristalls angezeigt. In der Abbildung können auch die weiteren anpassbaren Parameter gesehen werden. So kann ein PID-Regler konfiguriert und eingeschaltet werden und der Kristalldurchmesser, die Wachstumsgeschwindigkeit und weiteres eingestellt werden. Ein weiteres Feature ist die Angabe der Zeit für ein Intervall, welche sich aus Wachstumsgeschwindigkeit und Intervalllänge berechnet. Eine Besonderheit dieser Steuerung ist die Angabe des Gewichtes. Die Regelung dieser Anlage funktioniert mit einer sogenannten Tiegelwaage. Hier werden die Variablen Ziehgeschwindigkeit und Heizleistung geregelt.

Bei der Abbildung 8 wird der Plan für die Medien wie Kühlwasser und Gase gezeigt. Hierbei werden Ventile und andere Geräte gezeigt. Dies ist auch bei der Nemo-1-Anlage vorhanden.

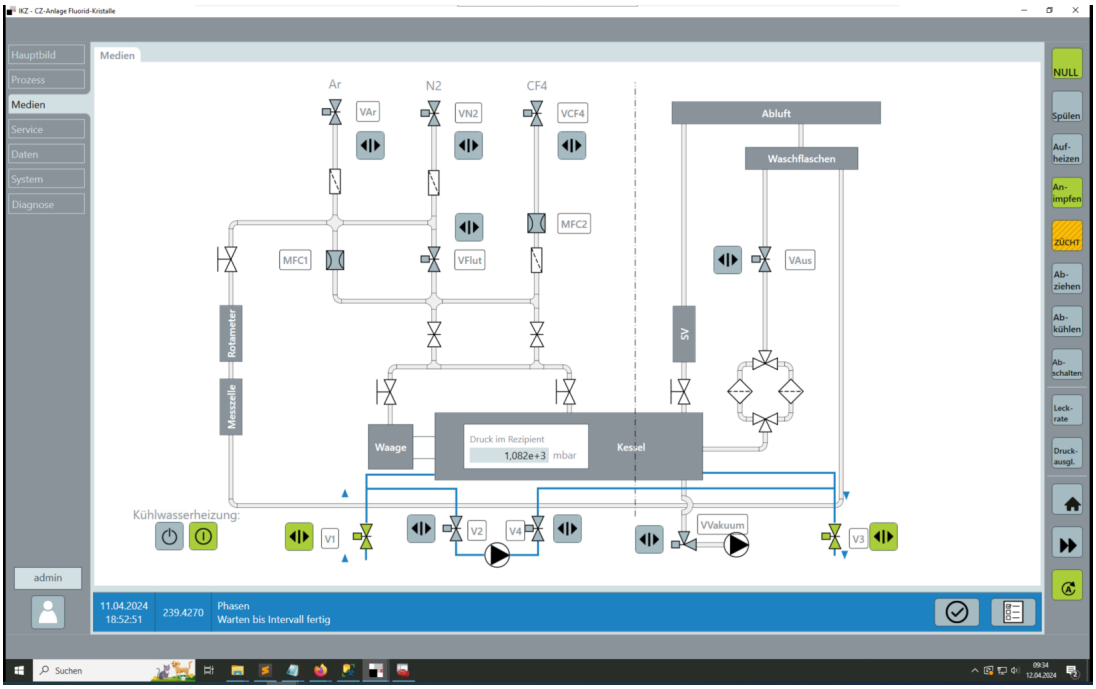
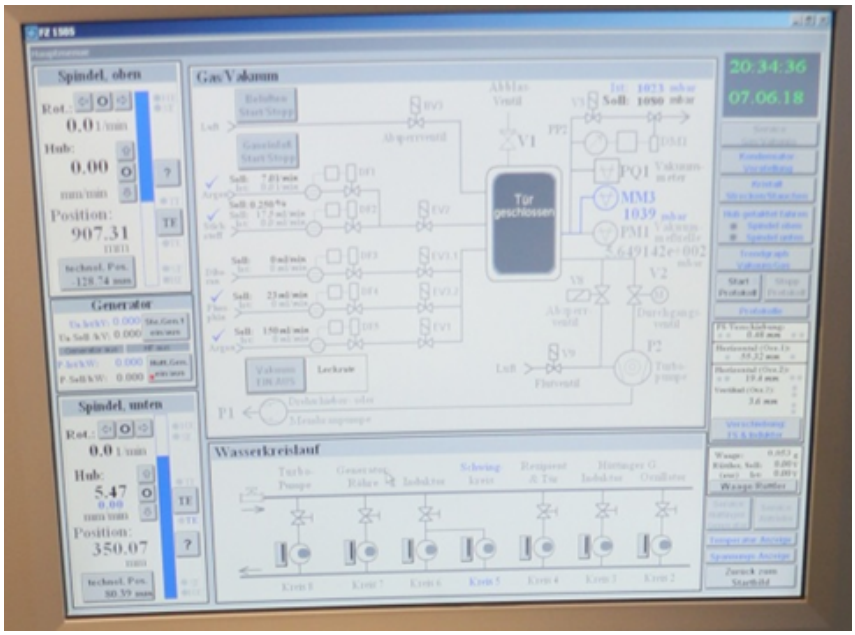


Abbildung 8: GUI der Anlage der Fluorid-Gruppe - Übersicht Medien (Quelle: [Perh])

Als letztes soll noch die Steuerung der Gruppe FZ-IV (Sektion Halbleiter) gezeigt werden. Dabei zeigt die Abbildung 9 die Steuerung der Anlage dieser Gruppe. Die Steuerung besteht diesmal aus einer GUI (Abbildung 9a) und einem Steuerpult (Abbildung 9b). Auch hier wird kein Programm verwendet, das im IKZ erstellt wurde. Auf der GUI sind das Gas- und Wassersystem (mit Einstellung des Gasflusses) und die Spannungswerte in kV (Generator) wiederzufinden. Für die beiden Spindeln gibt es eine Anzeige für Rotation und Hub sowie eine Positionsanzeige und einen Fortschrittsbalken. Die Bewegung der Antriebe kann dabei über die GUI oder das Bedienpult ausgelöst werden. Mit dem gezeigten Schaltpult können die Spindelantriebe für Keim und Vorratsstab wie gesagt rotiert und auf und ab bewegt werden. Neben diesem Antrieb kann der Induktionsheizer noch in 4 verschiedene Richtungen (nach oben, nach unten, hinein, heraus) bewegt werden. Die Antriebe können über einen einzigen Knopf auf dem Pult gleichzeitig angehalten werden, aber auch durch einen anderen Knopf für wenige Sekunden in dieselbe Richtung fahren. Auch der Generator kann über das Pult ein- und ausgeschaltet werden. Neben einem Not-Aus-Taster hat das Pult noch einen Touchscreen, über den die Leistung und Geschwindigkeit eingestellt werden können.



a) GUI

b) Pult

Abbildung 9: Steuerung der Gruppe FZ-IV - Pult und GUI (Quelle: [Perc])

Mit diesem Wissen zu den verschiedenen Steuerungen am IKZ soll nun die neue Steuerung VIFCON entworfen werden. Der Entwurf der Steuerung von VIFCON richtet sich dann nach den Wünschen der Modellexperimente-Gruppe und den genutzten Geräten. Weiterhin spielen noch Limitierungen durch das genutzte Programm und eben auch die Geräte eine Rolle beim Entwurf der Steuerung.

2.2 Speicherprogrammierbare Steuerung (SPS)

Mit der SPS oder im englischen PLC können Aufgaben zur Regelung oder Steuerung an Maschinen und Anlagen erfüllt werden. Dafür wird eine solche SPS digital programmiert. Die grundlegendsten Bauteile einer SPS sind die Eingänge, Ausgänge, das Betriebssystem und die Schnittstelle, über die das Programm auf die SPS hochgeladen wird. In dem Programm sind dann die Verknüpfungen zwischen den Ein- und Ausgängen beschrieben. (Quelle: [BT20] S. 2-3)

Für die SPS gibt es verschiedene Hersteller. Die bekanntesten sind Siemens, Wago und Schneider Electric. In der HTW werden SPSs aller drei Hersteller zu Lehrzwecken bereitgestellt. Neben dem Hersteller muss auch das Programm gewählt werden. In der HTW wurden die Programme CODESYS der CODESYS Group [Spsa] und e!cockpit von Wago [Spsc] zu Verfügung gestellt. In der Nemo-1-Anlage wird eine SPS von Siemens (Simatic S7-300) mit dem Programm WinCC der Version 8 verwendet.

Bei der Bauform einer SPS kann es Unterschiede geben. Die SPS kann z.B. modular oder kompakt sein. Der modulare Aufbau hat den Vorteil, dass sich weitere Module wie Eingangs- und Ausgangsbaugruppen anfügen lassen. Diese Baugruppen, der Programm-Speicher und die Zentraleinheit sind über den sogenannten Bus miteinander verbunden. Der Aufbau und die genannten Teile sind in Abbildung 10 zu sehen.

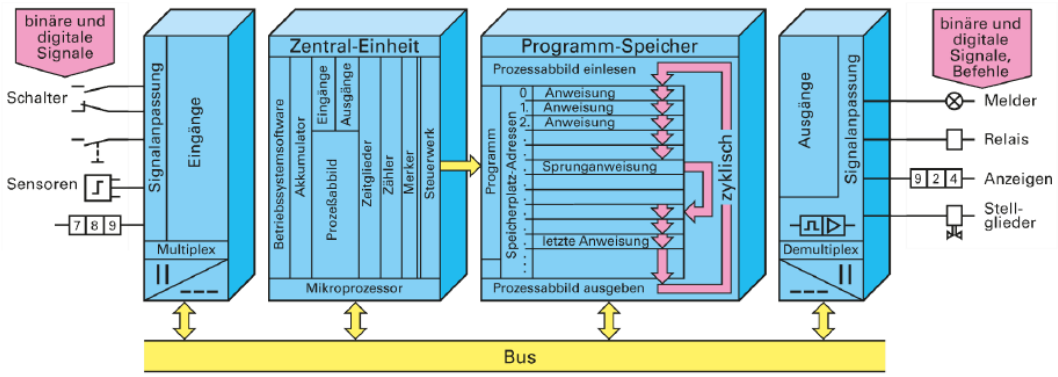


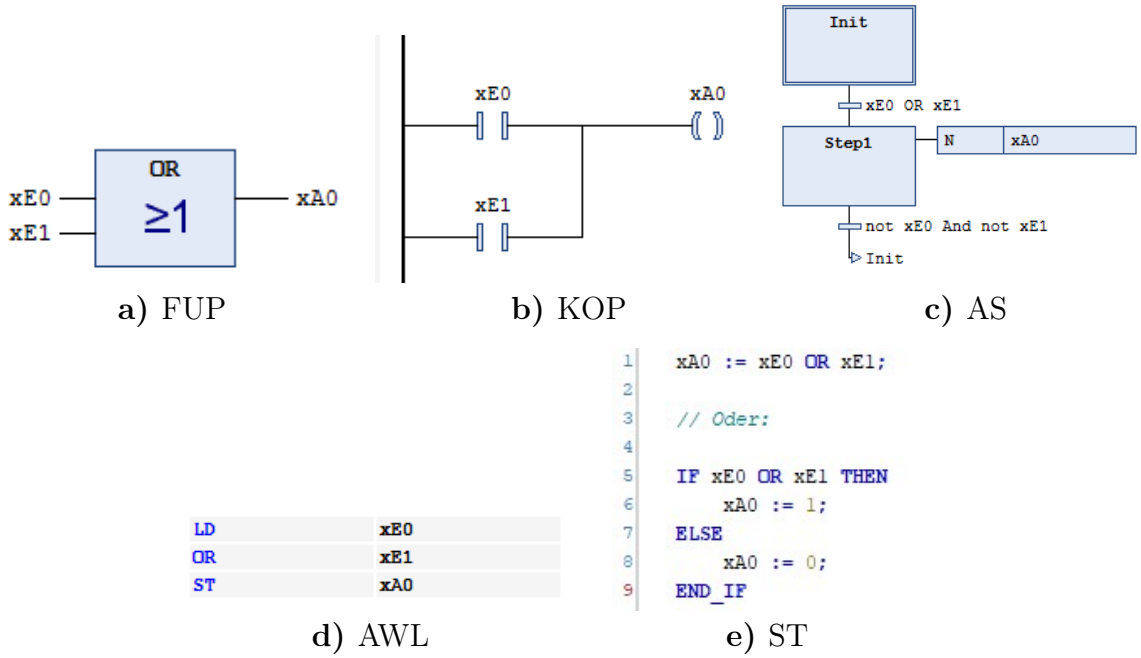
Abbildung 10: Aufbau einer SPS (Quelle: [BT20] S. 5)

Die Funktionsweise der SPS bezieht sich auf das sogenannte EVA Prinzip (Eingabe-Verarbeitung-Ausgabe). Wenn sich zu diesem Prinzip die Abbildung 10 angesehen wird, so kann dieses Prinzip wie folgt erläutert werden. Als erstes liest die SPS alle Eingänge (Abbildung links) aus und bestimmt den Zustand dieser. Darauf folgend wird das Programm mit Berücksichtigung dieser Eingänge abgearbeitet. Nachdem dies geschehen ist, werden dem Ausgang (Abbildung rechts) die jeweiligen Signale zugeordnet. Dieser Vorgang ist zyklisch, weshalb bei einer SPS auch von einer zyklischen Abarbeitung gesprochen wird. Wenn eine geringe Zykluszeit in der SPS eingestellt ist, kann die Echtzeitfähigkeit erreicht werden. (Quelle: [BT20] S. 6)

Neben der SPS selbst, haben auch die Programme einen ähnlichen Aufbau. Ein SPS-Programm besteht aus den sogenannten Netzwerken in dem dann die Teile des Codes eingefügt werden. Durch die verschiedenen Variablen entstehen dann Verbindungen zwischen den Netzwerken. Bei einer SPS wird das Programm in den besagten Programmen erstellt und dann auf die SPS übertragen. Neben dem Hauptprogramm (Codesys: PLC_PRG) können noch verschiedene Elemente erstellt werden. So können dem Programm z.B. eigene Funktionsblöcke, Funktionen, globale Variablenlisten und auch Visualisierungen hinzugefügt werden. Die ersten beiden genannten Beispiele werden als POU bezeichnet. (Quelle: [BT20] S. 7)

Bei den Programmen müssen nun drei Dinge beachtet werden. Diese sind die Programmsprache, der Variablentyp (z.B. BOOL, INT) und die Definition der Variablen (z.B. Input, Output). Die Sprachen und auch weitere grundlegende Dinge zur SPS werden in der Norm EN 61131 (Quelle: [Spsb]) beschrieben.

Insgesamt gibt es fünf Sprachen, die in Abbildung 11 zu sehen sind. Zur Darstellung wurde eine Oder-Verknüpfung verwendet. Bei den Sprachen wird in textbasierte (Anweisungsliste AWL - Abbildung 11d und Strukturierter Text ST - Abbildung 11e) und grafikbasierte (Kontaktplan KOP - Abbildung 11b, Funktionsplan FUP - Abbildung 11a und Ablaufsprache AS - Abbildung 11c) unterschieden. (Quelle: [BT20] S. 10)



a) FUP **b)** KOP **c)** AS

d) AWL **e)** ST

Abbildung 11: SPS-Sprachen (Programm: CODESYS V3.5 SP16 Patch 3) (Quelle: eigene Darstellung)

2.3 Schnittstellen und Kommunikationsprotokolle

Bevor die einzelnen Schnittstellen und deren Kommunikationsprotokolle, die im Zusammenhang mit dieser Arbeit stehen, erläutert werden, sollen drei Begriffe geklärt werden. Dafür werden drei Zitate gezeigt.

Begriffserklärungen:

„**Schnittstelle** (engl. interface), *Informatik:* Verbindungsstelle (Übergangsstelle) zw. zwei unterschiedlich arbeitenden Teilsystemen, über die der Austausch von Daten, Signalen u.a. erfolgen kann. Wichtige S. bei Computern sind **Hardware-S.** (z.B. Steckverbindungen), **Software-S.** zw. Programmen bzw. Programmteilen und **Mensch-Maschine-S.** (Benutzer-S., z.B. Tastatur, Maus, Menüs). - Nach der Art der Datenübertragung unterscheidet man zw. **seriellen** und **parallelen S.**, bei denen Bits nacheinander bzw. gleichzeitig übertragen werden.“ Zitat: [wis10c] S. 330

„**seriẹll,** [...] **2)** *Informatik:* zeitlich nacheinander geordnete bzw. erfolgende Darstellung, Verarbeitung und Übertragung von Informationselementen (im Ggs. zu gleichzeitig oder → parallel); in diesem Sinn wird s. häufig synonym zu **sequenziell** verwendet.“ Zitat: [wis10c] S. 507

„**parallẹl** [grch.], [...] **2)** *Informatik:* gleichzeitig und unabhängig voneinander ablaufend. Beispiele solcher Prozesse sind die parallele Datenübertragung (Parallelübertragung), bei der Zeichen (Bits, Bytes) p. voneinander über eine entsprechende Anzahl von Leitungen übertragen werden, sowie die parallele Datenverarbeitung (→Parallelverarbeitung).“ Zitat: [wis10b] S. 64-65

Ein Kommunikationsprotokoll ist im Bereich der Informatik ein Abkommen, welches bestimmt, wie die Datenübertragung vonstatten geht. Dabei kann die Kommunikation zwischen zwei oder mehreren Parteien ablaufen. Somit kann dieses Protokoll festlegen, wie z.B. die Syntax, die Semantik oder auch die Synchronisation bei der Kommunikation aussehen soll. Die Implementation dieser Protokolle erfolgt durch Hardware und/oder Software. Dabei wird auf der untersten Ebene das Verhalten der zu verbindenden Hardware definiert. (Quelle: [Scha])

In dieser Arbeit wird die Schnittstelle über Python (Bibliotheken) definiert und somit verwendet. Die speziellen Bibliotheken arbeiten dabei zum einem mit der RS232-Schnittstelle (Kapitel 2.3.1) und zum anderen direkt mit dem Kommunikationsprotokoll Modbus TCP (Kapitel 2.3.2). Die letztere übernimmt dabei die Kommunikation über die Ethernet-Schnittstelle.

2.3.1 Serielle Schnittstellen (RS232 (EIA-RS-232))

In der Allgemeinsprache wird in der Regel unter einer seriellen Schnittstelle die RS232-Schnittstelle mit einem 9-pin D-Sub-Stecker (Abbildung 12) gemeint. Weitgehend ist die RS232-Schnittstelle durch die USB-Schnittstelle ersetzt worden. Auch die USB-Schnittstelle gehört zu den seriellen Schnittstellen. Zwischen den beiden Schnittstellen Typen gibt es Adapter. So kann z.B. ein RS232-Anschluss auch an PCs ohne RS232 genutzt werden, da diese einen USB-Anschluss haben. (Quelle: [Schc])

Bei diesen seriellen Schnittstellen gibt es viele verschiedene Erscheinungsformen und Varianten. So existieren z.B. Punkt-zu-Punkt-Verbindungen, zu denen die RS232 und RS422 gehören, und Netzwerk- und Busschnittstellen (Ethernet, CAN-Bus). Somit unterscheiden sich diese Schnittstellen in verschiedenen Punkten. Diese sind:

- die Steckverbindung,
- die elektrischen Übertragungsparameter,
- die Methoden zur Übertragungssteuerung,
- die Datenflusskontrolle und,
- das Kommunikationsprotokoll. (Quelle: [Schc])

Im Weiteren soll nun nur noch auf die RS232-Schnittstelle eingegangen werden, da diese großteils in dieser Arbeit genutzt wird. Die Abbildung 12 zeigt die beiden Seiten eines Steckers. Die Innenseite (Abbildung 12b) zeigt die Stellen, an denen die Leitungen angelötet werden können. Durch solche Bauteile können die Verdrahtungen selbst durchgeführt werden. Die internen Verdrahtungen bei solchen Kabeln können von Hersteller zu Hersteller bzw. bei den Geräten anders sein. Solche Fälle werden in Kapitel 3 auftreten. Weiterhin kann an dem Bild die Nummerierung der Pins gesehen werden. Die Abbildung 12a zeigt eine der beiden Varianten des RS232-Steckers. In der Fachsprache wird hier bei von einem Female-Stecker gesprochen. Das Gegenstück wird als Male bezeichnet.

a) Stecker Außenseite

b) Stecker Innenseite

Abbildung 12: RS232-Stecker mit 9 Pins (Quelle: eigene Darstellung)

Bei diesen Steckern gibt es neben dem gezeigten 9poligen Sub-D-Stecker auch einen 25poligen Sub-D-Stecker. Die RS232-Schnittstelle wird im Deutschen auch als V-24-Schnittstelle bezeichnet. Für diese Schnittstellen wurden Normen wie die DIN 66020, 66021 und 66259 festgelegt. Der Unterschied der Stecker ist einfach erklärt. Der 9polige Stecker beinhaltet nur die wichtigsten bzw. für die Kommunikation bedeutendsten Signale. Die Tabelle 1 zeigt dabei die wesentlichen Pins und deren Funktion. (Quellen: [Thi96] S. 99-101)

Tabelle 1: Stecker-Pin Signal (Quelle: In Anlehnung an [Kai97] S.13 und [Thi96] S.102)

Pin (9 pol.)	**Pin (25 pol.)**	**Ein-/Ausgang**	**Bezeichnung**	**Funktion**
1	8	Ein	DCD	Empfangssignalpegel/Träger erkannt
2	3	Ein	RXD	Empfangsdaten
3	2	Aus	TXD	Sendedaten
4	20	Aus	DTR	Endgerät betriebsbereit/DEE bereit
5	7	GND	GND	Betriebserde/Signalmasse
6	6	Ein	DSR	Betriebsbereitschaft/Betriebsbereit
7	4	Aus	RTS	Sendeteil einschalten/Sendeanforderung
8	5	Ein	CTS	Sendebereitschaft
9	22	Ein	RI	Ankommender Ruf

Bei der Übertragung von Daten sind besonders die Signale TXD und RXD wichtig. Aus dem TXD-Ausgang kommen die zu sendenden Daten und bei RXD werden Daten empfangen. Mit der GND Leitung können diese drei Pins bereits eine Datenübertragung ermöglichen. In dem Kabel werden dabei die GND-Pins über Leitungen direkt verbunden. Die RXD und TXD werden dabei gekreuzt verbunden. Das heißt, dass das RXD am anderen Ende mit TXD verbunden wird und TXD dann mit RXD. (Quelle: [Thi96] S.103)

Neben diesen drei genannten Leitungen können noch sechs weitere definiert werden. Diese sechs Leitungen werden als Handshake-Leitungen bezeichnet. Der Name kommt daher, dass diese Leitungen für die Vorbereitung und Quittierung der Datenübertragung verwendet werden können. Dabei sind zwei dieser Leitungen als Ausgänge (DTR, RTS) und vier als Eingänge (CTS, DSR, DCD, RI) definiert. (Quelle: [Kai97] S. 13)

Die Datenkommunikation zwischen Computer und Gerät kann z.B. durch ein Oszilloskop betrachtet werden. In Abbildung 13 wird ein solches Beispiel gezeigt. Um die Datenübertragung zu gewährleisten, beinhaltet die Nachricht ein Startbit vor dem ersten Datenbit und auch Stoppbits. Das Startbit muss dabei an einem Pegelwechsel erkannt werden können. In dem Beispiel wird eine Baudrate von 1200 Baud verwendet. Das erste Bit in der Nachricht ist das niederwertigste, welches in der Abbildung mit „Bit 0“ bezeichnet ist. Das 8 Bit („Bit 7“) ist dementsprechend das höchstwertige Bit. Normalerweise werden die Spannungspegel verwendet, die durch die RS232-Norm definiert sind. Dabei entspricht die logische Eins einem Pegel von -12 V und die logische Null dann dem Pegel von 12 V. Beim Startbit wird eine logische Null und beim Stoppbit eine logische Eins gesendet. Somit bildet die logische Eins die Ruhelage des Pegels. (Quelle: [Kai97] S. 11-12)

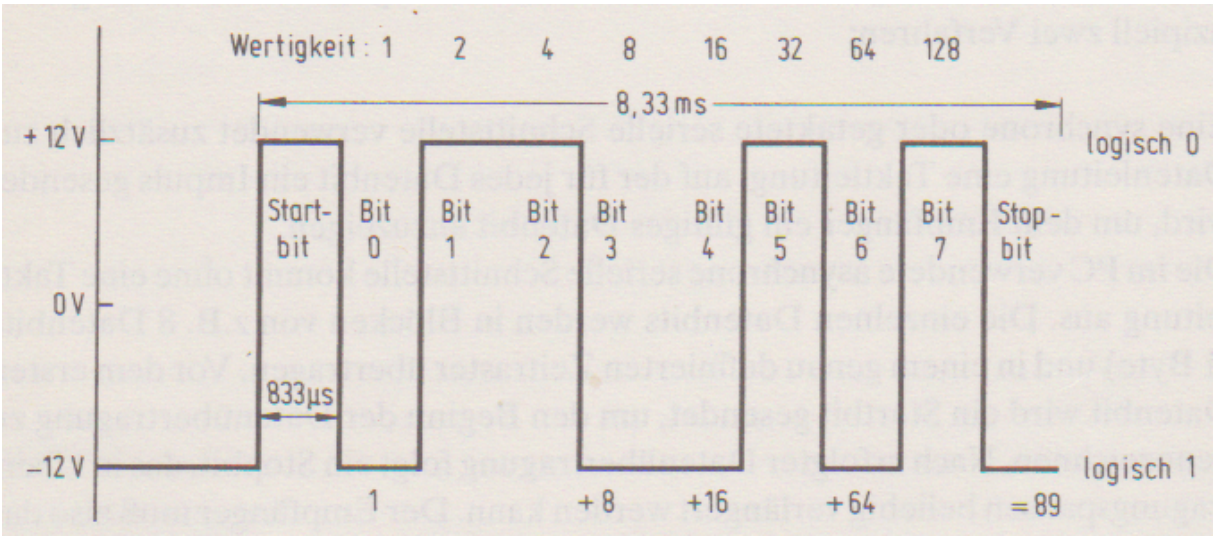


Abbildung 13: Oszillogramm des seriell gesendeten Zeichens Y (Quelle: [Kai97] S. 12)

Um eine Schnittstelle aufbauen bzw. definieren zu können müssen bestimmte Eigenschaften gesetzt werden. Dabei ist es wichtig, dass diese Eigenschaften auf beiden Seiten der Kommunikation identisch sind. In den Kapiteln 2.3.1.1 bis 2.3.1.3 werden diese Eigenschaften erläutert.

2.3.1.1 Ports

Um eine Schnittstelle ansprechen zu können, benötigt es einen sogenannten Port. Dieser Port kann von Betriebssystem und Schnittstelle abhängen. So werden diese bei Windows mit COM (z.B. COM12) angegeben. Bei Linux kann es wie folgt aussehen: /dev/ttyUSB2. Diese Art Port, die z.B. bei RS232 genutzt wird, ist ein Hardware-Port. Bei der Nutzung des Modbus-Protokolls wiederum wird eine Netzwerkadresse benötigt (Port und IP-Adresse).

2.3.1.2 Baudrate

Bei der Baudrate handelt es sich um eine Geschwindigkeit. Diese beschreibt, wie schnell Daten während einer Kommunikation übertragen werden. In dem Sinne werden die Zustände oder auch Pegel des Signals betrachtet. Die Baudrate zeigt somit auch die Häufigkeit der Zustandsänderung je Sekunde an. Ein Zustand kann dabei durch Frequenzen, Spannungspegel oder auch Frequenzphasenwinkel beschrieben werden. (Quelle: [Bauc] und [Baub])

Eine andere Größe gegenüber der Baudrate ist die sogenannte Bitrate. Hier geht es um die Übertragung von Bits je Sekunde (bps). Aus dem Grund ist die Baudrate mit dieser nicht gleichzusetzen. Die Baudrate wird in Baud (Bd) angegeben, was die Anzahl von Signalwechseln je Sekunde widerspiegelt. Beide Größen sind identisch, wenn z.B. wie bei der RS232-Schnittstelle nur mit zwei Spannungspegeln gearbeitet wird. Somit kann mit jedem der Pegel nur jeweils ein Bit übertragen werden, wodurch ein Baud einem bps entspricht. (Quelle: [Bauc] und [Baua])

Die Abbildung 14 zeigt dies. Bei zwei Pegeln, kann jeder Pegel nur durch einen Bit dargestellt werden, Null und Eins. Dies ist z.B. bei der RS232 so. Die Baudrate kann aber auch durch mehr Pegel dargestellt werden, was in dieser Arbeit nicht angewendet wird.

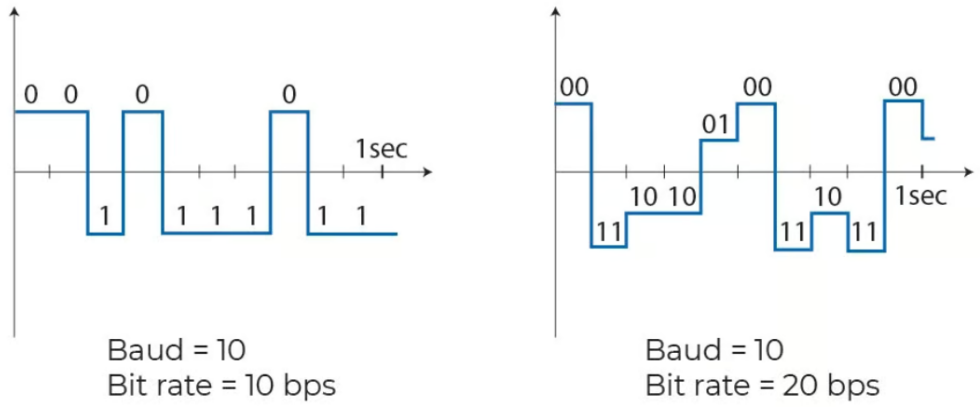


Abbildung 14: Darstellung von Baudrate und Bitrate (Quelle: [Bauc])

Folgende Baudraten haben sich als Standard durchgesetzt: 75, 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 und 115200. (Quelle: [Baua])

Wie erwähnt wurde bei dem Beispiel aus Abbildung 13 eine Baudrate von 1200 Baud verwendet. Umgewandelt bedeutet dies 1200 Bits/s. Somit hat jedes Bit (Start-, Daten- und Stoppbit) eine Zeitspanne von 833,3 µs. (Quelle: [Kai97] S. 12)

2.3.1.3 Arten von Bits

Für die Definition der Schnittstelle müssen im Sinne der Bits bzw. Bytes drei Größen angegeben werden. Diese sind die **Datenbytes**, das **Paritäts-Bit** und das **Stoppbit**. Neben diesen gibt es noch das Startbit, welches aber nicht definiert wird, sondern immer schon definiert ist. Dies wurde im Beispiel zu Abbildung 13 erläutert.

Als Datenbytelänge (Anzahl der Bits) können die Varianten 5, 6, 7 oder 8 Bits gewählt werden. Die ersten beiden Bytelängen gewährleisten eine Zusammenarbeit mit Fernschreibern. (Quelle: [Kai97] S. 12)

Die Parität beschreibt ein Bit in der Nachricht und ergänzt diese. Diese Parität kann gerade (E - Even), ungerade (O - Odd) oder nicht gesetzt (N - None) sein. Genutzt wird dieses Bit, um Übertragungsfehler zu erkennen, was auch als Paritätsprüfung bezeichnet wird. Mit der Methode ist keine Fehlerkorrektur möglich, da die fehlerhafte 1 oder 0 in der binären Nachricht nicht gefunden werden kann. (Quelle: [Schb])

Um dieses Paritätsbit zu erstellen werden alle 1 in der binären Nachricht gezählt. Wenn diese Anzahl z.B. bei gerader Parität ungerade ist, wird das Paritätsbit (Wert 1_2) angefügt, sonst eine Binäre Null. Bei der ungeraden Parität wird dies bei einer geraden Anzahl von 1 getan. (Quelle: [Schb])

Im Unterschied zu dem Startbit, kann es auch 2 Stoppbits geben. Wenn 8 Datenbits und ein Paritätsbit gewählt wurde, ist meist auch nur ein Stoppbit möglich. (Quelle: [Kai97] S. 12)

Die Stoppbits haben die Aufgabe einer Übertragungspause. Somit wird, bevor das nächste Startbit folgt, eine zeitliche Pause eingefügt. Quelle: [Thi96] S. 104)

2.3.2 Kommunikationsprotokoll Modbus

Beim Modbus handelt es sich nicht um eine Schnittstelle, sondern um ein Kommunikationsprotokoll. Hierbei wird eine Client-Server-Architektur verwendet. Diese Architektur wird auch als Master-Slave-Architektur bezeichnet. Zuerst aufgetreten ist Modbus 1979 in der SPS von Gould-Modicon. (Quelle: [Sch20] S. 84)

Bei Modbus existieren drei verschiedene Arten der Datenübertragung. Diese Arten sind:

1. Modbus TCP,
2. Modbus RTU und,
3. Modbus ASCII. (Quelle: [Sch20] S. 84)

In dieser Arbeit wird der Modbus-TCP verwendet, weshalb dieser im Folgenden näher erläutert wird. Die Umsetzung von Modbus und des SPS-Programms zur Steuerung der Nemo-1-Anlage (siehe Abbildung 2) wird durch die Firma AUTEAM Industrie-Elektronik GmbH durchgeführt. Die Verbindung zwischen Client (PC) und Server (SPS) wird durch das Ethernet ausgeführt.

In den meisten Fällen wird TCP mit dem IP verbunden. Somit bezeichnet der Begriff TCP/IP eine Gruppe von vielen verschiedenen Protokollen, zu denen eben auch TCP und IP gehören. (Quelle: [Fur03] S. 78)

Die Architektur von TCP/IP kann durch das sogenannte OSI-Referenzmodell dargestellt werden. Dieses Modell wurde von ISO im Jahre 1984 standardisiert. Dieses Modell besteht aus 7 Schichten, die in Abbildung 15 zu sehen sind. Neben diesem Umstand wird das OSI-Modell auch mit der TCP/IP-Architektur in der Abbildung verglichen. Obwohl es TCP/IP vor dem Referenzmodell gab, kann das Referenzmodell auf TCP/IP angewendet werden. Die Schichten 1 bis 3 werden jedoch bei TCP/IP als Anwendungsschicht zusammengefasst. (Quelle: [Fur03] S. 79)

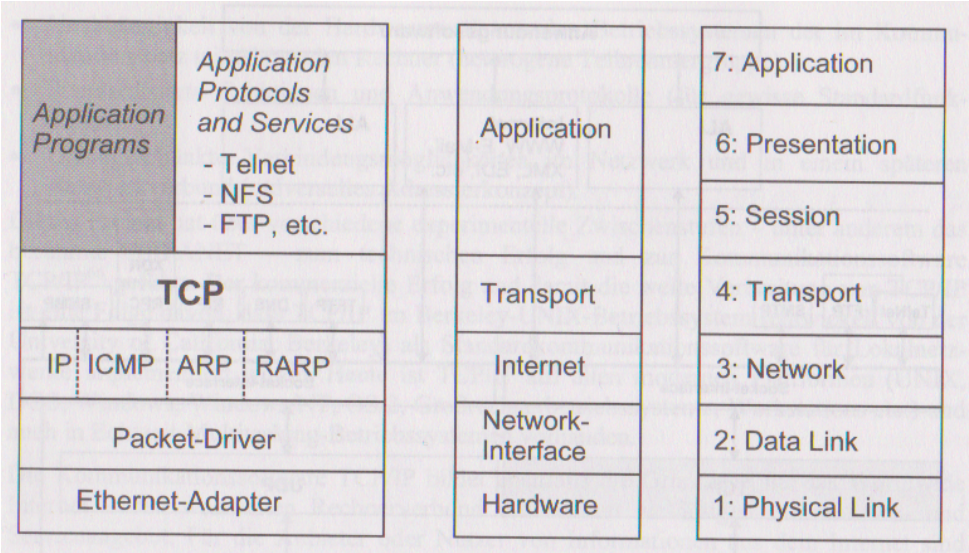


Abbildung 15: Aufbau TCP/IP (l, m) und OSI-Schichtenmodell (r) (Quelle: [Fur03] S. 80)

In Abbildung 15 kann gesehen werden, dass das TCP in der Transportschicht und das IP in der Network-Schicht liegt. Zwischen den Schichten aus dieser Abbildung werden sogenannte Telegramme hin und her gesendet. Alle Schichten haben ihren eigenen Header (Protokollbytes). Weiterhin benutzen die höheren Schichten immer nur das Datenfeld der untergeordneten Schicht. In dieses Datenfeld der untergeordneten Schicht werden dann die Telegramme (Frames) der übergeordneten Schicht eingefügt. (Quelle: [Fur03] S. 80)

In den folgenden Kapiteln soll die nötigen Punkte zur gesendeten Nachricht beschreiben werden.

2.3.2.1 Register

Um Modbus nutzen zu können, werden Objekte genutzt. Diese Objekte sind entweder 16 Bit Register oder 1 Bit Adressen. Über diese Objekte kann der Client dann auf den Server zugreifen. Somit sind Schreib- und Leseaufgaben möglich. Die Objekte können in Tabelle 2 gefunden werden. (Quelle: [Moda])

Tabelle 2: Modbus Objekte (Quelle: [Moda])

Objekttyp	Zugriff	Größe	Funktionscode
Einzelner Ein-/Ausgang „Coil“	Lesen & Schreiben	1-bit	01 / 05 / 15
Einzelner Eingang „Discrete Input“	nur Lesen	1-bit	02
(analoge) Eingänge „Input Register“	nur Lesen	16-bits	04
(analoge) Ein-/Ausgänge „Holding Register“	Lesen & Schreiben	16-bits	03 / 06 / 16

2.3.2.2 Aufbau der Nachrichten

In der Abbildung 16 wird der Aufbau der gesendeten und empfangenden Nachrichten gezeigt. Aus Tabelle 2 können die Funktionscodes abgelesen werden. Neben dem Funktionscode müssen nun auch die Registeradresse und die Anzahl der genutzten Register angegeben werden. Wenn z.B. 4 Datenwerte gesendet werden, werden diese Daten ab der gewollten Adresse in die Register geschrieben.

Als Beispiel soll angenommen werden, dass die Zahl 1,5 in ein Holding-Register geschrieben werden soll. Aus Tabelle 2 kann der Funktionscode für dieses Objekt abgelesen werden. In dem Fall ist es die 16_{10}. Beim Senden muss nun natürlich beachtet werde, in welchem Format gesendet wird. Hierbei wird im Beispiel nun das Hexadezimal-Format angenommen, wodurch die 16_{10} zu 10_{16} wird. Folgende Byte-Folge ist nun für das Beispiel entstanden:

```
1  10 00 04 00 02 04 3F C0 00 00
```

Alle Werte sind in Hexadezimal angegeben. Mit der Abbildung 16 können die einzelnen Komponenten entschlüsselt werden. So ist:

1. die 10 der Funktionscode 16 (Byte 0),
2. die 00 04 die Startregisteradresse (Byte 1 und 2),
3. die 00 02 die Anzahl der Register (N) (Byte 3 und 4),
4. die 04 der Byte-Count (Byte 5) und,
5. 3F C0 00 00 der Wert 1,5 (ab Byte 6 bis 2N+5).

In Kapitel 2.3.2.1 wurde erwähnt, dass jedes Register bei Holding 16 Bit, also 2 Byte groß ist. Jede Hexadezimalzahl beinhaltet 4 Bits, was bedeutet das ein Byte durch zwei Hexadezimalzeichen (z.B. 10_{16}) gegeben werden. In dem Beispiel werden 2 Register ab Register 4, also Register 4 und 5 beschrieben. Das bedeutet das 4 Bytes für die Füllung der Register gebraucht werden. Die Umwandlung der gezeigten 1,5 und das Beispiel selbst wurden durch die **pyModbusTCP**-Bibliothek von Python erstellt. Weitere Beispiele können daher in Anhang I für das Senden, Empfangen und Formatieren gefunden werden.

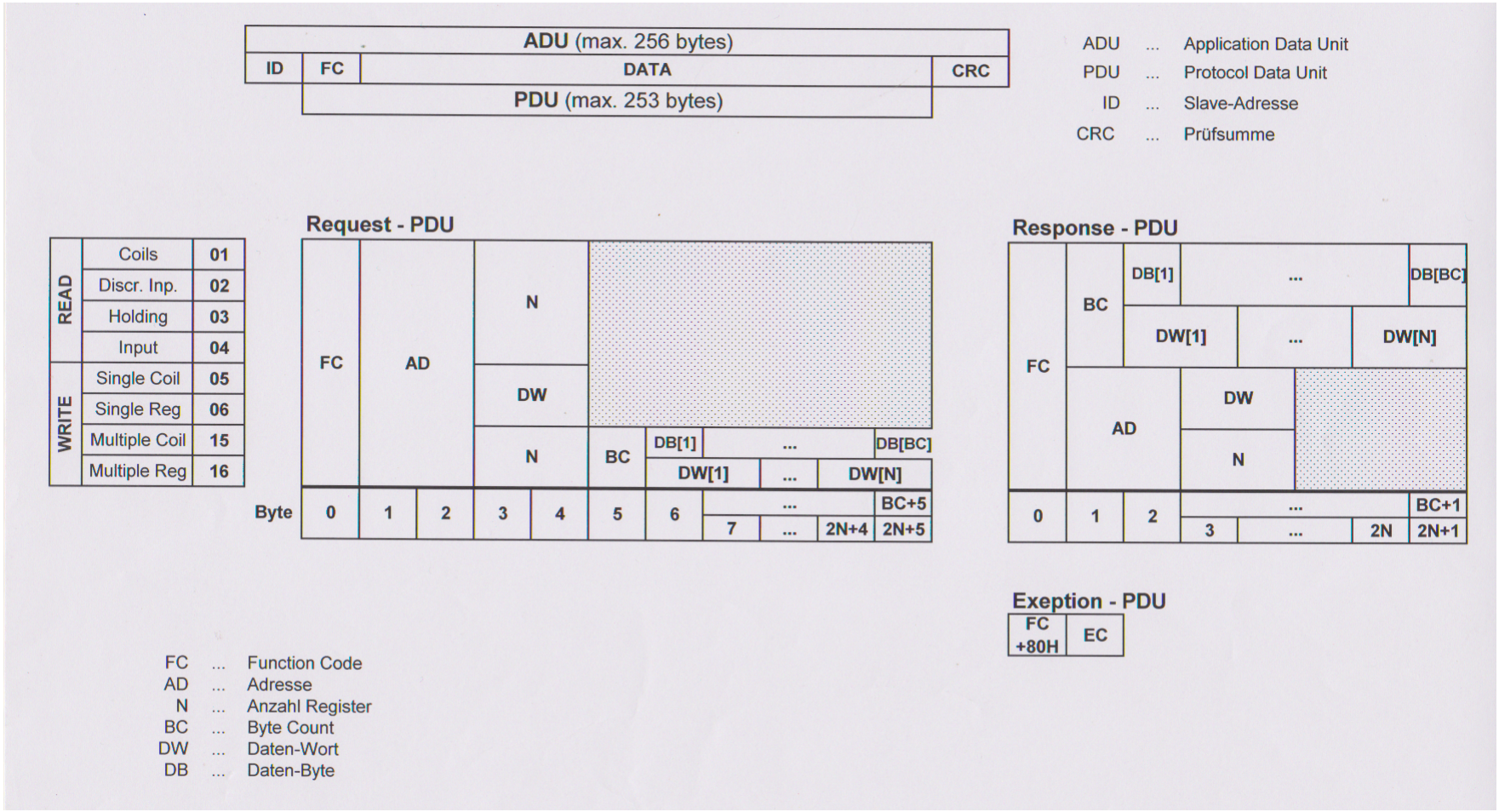


Abbildung 16: Modbus-Befehls-Aufbau (Quelle: [Perb])

3 Geräte für die Steuerung

In dieser Arbeit soll eine Steuerung namens VIFCON erstellt werden. Die Hintergründe dazu werden in Kapitel 4 und 5 dargestellt. Mit der Steuerung sollen nun verschiedene Anlagen bzw. Experimente angesteuert werden. Für diesen Zweck benötigt es die verschiedensten Geräte. Diese Geräte werden dann später über eine Config-Datei konfiguriert und somit über eine GUI vom Nutzer bedienbar sein. Diese Bedienung soll sowohl das Auslesen von Werten als auch das Auslösen von Tätigkeiten umfassen.

In VIFCON werden verschiedene Geräte verwendet, die in diesem Kapitel nun näher beschrieben werden. Zu VIFCON selbst wird Näheres in den Kapiteln 4 und 5 erläutert. Bei der Beschreibung der Geräte soll neben den Geräten selbst auch die Kommunikation mit diesen dargestellt werden. Somit umfassen die Erläuterungen im Folgenden die genutzten Befehle, den allgemeinen Aufbau besagter Befehle sowie die Funktionsweise der Kommunikation und auch Besonderheiten der Geräte und ihrer Schnittstellen.

Die VIFCON-Steuerung besitzt zwei Grundlegende Teile. Diese Teile dienen der Steuerung und dem Monitoring. Beim Monitoring werden Geräte für Kühlung und Gas ausgelesen, wie z.B. Pumpen. Bei der Steuerung werden Werte nicht nur gelesen, sondern auch durch den Benutzer geändert. Der Teil der Steuerung wird in der GUI in zwei Hälften geteilt. Somit umfasst die eine Seite Regler und Generatoren und die andere Seite die Antriebe.

Die Regler und Generatoren werden unter dem Geräte-Typ Generator in VIFCON zusammengefasst. Dies wird getan, da ein Regler indirekt als Generator genutzt werden kann. Zum Beispiel kann durch einen PID-Regler eine Ausgangsleistung durch den Ist-Sollwert-Vergleich einer Messgröße geregelt und somit ein Generator angesteuert werden. Somit können Generatoren direkt und indirekt angesteuert werden. Über die Regler erfolgt die indirekte Ansteuerung eines Generators.

Die Antriebe haben eine andere Funktion. Unter diesem Geräte-Typ können Achsen und Motoren verstanden werden, die entweder lineare oder Drehbewegungen ausführen. Mit diesen können bestimmte Positionen erreicht werden, aber auch die Züchtung eines Kristalls beeinflusst werden. In dem Sinne sind z.B. die Ziehgeschwindigkeit und die Drehbewegung bei der Züchtung gemeint.

In Kapitel 2.1 wurden die Anlagen am IKZ und die Züchtung von Kristallen erläutert. Daran ist zu sehen, warum Generatoren, Regler und Antriebe so wichtig sind für die Steuerung eines Kristallzüchtungsprozesses.

3.1 Eurotherm-Regler

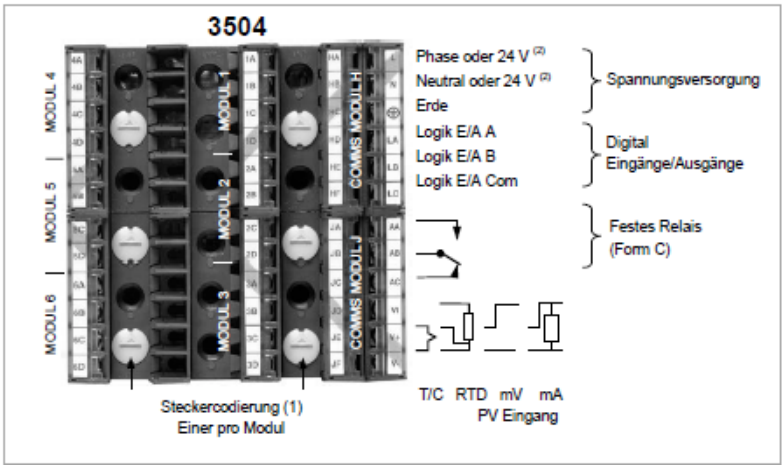
Das Eurotherm-Gerät gehört zu den Reglern (engl.: Controller) und wird in dem Tab Steuerung auf der Generator-Seite zu finden sein. Das Gerät wird meist als Eurotherm-Regler bezeichnet. In der Abbildung 17 ist der in dieser Arbeit genutzte Eurotherm-Regler zu sehen. Genutzt wurde das Modell 3504. Das Eurotherm-Gerät (bzw. ein älteres Modell) wurde bereits in der Bachelorarbeit [Fun22] verwendet und beschrieben. In dieser Bachelorarbeit finden sich zum Eurotherm Beschreibungen auf den Seiten XV und 14 bis 19.

Abbildung 17: Eurotherm-Regler vom Modell 3504 (Quelle: eigene Darstellung)

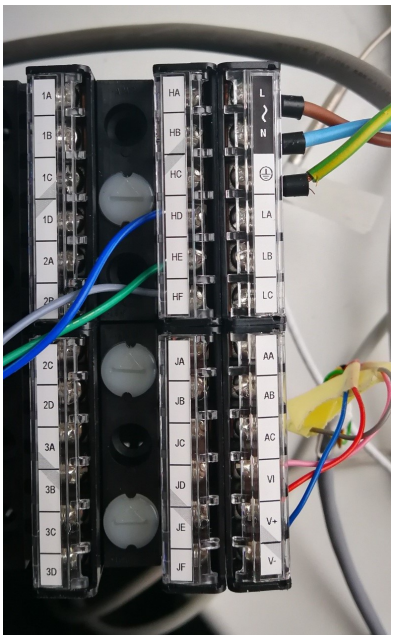
3.1.1 Funktionsweise und Aufbau

Der Eurotherm-Regler regelt einen Leistungsausgang. Diese Ausgangsleistung (in %) wird über einen PID-Regler und den Ist-Sollwert-Vergleich einer Messgröße bestimmt. Die Modellexperimente-Gruppe nutzt die Temperatur als Messgröße in Zusammenhang mit dem Eurotherm, weshalb Temperaturmessgeräte wie z.B. Thermoelement und Widerstandsthermometer angeschlossen werden können. Neben der Bestimmung der Ausgangsleistung mit PID-Regler (automatischer Modus), kann die Ausgangsleistung auch direkt über den manuellen Modus eingestellt werden.

In Abbildung 18 kann die Rückseite des Eurotherm-Reglers gesehen werden. Die Abbildung zeigt die Anschlüsse für die Module, die Schnittstelle und auch für die Spannungsversorgung. Wie das Eurotherm verdrahtet wurde bzw. wie die Verbindung zum Computer ist, kann in Abbildung 18b gesehen werden. Die Versorgung hängt an den Pins Phase, Neutral und Erde. Die Slots VI, V+ und V- sind für das Messgerät vorgesehen. Auf der Seite 19 in [Doki] können die Verdrahtungen an den Slots für verschiedene Messgeräte gefunden werden. In Abbildung 18b ist ein Widerstandsthermometer angeschlossen.



a) Doku (Quelle: [Doki] S.18)



b) Genutzt (Quelle: eigene Darstellung)

Abbildung 18: Eurotherm-Regler - Anschlüsse vom Modell 3504

3.1.2 Spezifikation für Hardware

Beim Aufbau des Eurotherm muss zunächst Folgendes beachtet werden. Ein Eurotherm-Regler besteht aus Steckmodulen, die bei der Bestellung festgelegt werden können. Die Art der Module kann durch den Bestellcode auf dem Gehäuse entnommen werden. Bei dem Modell 3504 kann es bis zu sechs dieser Steckmodule geben. Auch die Funktion des Reglers wird durch den Bestellcode erläutert. (Quelle: [Doki] S. 12)

Die Bedeutung des Bestellcodes ist in [Doki] auf Seite 13 bis 15 nachzulesen. Für den genutzten Eurotherm-Regler ist die Bestellcodierung folgende:

3504 /CC /VH /1 /XX /10 /4 /XXX /G /D4 /XX /XX /XX /XX /XX /AE /XX /XX /ENG /ENG /XXXXX /XXXXX /XXXXX /XXXXXX /STD ////////////////////////

Somit hat dieser Eurotherm-Regler z.B. eine Versorgung von 100-230 V_{AC} (/VH), nur ein Eingangs- und Ausgangsmodul und zwar ein DC Stetigausgang (/D4) und als H Comms Slot das 232 EI-Bisynch (/AE). Alle anderen Bezeichnungen können auf den genannten Seiten nachgelesen werden.

In der Dokumentation [Doki] des Eurotherms können auf den Seiten 22 bis 26 die verschiedenen Einsteckmodule für Ein- und Ausgang gefunden werden. Auf der Seite 23 findet sich dann auch der oben genannte DC Stetigausgang. Nach der Darstellung auf der Seite 23 in [Doki] beschreibt dieses Modul ein Stellglied und umfasst folgende Wertebereiche: Strom 0 bis 20 mA, Spannung 0 bis 10 V_{DC}.

Neben dem DC Stetigausgang verfügt Eurotherm auch über Standardanschlüsse. In der Serie des genutzten Eurotherm existieren:

1. ein PV Eingang (Messeingang),
2. ein Digital E/A (Logieingänge und Schließkontakteingänge),
3. Digital (Logik) Ausgänge (zur Versorgung eines externen 2-Leiter, 3-Leiter oder 4-Leiter Transmitters),
4. ein Relaisausgang und,
5. Versorgungsanschlüsse. (Quelle: [Doki] S. 19-21)

Bei dem Messeingang können so:

1. Thermoelemente,
2. Pyrometer,
3. Widerstandsthermometer (RTD) und,
4. ein sogenannter Lineareingang für Strom oder Spannung angeschlossen werden. (Quelle: [Doki] S. 19)

In Abbildung 18a können diese Anschlüsse gesehen werden. Die Abbildung 18b zeigt die genutzten Module. In der Abbildung werden der PV-Anschluss mit RTD, die Spannungsversorgung und die Schnittstelle genutzt. Wie bereits erwähnt konnte die Schnittstelle auch aus dem Bestellcode abgelesen werden. Somit benötigt die RS232-Schnittstelle die Slots HD, HE und HF, da der Bestellcode bei der Kategorie „H Comms Slot“ ([Doki] S.13) belegt worden ist. Die „J Comms Slot“ wurden bei dem Gerät nicht belegt, was durch das /XX nach dem /D4 gezeigt wird ([Doki] S.13). Die genaue Verdrahtung ist auf Seite 27 in [Doki] zu finden. An HD kommt die Com-Leitung, an HE das Tx und an HF das Rx (Erläuterung in Kapitel 2.3.1). Die Verdrahtung ist in Abbildung 18 zu sehen.

Neben dem genutzten EI-Bisynch der RS232-Schnittstelle könnten beim Eurotherm noch die Schnittstellen bzw. Kommunikationsprotokolle (wenn im Bestellcode ausgewählt) genutzt werden. Diese wären:

1. Modbus (232, 485 (2 und 4-Leiter)),
2. Modbus Master (232, 485 (2 und 4-Leiter)),
3. EI-Bisynch (232, 485 (2 und 4-Leiter)),
4. Ethernet Modbus (TCP IP),
5. Profibus und,
6. DeviceNet. (Quelle: [Doki] S.13)

3.1.3 Spezifikation für Schnittstellen

Die Spezifikationen für das Ansprechen des Eurotherms über die RS232-Schnittstelle sind festgelegt mit:

1. einer Baudrate von 9600 Baud,
2. einer Bytegröße von 7,
3. einer Anzahl der Stoppbits von 1 und,
4. einer geraden Parität (E - Even).

3.1.4 Kommunikationsprotokoll

In der Arbeit wird der Eurotherm-Regler über die serielle Schnittstelle RS232 angesprochen. Verwendet wird das Kommunikationsprotokoll EI-Bisynch.

Da in dieser Arbeit das EI-Bisynch Kommunikationsprotokoll benutzt wird, soll dieses auch kurz erläutert werden. Das Protokoll wurde von Eurotherm selbst erstellt. Es basiert auf dem ANSI X3.28-2.5 A4 Standards für Message Framing. Das Protokoll ist basierend auf ASCII und ist asynchron. Die Befehle werden in dem Fall „Mnemonic“ genannt. In Kapitel 3.1.6 können diese Mnemonic-Befehle gesehen werden. Meist bestehen diese aus zwei Zeichen. (Quelle: [Doki] S.133)

Im folgenden Kapitel kann der Aufbau der Nachrichten gesehen werden. Bei der Kommunikation mit Eurotherm ist zu beachten, dass diese eine Checksummenprüfung erfordert und Nachrichten durch ACK und NAK quittiert werden.

3.1.5 Aufbau der Nachrichten

Der Befehl zum Schreiben und zum Lesen unterscheidet sich vom Aufbau, besteht aber aus ähnlichen Zeichen. Gesendet wird dem Gerät ein String aus ASCII- und Steuerzeichen (Tabelle 3).

Tabelle 3: Steuerzeichen (Quelle: [Dokd] S. 11)

Hex value	Name	Usage
02	STX	Start of data in a message
03	ETX	End of message
04	EOT	End of transmission sequence
05	ENQ	Enquiry for a value
06	ACK	Positive Acknowledge
15	NAK	Negative Acknowledge
01	SOH	Start of a block transmission
17	ETB	End of block transmission
1C	FS	File separator in composite messages
1D	GS	Group separator in composite messages
1E	RS	Record separator in composite messages
1F	US	Unit separator in composite messages

Der Schreib-Befehl hat folgenden Aufbau nach [Dokd] S. 18 und besteht somit aus:

1. dem Steuerzeichen EOT (\x04),
2. zweimal der Gruppennummer,
3. zweimal der Unit Nummer,
4. dem Steuerzeichen STX (\x02),
5. dem Mnemonic-Befehl,
6. den Daten,
7. dem Steuerzeichen ETX (\x03) und,
8. der BCC (Prüfsumme).

Die Steuerzeichen-Bedeutung kann in Tabelle 3 gesehen werden. Bei der Prüfsumme handelt es sich um eine XOR-Verknüpfung von allen Zeichen des Mnemonic-Befehls, den Daten und dem Steuerzeichen ETX. Dieser Wert wird mit an das Eurotherm gesendet, damit dieser diesen Wert verifizieren kann. In der Arbeit sind die Gruppennummer und die Unit Nummer Null groß. Als Antwort kommt entweder das Steuerzeichen ACK oder das Steuerzeichen NAK zurück.

Der Lese-Befehl und die Geräte-Antwort sind nach [Dokd] S. 15 folgendermaßen aufgebaut. Der Befehl besteht aus:

1. dem Steuerzeichen EOT (\x04),
2. zweimal der Gruppennummer,
3. zweimal der Unit Nummer,
4. dem Mnemonic-Befehl und,
5. dem Steuerzeichen ENQ (\x05).

Die Antwort wiederum besteht aus:

1. dem Steuerzeichen STX (\x02),
2. dem Mnemonic-Befehl,
3. den Daten,
4. dem Steuerzeichen ETX (\x03) und,
5. der BCC (Prüfsumme).

3.1.6 Genutzte Befehle

In der Quelle [Dokd] finden sich in dem Kapitel „3 902, 903, and 904 mnemonics“ auf den Seiten 26 bis 42 die Mnemonic-Befehle wieder. Bei Quelle [Doki] sind es die Seiten 386 bis 393 (Modell 3504). Die Befehle sind bei den Modellen identisch. Die genutzten Befehle können dabei in drei Kategorien unterteilt werden. Diese sind:

1. Werte für die Steuerung (Kapitel 3.1.6.1),
2. Werte für Informationen (Kapitel 3.1.6.2) und,
3. Werte für das Rezept (Kapitel 3.1.6.3).

3.1.6.1 Befehle für die Steuerung

Die folgenden Befehle werden von VIFCON für die Steuerung des Eurotherm benutzt und werden mit den Mnemonics:

1. PV – Isttemperatur (Lesen),
2. SL – Solltemperatur (Lesen, Schreiben),
3. OP – Ausgangsleistung (Lesen, Schreiben),
4. HO – maximale Ausgangsleistung (Lesen, Schreiben) und,
5. SW (Lesen) und SW> (Schreiben) – Statuswort bezeichnet.

Bei der Ausgangsleistung OP muss nach [Dokd] S. 27 beachtet werden, dass dieser Wert nur im Manuellen Modus schreibbar ist. Durch das Setzen des Statuswortes SW kann dies erreicht werden. In Tabelle 4 kann der Aufbau des Statuswortes SW gesehen werden. Es besteht aus 16 Bit, also 2 Byte. Der Bit 15 ist hierbei der Bit, der zum Umschalten zwischen Automatik und Manuell genutzt werden kann. Im manuellen Betrieb ändert sich OP nur durch den Bediener (bzw. auch durch VIFCON), im automatischen Betrieb durch den Regler. Für die Steuerung mit VIFCON muss das Statuswort also umstellbar sein. Zu der Tabelle noch eine kurze Erklärung: R/O bedeutet „Read Only“ (Quelle: [Dokd] S. 27).

Tabelle 4: Statuswort (SW) (Quelle: [Dokd] S. 31)

Digit	Bit	Function	Access	Clear/Set
A	15	Manual Active	R/W	Auto/Manual
A	14	Remote Active	R/W	Local/Remote
A	13	SP2 Active	R/W	SP1/SP2
A	12	AL1 or AL2	R/O	No Alarm/Alarm 1 or 2
B	11	N/A		
B	10	Alarm 1 State	R/O	On/Off
B	9	N/A		
B	8	Alarm 2 State	R/O	On/Off
C	7	N/A		
C	6	N/A		
C	5	Parameter Change	R/O	Not Changed/Changed
C	4	N/A		
D	3	N/A		
D	2	Key Lock	R/W	No/Yes
D	1	T/C Break	R/O	No/Yes
D	0	Data Format	R/W	Free/Fixed

Beim Senden des Befehls werden die Daten z.B. als 0000 angegeben. Jede Null steht dabei für 4 Bit. Die Reihenfolge der genutzten Bits ist ABCD. Wenn also der 15 Bit gesetzt werden will und alle anderen Null sind, dann muss bei SW> eine 8000 geschrieben werden. Die Daten sind in Hexadezimal angegeben! Die Hexadezimalzahl 8 entspricht dem Binärcode 1000.

3.1.6.2 Befehle für Informationen

Die zweite Gruppe von Befehlen ist in Zusammenhang mit der Initialisierung bzw. dem Start von VIFCON zu sehen. Somit werden bei gewollter Initialisierung des Gerätes Werte als Information gespeichert. Die Befehle dazu sind:

1. II - Instrument Identity,
2. V0 - Software Version,
3. IM - Instrumenten Modus,
4. 1L - Display Minimum,
5. 1H - Display Maximum,
6. LS - Sollwert Minimum,
7. HS - Sollwert Maximum,
8. XP - Proportional Band,
9. TI - Integral Time und,
10. TD - Derivative Time.

Während der Initialisierung werden auch das Statuswort SW und das Maximum HO (Leistung) ausgelesen und gegebenenfalls beschrieben. Unter den zuletzt genannten Befehlen sind auch die PID-Parameter P (XP), I (TI) und D (TD) vorhanden. Nach der Seite 225 aus [Doki] ist der P-Anteil einheitenlos und der I- und D-Anteil werden als Zeit in Sekunde angegeben.

3.1.6.3 Befehle für die Eurotherm-Rampe

In dem Kapitel sollen nicht nur die Befehle für die Eurotherm eigene Rezept-Funktion gezeigt werden, sondern dies auch kurz erläutert werden. In VIFCON gibt es eine Rezept-Funktion, die in Kapitel 5.5 erläutert wird. Aus dem Bestellcode, welcher in Kapitel 3.1.2 gezeigt wurde, geht hervor, dass das Eurotherm 3504, welches genutzt wird, 10 Programme und somit 500 Segmente ([Doki] S. 13) für den Programmgeber hat. Um mit den Programmen arbeiten zu können benötigt es auch weitere Mnemonics. Somit werden:

1. r1 - r8 - Steigung der 8 Rampen,
2. l1 - l8 - Zielsollwert der 8 Rampen,
3. t1 - t8 - Haltezeit und,
4. OS (Schreiben: OS>) - Optionales Statuswort verwendet.

Zuerst soll das neue Statuswort OS erläutert werden, welches in Tabelle 5 zu sehen ist. Für VIFCON ist nur der D-Anteil bedeutend, welcher in Tabelle 6 zu sehen ist. Aus diesem werden Reset (0_{16}) und Run (2_{16}) benötigt. Auch dieses Statuswort (OS) baut sich wie das andere Statuswort (SW) aus Tabelle 4 auf. Somit wird dem Eurotherm für den Start die 0102 gesendet und für den Reset die 0100. Die B- und D-Anteile des Bytes besitzen jeweils vier Bits, welche durch MSB und LSB eine Lese-Richtung erhalten. Nach Seite 32 in [Dokd] können bei D nur die Werte 0 bis 6 (siehe Tabelle 6) und bei B nur die Werte 1 bis 8 gesetzt werden. Nach Tabelle 5 geben die B-Bits das aktuelle Segment an. In dem genannten Beispiel wird somit auf das erste Segment verwiesen. In den beiden Tabellen werden Verweisungen (1, 2, *) gezeigt, diese sind auf der besagten Seite in [Dokd] zu finden. Bei 1 und 2 wird nur der Wertebereich genannt.

Tabelle 5: Optionales Statuswort (OS) (Quelle: [Dokd] S. 32)

Digit	Bit	Function	Access	Clear/Set
A	15	Digital Input 1	R/O	Off/On
A	14	Digital Input 2	R/O	Off/On
A	13	Dig Output Channel 3	R/W	Off/On
A	12	Dig Output Channel 4	R/W	Off/On
B	11	Segment No. (MSB)	R/O	see[1]
B	10	Segment No.	R/O	see[1]
B	9	Segment No.	R/O	see[1]
B	8	Segment No. (LSB)	R/O	see[1]
C	7	Digital I/P Inhibit	R/W	None/Inhibit
C	6	Ramp/Dwell	R/O	Ramp/Dwell
C	5	Skip Segment	R/W	Remain/Skip
C	4	Hold Logged	R/W	Continue/Hold
D	3	Prog/Ramp Status (MSB)		see[2]
D	2	Prog/Ramp Status		see[2]
D	1	Prog/Ramp Status		see[2]
D	0	Prog/Ramp Status (LSB)		see[2]

Tabelle 6: Optionales Statuswort (OS) - Bedeutung der D-Bits (Quelle: [Dokd] S. 32)

Value	Meaning	Access
0	Rest	R/W
1	N/A	
2	Run	R/W
3	Hold	R/W
4	Program end	R/O
5	Ramp End (Still active*)	R/O
6	Holdback Active	R/O

Auf Seite 388 in Quelle [Doki] finden sich die Tabellen von OS und SW für das Eurotherm 3504. Dabei werden die Funktionen mit der alten 900-Serie verglichen, deren Tabellen gezeigt worden sind und den Tabellen der 3500-Serie sehr ähnlich sind. Alle beschriebenen Funktionen zu den Statuswörtern sind identisch.

Bei dem Programmgeber mit den Mnemonics können je Programm 16 Segmente definiert werden, die dem eingebauten Sollwertgenerator seine Werte vorgeben. Dieser Generator liefert dem Regelkreis dann den Vergleichswert. Wenn sich die Mnemonics angesehen werden, dann ist die Reihenfolge wie folgt. Zuerst wird eine Rampe gesendet, die den Zielsollwert l1 und die Steigung r1 erhält. Danach folgt eine Haltezeit (Dwell) t1. Somit wechseln sich die beiden Teile immer ab, was auch in Abbildung 19 gezeigt wird. Die Rampe nimmt die ungeraden Segmente und die Haltezeit die geraden ein. (Quelle: [Dokaa] S. 6-1 bis 6-2)

In VIFCON wurde sich dazu entschieden, immer nur eine Rampe zu nutzen. Somit benötigt VIFCON nur l1, r1 und t1. Das t1 ist in dem Fall besonders, da es das Ende des Programms angibt. Solche Besonderheiten werden in [Dokd] auf Seite 28 angesprochen. Dies kurz zusammengefasst bedeutet, dass wenn die Rampe das Ende ist, so wird in tn eine -0,01 übergeben. Interessanterweise wird dabei eine -0,02 im Eurotherm-Gerät übernommen. Sollte die Haltezeit das letzte Segment sein, so wird in rn eine -0,02 übergeben. Beim Ende muss beachtet werden, dass das Programm dann aufhört zu arbeiten und auf den Reset oder ein neues Programm wartet. Für einen Sollwertsprung wird in rn eine Null verwendet. Das n steht hierbei für die Mnemonic Nummer bei t und r.

Wie schon erwähnt sind in diesen Programm so Rampen, Platos (Haltezeit) und auch Sprünge realisierbar. In VIFCON wird hierbei nur die Rampe benötigt. Die Rampe besteht aus deren Zielwert (ln) und aus der Steigung (rn). Die Rampe startet dabei vom aktuellen Wert, wodurch auch die Neigung der Rampe, ob sie steigt oder fällt, bestimmt wird. Im Eurotherm-Gerät muss die Zeiteinheit für die Steigung manuell angegeben bzw. eingestellt werden. Somit kann die Steigung in °C/s, °C/min und °C/h angegeben werden. (Quelle: [Dokaa] S. 6-3)

Die Abbildung 19 zeigt dabei ein Beispiel einer solchen Rampe. In dieser kann die Abfolge der Segmente gesehen werden. Dabei entspricht Pl dem l, Pr dem r und Pd dem t bei den Mnemonics. Oben wird auch erläutert, wie das Programm beendet wird. Das bedeutet jedoch nicht, dass es aus dem Zustand des Programmgebers geht. Bei Programmende wird der Zustand (Sollwert) weiterhin beibehalten, welcher erst durch den besagten Reset verlassen wird, wodurch dann der alte Betrieb wieder aufgenommen werden kann. Bei Versuchen mit der Rampe konnte gesehen werden, dass das Symbol RUN auf dem Eurotherm Bildschirm gezeigt wird, wenn ein Programm läuft. Mit Beendigung des Programms blinkt das Symbol, doch der Sollwert bleibt wie zu Programmende. Mit dem Reset wird jedoch der Sollwert vor dem Programm wieder eingestellt. (Quelle: [Dokaa] S. 6-10)

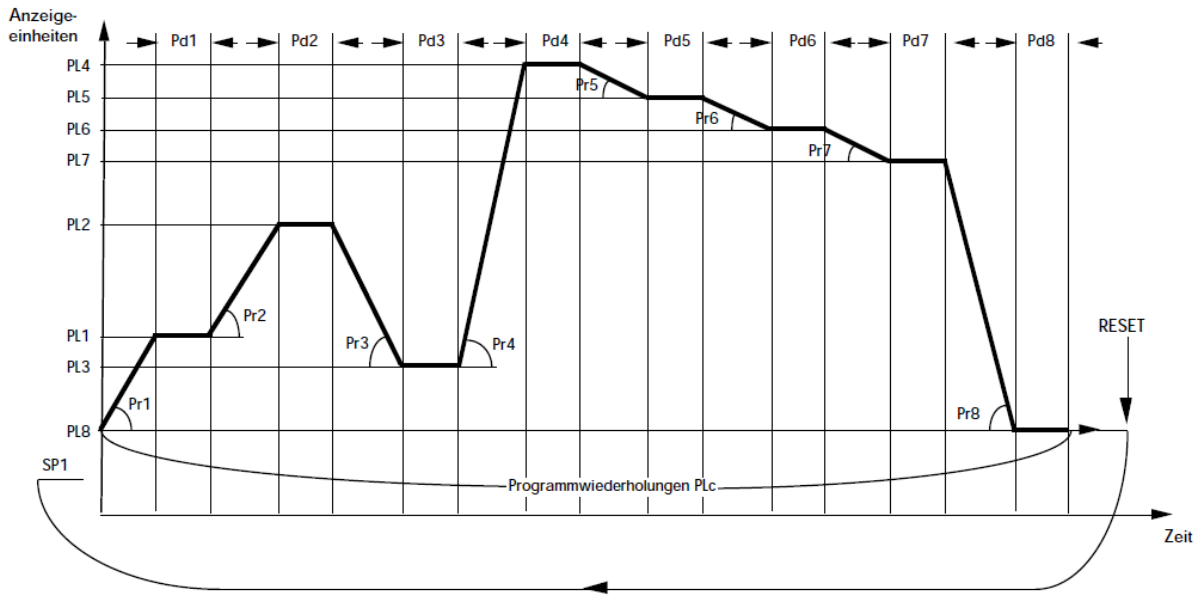


Abbildung 19: Beispiel einer Eurotherm-Rampe (Quelle: [Dokaa] S. 6-2)

3.1.7 Beispiele von Befehlen

In den folgenden Kapiteln werden die gesendeten Befehle anhand von Beispielen gezeigt. Diese Nachrichten bestehen aus ASCII-Zeichen und den Steuerzeichen aus Tabelle 3. Um die Befehle in beide Richtungen zu zeigen werden ein Lese-Befehl (Kapitel 3.1.7.2) und ein Schreib-Befehl (Kapitel 3.1.7.1) gezeigt. Das Beispiel wird anhand des Sendens und des Empfangens in Python gezeigt. Näheres wird in Kapitel 3.1.8 erläutert.

3.1.7.1 Beispiel Schreiben/Senden

Befehl: '\x04 0 0 0 0 \x02 S L 1 0 \x03 \x1D'
Antwort: '\x06'

Das Beispiel zeigt einen Solltemperatur-Schreib-Befehl. Es soll eine Temperatur von 10°C eingestellt werden. Dieser String (Befehl) wird dann an das Eurotherm gesendet. Das \x1D ist die Prüfsumme, die in ASCII ↔ und als Dezimalzahl 29 entspricht. Die Antwort des Gerätes ist hierbei entweder ACK oder NAK. Das ACK zeigt dem Nutzer, dass das Gerät den Befehl verstanden und ausgeführt hat.

3.1.7.2 Beispiel Lesen

Befehl: '\x04 0 0 0 0 P V \x05'
Antwort: '\x02 P V 2 2 . 2 2 \x03 +'

Mit dem Lese-Befehl soll die aktuelle Temperatur ausgelesen werden. In der Antwort ist nach dem PV dann die Isttemperatur zu finden. Das Plus am Ende ist die Prüfsumme, die von Eurotherm gesendet wird. Anders als beim Schreiben wird hier ein solches ASCII Zeichen als BCC angegeben. Beim Schreiben war das \x1D ein Steuerzeichen, welches durch *encode* in die hexadezimale Schreibweise gewandelt wurde.

3.1.7.3 Prüfsummen-Beispiel (XOR)

Durch eine XOR-Verknüpfung der Mnemonic Zeichen, der Daten und des ETX wird beim Senden das BCC ermittelt. Beim Senden eines Lese-Befehls wird dieses BCC nicht benötigt.

Kap. 3.1.7.1: S XOR L XOR 1 XOR 0 XOR \x03
$\widehat{=}$ \x53 XOR \x4C XOR \x31 XOR \x30 XOR \x03
= \x1D $\widehat{=}$ Group separator
Kap. 3.1.7.2: P XOR V XOR 2 XOR 2 XOR . XOR 2 XOR 2 XOR \x03
$\widehat{=}$ \x50 XOR \x56 XOR \x32 XOR \x32 XOR \x2E XOR \x32 XOR \x32 XOR \x03
= \x2B $\widehat{=}$ +

Ein ausführliches Beispiel mit der Betrachtung des Binärcodes der Hexadezimalzeichen ist in der Bachelorarbeit [Fun22] auf Seite 17 zu finden.

3.1.8 Umsetzung in Python

Für die Kommunikation mit dem Gerät wird die Python-Bibliothek **serial** verwendet. Insgesamt werden nur die beiden Methoden *write* und *readline* verwendet. Ein Beispiel soll das im Folgenden zeigen:

```
1 def read(self):
2     try:
3         self.serial.write(self.read_temperature.encode())
4         temperature = float(self.serial.readline().decode()[3:-2])
5     # Teil hier nicht gezeigt
```

Wie an dem Beispiel gesehen werden kann, werden hier noch die Methoden *encode* und *decode* verwendet. Folgende Funktionen sind im VIFCON für das Eurotherm definiert worden:

1. write,
2. write_read_answer,
3. write_EuRa,
4. bcc,
5. read,
6. check_HO und,
7. Start_Werte.

Das oben gezeigte Beispiel stammt aus der Funktion *read*. Diese beiden Zeilen werden für jede Größe oder jeden Auslese-String aufgerufen. Weiterhin wird beim Lesen keine Checksumme geprüft. Würde das noch eingefügt werden, würde sich eine neue Funktion als nützlich erweisen.

Beim Schreiben wird die Funktion *write* aufgerufen. Diese Funktion und auch *read* müssen bei allen Geräten gleich heißen. In *write* wird dann je nach Aufgabe *write_read_answer*, *Start_Werte*, *check_HO* und/oder *write_EuRa* aufgerufen. Die Funktion *write_read_answer* sendet dabei den gewollten Wert an Eurotherm und bestimmt die Checksumme. Auch die Antwort wird durch diese ausgelesen. Die Funktion *write_EuRa* sendet genau ein Rezept-Segment an den Eurotherm. Die Funktion *Start_Werte* liest die Werte aus Kapitel 3.1.6.2 aus und wird zudem auch durch *init_device* aufgerufen. Durch *check_HO* wird die max. Leistung ausgelesen.

Die Funktion *bcc* bestimmt die Checksumme wie in Kapitel 3.1.7.3 beschrieben. Diese Funktion wurde der Bachelorarbeit [Fun22] von Seite 18 entnommen. Wie schon erwähnt wurde eine ältere Version des Eurotherms (902P, 905S) in dieser Bachelorarbeit über eine RS232-Schnittstelle bereits angesprochen und ausgelesen.

Somit sind die Hauptfunktion *read* und *write*, da die anderen Funktionen durch diese aufgerufen werden. Bei der Erklärung konnte gelesen werden, dass auch bei *write* einige Informationen von dem Gerät ausgelesen werden. Dies wird getan, da die Initialisierung nur einmal vorkommt und weil die maximale Leistung (HO) für die Kontrolle der Werte gebraucht wird. HO liefert die obere Leistungsgrenze. Diese kann durch VIFCON oder den Nutzer verändert werden, wodurch es auch eine Auslese-Funktion geben muss. Weiterhin wird die Rezept-Funktion nicht durch *write_read_answer* geregelt, sondern durch *write_EuRa*. Der Grund dafür ist die Struktur der internen Rezepte, die in Kapitel 3.1.6.3 vorgestellt wurde. Außerdem wurde sich dazu entschlossen, immer nur eine Rampe zu senden. Das bedeutet, dass das zweite Segment (t1) immer als Ende definiert wird. Insgesamt werden dann hier nur die Mnemonics r1, l1 und t1 verwendet. In der Funktion wird auch sichergegangen, dass alle drei Werte richtig gesetzt werden, indem auch hier die Antwort kontrolliert wird. Auch hier wird *bcc* aufgerufen.

3.2 PI-Achse und Mercury-DC-Controller

Bei der PI-Achse handelt es sich um eine Achse (MP113D M-5x1 Versteller) mit einem Mercury DC-Motor Controller. Das PI steht für den Hersteller „Physik Instrumente“. Die genutzten Controller sind vom Modell C-862 [Dokj] und C-863 [Dokk]. Die beiden Modelle unterscheiden sich in Kleinigkeiten wie z.B. die Bedeutung der LEDs. Die Schreib- und Lese-Befehle sind fast immer identisch. Der Controller und die Achse sind in Abbildung 20 zu sehen. Diese Achse ist das erste Gerät, welches in die Kategorie Antrieb fällt.

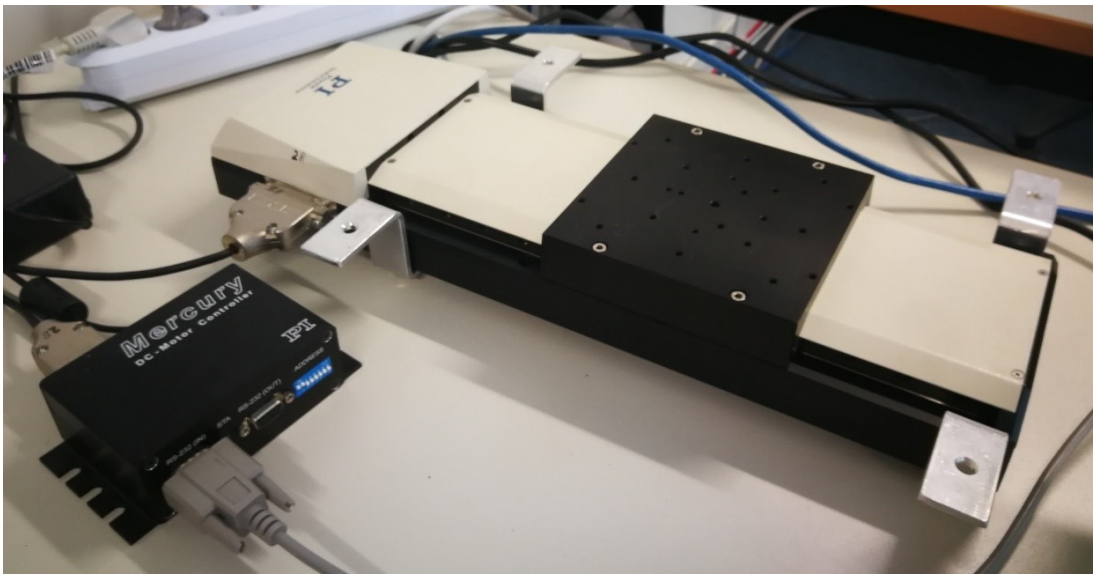


Abbildung 20: PI-Achse mit Mercury-DC-Controller des Modells C-862 (Quelle: eigene Darstellung)

3.2.1 Funktionsweise und Aufbau

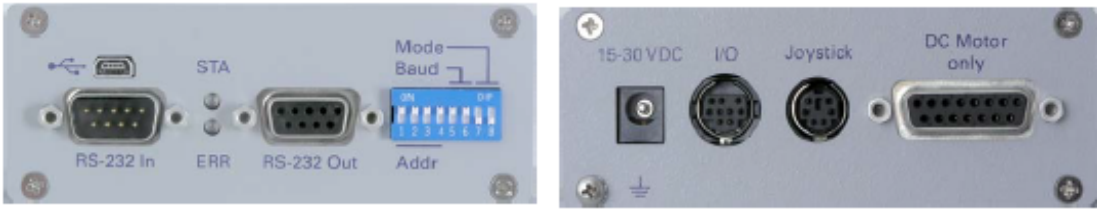
Bei den Achsen handelt es sich um Mikrostelltische mit Kugelumlaufspindel. Diese Bezeichnung wird bei dem Nachfolger-Modell der Achsen (M5x1) genannt. Bei dem Nachfolger-Modell wurden Punkte wie die Positionsmessung, die Positionsauflösung, die maximale Geschwindigkeit und das Anfahren der Referenzposition neben anderen Dingen verbessert. Da keine Dokumentation für das alte Modell gefunden wurde, wird die Funktionsweise nach eigener Erfahrung und Beobachtung der Achse kurz im Folgenden beschrieben. (Quellen: [Piab] S. 9, [Piac] S. 1)

Die Funktion der Achse ist ungefähr so zu verstehen: Die Geschwindigkeit wird in counts je Sekunde bzw. steps je Sekunde angegeben. Im Inneren der Achse befindet sich eine Gewindestange, die durch den Motor gedreht wird. Der daran montierte Aufsatz bewegt sich somit durch die Drehung des Gewindes.

Um die Kommunikation zwischen der Achse und dem Computer zu gewährleisten, benötigt es noch einen Mercury-Controller. Somit wird die Schnittstelle über den Mercury-Controller und den Computer eingerichtet. Die Befehle werden an den Controller gesendet und der Controller wird von dem Computer ausgelesen. In Abbildung 20 kann gesehen werden, dass ein Stecker in den Mercury hineingeht und auf der anderen Seite wieder herausgeht. Für den C-863 wird dies in Abbildung 21 gezeigt. Auf der Front-Seite (Abbildung 21a) kann zum einem RS232 In und zum anderen RS232 Out gesehen werden. In Kapitel 6.1 wird auch der Stecker RS232 Out verwendet. Mit diesem können mehrere Controller an einer Schnittstelle hängen. Beim Back-Panel (Abbildung 21b) wird dann „DC Motor only“ mit der Achse verbunden.

Somit können bis zu 16 Controller über dieselbe Schnittstelle bearbeitet werden. Was im Sinne der Befehle dafür getan werden muss, wird in Kapitel 3.2.5 erläutert. Dabei kann jeder der Controller einzeln und individuell angesprochen werden. Wenn die Kommunikation mit einem Controller läuft, wird die Kommunikation mit den anderen aufgehoben. Der Wechsel zwischen den Controllern erfolgt durch die Adresse, die im besagten Kapitel erläutert wird. Das Prinzip der Verbindung wird als „Daisy Chain“ bezeichnet. (Quelle: [Dokj] S. 8)

Dabei beschreibt das „Daisy Chain“ Prinzip, die serielle Verkettung von Hardwarekomponenten. Bei dieser Verkettung wird die erste Hardwarekomponente mit z.B. dem Bussystem oder dem Computer verbunden, die folgenden Komponenten sind immer mit dessen Vorgänger verknüpft. Diese Verkettung kann in Kapitel 6 in Abbildung 66 gesehen werden. (Quelle: [Piaa])



a) Front-Seite (Quelle: [Dokk] S. 11) **b)** Back-Seite (Quelle: [Dokk] S. 12)

Abbildung 21: Mercury -Controller C-863 Panels

Bei dem Mercury handelt es sich um einen Servoregler, welche über einen PID die Position, Geschwindigkeit und die Beschleunigung regeln kann. (Quelle: [Dokj] S. 5)

Die Position wird bei der Achse in steps oder counts und die Geschwindigkeit in counts je Sekunde angegeben. Ob steps oder counts genutzt werden, hängt von der Hardware ab, die an dem Controller hängt. Für das Positions-Feedback, werden die Counts durch einen integrierten Encoder definiert. Die Steps beziehen sich auf Schrittmotoren. (Quelle: [Dokl] S. 6)

Weiterhin ist zu beachten, dass der Mercury eine Zielposition berechnet. Immer wenn die Position vom Computer gesetzt wird, wird intern die Zielposition gesendet. Sobald der Motor-Servo-Loop aktiv ist, will der Controller diese Position erreichen. (Quelle: [Dokj] S. 46)

Der Mercury-Controller besitzt LEDs. Die STA-LED steht für den Status. Diese LED wird rot wenn der Motor Servo-Loop ausgeschaltet ist und grün wenn die Achse sich bewegt bzw. der Motor Servo-Loop aktiv ist. Bei der ERR-LED handelt es sich um den Fehler. Wenn ein fehlerhafter Befehl gesendet wird, so wird diese LED rot. Sobald ein richtiger Befehl kommt, wird sie wieder grün. Somit kann der Nutzer nur indirekt erfahren, ob der Befehl angekommen ist. (Quelle: [Dokj] S. 9)

Auf S. 19 in Quelle [Dokk] ist etwas zu den LEDs beim Modell C863 erläutert worden.

3.2.2 Spezifikation für Hardware

In der Einleitung zu den Achsen wurde erwähnt, dass diese nach [Piab] zu dem MP113D M-5x1 Versteller gehört. Insgesamt werden 4 dieser Achsen verwendet. Auf einer dieser Achsen konnte die Modell-Nummer **M-525. 22** abgelesen werden. Nach [Piac] Seite 1 ist das genannte Modell der Vorgänger für die in [Piab] gezeigten Achsen.

Über die Modell-Nummer kann der Stellweg abgelesen werden. Die mittlere Nummer, in dem Fall die 2, steht für den Stellweg, welcher hier 204 mm groß ist. Dieser Fakt scheint sich nicht geändert zu haben, da dieser Wert bei den genutzten Achsen nachgewiesen werden konnte. (Quelle: [Piab] S. 9)

Bei dem Mercury-Controller von Typ C-862 gibt es für die Geschwindigkeit einen angegebenen Wertebereich. Dieser wäre von 0 bis 500000 counts/s angegeben. (Quelle: [Dokj] S. 43)
Um diesen Wert in mm/s anzugeben, benötigt es einen Umrechnungsfaktor (auch beim Weg). Dieser kann sich von Achse zu Achse minimal unterscheiden. Der Wert 29752 cpm (Counts per mm) ist einer der Werte und wurde von Dr. Kaspars Dadzis [Perc] bestimmt. Die Berechnung kann in Formel 3.1 gesehen werden. Somit kann die Achse bis ca. 17 mm/s betrieben werden. Beim C-863 wird kein Wert dafür angegeben (Dokumentation [Dokl] S. 35).

$$v_{\mathsf{mm/s}} = \frac{v_{\mathsf{counts/s}}}{\mathsf{Faktor}} = \frac{500000\frac{counts}{s}}{29752\frac{counts}{mm}} = 16,81\frac{mm}{s} \tag{3.1}$$

3.2.3 Spezifikation für Schnittstellen

Die Schnittstellendaten für das Modell C-862 sind auf der Seite 13 der Quelle [Dokj] zu finden. Bei C-863 stehen diese´in [Dokk] auf Seite 23 . Die Schnittstellendaten beider Motor-Controller sind identisch und haben diese Werte:

1. Baudrate mit 9600 Bits/s,
2. Bytegröße mit 8 Bits,
3. ein Stoppbit und,
4. einer Parität von None (N).

Bei den ersten Test mit der Achse wurde festgestellt, dass dieses Gerät ein speziell gekreuztes RS232-Schnittstellenkabel verwendet. Nach dem Prüfen der Schnittstellenpins mit einem Multimeter (Widerstand) können die Pins nach Abbildung 22 und Tabelle 7 zugeordnet werden. Die Abbildung 22 soll zunächst beide Buchsen des RS232-Kabels zeigen. Diese Buchsen haben auf beiden Seiten Pins. Die Nummerierung ist beidseitig identisch (dies zeigen die Farben). Weiterhin gibt es auf den Enden Buchstaben, in dem Beispiel F und A.

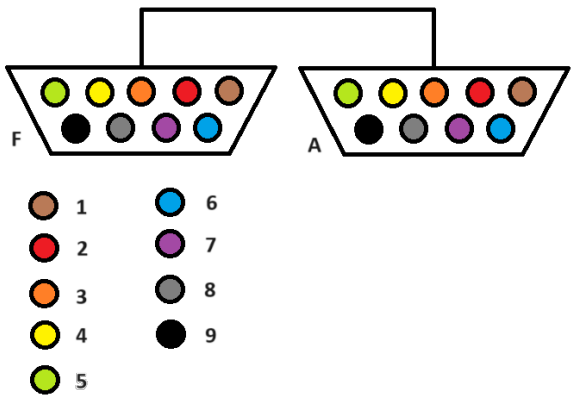


Abbildung 22: Darstellung der RS232-Buchsen mit Pins (Quelle: eigene Darstellung)

In der Tabelle 7 kann nun die interne Verknüpfung der Pins in dem RS232-Kabel gesehen werden. Mit dem Wissen aus Tabelle 7 und Abbildung 22 kann das gewollte Kabel durch z.B. Lötung hergestellt werden.

Tabelle 7: spezielle RS232-Kabel-Kreuzung für PI-Achse (Quelle: eigene Darstellung)

Seite F des RS232-Kabels	Seite A des RS232-Kabels
1	4
2	3
3	2
4	1 und 6
5	5
6	4
7	8
8	7
9	9

3.2.4 Kommunikationsprotokoll

Für das Senden gibt es zwei Befehlssätze, zum einen das hier genutzte „Native Command Set“ und zum anderen das „GCS Command Set“, auf das nicht weiter eingegangen wird. Kurze Beschreibungen sind in Quelle [Dokk] auf Seite 8 zu finden. Die Native Commands benutzen wie bei Eurotherm (Kapitel 3.1) sogenannte Mnemonic-Befehle, die aus meist zwei Buchstaben bestehen.

Beim „Native Command Set“ werden die Befehle als ASCII-Zeichen gesendet und von der Firmware des Mercury-Controllers direkt verstanden. Auch das andere Set wird mit ASCII-Befehlen verwendet. (Quelle: [Dokl] S. 3)

Bei der Kommunikation ist zu beachten, dass bei mehreren in Serie geschalteten Motor-Controllern jeder einen eigenen Adressauswahlcode hat. Dieser Adressauswahlcode (erläutert in Kapitel 3.2.5) muss das erste sein, das in einem Befehl an den Controller gesendet wird. Weiterhin ist beim Senden auch ein Abschlusszeichen zu beachten. Anders als bei Eurotherm (Kapitel 3.1) gibt die Achse keine direkte Bestätigung wieder, ob der Befehl angekommen ist. Durch die ERR-LED ist es aber möglich zu schauen, ob ein Befehl falsch einging.

3.2.5 Aufbau der Nachrichten

Der Aufbau von Schreib- und Lese-Befehl ist bei diesem Gerät identisch. Nur die Antworten unterscheiden sich. Der Befehl selbst wird als Bytearray an die Achse gesendet und besteht aus folgenden drei Teilen:

1. dem Adressauswahlcode,
2. dem Befehl und den Daten und,
3. einem Abschlusszeichen.

Um den Adressauswahlcode zu bestimmen, werden Schalter am Mercury verwendet. Diese Schalterfläche (blau) ist in der Abbildung 20 zu sehen. Die Abbildung 23 zeigt die Stellung der Schalter und auch die beiden LEDs. Weiterhin ist hier auch die RS232-Schnittstelle zusehen. Der Mercury C-863 (grau) sieht ähnlich aus, nur dass die Schalter und die LEDs anders funktionieren.

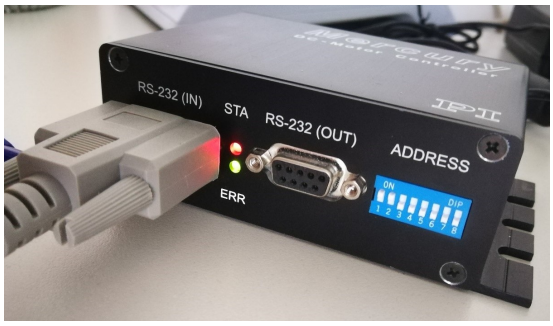


Abbildung 23: Mercury-DC-Controller - Schalterfläche (Modell C-862) (Quelle: eigene Darstellung)

Die Interpretation für die Schalter kann in Quelle [Dokj] auf den Seiten 12 und 13 für C-862 und in Quelle [Dokk] auf S. 23 für C-863 gefunden werden. Der Adressauswahlcode ist aus den Schaltern 1 bis 4 abzulesen. Die Kombination (Abbildung 23) An, An, Aus, Aus ergibt die Adresse 3. Diese Adresse hat die Gerätenummer 4 und den Hexadezimalcode 0x33. Der Adressauswahlcode wird mit der Hexadezimalzahl 0x01 eingeleitet. Somit ergibt sich der Adressauswahlcode **0133**. Der Adressauswahlcode besteht aus zwei Zeichen. Wie bereits erwähnt ist der erste die Hexadezimalzahl 0x01 und danach folgt das Adresszeichen, welches im Beispiel die 0x33 ist. (Quelle: [Dokj] S. 16)

Diese 01_{16} hat eine wesentliche Bedeutung für das Nutzen von mehreren Achsen. Um zwischen den Controllern zu wechseln, benötigt es genau diese Hexadezimalzahl. (Quelle: [Dokj] S. 8)

Die Schalter 5 und 6 geben die Baudrate an. Durch das Setzen beider auf Aus, wird als Baudrate die 9600 festgelegt. Neben diesen Schaltern gibt es noch die Schalter 7 und 8, welche beim C-862 keinen Nutzen bekommen haben. Diese Beschreibungen sind auf den besagten Seiten in den Dokumenten zu finden.

In der Dokumentation [Dokl] auf der Seite 18 findet sich noch eine Besonderheit. Bei den sogenannten „Report Commands“ werden drei Endzeichen mitgesendet. Diese sind CR, LF und ETX. Das ETX kann in der Tabelle 3 gefunden werden und gibt das Ende der Nachricht an. CR bedeutet „Carriage Return“, was in Hexadezimal der \0D und als ASCII-Symbol dem \r entspricht. Das LF steht für „Line feed“, was in Hexadezimal dem \0A und als ASCII-Symbol dem \n entspricht. Das Abschlusszeichen, das beim Senden einer Nachricht mitgesendet werden muss, ist das Carriage Return (CR).

Bei den Lese-Befehlen erhält das Programm nach Auslesen der Antwort aus der Schnittstelle einen String. Bei den Befehlen in Kapitel 3.2.6 wird dies gezeigt. Jeder Lese-Befehl hat einen anderen Buchstaben. Z.B. hat der Befehl TT (Tell Target) ein T als Antwort und TY (Tell programmierte Geschwindigkeit) ein Y. Nach diesen folgt ein Doppelpunkt und das Vorzeichen des Wertes. Im Folgenden kommen 10 Ziffern, die den ausgelesenen Wert darstellen.

Wie gesagt muss zwischen Lese- und Schreib-Befehlen unterschieden werden. Weiterhin muss nicht nach jedem Befehl ein Wert kommen. Zum Beispiel kann der Lese-Befehl TV (Geschwindigkeit lesen) einen Wert zwischen 1 und 65535 erhalten. Wenn kein Wert angegeben ist, wird bei Default eine Zeit von 1000 ms verwendet. Der Befehl zählt die Schritte in der angegebenen Zeit (in ms). Der Befehl ist in Quelle [Dokj] auf S. 46 zu finden. Manche Schreib-Befehle wie z.B. GH (Go Home) benötigen keine Daten, während z.B. MR (Relative Bewegung) ohne Wert nicht funktioniert. Die genutzten Befehle werden in Kapitel 3.2.6 gezeigt.

3.2.6 Genutzte Befehle

Die Befehle können in [Dokj] auf Seite 32 bis 49 und in [Dokl] auf Seite 17 bis 44 gefunden werden. Folgende Befehle werden in der Steuerung verwendet. Diese sind:

1. AB - Stoppe Motor abrupt und speichere Position als Zielposition (Schreiben) ([Dokj] S. 35, [Dokl] S. 22),
2. AB1 - Stoppe Motor sanft und speichere Position als Zielposition (Schreiben) ([Dokj] S. 35),
3. DH – Definiere aktuelle Position als Home-Position (Schreiben)([Dokj] S. 37, [Dokl] S. 24-25),
4. MF – Motor Off – Halte den Motor an, speichere die Position aber nicht als Zielposition (Schreiben) ([Dokj] S. 40-41, [Dokl] S. 31),
5. MRn – Move Relative – Bewege die Achse um n counts (Schreiben) ([Dokj] S. 40, [Dokl] S. 31-32),
6. SVn – Set Velocity ($0 < n < 500.000$) – Stelle die Geschwindigkeit ein, n in counts/s (Schreiben) ([Dokj] S. 43, [Dokl] S. 35),
7. TP – Tell Position (Antwort: P:+0000005555) (Lesen) ([Dokj] S. 45, [Dokl] S. 39) und,
8. TVn/TV – Tell actual Velocity ($1 < n < 65.535$) (Antwort: V:+0000020006) (Lesen) ([Dokj] S. 46, [Dokl] S. 41).

Hierbei gibt es bereits Unterschiede zwischen den beiden Controller-Modellen. Bei C-862 muss bei TV ein Wert mit angegeben werden. Dieser Umstand wurde am Ende von Kapitel 3.2.5 gezeigt. Bei C-863 muss dieser Wert nicht mehr angegeben werden, wodurch das Auslesen der Geschwindigkeit bei diesem deutlich schneller funktioniert.

Neben diesem Umstand hat der C-863, den AB1-Befehl nicht. Weiterhin kann bei diesem Modell der DH-Befehl als DHn-Befehl ausgeführt werden. Nach [Dokl] auf Seite 25 (DH Beispiel) kann die aktuelle Position auch einen anderen Wert als Null bekommen.

Auch bei der PI-Achse werden während der Initialisierung noch weitere Befehle zur Achse ausgelesen. Diese sind:

1. TB - Board Adresse (Lesen) ([Dokj] S. 43, [Dokl] S. 36),
2. TS – Status (Lesen)([Dokj] S. 45-46, [Dokl] S. 39-40)) und,
3. VE – Version (Lesen) ([Dokj] S. 47, [Dokl] S. 42).

Zusätzlich wird auch die Start-Position (TP) ausgelesen. Außerdem wurde noch der Lese-Befehl TT eingefügt. Dieser liest die aktuelle Zielposition aus. Zu finden ist der Befehl in [Dokj] auf S. 46 und in [Dokl] auf S. 40.

3.2.7 Beispiele von Befehlen

Um die Kommunikation bzw. das Senden und Empfangen von Daten zu erläutern, werden hier Beispiele gezeigt. Wie aus den vorherigen Kapiteln hervorging, sendet die Achse keine Antwort nach einem Schreib-Befehl. Nur durch die LED (siehe Kapitel 3.2.1) kann das Fehlschlagen eines Befehls gesehen werden.

3.2.7.1 Beispiel Schreiben/Senden

In dem Beispiel wird der Befehl MR1000 ausgelöst. Wenn die Achse steht oder auch der Servo-Motor-Loop aus ist, wird dieser angeschaltet und die Achse bewegt sich um 1000 steps oder counts in die positive Richtung. Dabei müssen der Adressauswahlcode und das Abschlusszeichen in Hexadezimal vorliegen. Der Befehl an sich wird in ASCII gesendet. Somit ähneln das Senden und Empfangen den Eurotherm-Befehlen. Bei einem unbekannten Befehl oder einem Fehler im Befehl wird die ERR-LED rot leuchten.

Befehl:	`b'\x01 3 M R 1 0 0 0 \r'`
Antwort:	ERR-LED leuchtet grün! STA-LED wechselt zu grün!

3.2.7.2 Beispiel Lesen

In dem Beispiel soll die Istposition der Achse ausgelesen werden. Dies passiert mit dem Befehl TP (Tell Position). Die Antwort enthält 10 Ziffern und ein Vorzeichen. In Kapitel 3.2.6 wurde bereits erwähnt, dass die Nachfrage-Befehle immer zu Beginn einen Buchstaben und einen Doppelpunkt haben. In dem Fall ist dieser Buchstabe das P (Position). Am Ende der Antwort finden sich die besagten drei Zeichen wieder.

Befehl:	`b'\x01 3 T P \r'`
Antwort:	`b'P : + 0 0 0 0 0 0 5 5 5 5 \r \n \x03'`

3.2.8 Umsetzung in Python

Da es von der Achse bzw. vom Motor-Controller keine Rückmeldung oder Prüfsumme gibt, benötigt es keine weiteren Funktionen für das Schreiben. Somit beinhaltet das Programm folgende Funktionen für das Schreiben und Lesen. Diese Funktionen heißen:

1. write,
2. stopp,
3. read,
4. read_TP,
5. read_TV,
6. send_read_command,
7. entferneSteuerzeichen und,
8. wertSchneiden.

Auch hier wird die Python-Bibliothek **serial** verwendet. Die Funktionen *write* und *read* sind dabei die Haupt-Funktionen. In beiden Funktionen werden die gewünschten Größen linear abgearbeitet. Die Funktionen *stopp*, *read_ TP* und *read_ TV* sind dabei spezielle Funktionen, die auch anderweitig aufgerufen werden können. So wird z.B. ein Bestimmen der einzelnen Messgrößen in VIFCON benötigt, um bestimmte Funktionen zu realisieren. Bei den Funktionen *entferne-Steuerzeichen* und *wertSchneiden* handelt es sich um Funktionen, die die Antwort (siehe Kapitel 3.2.7.2) bearbeiten, sodass der Nutzer einen Wert zurückbekommt.

Um den Befehl zu senden, wird die *write*-Methode verwendet. In Kapitel 3.2.7.1 wurde beschrieben, dass der Beginn und das Ende des Befehls im Hexadezimal-Format vorliegen müssen. Diese Umwandlung wird durch die Nutzung der *fromhex*-Methode des *bytearray*-Objektes durchgezogen. Der Befehl selbst wird durch die *encode*-Methode in ein Bytearray gewandelt. Dabei bleiben die ASCII-Zeichen erhalten. Der gesendete Befehl ist in Kapitel 3.2.7.1 zu sehen.

```
1  self.t1 = bytearray.fromhex('0133')
2  Befehl = 'MR1000'
3  self.t3 = bytearray.fromhex('0D')
4  self.serial.write(self.t1+Befehl.encode()+self.t3)
5  ant = self.serial.readline().decode()
6
7  def entferneSteuerzeichen(self, String):
8      weg_List = ['\r', '\n', '\x03']
9      for n in weg_List:
10         String = String.replace(n, '')
11     return String
```

Die Zeilen 1 bis 5 des gezeigten Codes sind allgemein gehalten, da die ersten drei Teile die Zusammensetzung des gesendeten Befehls (Adressauswahlcode, Befehlszeichen und Abschlusszeichen) vorbereiten. Die Zeilen 4 und 5 werden in einigen der genannten Funktionen verwendet. Im Kapitel 3.2.7.2 wird das Argument der *serial.write*-Methode als Bytearray gezeigt. Zum Lesen der Antwort wird die *readline*-Methode und zusätzlich die *decode*-Methode verwendet. Das *readline* liest dabei die Antwort des Gerätes nur bis zum Line Feed, wodurch das ETX eine Zeile später kommt. Aus dem Grund läuft die Funktion *entferneSteuerzeichen* (oberer Code Zeilen 7 bis 11) über die Antwort. Somit wird sichergestellt, dass die Antwort sauber bleibt. Das folgende Beispiel zeigt zwei Lese-Zyklen und die Nicht-Nutzung von *entferneSteuerzeichen*. Das *decode* entfernt die Steuerzeichen nicht, sondern löst sie aus. Das Line-Feed ist durch die Leerzeile zu sehen und das ETX wird im Folge-Befehl von Python als Herz dargestellt. Mit **vor** ist der String gemeint, der ausgelesen wurde und mit **nach** der bearbeitete String!

```
1  vor: P:-0000000008
2
3  nach: P:-0000000008
4  vor: ♥V:+0000000000
5
6  nach: V:+0000000000
```

Wie beim Eurotherm besitzt auch die PI-Achse die Funktion *Start_ Werte*, die von *write* und *init_ device* aufgerufen werden kann. Bei dieser Funktion werden die speziellen Informationen und auch die Start-Position ausgelesen.

3.3 Induktionsgenerator

Das dritte Gerät ist ein Induktionsgenerator, speziell der TruHeat Generator von Typ HF 5010. Der Generator stammt von der Firma Trumpf. Die Dokumentation dieses Gerätes ist in der Quelle [Dokb] zu finden. Die Abbildung 24 zeigt den Generator.

Abbildung 24: TruHeat HF 5010 (Quelle: eigene Darstellung)

Der Induktionsgenerator TruHeat gehört zum Geräte-Typ Generator. In der Steuerung soll es dann möglich sein, die drei Sollwerte Spannung, Strom und Leistung setzen zu können, über welche die Regelung des Ausgangs des Generators erfolgt. Neben den Sollwerten sollen auch die Istwerte ausgelesen werden. Hierbei gibt es noch die vierte Größe Frequenz. In den folgenden Kapiteln werden Sachverhalte zu dem Gerät und der Schnittstelle erläutert.

3.3.1 Funktionsweise und Aufbau

Der TruHeat-Generator besitzt ein Display, auf dem verschiedene Konfigurationen und auch die Ist- und Sollwerte abgelesen werden können. Die Abbildung 25 zeigt dieses Display. Hier sind bestimmte Geräte-Spezifikationen zu sehen.

Abbildung 25: TruHeat Konsole - Geräte-Spezifikation (Quelle: eigene Darstellung)

Die Bedeutung der Schaltfläche wird in Abbildung 26 gezeigt. Über diese Schaltfläche können Stör- und Fehlermeldungen eingesehen und das gültige Interface eingestellt werden.

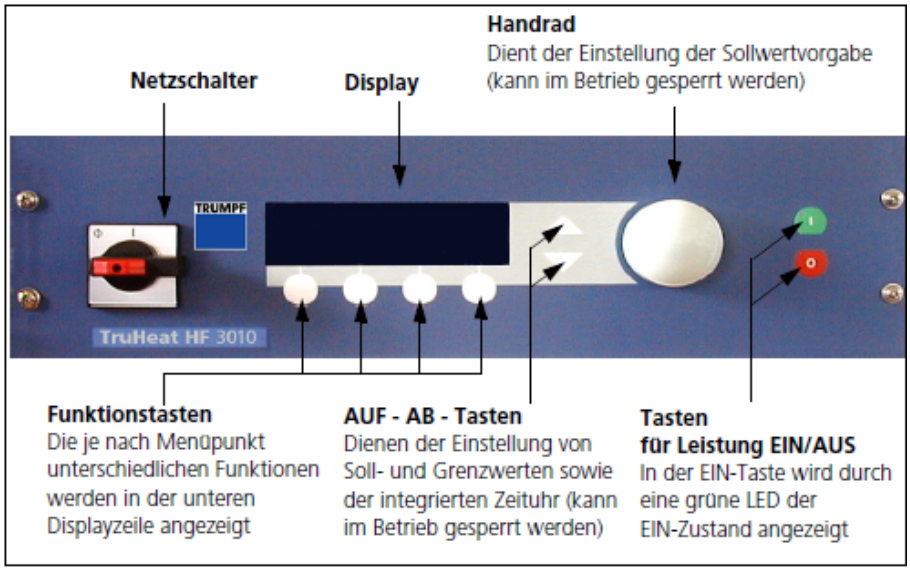
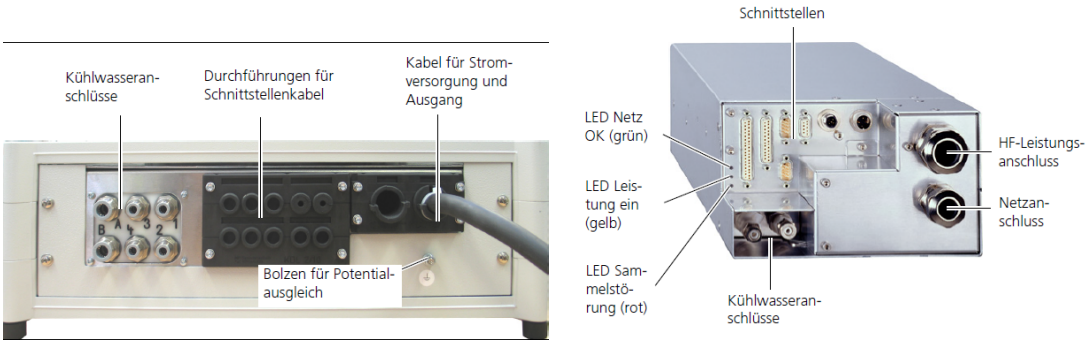


Abbildung 26: Erläuterung der Schaltfläche am TruHeat (Quelle: [Dokb] S. 72)

Bei dem Generator handelt es sich um einen Hochfrequenzgenerator (HF), welcher volltransistorisiert und μC-gesteuert (μC steht für Mikrocontroller) ist. Um den Generator zu bedienen, kann entweder die gezeigte Schaltoberfläche (Abbildung 26) oder ein Fernzugriff durch Schnittstellen (Kapitel 3.3.2) genutzt werden. Für die Nutzung des Generators muss zunächst die Drehspannung gleichgerichtet werden. Für diesen Zweck gibt es im Netzteil einen EMV-Filter, eine Sechspulsbrücke und eine Filterdrossel. Danach wird die Gleichspannung wieder zu einer Wechselspannung gemacht, die eine veränderliche Frequenz besitzt. Für diesen Zweck wird eine Wechselrichterbrücke benutzt. Diese Spannung mit der Taktfrequenz liegt an dem Ausgang des Netzteils an. Hierbei wird dann dieser Ausgang mit einem Außenkreis über ein geschirmtes HF-Kabel verbunden. In dem Außenkreis gibt es noch Transformatoren zur Potentialtrennung und Kondensatoren. Diese Kondensatoren bilden mit einem angeschlossenen Induktor (am Ausgang) einen Serienkreis. Sollte dieser durch eine Wechselspannung angeregt werden, so entsteht eine periodische Schwingung von Strom und Spannung. Je näher sich Taktfrequenz und Resonanzfrequenz des Serienschwingkreises sind, desto höher wird die Amplitude der beiden Größen. Genau diese Eigenschaft wird zur Leistungssteuerung verwendet. (Quelle: [Dokb] S. 31)

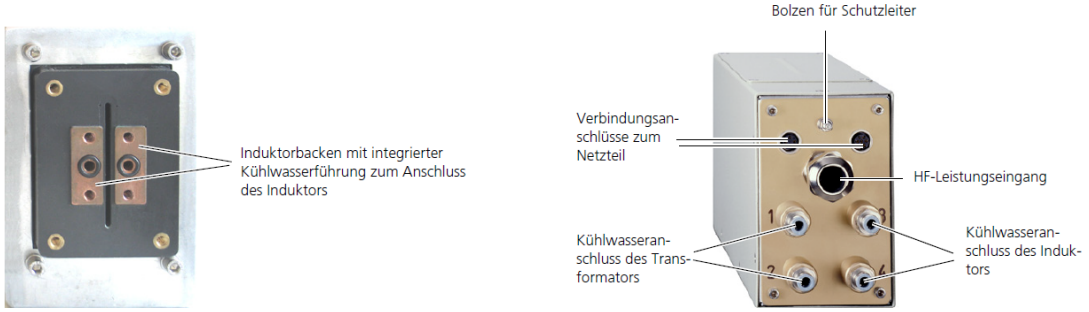
Der Generator dient der Energieversorgung von Anwendungen, die Induktionserwärmung nutzen. Somit ist die Last des Generators in dem meisten Fällen ein Material, das durch ein Magnetfeld induktiv erwärmt wird. (Quelle: [Dokb] S. 33)

In den Abbildungen 27 und 28 werden weitere Anschlüsse mit Bezeichnung gezeigt. Die Abbildung 27a zeigt dabei die Rückseite des Generators, in dem dann das Modul aus Abbildung 27b versteckt ist. Dieses wird in Kapitel 3.3.2 noch einmal aufgegriffen. Die Abbildung 28 zeigt den Außenkreis bzw. den Anschluss für den Induktor (Abbildung 28a).



a) Generators **b)** Moduls

Abbildung 27: Bezeichnungen an den Rückseiten des ... (Quelle: [Dokb] S. 29)



a) Vorne **b)** Hinten

Abbildung 28: Bezeichnungen am Außenkreis (Quelle: [Dokb] S. 30)

Auch hier gibt es eine Regelung im Hintergrund. Nach welcher Größe der Generator regelt, hängt von den Sollwerten bzw. von dem Generator ab. Der Nutzer kann die aktuelle Regelung anhand des Formelzeichens sehen, das markiert wird (Inverse Darstellung). (Quelle: [Dokb] S. 36)

Der Generator verfügt auch über die Anzeige von Fehler- und Warnmeldungen. Dabei sind diese zu unterscheiden. Bei einer Störung/einem Fehler wird der Generator sofort abgeschaltet, wenn dieser im Leistungsbetrieb war. Das Einschalten ist in dem Fall nicht möglich, bis alle Störungen behoben sind. Die Fehlermeldungen mit geringer Gefahr lösen dies nicht aus und werden als Warnmeldung angezeigt. Die verschiedenen Meldungen können in [Dokb] auf den Seiten 110 bis 119 gefunden werden. (Quelle: [Dokb] S. 110)

3.3.2 Spezifikation für Hardware

Der TruHeat besitzt verschiedene Schnittstellen, welche in Abbildung 29 zu sehen sind.

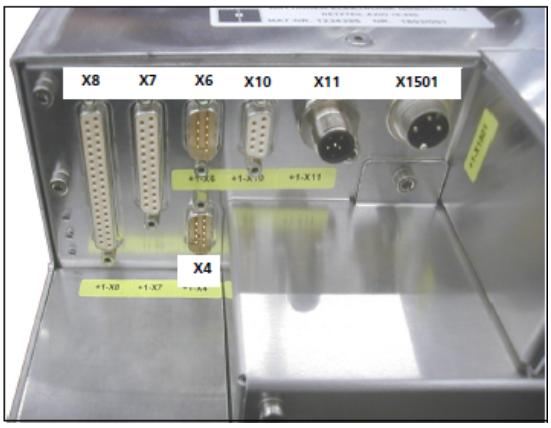


Abbildung 29: TruHeat Schnittstellen (Quelle: [Dokb] S. 58)

Die Bezeichnungen in der Abbildung 29 sind in [Dokb] auf Seite 58 zu finden und bedeuten folgendes:

- X8 AD-Schnittstelle (mit Interlock)
- X7 TRUMPF Hüttinger Bus
- X6 RS232 - Service
- X4 RS232
- X10 Profibus
- X11 CAN Bus
- X1501 Spannungsmessung

Bei VIFCON wird die Schnittstelle X4 verwendet. Diese Schnittstelle ist eine RS232, die in Kapitel 2.3.1 näher erläutert wurde. Neben dieser Schnittstelle wurde die X6 Schnittstelle verwendet. Diese Schnittstelle kann mit einem speziellen Programm genutzt werden. Das Programm ist ein Terminalprogramm und heißt Tera Term (ttempro.exe). Eine Beschreibung der beiden RS232-Schnittstellen ist auf den Seiten 132 und 133 in [Dokb] zu finden. Das Terminal ist in Abbildung 30 zu sehen. Wenn die Schnittstellenparameter in dem Programm richtig gesetzt worden sind, wird die Konsole auf dem Generator angezeigt. Die Abbildung zeigt die Ist- und Sollwerte der 4 Messgrößen.

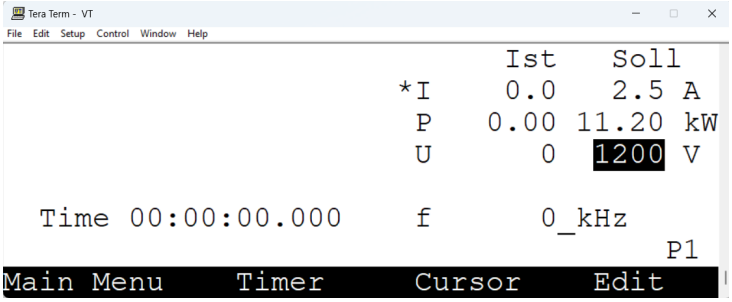


Abbildung 30: Terminalprogramm Tera Term (Quelle: eigene Darstellung)

Die Tabelle 8 zeigt verschiedene Daten für die 4 Messgrößen. Dabei werden Daten aus dem Datenblatt [Dokz] und selbst ausgelesene Werte gezeigt. Auf der Internetseite [Dokz] sind die Werte unter TruHeat HF 5010 (Technische Daten) oder in dem herunterladbaren PDF (Technisches Datenblatt) zu finden. Die Befehle werden in Kapitel 3.3.6 gezeigt. Neben den Messgrößen konnten so auch die Software-Version 01.21.00.00, die Seriennummer der Stromversorgung 201215795 und der Netzteil-Typ 10 kW, 3x400 V bestimmt werden. Nach [Dokb] S. 16 bedeutet UC Kondensatorspannung.

Tabelle 8: TruHeat HF 5010 - Daten Übersicht (Quelle: eigene Darstellung)

Größe	**Wert aus [Dokz] S. 2** (PDF)	Wert Ausgelesen
Leistung	**Netzanschlussdaten**: Netzaufnahmeleistung = 12,5 kVA Leistungsfaktor = 0,95 **Ausgang:** Nennleistung = 10 KW	max. Leistung = 112 kW
Spannung	**Ausgang:** max. Spannung = 1500 V eff	min. UC = 0 V max. UC = 1200 V
Strom	**Ausgang:** max. Strom = 560 A eff	min. Ausgangsstrom = 2,5 A max. Strom (HF) = 35 A
Frequenz	**Netzanschlussdaten**: Netzfrequenz = 50 Hz - 60 Hz **Ausgang:** Nennbetriebsfrequenz = 50 kHz - 800 kHz	/
Kühlung	Max. Wasserdruck = 6 bar Min. Druckdifferenz = 3 bar Min. Durchflussmenge = 9,5 l/min max. Temperatur Kühlmedium = 35°C	/
Umgebungs-bedingungen	Außentemperatur = 10°C - 45°C	/

3.3.3 Spezifikation für Schnittstellen

Um mit einer Schnittstelle zu kommunizieren müssen bestimmte Werte gesetzt werden. Für die RS232-Schnittstelle sind diese in Kapitel 2.3.1 erläutert. Diese Werte sind in Tabelle 9 zu sehen. In der Dokumentation [Dokb] sind diese Werte auf den Seiten 132 (X6) und 133 (X4) zu finden. Nur die Baudrate für X6 ist dort nicht beschrieben.

Tabelle 9: Spezifikationen der RS232-Schnittstellen des TruHeat (Quelle: eigene Darstellung)

	X4 - RS232	**X6 - RS232 - Service**
Baudrate	19200	57600
Bytegröße	8	8
Stoppbits	1	1
Parität	None (N)	None (N)

Neben dieser Einstellung muss an dem Generator noch etwas beachtet werden. Bei der Konfiguration muss ein Interface eingestellt werden. Im Sinne einer Kommunikation mit einem PC kann der Generator nur über die aktive Schnittstelle bedient und gesteuert werden. Weiterhin kann der Generator auch direkt über das Bedienpanel (Abbildung 26) verwendet werden. (Quelle: [Dokb] S. 154)

Nach [Dokb] S. 88 und S. 154 gibt es die Schnittstellen:

1. Bedienpanel,
2. RS232,
3. PROFIBUS,
4. CANopen,
5. Terminal,
6. AD-Schnittstelle,
7. Regulus (Temp) und,
8. User 1 ... User 4.

Wie bei der PI-Achse (Kapitel 3.3) war die interne Kreuzung der Leitung im RS232-Kabel speziell. Die interne Kreuzung ist in Abbildung 31 zu sehen.

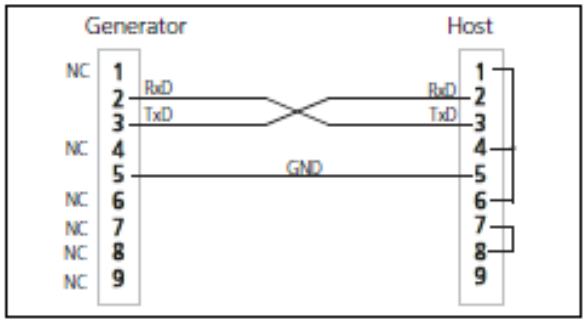
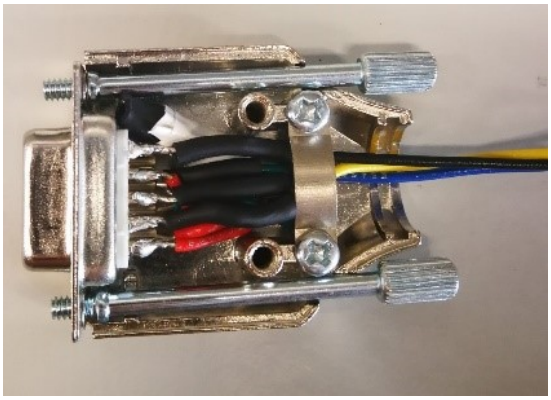


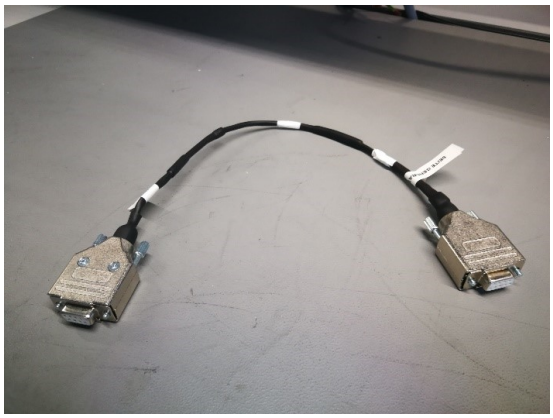
Abbildung 31: Schnittstellen-Kreuzung RS232 (Quelle: [Dokb] S. 133)

Für diesen Zweck wurde ein Adapter gebaut, der in Abbildung 32 zu sehen ist.



a) PC-Seite

b) Generator-Seite



c) gebauter Adapter

Abbildung 32: RS232-Adapter (Quelle: eigene Darstellung)

3.3.4 Kommunikationsprotokoll

Bei der Abfrage oder dem Beschreiben eines Wertes muss ein Nachrichtenblock (Aufbau: Kapitel 3.3.5) vom Computer (Host-PC) zum Generator gesendet werden. Zuerst nimmt der Generator die Überprüfung von Adresse und Prüfsumme vor. Sollte die Adresse nicht stimmen, erhält der PC keine Antwort vom Generator. Ist die Adresse korrekt, die Prüfsumme aber falsch, dann bekommt der PC ein NAK (Hexadezimal: 15). Wenn beides stimmt, wird ein ACK (Hexadezimal: 06) vom Generator ausgegeben. Nachdem das ACK gesendet wurde, wird je nach Befehl die Antwort erstellt. Diese Antwort beinhaltet im Fall einer Informationsabfrage die aufbereiteten Daten und im Fall eines Schreib-Befehls eine Quittierungsmeldung mit einem Datenbyte. Die Bedeutung der Quittierungsmeldungen kann auf Seite 149 in [Dokb] nachgelesen werden. (Quelle: [Dokb] S. 136)

Die Antwort des Generators wird nun vom PC empfangen und ausgewertet. Wie auf einen Fehler vom PC reagiert wird, hängt von dem Code oder dem PC ab. Sendet der PC ein ACK zurück, geht der Generator in den Zustand zurück, in dem er auf einen Befehl wartet. Sendet der PC ein NAK an den Generator, so wiederholt der Generator den letzten Nachrichtenblock, den er erhalten hatte. Wenn kein NAK oder ACK kommt, dann wird nach einer internen Wartezeit der Zustand erreicht, in dem der Generator auf einen neuen Befehl wartet. (Quelle: [Dokb] S. 136 bis 137)

3.3.5 Aufbau der Nachrichten

Wie die anderen Geräte hat auch der TruHeat einen speziellen Aufbau für den gesendeten Befehl und die erhaltene Antwort. Der Befehl wird wie bei der PI-Achse als Bytearray formatiert, nur dass bei TruHeat jedes Byte einzeln gesendet wird. Die Befehle bestehen grundsätzlich aus den Teilen:

1. Header,
2. Befehl,
3. optionales Längenbyte,
4. Datenbyte oder Datenbytes und,
5. Checksumme (Prüfsumme).

Die Beschreibung des Nachrichtenblocks ist in Quelle [Dokb] auf den Seiten 134 und 135 zu finden. Die genannten Teile bestehen dabei meist aus einem Byte, also 8 Bit. Der Header ist dabei so definiert, dass in diesem die Adresse und die Anzahl der Datenbytes gefunden werden können. Ein Beispiel für den Header-Aufbau wird in Kapitel 3.3.7 gezeigt. Die ersten 5 Bit des Headers sind die Adresse des Generators. Nach der Dokumentation [Dokb] Seite 134 ist diese Adresse für den Generator fest bei Eins. Die letzten drei Bits in dem Header-Byte geben die Anzahl der Datenbytes wieder. Das zweite Byte im Nachrichtenblock gibt den Befehl wieder. Alle Befehle sind in der Dokumentation [Dokb] auf den Seiten 149 bis 170 beschrieben. Die in VIFCON genutzten Befehle für den TruHeat sind in Kapitel 3.3.6 nachzulesen.

Nach den beiden Bytes kann das sogenannte optionale Längenbyte gefunden werden. Dieses wird genutzt, wenn die drei Bits im Header für die Anzahl der Datenbytes nicht ausreichen. Sobald die binäre Zahl 111 überschritten wird, muss dieses Byte in den Befehl eingefügt werden. In dem Fall bleibt die 111_2 im Header stehen. (Quelle: [Dokb] S. 135)

Die Datenbytes kommen dann danach. Wie sie gelesen bzw. gesendet werden müssen, hängt von den Befehlen ab. Im Fall von Zahlen (z.B. Messdaten, Seriennummer) ist die Reihenfolge der Bytes die folgende. Das erste Byte, welches im Befehl / in der Antwort gefunden wird, ist das Lower Byte. Dieser Umstand wird in Kapitel 3.3.7 näher aufgegriffen.

Als letztes steht im Befehl die sogenannte Prüfsumme. Bei dieser Prüfsumme wird eine XOR-Verknüpfung der Bytes (Header bis Datenbyte) durchgeführt. Diese Summe dient der Überprüfung, ob der Befehl richtig ist. Beim Eurotherm (Kapitel 3.1) gibt es auch eine Prüfsumme, die über XOR bestimmt wird. (Quelle: [Dokb] S. 135)

Die Antwort des Gerätes hat genau denselben Aufbau wie das Senden des Befehls. Die Kommunikation erfolgt nicht über das Senden von ASCII-Zeichen, sondern über eine reine Binärecodierung.

3.3.6 Genutzte Befehle

Von den Befehlen in [Dokb] auf den Seiten 149 bis 170, wurden nur bestimmte verwendet. Diese Befehl sind:

1. Schreibe Stromsollwert – Befehl 31_{16} ([Dokb] S. 155),
2. Schreibe Leistungssollwert – Befehl 32_{16} ([Dokb] S. 155),
3. Schreibe Spannungssollwert – Befehl 33_{16} ([Dokb] S. 155),
4. Lese Sollwert HF-Strom – Befehl $8D_{16}$ ([Dokb] S. 161),
5. Lese Sollwert Leistung – Befehl $8E_{16}$ ([Dokb] S. 161),
6. Lese UC-Spannung – Befehl $8F_{16}$ ([Dokb] S. 161),
7. Lese Istwert HF-Strom – Befehl $A4_{16}$ ([Dokb] S. 163),
8. Lese Istwert Leistung – Befehl $A5_{16}$ ([Dokb] S. 163),
9. Lese Istwert Kondensatorspannung – Befehl $A6_{16}$ ([Dokb] S. 164) und,
10. Lese Istwert Frequenz – Befehl $C4_{16}$ ([Dokb] S. 166).

Diese Befehle beinhalten die Abfrage der Messdaten (Soll und Ist) und das Schreiben der Sollwerte. Bei den Schreib-Befehlen und der Antwort auf einen Lese-Befehl werden zwei Datenbytes verwendet. Hierbei muss die Reihenfolge der Bytes beachtet werden. Zuerst kommt das Lower Byte und dann das Higher Byte. Was das bedeutet wird im Beispiel (Kapitel 3.3.7) näher erläutert.

Neben den genannten Befehlen werden noch weitere Befehle verwendet. Diese Befehle geben Auskünfte über das Gerät und die Limits. Diese Befehle sind:

1. Lese Softwareversion – Befehl $C6_{16}$ ([Dokb] S. 167),
2. Lese Seriennummer der Stromversorgung – Befehl $C7_{16}$ ([Dokb] S. 167),
3. Lese Netzteil-Typ – Befehl 80_{16} ([Dokb] S. 161),
4. Lese SIMIN – Befehl $D0_{16}$ ([Dokb] S. 168),
5. Lese SUMIN – Befehl $D2_{16}$ ([Dokb] S. 168),
6. Lese HF-Strom und Leistungsgrenzwert – Befehl 82_{16} ([Dokb] S. 161),
7. Lese Max Sollwert UC – Befehl CA_{16} ([Dokb] S. 167),
8. Lese Max Sollwert HF Strom – Befehl CB_{16} ([Dokb] S. 167) und,
9. Lese Aktives Interface – Befehl $9B_{16}$ ([Dokb] S. 162).

Auf den genannten Seiten werden SIMIN und SUMIN so bezeichnet. Die direkte Bedeutung der Abkürzung wird nicht im Dokument [Dokb] erwähnt. Die Bedeutung ist zu finden und sieht wie folgt aus:

1. SIMIN - minimaler Sollwert des HF-Stromes ([Dokb] S. 157)/ minimaler Ausgangsstrom-Sollwert ([Dokb] S. 92)
2. SUMIN - minimaler Sollwert der Kondensatorspannung ([Dokb] S. 157 und S. 93)

Bei den gerade gezeigten Befehlen muss etwas mehr beachtet werden. Zu beachten ist das Aussehen der Antwort dieser Abfragen. Dies kann auf den genannten Seiten gefunden werden. Besonders hierbei sind die Softwareversion, der Typ des Netzteils und der Kombi-Befehl für Strom und Leistung. Bei den ersten beiden werden ASCII-Zeichen vom Generator gesendet. Diese werden von links nach rechts gelesen und ergeben einen zusammenhängenden String. Bei den anderen Befehlen musste die Reihenfolge von Lower und Higher Byte beachtet werden. Der Kombi-Befehl besitzt zwei Messgrößen bzw. Limits in 4 Datenbytes. In der Beschreibung des Befehls wird von Byte Null bis Drei geredet. Byte Null ist nach der Beschreibung das erste Datenbyte, das von links gelesen in der Antwort steht.

Bei den Befehlen, wo ein Messwert ausgelesen wird, muss noch eine Auflösung beachtet werden, damit der Wert richtig interpretiert werden kann. Diese steht aber bei der Erläuterung der Befehle auf den genannten Seiten. Weiterhin wird dies dort mit Beispielen veranschaulicht.

Die dritte Gruppe der genutzten Befehle umfasst die Befehle zur Kontrolle des Generators. Genutzt werden die Befehle:

1. Lese RS232-User-Watchdog - Befehl 91_{16} ([Dokb] S. 162),
2. Schreibe RS232-User-Watchdog - Befehl $2D_{16}$ ([Dokb] S. 155),
3. Schreibe Einschalten - Befehl 02_{16} ([Dokb] S. 150) und,
4. Schreibe Ausschalten - Befehl 01_{16} ([Dokb] S. 150).

Hierbei ist Folgendes zu beachten:

- Das Einschalten des Generators kann nur dann geschehen, wenn keine Störmeldung anliegt. Dies kann an der Quittiermeldung 7 gesehen werden. Diese lautet: „mindestens eine Störmeldung steht (noch) an“ Zitat: [Dokb] S. 149
- Der Watchdog wird eine Störmeldung auslösen, wenn dieser triggert. Zum Watchdog selbst wird mehr im Kapitel 5.8.5 gesagt.
- Eine Störmeldung schaltet den Leistungsbetrieb des Generator sofort aus! (Quelle: [Dokb] S. 110)

3.3.7 Beispiele von Befehlen

In dem Kapitel soll ein Beispiel zu den Befehlen erbracht werden. Dabei werden ein Lese-Befehl (Kapitel 3.3.7.1) und ein Schreib-Befehl (Kapitel 3.3.7.2) gezeigt. Das Beispiel wird anhand des Sendens und des Empfangens in Python gezeigt. Näheres wird in Kapitel 3.3.8 gezeigt.

3.3.7.1 Beispiel Lesen

Befehl: `[b'\x08', b'\x8F', b'\x87']`
Antwort: `[b'\x06', b'\n', b'\x8F', b'\xB0', b'\x04', b'1']`

In dem Beispiel wurde die Abfrage nach der Sollspannung angefordert. Diese ist z.B. in Abbildung 30 zu sehen. Sie beträgt bei der Messung 1200 V. Bei der Interpretation der Antwort muss aufgepasst werden. Python hat alle Hexadezimalzahlen in ASCII umgewandelt, die es umwandeln konnte. Aus dem Grund werden das Newline (Byte 2) und auch die 1 (Byte 6) angezeigt.

Das erste Byte zeigt die Prüfung des Befehls. Die Hexadezimalzahl 6_{16} entspricht dem ACK, was bedeutet, dass der gesendete Befehl richtig erkannt wurde und der Generator im Folgenden den Befehl bearbeitet. Das folgende Byte entspricht dem Header. Der Unterschied in den beiden Headern ist die Anzahl der Datenbytes, wie es folgend gezeigt wird.

Header:

$08_{16} \mathrel{\widehat{=}} 00001000_2$ → 00001_2 = Adresse = 1_{10} und 000_2 = Anzahl Datenbytes = 0_{10}
\n $\mathrel{\widehat{=}} 0A_{16} \mathrel{\widehat{=}} 00001010_2$ → 00001_2 = Adresse = 1_{10} und 010_2 = Anzahl Datenbytes = 2_{10}

Das heißt, dass der gesendete Befehl keine Datenbytes enthält, die Antwort aber 2 beinhaltet. Das dritte Byte in der Antwort sowie das zweite Byte in dem Befehl müssen identisch sein, da diese den Befehl widerspiegeln. Der Befehl 8F steht für die Abfrage der Sollspannung ([Dokb] S. 161).

Bei der Antwort folgen nun die beiden Datenbytes. Da die Anzahl der Datenbytes kleiner als 7_{10} (111_2) ist, gibt es hier kein optionales Längenbyte. Wie es bereits erwähnt wurde, müssen diese Werte anders interpretiert werden. Das erste Datenbyte in der Antwort ist das Lower Byte und das zweite Datenbyte das Higher Byte (siehe oben). Somit ist die Bytereihenfolge der Antwort von links nach rechts gleichbedeutend mit dem niedrigsten zum höchsten Byte für die Interpretation der Datenbytes. Siehe auch:

Datenbytes: $04B0_{16} \mathrel{\widehat{=}} 1200_{10}$

Die letzten Bytes in Sendebefehl und Antwort sind die Checksumme bzw. Prüfsumme. Diese Summe wird durch die XOR-Verbindung aller vorherigen Bytes (außer Checksumme und ACK/NAK) gebildet. Bei der Überprüfung kann das Checksummen-Byte auch mit betrachtet werden. Die Antwort muss dann aber Null ergeben, wie folgend bei dem Antwort-Beispiel gezeigt wird.

Checksumme:

Befehl:	08_{16} XOR $8F_{16}$	$= 87_{16}$
Antwort:	$0A_{16}$ XOR $8F_{16}$ XOR $B0_{16}$ XOR 04_{16} XOR 31_{16}	$= 0_{16}$

Die ASCII 1, die als letztes Byte in der Antwort gefunden wird, entspricht in Hexadezimal der 31. Um den Unterschied zu der anderen Interpretation zu zeigen, wird hier noch der Befehl für die Software-Version gezeigt:

Befehl:	`[b'\x08', b'\xc6', b'\xce']`
Ant.-Start:	`[b'\x06', b'\x0f', b'\xc6', b'\x0b']`
Ant.-Daten:	`[b'0', b'1', b'.', b'2', b'1', b'.', b'0', b'0', b'.', b'0', b'0']`
Ant.-Prüfsumme:	`[b'\xee']`

An dem Beispiel können nun zwei Dinge gesehen werden. Zum einen wird das optionale Längenbyte verwendet und zum anderen müssen die Datenbytes nicht von rechts nach links gelesen werden. Der String der Softwarenummer kann in Abbildung 25 gesehen werden und bestätigt die Betrachtung somit.

Header:	$0f_{16} \mathrel{\widehat{=}} 00001111_2 \rightarrow 00001_2 = \text{Adr.} = 1_{10}$ und $111_2 = \text{Anz. Datenbytes} = 7_{10}$
op. Längenbyte:	$0b_{16} \mathrel{\widehat{=}} 00001011_2 \mathrel{\widehat{=}} 11_{10}$
Datenbytes:	01.21.00.00

3.3.7.2 Beispiel Schreiben/Senden

Befehl:	`[b'\x0a', b'\x33', b'\xe8', b'\x03', b'\xd2']`
Antwort:	`[b'\x06', b'\t', b'3', b'\x16', b',']`

Bei dem gesendeten Befehl ist dasselbe zu beachten wie bei der Antwort des Lese-Befehls. Die Datenbytes müssen bei Zahlenwerten umgedreht werden. Das heißt, dass der gesendete Wert in Hexadezimal umgewandelt werden muss und die Bytes in verkehrter Reihenfolge gesendet werden müssen. Auch dieser Umstand kann in der Befehlsbeschreibung nachgelesen werden.
Bei dem Befehl zum aktiven Interface gibt es auch einen Schreib-Befehl. Dieser erwartet z.B. eine Zahl zwischen 0 und 9 (`\x00` bis `\x09`), um das Interface freizugeben. (Quelle: [Dokb] S. 154)

Das Beispiel zeigt den Befehl zum Setzen der Sollspannung auf 1000 V. Bei der Spannung ist die Auflösung mit 1 V gegeben. Die Dezimalzahl 1000 entspricht der Hexadezimalzahl 3E8. Die Antwort auf den Befehl ist eine Quittierungsmeldung (1 Datenbyte). Bei einer Null an der Stelle wäre alles in Ordnung. Die 16_{16} entspricht einer 22_{10}. Diese 22 sagt dem Anwender, dass das Interface ungültig ist. Das bedeutet, Schreiben geht nur, wenn auch RS232 als Interface ausgewählt ist!

3.3.8 Umsetzung in Python

In Python wird für diese Schnittstelle bzw. für die Kommunikation mit dem Gerät die Bibliothek **serial** verwendet. Die Umsetzung der Definition der Schnittstelle wird hier nicht erläutert. Dies wird im Kapitel 5.3 gezeigt. In dem Kapitel sollen die besonderen Teile zum Senden, Lesen und der Interpretation erläutert werden.

Anhand der Beispiele in Kapitel 3.3.7 kann gesehen werden, wie die Daten gesendet und empfangen werden.

3.3.8.1 Funktionen für Lesen und Schreiben

Das Senden der Werte erfolgt über die *write*-Methode und eine For-Schleife. Dabei werden hintereinander alle Bytes einzeln gesendet. Diese müssen in die binäre Codierung (Bytearray) umgewandelt werden. Durch den Header (Anzahl Datenbytes) und die Checksumme ist das Ende der Daten bekannt. Folgend wird der Code-Teil für das Senden gezeigt:

```
1 def write_read_answer(self, befehl, value, res, um, Anz_DatBy, op_Anz_DatBy = ...
      '00000000'):
2     # Teil hier nicht gezeigt
3             for n in write_list:
4                 self.serial.write(bytearray.fromhex(n))
5     # Teil hier nicht gezeigt
```

Für das Auslesen wird auch wieder eine For-Schleife verwendet, sowie die Methode *read*. Bei der For-Schleife hängt die Menge der ausgelesenen Daten von der Anzahl der Datenbytes ab. Folgend ist der Code-Teil für das Lesen zu sehen:

```
1 def read_send(self, befehl, dat_Anz, res, um, dreh = True):
2     # Teil hier nicht gezeigt
3         for ans_byte in range(1,dat_Anz + 6):
4             ans = self.serial.read()
5     # Teil hier nicht gezeigt
```

In VIFCON wird das Schreiben und Lesen der in Kapitel 3.3.6 gezeigten Befehle durch die Funktionen:

1. write,
2. write_read_answer,
3. write_checksum
4. hex_schnitt,
5. byte_xor,
6. read,
7. read_send und,
8. Start_Werte

gewährleistet. Wie auch schon bei Eurotherm und der PI-Achse sind *write* und *read* die Funktionen, die von dem Controller ausgelöst werden. In *write* werden hierbei die Funktionen *write_read_answer* und *Start_Werte* aufgerufen. Die letzte ist dabei genauso wie bei den anderen

Geräten verwendet worden. Da das Senden der Befehle und das Empfangen der Daten hier etwas spezieller sind, wird der ungefähre Ablauf des Lesens und Schreibens in den Abbildungen 33 (Schreiben) und 34 (Lesen) gezeigt. An den Abbildungen kann nun auch gesehen werden, dass die Funktionen *write_read_answer* und *read_send* die Kommunikation mit dem TruHeat durchführen.

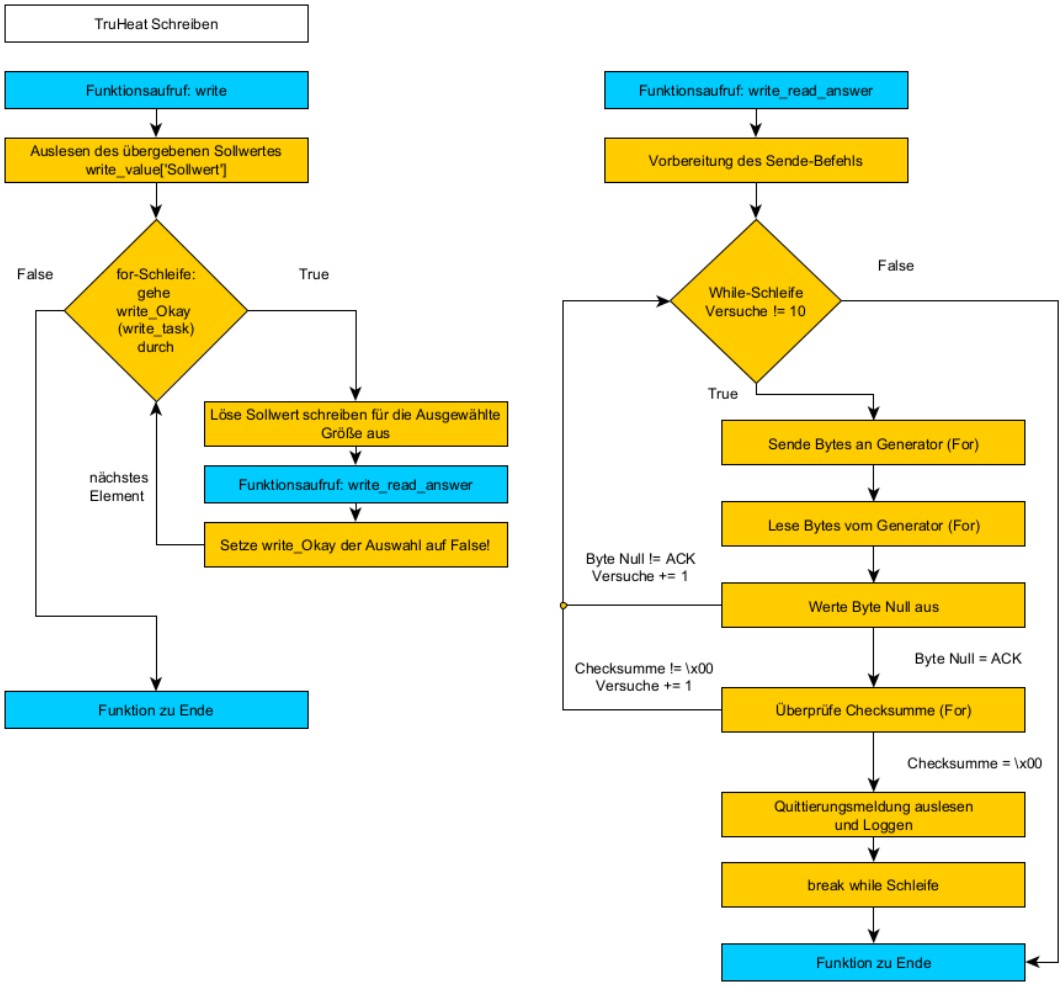


Abbildung 33: TruHeat Sende Wert - Programmablauf (Quelle: eigene Darstellung)

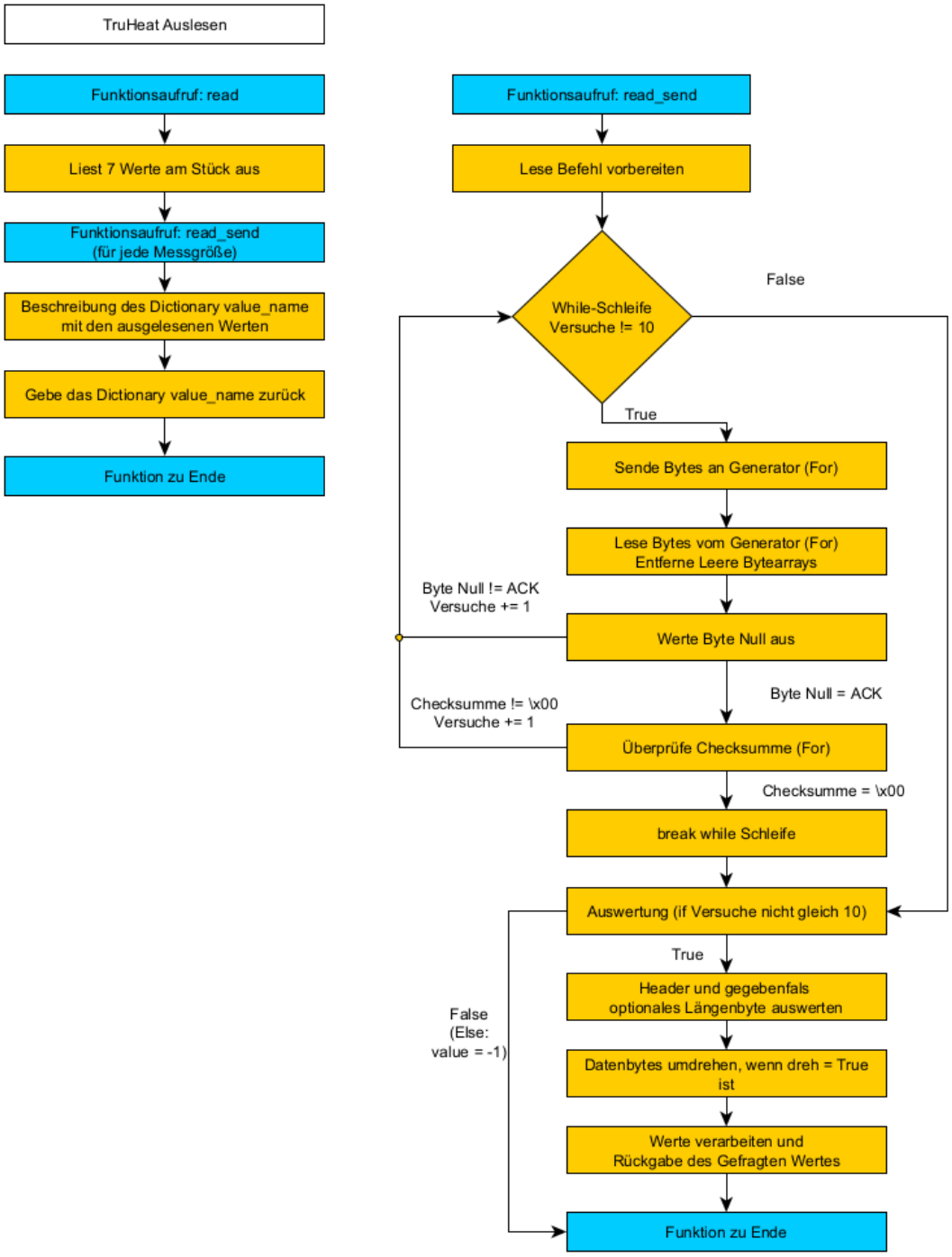


Abbildung 34: TruHeat Lese Wert - Programmablauf (Quelle: eigene Darstellung)

Die Funktionen *hex_ schnitt*, *write_ checksum* und *byte_ xor* werden in den beiden Abbildungen 34 und 33 nicht gezeigt, da diese für die Vorbereitung der Bytes, der Bestimmung der Checksumme gebraucht und mehrfach aufgerufen werden. Diese werden folgend kurz erläutert.

In *hex_ schnitt* werden die Hexadezimal-Zahlen mit dem Präfix 0x bearbeitet. Dieser Präfix muss entfernt werden. Weiterhin werden die Hexadezimal-Zahlen, die nur aus einem halben Byte bestehen (0 bis F) um eine Null erweitert (00 bis 0F). Die Methode *fromhex* erwartet diese Änderungen.

Die beiden Funktionen *write_ checksum* und *byte_ xor* dienen der Bestimmung bzw. Überprüfung der Checksumme. Der Unterschied zwischen den beiden Funktionen ist der Typ der Variablen. Die Funktion *write_ checksum* bestimmt anhand von Integer-Werten (ganzen Zahlen) die Checksumme, die an den Generator gesendet werden muss. Bei *byte_ xor* wird die Checksumme der Antwort aus den Bytearrays berechnet. Beide Fälle werden in For-Schleifen aufgerufen. Das XOR wird in Python mit „^ “ ausgeführt. Die Funktion *byte_ xor* stammt aus der Quelle [Xor] und wurde nur um Kommentare erweitert. Folgend wird *write_ checksum* gezeigt:

```
1 def write_checksum(self, liste):
2     cs_alt = 0
3     for n in liste:
4         cs = int(n,16) ^ cs_alt
5         cs_alt = cs
6     cs = hex(cs)
7     return cs
```

Die Funktion *read_ send* wird auch aus der Funktion *Start_ Werte* aufgerufen, wodurch ein paar If-Anweisungen in der Datenverarbeitung entstanden sind. In Kapitel 3.3.7 und 3.3.5 wurde bereits erläutert und gezeigt, dass die Datenbytes nicht immer gedreht werden müssen. Genau dies wird durch die boolesche Variable *dreh* entschieden. Wird diese auf False gesetzt, dann werden die Datenbytes von links nach rechts verbunden. Weiterhin erhält der Befehl 82_{16} eine Sonderbehandlung, weil dort nur der Leistungswert ausgelesen werden muss, da der Stromwert einen separaten Befehl hat.

3.3.8.2 Datenumwandlung

Als letztes muss nun noch die Umwandlung der Werte gezeigt werden. Insgesamt benötigt es die Umwandlungen:

- Hexadezimal zu Dezimal (Funktion: `int`),
- Binär zu Hexadezimal (Funktion: `hex`),
- Bytearray zu Hexadezimal (Funktion: `hex`),
- Dezimal zu Binär (Funktion: `bin`) und,
- Hexadezimal zu Bytearray (Methode: `bytearray.fromhex`).

Hier ein Beispiel für alle Umwandlungen:

```
1     1. int('A8',16)              ans: 168
2     2. hex(int('00001010',2))    ans: 0xa
3     3. b'\x08'.hex()             ans: 08
4     4. bin(int('0A',16))         ans: 0b1010
5     5. bytearray.fromhex('06')   ans: b'\x06'
6        bytearray.fromhex('6')    ans: ValueError: non-hexadecimal number found ...
             in fromhex() arg at position 1
```

Bei der *int*-Funktion werden Strings oder Zahlen in Integer umgewandelt. Dabei muss bei der Funktion, bei Zahlenwerten wie Binär (2) oder Hexadezimal (16) eine Base mit angegeben werden.

Die *hex*-Funktion wandelt einen Integer in einen Hexadezimal String mit dem Präfix 0x. Bei der Nutzung mit den Bytearrays (Präfix \x) wird der Präfix einfach nur entfernt.

Die *bin*-Funktion tut dasselbe wie die *hex*-Funktion. Die Funktion erwartet einen Integer und gibt einen String in Binär zurück. Hierbei wird der Präfix 0b angefügt.

Die Methode *fromhex* des *bytearray*-Objekts erreicht die Umwandlung von einem Hexadezimal-String in ein Bytearray. Dabei darf der String keinen Präfix haben und nicht nur aus einem halben Byte (siehe Beispiele) bestehen.

3.4 Nemo-1-Anlage

Die Nemo-1-Anlage (andere Bezeichnung: Test-CZ) ist eine der Anlagen, die bereits in Kapitel 2.1 gezeigt wurden. Die Anlage steht in dem Labor der Gruppe Modellexperimente und wird für verschiedene Experimente genutzt. Insbesondere wird die Anlage für die Züchtung von Kristallen mit dem CZ-Verfahren verwendet. Neben dieser Anwendung wurden mit dieser Anlage in der Bachelorarbeit [Fun22] Magnetfeldmessungen an einem induktiven Heizer und Emissivitätsmessungen an verschiedenen Materialien durchgeführt. Die Anlage ist in Abbildung 35 und Abbildung 2 zu sehen.

a) Anlage **b)** Schaltschrank

Abbildung 35: Nemo-1-Anlage (Quelle: eigene Darstellung)

Durch diese Anlage erhält VIFCON neue Geräte für die Gerätetypen Antriebe und Monitoring. Bei den Antrieben kommen nun eine Achse für Hub (Lineare) und für Rotations-Bewegungen hinzu. Neben diesen werden die verschiedenen Punmpenwerte (Durchfluss, Druck, Drehzahl) in dem Tab Monitoring angezeigt. Anders als bei dem Tab Steuerung soll der Monitoring-Tab nur eine Anzeige der Werte ermöglichen.

3.4.1 Funktionsweise und Aufbau

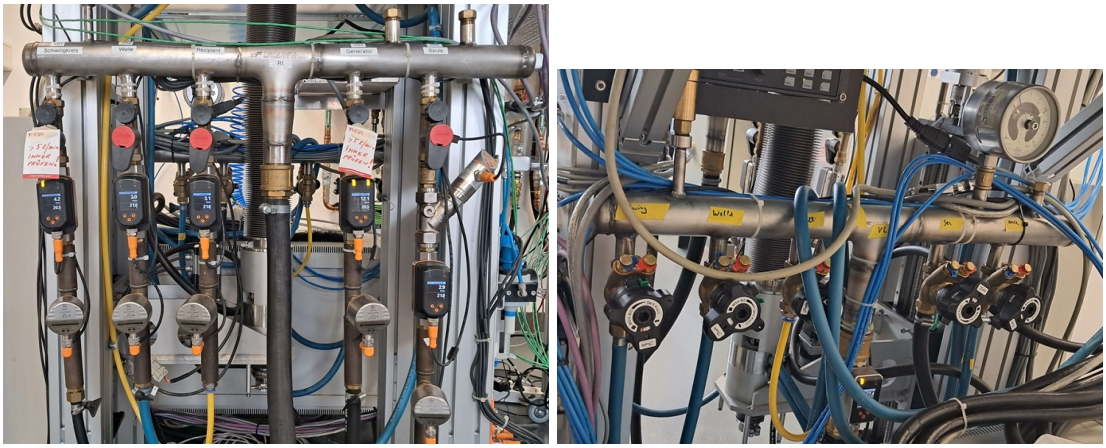
Die Anlage wird über den Schaltschrank (Abbildung 35b) gesteuert. Dabei gibt es ein Bedienpanel, das in Kapitel 2.1.3 näher erläutert wurde. Die angeschlossenen Geräte in der Anlage lassen sich so bearbeiten. So können in dem Bedienfeld die Werte für die Pumpen eingesehen werden, sowie auch die Antriebe für Hub und Rotation gestartet werden.

In dem Schaltschrank befindet sich eine SPS (Simatic S7-300) , die in Abbildung 36 zu sehen ist. Über diese wird die Ansteuerung der Antriebe und von vielen anderen Anlagenteilen ermöglicht. Alles was in den Bedienfeld-Seiten zu sehen ist, wird mit der SPS gesteuert. Neben diesem Umstand wird auch die Kommunikation mit einem Host-Rechner über Modbus-TCP ermöglicht.

Abbildung 36: Siemens SPS der Nemo-1-Anlage (Quelle: eigene Darstellung)

Somit können verschiedene Geräte angeschlossenen werden. Entweder werden diese über den Schaltschrank oder nebenstehend betrieben. Beim letzteren sind Geräte wie Eurotherm oder auch Messgeräte gemeint. Über den Eurotherm kann, wie es in Kapitel 3.1 erwähnt wurde, auch ein Generator (z.B. Leistungseinheit) gesteuert werden. Auch die Wahl des Heizers kann von Experiment zu Experiment unterschiedlich sein. Je nach Heizer (Induktiv, Resistiv) muss auch die Versorgung und die Kühlung geändert werden.

In dem Aufbau der Anlage gibt es auch eine Kühlung. Diese Messgeräte sind in Abbildung 37 zu sehen. Die Kühlung wird für den Schwingkreis, die Tiegelwelle, den Rezipient (Züchtungsraum, Vakuumkammer), den Generator und die Spule genutzt. In den Abbildungen 37a und 37b sind diese Bezeichnungen von links nach rechts den Anschlüssen der Kühlung zuzuordnen.

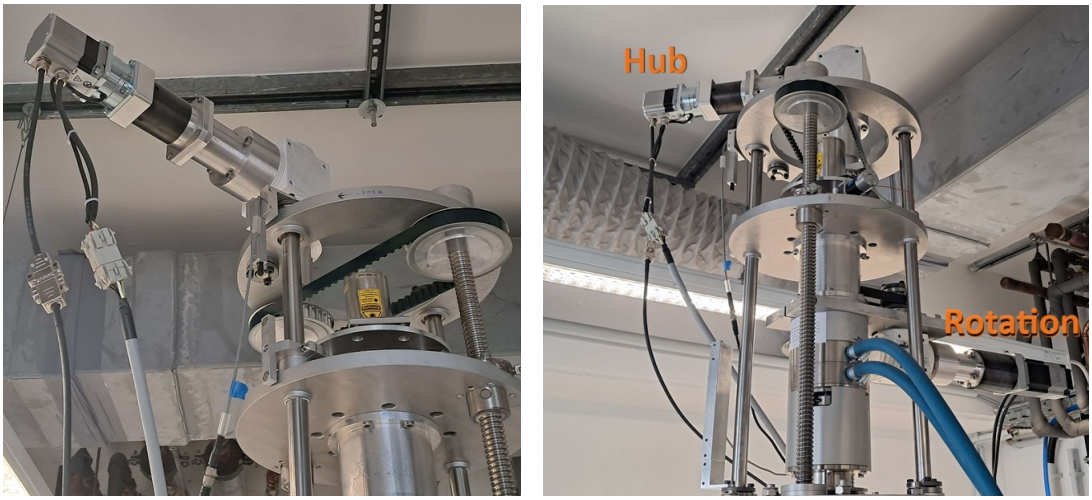


a) Rücklauf **b)** Vorlauf

Abbildung 37: Kühlung der Nemo-1-Anlage (Quelle: eigene Darstellung)

3.4.2 Spezifikation für Hardware

In VIFCON werden die Antriebe und die Pumpen eingearbeitet, die direkt an der Anlage eingebaut sind. Bei den Achsen gibt es für die Spindel und für den Tiegel jeweils zwei Antriebe. Im Sinne der Kristallzüchtung kommt an die Spindel der Impfkristall und beim Tiegel das Rohmaterial. Dabei können die Antriebe entweder für Rotation oder Hub genutzt werden, wodurch bestimmte Eigenschaften am Kristall geändert werden können. Zum Beispiel kann durch die Ziehgeschwindigkeit (Hub-Bewegung) der Durchmesser des Kristalls beeinflusst werden.



a) Hub-Antrieb **b)** Hub- und Rotations-Antrieb

Abbildung 38: Nemo-1-Anlage - Spindel Antriebe (Quelle: eigene Darstellung)

In Abbildung 38 können die beiden Antriebe für die Spindel gesehen werden. Die abgebildeten Antriebe sind auf der Anlage, die in Abbildung 35a zu sehen ist. Einerseits gibt es einen Antrieb für die vertikale Bewegung, den Hub, und andererseits einen Antrieb für die Rotation der Spindel. Die Abbildung 39 zeigt die beiden Antriebe für den Tiegel. Diese Antriebe befinden sich unter der Anlage zwischen den Kühlungsleitungen (Abbildung 37). Auch hier sind diese für Hub und Rotation da. Bei dem Hub-Antrieb für die Spindel, Abbildung 38a, wird ein Keilriemen bewegt, der über ein Gewinde die Spindel auf und ab bewegen lässt. In Abbildung 39b kann der Teil des Tiegel-Hub-Antriebes gesehen werden, der die Bewegung auslöst. Hier wird eine Gewindestange gedreht, wodurch sich der Tiegel dann auf und ab bewegen kann. Auch die Tiegel-Rotation besitzt einen Keilriemen.

Die Erklärungen zu einigen Teilen (Antriebe und Pumpen) der Anlage stammen vom IKZ-Mitarbeiter Adrian Wagner [Pera]. In der Nemo-1-Anlage sind wie erwähnt 4 Antriebe verbaut. Die Abbildung 38a zeigt dabei die Anschlüsse an dem Motor. Insgesamt hat jeder Antrieb drei Anschlüsse an zwei Schnittstellen. Diese drei Anschlüsse sind für den Encoder, den Schrittmotor und die Bremse. Der Encoder hängt an der silbernen (metallischen) Schnittstelle und misst die Umdrehungszahl des Schrittmotors. Dieser Wert wird dann für die SPS umgewandelt und bereitgestellt. Der andere Anschluss (grau/weiß) ist die Spannungsversorgung und die Bremse. Weiterhin wird hierüber die Kommunikation mit dem Antrieb ermöglicht. Der Frequenzumwandler wertet den Encoder aus und ist mit Profibus (Bus-System) verbunden.

a) Hub-Antrieb **b)** Hub- und Rotations-Antrieb

Abbildung 39: Nemo-1-Anlage - Tiegel Antriebe (Quelle: eigene Darstellung)

Neben den Antrieben werden auch Messgeräte für die Atmosphäre benötigt. Bei der Atmosphäre wird der Rezipient mit Gasen oder einem Vakuum gefüllt. Für diesen Zweck benötigt es Pumpen. In der Anlage können die Gase Stickstoff (N2) und Argon (Ag) verwendet werden. Die Pumpen, Ventile und dazugehörige Messgeräte werden in Abbildung 40 gezeigt. Auch die verschiedenen Messgeräte für die Messung von Druck und Durchfluss sind in dieser Abbildung zu sehen. In der Abbildung 40 wird in der Mitte die GUI des Schaltschrankes für die Gase aus Abbildung 4 gezeigt. Die Abbildung zeigt somit auch eine Verbindung zur Anlagen-GUI und der wahren Anlage. Die MFC (Mass Flow Controller) sind dabei über Profibus mit dem Schaltschrank (Abbildung 35b) verbunden. Diese Geräte werden in Abbildung 40 in deren realen Reihenfolge gezeigt. Oben ist das MFC27 und ganz unten der MFC25. Spezifische Daten zu den gezeigten Geräten (MFC, Ventile, Pumpen, Antriebe und SPS) werden in den Tabellen 10 bis 13 gezeigt. Die Werte wurden dabei von den verschiedenen Geräten (Abbildung 36, 38, 39, 40) bzw. deren Typenschilder und Beschriftungen abgelesen. Neben diesen wurden auch Werte eingetragen, die von Adrian Wagner [Pera] stammen.

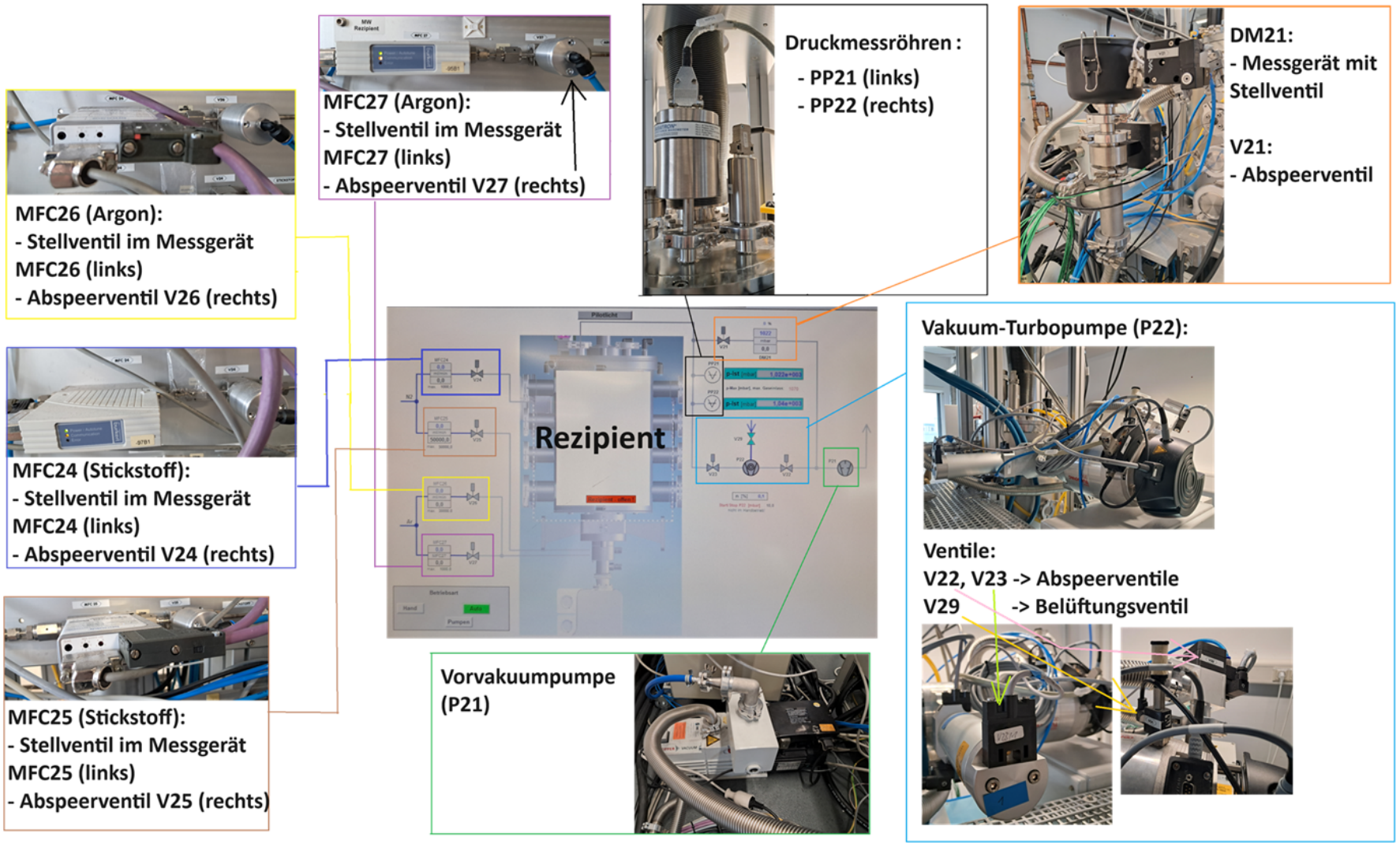


Abbildung 40: Nemo-Gase - Pumpen und Gaseinfuhr (Quelle: eigene Darstellung)

Tabelle 10: Nemo-1-Anlage - Geräte-Komponenten Beschreibung 1 (Antriebe) (Quelle: eigene Darstellung)

Geräte-Bezeichnung	**Teil des Gerätes**	**Hersteller**	**Modell-Nummer**	**Weitere Werte**
Spindel-Antrieb-Hub	Servomotor (Quelle: [Herb]) (2-phase 50 Pole-Pair Hybrid Servo Motor)	Jenaer Antriebstechnik GmbH	23S16-0560-8B5P7-PLE60-521-15‘-55 (GAXXX00232125098)	P_{cont} = 57 W | n_{cont} = 1500 rpm I_{cont} = 4,9 A | T_{cont} = 0,36 Nm U_{drv} = 60 V DC link n_{max} = 3000 rpm I_{stall} = 5,7 A | T_{hold} = 1,5 Nm Holding brake: 24 V DC IP40
	Kegelradgetriebe (Quelle:[Hera])	Mädler	41200404 Serien-Nr: 07210117	/
Spindel-Antrieb-Rotation	Servomotoren (Quelle: [Herb])	Jenaer Antriebstechnik GmbH	23S31-0650-807W7-MP060-60-15-55(30) (GAXXX00112722358)	U_{max} ≤ 90 V DC
	Schrittmotor	MOONS’	Moons Stepping Motor Type 23HS3414 60073724 11/03/07	/
	Kegelradgetriebe (Quelle: [Hera])	Mädler	PN: 41200100 SN: 06150842	/
Tiegel-Antrieb-Hub	Servomotor (Quelle: [Herb])	Jenaer Antriebstechnik GmbH	23S31-0650-805P7-MP060-60-15‘-55(30) (GAXXX00091079773)	U_{max} ≤ 90 V DC
Tiegel-Antrieb-Rotation	Servomotor (Quelle: [Herb])	Jenaer Antriebstechnik GmbH	23S31-0650-807W7-55 (30) (GAXXX00090869154)	/

Tabelle 11: Nemo-1-Anlage - Geräte-Komponenten Beschreibung 2 - SPS, MFC-Geräte (Werte Durchfluss von Adrian Wagner [Pera]) (Quelle: eigene Darstellung)

Geräte-Bezeichnung	Teil des Gerätes	Hersteller	Modell-Nummer	Weitere Werte
SPS	SPS (CPU, Baugruppen)	Siemens	Simatic S7-300 (317-2EK14-0AB0)	CPU: CPU317-2 PN/DP Baugruppen (Module): 2x 321-1BH02-0AA0 2x 322-1BH01-0AA0 1x 334-0CE01-0AA0 1x 355-0VH10-0AE0 (FM355 C PID Control)
MFC24 (Stellventil, Messgerät)	8711 MFC N2 1000 SCCM	bürkert	S/N 1001 00209816	24 V DC OP Durchfluss (Stickstoff): bis max. 1000 ml/min
MFC25 (Stellventil, Messgerät)	GF Series Thermal Mass Flow	BROOKS® INSTRUMENT	GF040CXXC-SA50050L-VXVVP4-XXXXAX-00C S/N: M3207301001 Version: 4.30	Gas: 13-N2 Range: 50000 sccm Ref: 0°C/760 Torr Response: 300 msec Valve: Normally Closed Durchfluss (Stickstoff): bis max. 50000 ml/min
MFC26 (Stellventil, Messgerät)	GF Series Thermal Mass Flow	BROOKS® INSTRUMENT	GF040CXXC-SA48030L-T6VVP5-XXXXAX-00C S/N: M3164501001 Version: 4.30	Gas: 13-N2 Range: 30000 sccm Ref: 0°C/760 Torr Response: 1 Seconds Valve: Normally Closed Durchfluss (Argon): bis max. 30000 ml/min
MFC27 (Stellventil, Messgerät)	8711 MFC Argon 1000 SCCM	bürkert	S/N 1001 00209805	24 V DC OP Durchfluss (Argon): bis max. 1000 ml/min

Tabelle 12: Nemo-1-Anlage - Geräte-Komponenten Beschreibung 3 - Ventile (Quelle: eigene Darstellung)

Geräte-Bezeichnung	Teil des Gerätes	Hersteller	Modell-Nummer	Weitere Werte
DM21 (Stellventil, Messgerät)	Ventil	MKS Instruments Deutschland GmbH	Serial#: 000214427 Model#: 253B-1-40-1	/
	Dust separator SAS 25	Pfeiffer Vacuum	Mod.-Nr.: PK Z60 508	/
V21 – Absperrventil	Ventil	VAT	Fabrication No.: 26428-KA41-0001/3354	/
V22 – Absperrventil	Ventil	VAT	Fabrication No.: 12136-PA44-0001/1391 A-1865735	/
V23 – Absperrventil	Ventil	VAT	Fabrication No.: 12136-PA44-0001/1391 A-1865735	/
V24 – Absperrventil	/	/	/	/
V25 – Absperrventil	/	/	/	/
V26 – Absperrventil	/	/	/	/
V27 – Absperrventil	/	/	/	/
V29 – Belüftungsventil	/	/	/	/

Tabelle 13: Nemo-1-Anlage - Geräte-Komponenten Beschreibung 4 - Pumpen (Werte Messbereich von Adrian Wagner [Pera]) (Quelle: eigene Darstellung)

Geräte-Bezeichnung	Teil des Gerätes	Hersteller	Modell-Nummer	Weitere Werte
P21 – Vorvakuumpumpe	Pumpe	Pfeiffer Vacuum	/	/
	Motor	Leroy-Some	LS80PR/T	/
PP21 – Druckmessröhre	Baratron® Capacitance Manometer	MKS Instruments Deutschland GmbH	SN: G213894G10 Model: 627BX13MDC1B	Power Input +-15 V DC Output 0-10 V DC 1000 mbar Messbereich: 5 mbar bis ca. atmosphärischer Druck
PP22 – Druckmessröhre	974B QuadMag Transducer	MKS Instruments Deutschland GmbH	P/N: 974B-21030 S/N: 974A000524	Messbereich: 10^{-9} mbar bis ca. atmosphärischer Druck
P22 - Vakuum-Turbopumpe	leybold TURBOVAC 350 i	Oerlikon Leybold Vacuum	/	/
	TURBO. POWER integra	Oerlikon Leybold Vacuum	800100V0003	IN: 100-240 V AC 50/60 Hz 403 VA OUT: 24 V DC +/- 5 % 10 A 288 W

3.4.3 Spezifikation für Schnittstellen

Bei der Nemo-1-Anlage wird die gesamte Kommunikation durch Python geregelt. Hierbei müssen lediglich Spezifikationen für die Nutzung des Kommunikationsprotokolls getroffen werden, die im folgenden Kapitel 3.4.4 gezeigt werden. Im Sinne der Schnittstelle wird lediglich eine Verbindung über eine Ethernet-Schnittstelle von Computer zur SPS benötigt. Ob diese nun direkt oder über einen Switch verbunden werden, kommt auf den Aufbau an.

3.4.4 Kommunikationsprotokoll

Anders als bei den RS232-Schnittstellen müssen bei dem Modbus-Protokoll nur eine Server-Adresse und ein Port definiert werden. In dem Fall ist die IP-Adresse des Servers (SPS) 192.168.116.99 und der anzusprechende TCP-Port ist 502. Neben diesen Werten gibt es nur noch die sogenannten Register und Coils. In diesen Bereichen der SPS werden die Werte eingetragen und ausgelesen. Die Definition und Belegung wird durch den Programmierer des SPS-Programms (in dem Fall AUTEAM) festgelegt. Die Input-Register, Holding-Register und Coils werden in Kapitel 3.4.6 in den Tabellen 15 bis 16 vorgestellt. Über Modbus an sich wurde in Kapitel 2.3.2 Näheres erläutert.

3.4.5 Aufbau der Nachrichten

Der Aufbau der Nachrichten wurde in Kapitel 2.3.2.2 in Abbildung 16 gezeigt. Kurzgefasst ist der Aufbau des Befehls abhängig von dem Funktionscode, der das Lesen oder Schreibens eines der 4 Objekte (Coil, Discrete Input, Input-Register, Holding-Register) regelt. Dasselbe gilt für die Antwort. Das Besondere bei der Umsetzung der Geräte der Nemo-1-Anlage ist, dass die genutzte Python-Bibliothek **pyModbusTCP** einen Großteil der Kommunikation übernimmt. Der Programmierer muss nur noch die richtige Schnittstellen-Spezifikation (Kapitel 3.4.3) treffen, die richtigen Register setzen und die Werte richtig formatieren. Selbst bei der Formatierung hilft die Bibliothek sehr weiter. Der Aufbau der Nachricht kann durch das Setzen eines Default-Wertes in der Konsole angezeigt werden. In Kapitel 3.4.7 werden diese Ausgaben gezeigt und kurz erklärt.

3.4.6 Genutzte Befehle

Anders als bei den Geräten aus den Kapiteln 3.1 bis 3.3 gibt es hier keine direkten Befehle. Da hier Modbus verwendet wird, werden Register und Adressen beschrieben, die dann von der SPS ausgelesen und bearbeitet werden. In Kapitel 3.4.1 wurde die SPS der Anlage in Abbildung 36 gezeigt.

Wie in den Kapiteln 2.3.2.1 und 3.4.4 erläutert existieren 4 Objekte in Modbus. Diese 4 Objekte sind in Tabelle 2 zu finden. Insgesamt werden nur 3 der 4 Objekten benutzt. Die Objekte Coil und Holding-Register beinhalten die von VIFCON ausgelösten Aktionen und gesendeten Werte. In den Coils werden die Bewegungsaktionen für die Achsen gespeichert und von der SPS ausgeführt. Die Soll-Geschwindigkeiten für die Achsen werden in den Holding-Registern gespeichert.

Für das Auslesen werden die Input-Register verwendet. In diese schreibt die SPS die Werte für Ist-Geschwindigkeit und Position der Achsen, sowie die verschiedenen Werte der Pumpen. Um Änderungen der Sollgeschwindigkeit an der Anlage mit VIFCON zu bemerken, wird die Sollgeschwindigkeit für Hub und Rotation auch in den Input-Registern zu finden sein. Somit kann der Wert durch VIFCON oder an der Anlage angepasst werden und beide Teilnehmer bemerken die Änderung.

Das Programm und die Register-Belegung für die Modbus-Kommunikation wurde von der Firma AUTEAM übernommen, welche auch die Erklärung (speziell Bernd Eberhardt [Perb]) dieser Werte lieferte. Die Tabellen 14 bis 16 zeigen die Register-Adressen für die drei genutzten Modbus-Objekte. In diesen Tabellen finden sich in der ersten Spalte die Register-Modbus-Adressen. Wie in Kapitel 2.3.2 gezeigt sind die Register 16 Bit groß. Diese 16 Bit entsprechen 2 Bytes, also zweimal 8 Bit. Die Coils wiederum besitzen nur ein Bit, wodurch nur True oder False, also boolesche Werte gegeben werden können. In jedem Objekt wird wieder bei Null begonnen. Die Anzahl der Bytes für einen Wert kann der Spalte 3 entnommen werden. Auch hier werden die Bytes von Null an hochgezählt. Z.B. stehen Register-Modbus-Adresse 2 (Tabelle 14) 4 Bytes zur Verfügung (Byte-Adresse ist 4, folgende ist 8, Differenz sind 4 Bytes). Neben diesen Fakten können in den Tabellen die Namen der Variablen in der SPS und die jeweiligen Datentypen abgelesen werden. Insgesamt werden in den Tabellen drei Adressen genannt. Die SPS-Adresse ist etwas Internes, während die Register-Modbus-Adressen durch VIFCON und somit Python genutzt werden. Der Modbus arbeitet Word-orientiert (1 Word = 2 Byte = 16 Bit = 1 Register), während die SPS Byte-orientiert arbeitet.

Die Tabelle 14 zeigt die genutzten Input-Register. Diese Register werden von der SPS beschrieben, im Bedienfeld angezeigt und von VIFCON ausgelesen. Die ersten beiden Register bilden dabei extra Funktionen für den Test der Kommunikation (Zähler und Reserve). Danach folgen drei Gruppen. Diese sind Daten der Pumpen, Daten der Tiegel-Antriebe und Daten der Spindel-Antriebe. In diesen Daten findet sich viermal die Bezeichnung „Stat“ wieder, was für Status steht. Für diese Statusmeldung finden sich Informationen in den Tabellen 17 und 18. Bei diesen Daten müssen die einzelnen Bits gedeutet werden.

Tabelle 14: Nemo-1 Modbus-Objekte - Informationen zu den Input-Registern (Quelle: In Anlehnung an [Perb])

Register Modbus Adr.	**SPS-Adr.**	**Byte-Adr.**	**Variablen-Name SPS**	**Datentyp**
0	DB402.DBW	0	MbInpR.Cnt	DEZ
1	DB402.DBW	2	MbInpR._res	HEX
2	DB402.DBD	4	MbInpR.MFC24_Flow	GLEITPUNKT
4	DB402.DBD	8	MbInpR.MFC25_Flow	GLEITPUNKT
6	DB402.DBD	12	MbInpR.MFC26_Flow	GLEITPUNKT
8	DB402.DBD	16	MbInpR.MFC27_Flow	GLEITPUNKT
10	DB402.DBD	20	MbInpR.DM21_Druck	GLEITPUNKT
12	DB402.DBD	24	MbInpR.PP21_Druck	GLEITPUNKT
14	DB402.DBD	28	MbInpR.PP22_Druck	GLEITPUNKT
16	DB402.DBW	32	MbInpR.PP21_Stat	HEX
17	DB402.DBW	34	MbInpR.PP22_Stat	HEX
18	DB402.DBD	36	MbInpR.PP22_Ist	GLEITPUNKT
20	DB402.DBD	40	MbInpR.Tiegel.Hub.vIst	GLEITPUNKT
22	DB402.DBD	44	MbInpR.Tiegel.Hub.vSoll	GLEITPUNKT
24	DB402.DBD	48	MbInpR.Tiegel.Hub.posIst	GLEITPUNKT
26	DB402.DBD	52	MbInpR.Tiegel.Hub.posSoll	GLEITPUNKT
28	DB402.DBD	56	MbInpR.Tiegel.Hub.posMax	GLEITPUNKT
30	DB402.DBD	60	MbInpR.Tiegel.Hub.posMin	GLEITPUNKT
32	DB402.DBW	64	MbInpR.Tiegel.Hub.Stat	WORD
33	DB402.DBD	66	MbInpR.Tiegel.Rot.vIst	GLEITPUNKT
35	DB402.DBD	70	MbInpR.Tiegel.Rot.vSoll	GLEITPUNKT
37	DB402.DBW	74	MbInpR.Tiegel.Rot.Stat	WORD
38	DB402.DBD	76	MbInpR.Spindel.Hub.vIst	GLEITPUNKT
40	DB402.DBD	80	MbInpR.Spindel.Hub.vSoll	GLEITPUNKT
42	DB402.DBD	84	MbInpR.Spindel.Hub.posIst	GLEITPUNKT
44	DB402.DBD	88	MbInpR.Spindel.Hub.posSoll	GLEITPUNKT
46	DB402.DBD	92	MbInpR.Spindel.Hub.posMax	GLEITPUNKT
48	DB402.DBD	96	MbInpR.Spindel.Hub.posMin	GLEITPUNKT
50	DB402.DBW	100	MbInpR.Spindel.Hub.Stat	WORD
51	DB402.DBD	102	MbInpR.Spindel.Rot.vIst	GLEITPUNKT
53	DB402.DBD	106	MbInpR.Spindel.Rot.vSoll	GLEITPUNKT
55	DB402.DBW	110	MbInpR.Spindel.Rot.Stat	WORD

Die Tabelle 15 zeigt die Holding-Register. Diese Register beinhalten die geschriebenen Werte der Geschwindigkeiten der 4 Achsen der Nemo-1-Anlage. Diese Register werden durch das Bedienfeld und VIFCON beschrieben. Weiterhin liest die SPS diese Register aus. Die SPS leitet die Werte an die Antriebe weiter und lässt diese im Bedienfeld anzeigen. Da der Sollwert für die Geschwindigkeit im Input-Register ausgelesen wird, können diese Werte an zwei Stellen geändert werden.

Tabelle 15: Nemo-1 Modbus-Objekte - Informationen zu den Holding-Registern (Quelle: In Anlehnung an [Perb])

Register Modbus Adr.	**SPS-Adr.**	**Byte-Adr.**	**Variablen-Name SPS**	**Datentyp**
0	DB401.DBD	0	MbHoldR.Tiegel.Hub.vSoll	GLEITPUNKT
2	DB401.DBD	4	MbHoldR.Tiegel.Rot.vSoll	GLEITPUNKT
4	DB401.DBD	8	MbHoldR.Spindel.Hub.vSoll	GLEITPUNKT
6	DB401.DBD	12	MbHoldR.Spindel.Rot.vSoll	GLEITPUNKT

In Tabelle 16 finden sich die Coils wieder. Diese werden für die Aktionen der Antriebe benötigt. Da diese nur 1 Bit groß sind, wird durch die Betätigung des Knopfes die jeweilige Coil-Adresse auf True gesetzt. Durch eine interne Programmierung an der SPS wird dieser Wert nach Ausführung zurückgesetzt. Somit ist dies in VIFCON nicht notwendig. Da es bei Nemo-1 zwei Positionen (unten Tiegel, oben Spindel) gibt, ist die Tabelle 16 in zwei Kategorien unterteilt.

Tabelle 16: Nemo-1 Modbus-Objekte - Informationen zu den Coils (Quelle: In Anlehnung an [Perb])

Register Modbus Adr.	**SPS-Adr.**	**Byte-. Bit-Adr.**	**Variablen-Name SPS**	**Datentyp**
0	DB403.DBX	0.0	MbHoldR.Tiegel.cStopH	BOOL
1	DB403.DBX	0.1	MbHoldR.Tiegel.cAuf	BOOL
2	DB403.DBX	0.2	MbHoldR.Tiegel.cAb	BOOL
3	DB403.DBX	0.3	MbHoldR.Tiegel.cStopR	BOOL
4	DB403.DBX	0.4	MbHoldR.Tiegel.cRe	BOOL
5	DB403.DBX	0.5	MbHoldR.Tiegel.cLi	BOOL
16	DB403.DBX	2.0	MbHoldR.Spindel.cStopH	BOOL
17	DB403.DBX	2.1	MbHoldR.Spindel.cAuf	BOOL
18	DB403.DBX	2.2	MbHoldR.Spindel.cAb	BOOL
19	DB403.DBX	2.3	MbHoldR.Spindel.cStopR	BOOL
20	DB403.DBX	2.4	MbHoldR.Spindel.cRe	BOOL
21	DB403.DBX	2.5	MbHoldR.Spindel.cLi	BOOL

Die Tabellen 17 und 18 zeigen wie bereits erwähnt die Bedeutung der Status-Wörter. In der SPS entspricht der Datentyp Word 2 Bytes, also 16 Bit. Bei Tabelle 17 sind die Bits 9 bis 12 nur für die Pumpe mit der Bezeichnung P22 (Turbopumpe) definiert und können nur dort als diese gedeutet werden. Etwas Ähnliches gilt bei Tabelle 18. Der Status der Endlagen macht nur bei den Hub-Antrieben Sinn.

Tabelle 17: Nemo-1 Modbus-Objekte - Statusbits Pumpen (Quelle: [Perb])

Bit-Nr.	**Bit-Funktion im Status-Wort**	
0	Hand	Handbetrieb
1	Auto	Automatikbetrieb
8	Run	Lauf
9	Steht	steht (nur P22)
10	Norm	Normalbetrieb (nur P22)
11	Warn	Warnung (nur P22)
12	Err	Error (nur P22)

Tabelle 18: Nemo-1 Modbus-Objekte - Statusbits Antriebe (Quelle: [Perb])

Bit-Nr.	**Bit-Funktion im Status-Wort**	
0	sReady	Betriebsbereit
1	sRef	Achse referiert
2	sErr	Achse Fehler
3	sRun	Antrieb läuft
4	sLAuf	Antrieb läuft aufwärts
5	sLAb	Antrieb läuft abwärts
6	sObn	Achse Position oben
7	sUtn	Achse Position unten
8	sEndObn	Achse Endlage oben
9	sEndUtn	Achse Endlage unten
10	sSwEndOff	Software-Endlagen aus
11	sStop	Achse in Stopp

Zuletzt sollen einige Statusmeldungen noch näher erläutert werden. Die Erläuterung zu der Bedeutung der Tabellen in diesem Kapitel stammt von AUTEAM, speziell von Bernd Eberhard [Perb]. Zuerst wird zu Tabelle 17 etwas gesagt. „Lauf“ zeigt dem Nutzer, dass die Pumpe aktiv ist, „Steht“ wiederum ist nur auf die Turbopumpe anzuwenden und zeigt dem Nutzer, dass die Pumpe angehalten ist. Die Turbopumpe bleibt nicht abrupt stehen, sondern trudelt aus. Bei Tabelle 18 bedeutet das „Achse referiert“, dass ein Nullpunkt bzw. ein Referenzwert gesucht/angefahren wird. Die beiden Statusmeldungen „sLAuf“ und „sLAb“ sind zwar für den Hub gedacht, können aber auch für die Rotation genutzt werden. Hierbei wird z.B. in VIFCON der Text etwas angepasst, sodass dieser zur Bewegung passt. Die Meldung „Software-Endlagen aus“ hat eine SPS-interne Bedeutung.

3.4.7 Beispiele von Befehlen

Anders als bei den anderen Geräten übernimmt die Python-Bibliothek **pyModbusTCP** die komplette Kommunikation mit der SPS. Der Nutzer von VIFCON muss lediglich die Registeradressen aus den Tabellen 14 bis 16 und die Anzahl der zu beschreibenden Register richtig setzen und nicht noch bestimmte Strings selbst zusammensetzen. Weiterhin ist auch die Formatierung der Werte durch die Bibliothek gegeben. Hierbei müssen diese nur richtig angewendet werden. Für diese Zwecke gibt es die folgenden drei Funktionen:

1. write_single_coil(),
2. write_multiple_registers() und,
3. read_input_registers().

Bei Interesse können Beispiele für die Nutzung dieser Funktionen im Anhang I gefunden werden.

3.4.8 Umsetzung in Python

Die Umsetzung in der Software umfasst auch wieder verschiedene Funktionen, die später genannt werden. Die Daten-Interpretation wird auch wieder zum größten Teil durch **pyModbusTCP** mitgeliefert. Die Verwendung der dazugehörigen Funktionen

1. encode_ieee(),
2. decode_ieee() und,
3. word_list_to_long()

kann in Anhang I.3 gefunden werden. Hierbei werden noch Funktionen wie *hex* und **int** verwendet. Bei der Umsetzung in VIFCON wird nun somit eine andere Python-Bibliothek verwendet, um die Kommunikation bzw. die Schnittstelle aufzubauen. Einzelheiten zur Umsetzung der Schnittstelle werden in Kapitel 5.3 gezeigt.

Anders als bei den anderen Geräten in diesem Kapitel (Kapitel 3) umfasst die Nemo-1-Anlage 3 Gerätetypen für VIFCON. Diese Geräte sind nach VIFCON Nemo-Gase, Nemo-Achse-Linear und Nemo-Achse-Rotation. Da die Nemo-Gase unter den Geräte-Tab-Typ Monitoring fallen, unterscheidet sich diese von den Achsen, die sich wiederum sehr ähnlich sind.

Für die Schnittstelle werden die Funktionen:

1. write,
2. stopp,
3. write_v,
4. read und,
5. umwandeln_Float,

definiert. Dabei hat das Gerät Nemo-Gase nur die letzten beiden Funktionen, da hier keine Werte an die SPS gesendet werden müssen. Wie auch schon bei den anderen Geräten sind die Haupt-Funktionen *read* und *write*. Somit müssen diese so heißen. Der Inhalt der Funktionen ist bei den drei Geräten unterschiedlich.

In *read* werden die Input-Register (Tabelle 14) der jeweiligen Geräte ausgelesen. Für die Umwandlung wird die Funktion *umwandeln_ Float* verwendet. Der Inhalt dieser Funktion basiert dabei auf den **pyModbusTCP**-Funktionen, weswegen diese Umwandlung in Anhang I.3 in den Beispielen 2 und 3 beschrieben wird.

Bei der *write* Funktion werden bei den Achsen die Coils (Tabelle 16 und die Holding-Register beschrieben. Hierbei wird noch eine Berechnung des Weges (s) bzw. des Winkels (α) durchgeführt, da das Auslesen dieser Werte fehlt (Winkel) oder ungenau ist (Weg). Die lineare und die Rotationsbewegung unterscheiden sich nur in der Berechnung des Positionswertes und den zu setzenden Werten. Auch die Option Define-Home wird in dieser Funktion ausgeführt. Aus dieser Funktion werden die Funktionen *write_ v*, *stopp* und *Start_ Werte* ausgeführt.

In der Funktion *stopp* wird, wie es der Name schon sagt, die jeweilige Achse angehalten, indem das jeweilige Coil gesetzt wird. Bei *write_ v* wird das jeweilige Holding-Register mit dem Geschwindigkeitssollwert befüllt. Die ausgeführte Umwandlung wird in Anhang I.3 in Beispiel 1 gezeigt. *Start_ Werte* führt dasselbe aus wie bei den anderen genannten Geräten in Kapitel 3.

Die Berechnung des Weges und des Winkels sieht wie folgt aus. Sobald der Antrieb eine Geschwindigkeit erhält und die Bewegung gestartet wurde, wird als Erstes eine Zeitdifferenz bestimmt. Diese Zeitdifferenz wird durch die Messung bestimmt. Bei jedem Aufruf der Wert-Berechnung wird eine neue Startzeit ermittelt. Diese Startzeit wird beim folgenden Aufruf mit der aktuellen Zeit verglichen, wodurch eine Zeitdifferenz (t) für die Berechnung des Weges bzw. des Winkels erhalten wird. Für eine genauerere Rechnung wird in dem Moment auch die Istgeschwindigkeit (v) bzw. Istwinkelgeschwindigkeit (ω) der jeweiligen Achse ausgelesen. Nun besitzt das Programm beide Werte für die Berechnung. Beide grundlegenden Gleichungen sind in Formel 3.2 und 3.3 zu sehen. Bei beiden Formeln ist nun nur noch ein Faktor für die Einheit zu beachten. Beide Faktoren rechnen die Geschwindigkeiten um, sodass der Weg in Millimeter und der Winkel in Grad angegeben werden können. Weiterhin wird mit der Zeit in Sekunden gerechnet.

$$s = v * t * \frac{1}{60} \tag{3.2}$$

$$\alpha = \omega * t * \frac{360}{60} \tag{3.3}$$

Für den gesamten Weg bzw. Winkel wird nun anhand der Bewegungsrichtung addiert oder subtrahiert. Somit erhält der Nutzer eine ansteigende oder abfallende Kurve des Weges oder des Winkels.

Ein weiterer Unterschied zu den anderen Geräten ist der Fall der Status-Nachricht. Die Bedeutung und das benötigte Bit von dieser Nachricht wurde für die Antriebe und Gas-Geräte (Pumpen, Messgeräte) in den Tabellen 18 und 17 gezeigt. Im Sinne von Modbus stehen diese Werte auch in den Input-Registern (Tabelle 14). Der Wert wird in der Funktion *read* ausgelesen. Diese Meldung muss nun noch verarbeitet werden. Nach den Tabellen ist bekannt, dass hierbei nur ein Register, also 2 Byte, verwendet werden. Um den Wert nun der Status-Meldung zuzuordnen und diesen im Anschluss auch auszugeben, wurde die Funktion *status_ report_ umwandlung* erstellt. Als kleiner Vorgriff muss hierbei gesagt werden, dass diese Funktion in dem GUI-Teil der Geräte zu finden ist und nicht wie die anderen im Schnittstellen-Definitionsteil. Diese Funktion wird nicht von der Python-Bibliothek übernommen. Nur die Umwandlung beim Auslesen wird von dieser erledigt.

Um den erhaltenen Wert zu verstehen, müssen die folgenden Schritte unternommen werden:

1. Die Antwort (Integer) wird in Binär umgewandelt.
2. Die Bytes werden bestimmt und gegebenenfalls aufgefüllt.
3. Die Bytes werden in der richtigen Reihenfolge verbunden.
4. Die Reihenfolge der Bytes wird gespiegelt.

Im Folgenden werden diese Schritte anhand von Code-Segmenten erläutert. Diese Segmente entstammen der besagten Funktion *status_ report_ umwandlung*. Die komplette Funktion kann dem beiliegenden Programm (z.B. **view/nemoGase.py**) entnommen werden. Im folgenden Beispiel wird der ausgelesene Integer 512 betrachtet. Weiterhin wird auch das Ergebnis der *print*-Methode gezeigt.

Teil 1 - Antwort umwandeln:

```
lese_value = [512]
status = bin(lese_value[0])[2:]    --> print: 1000000000
```

Mit der Methode *bin* wird eine Integer-Zahl in eine Binär-Zahl mit dem Präfix „0b“ gewandelt. Dieser Präfix wird weggeschnitten. Das Problem bei der Zahl ist, dass nur 10 Bits angezeigt werden. In dem Beispiel wird sich auf die Tabelle 17 bezogen. Die 512 wurde ausgelesen, als die Pumpen im Automatikbetrieb waren. In dem Fall muss Bit 1 auf True (1) stehen.

Teil 2 - Bytes bestimmen und gegebenenfalls füllen:

```
l = len(status)
anz = int(l/8)
byte_list = []
for n in range(0,anz):
    byte_list.append(status[-8:])
    status = status[0:-8]

byte_list --> print: ['00000000']
```

Bei diesem Schritt ist es nun wichtig zu wissen, dass die Reihenfolge der Bytes wie folgt ist. Zuerst kommt das Lower-Byte und dann das Higher-Byte. Die 512_{10} wird mit 10-00000000_2 (Minus zur Darstellung der Bytes) übersetzt. In dem Code-Beispiel werden von hinten an die Bytes abgeschnitten. Somit bleibt am Ende nur noch der Byte-Teil mit der 10 übrig. Der folgende Code zeigt die Befüllung dieses Byte-Teils. Somit erhält das Programm am Ende eine Liste mit den vollständigen Bytes (jeweils 8 Bit je Byte).

```
if len(status) < 8:
    status = '0000000' + status
    byte_list.append(status[-8:])

byte_list --> print: ['00000000', '00000010']
```

Teil 3 - Zusammensetzen der Bytes:

```
byte_string = ''
for n in byte_list:
    byte_string = byte_string + n

byte_string --> print: 0000000000000010
```

Durch die For-Schleife werden die Listen-Elemente von *byte_ list* verbunden.

Teil 4 - Umdrehen der Bitfolge:

```
# Byte_Umdreh = byte_string[::-1] --> print: 0100000000000000

# Python-Liste (Pumpe):
def update_GUI(self, value_dict, x_value):
    # Teil hier nicht gezeigt
    status = ['Hand', 'Auto', '', '', '', '', '', '', 'Run', 'Norm', 'Warm', ...
        'Err', '', '', '', '']
    l = len(status_2)
    for n in range(0,l):
        if status_2[n] == '1':
            if not status[n] == '':
                label_s2 = label_s2 + f'{status[n]}, '
    if label_s2 == '':
        label_s2 = self.no_sta_str[self.sprache]
    else:
        label_s2 = label_s2[0:-2]
    self.labelDict['PP22Status'].setText(f'{label_s2}')
```

Dieser Schritt wird aufgrund der Listen-Definition in Python eingefügt. Die Tabellen 17 und 18 werden in Python als Liste (*status*) definiert. Die Bit-Folge wird von rechts nach links gelesen, während die Python-Listen von links nach rechts nummeriert werden. Auf den Schritt kann verzichtet werden, wenn die Python-Liste (z.B. die gezeigte) umgedreht wird und alle 16 Fälle enthält. In dem Fall würden Liste und Bit-Folge übereinstimmen.

Über eine For-Schleife (siehe Code) werden dann die einzelnen Zeichen der Bit-Folge geprüft und mit der Liste der Statusmeldungen verglichen. Bei einer Eins wird diese Statusmeldung an den Statusstring angehängt und angezeigt. Dies ist aber nicht mehr Teil der genannten Funktion, sondern Teil von der Funktion *update_ GUI*. In dem Beispiel würde somit der Status „Auto“ aktiv sein.

4 Methoden

Dieses Kapitel soll einen ersten Einblick auf die Methoden dieser Arbeit geben. Durch diese Methoden sollen die Ziele aus Kapitel 1.2 erreicht werden. Warum diese Methoden verwendet wurden und welche Alternativen es gibt, wird dann in Kapitel 5 gezeigt. Dieses und das Kapitel 6 zeigen dann die Ergebnisse dieser Arbeit. Das Kapitel 6 zeigt dabei spezielle Experimente, die von Mitgliedern der Modellexperimente-Gruppe durchgeführt worden sind, damit die Software optimiert werden kann und eventuelle Fehler gefunden werden.

Folgend wird nun noch ein Rückblick auf die Ziele aus Kapitel 1.2 gezeigt. Die Steuerung soll so erstellt werden, dass die GUI und auch das Programm **modular aufbaubar** sind. Dafür soll es auch eine **anpassbare Konfigurierbarkeit** für jeden Versuch geben. Somit kann jedes Experiment anders sein und an **verschiedenen Anlagen** genutzt werden. Weiterhin sollen mehrere Geräte die **selbe Schnittstelle nutzen** können. Auch die GUI soll **überschaubar und benutzerfreundlich** sein. Neben den verschiedenen Anlagen soll VIFCON auch auf **verschiedenen Betriebssystemen und Computer** wie Raspberry Pi, Linux und Windows laufen können. Besonders die Kompatibilität mit Raspberry Pi ist gewünscht. Das große Ziel ist, dass VIFCON am Ende eine **Kristallzüchtung durchführen** kann.

4.1 Programmiersprache Python

Aus den vorangegangen Kapiteln (1 bis 3) konnte gesehen werden, dass in den Anlagen des IKZ eine SPS genutzt wird. Durch diese werden die Anlagen-Komponenten gesteuert. Eins der Ziele dieser Arbeit ist die Erstellung einer flexibleren Steuerung. In den späteren Kapiteln werden auch andere Aufbauten von Anlagen erwähnt, die z.B. keine SPS besitzen. Die erzielte Steuerung soll somit mit Anlagen arbeiten können, die eine oder keine SPS haben. Also muss ein Programm gefunden werden, das dies umsetzen kann.

Genutzt wird die **Programmiersprache Python**. Python ist ein kostenloses Programmiersprache und beinhaltet sehr viele Packages bzw. Bibliotheken, wodurch die Nutzung dieses Programms favorisiert wird. Neben Python existieren z.B. noch Programmiersprachen wie Java und C++. Ein weiterer Vorteil dieser Sprache ist die große Community, die dahinter steht. Somit finden sich z.B. über GitHub neue Ansätze für die Programmierung und auch neue Bibliotheken.

Im Gegensatz zu den kostenpflichtigen und anfälligen oder alten Programmiersprachen bietet Python eine relativ einfache, sehr umfangreiche und gut lesbare Programmiersprache.

Für die VIFCON-Steuerung wird die **Python Version 3.8.5** verwendet. Die Tabelle 19 und 20 zeigen die Bibliotheken, die in den in Anhang A gezeigten Programmen genutzt werden. In der Tabelle finden sich auch die Versionsnummern und die Dokumentation der Methoden der Packages wieder. Python besitzt eine Standard-Bibliothek, die in Quelle [Sta] zu finden ist. Nicht alle Packages haben eine Versionsnummer in der Tabelle, da diese Standard-Bibliotheken sind, die mit der konkreten Python Version kommen. Die Installation eines Packages erfolgt durch **pip install** in der Konsole (z.B. Powershell).

Um ein Package in Python aufrufen zu können, muss dieses in dem Code importiert werden. Dabei kann es als Ganzes oder nur bestimmte Methoden (Funktion) des Packages eingefügt werden. Um ein Package nutzen zu können, muss dieses aber erst installiert worden sein. Um zu sehen, welche Packages installiert wurden, kann der Konsolen-Befehl **pip list** verwendet werden. Hierbei finden sich nicht die Standard-Bibliotheken wieder. Folgend soll ein Beispiel dazu gezeigt werden:

```
from PyQt5.QtCore import (
    QObject,
    QTimer,
    QThread,
    pyqtSignal,
    QMutex,
    QMutexLocker,
)
import logging
```

In dem Beispiel wird das Package **logging**, sowie einige Methoden des *PyQt5*-Packages importiert. Bei **PyQt** muss etwas genauer darauf geachtet werden, wo sich die einzelnen Methoden bzw. in dem Fall Objekte befinden. Neben *QtCore* gibt es z.B. noch *QtWidget* und *QtGui*.

Tabelle 19: Genutzte Python-Packages 1 (Quelle: eigene Darstellung)

Packages (-Version)	**Dok.**	**Beschreibung**
argparse	[Doka]	Durch das argparse-Package können besondere Start-Befehle einprogrammiert werden wie z.B. ein Testmodus oder die Verwendung von bestimmten Startwerten.
datetime	[Dokc]	Die Nutzung des datetime-Packages ermöglicht das Arbeiten mit Daten und Zeiten.
json	[Dokf]	Die Bibliothek dient dem Dekodieren und Codieren von Variablen.
logging	[Dokg]	Mit dem logging-Package wird ein Log-File erstellt, welches dann mit Informationen, Fehlern und anderem gefüllt wird. Jeder Eintrag hat seinen eigenen Zeitstempel.
os	[Dokn]	Das os-Package ermöglicht es Ordner, Dateien und Pfade zu erstellen und mit solchen zu arbeiten.
pygame - 2.5.2	[Dokm]	Mit dem pygame-Package von Python können Spiele erstellt werden. In VIFCON findet es Nutzen bei der Integration eines Spiele-Controllers.
pyModbusTCP - 0.2.1	[Dokac]	Mit dem pyModbusTCP-Package kann eine Kommunikation mit Modbus TCP hergestellt werden. Hierbei können Server und Clients erstellt werden. Als Client kann dann auf die Modbus-Register (z.B. Coil, Holding-Register) zugegriffen werden.
PyQt5 - 5.15.7	[Dokr]	Das PyQt5-Package wird genutzt, um eine graphische Oberfläche (GUI) zu erstellen. Neben der GUI können auch Threads, Signale und Timer erstellt werden.

Tabelle 20: Genutzte Python-Packages 2 (Quelle: eigene Darstellung)

Packages (-Version)	**Dok.**	**Beschreibung**
pyqtgraph – 0.13.3	[Doko]	Mit dem pyqtgraph-Package können Plots und Graphen erstellt werden, die dann in der GUI angezeigt werden können.
pySerial – 3.5	[Dokp]	Mit dem pySerial-Package wird die Nutzung der Schnittstellen ermöglicht. Über diese kann ein Gerät beschrieben und ausgelesen werden. Das Importieren wird mit **import serial** durchgezogen.
PyYAML - 6.0	[Dokq]	Mit dem yaml-Package wird das Arbeiten mit yml-Dateien ermöglicht. Das Importieren wird mit **import yml** durchgezogen.
random	[Doks]	Das random-Package wird verwendet, um Zufallsoptionen zu erstellen. So kann mit dem Package z.B. eine Zufallszahl generiert oder ein zufälliger Wert aus einer Liste gelesen werden.
shutil	[Doku]	Mit dem shutil Package wird das Arbeiten mit Dateien ermöglicht. Zum Beispiel ist so das Kopieren von Dateien möglich. Das Importieren wird mit **import shutil** durchgezogen.
socket	[Dokv]	Mit dieser Bibliothek wird das Senden von Nachrichten über ein Netzwerke erstellt.
sys	[Dokw]	Durch diese Bibliothek können systemspezifische Funktionen und Variablen aufgerufen werden. So kann z.B. bei einer Anwendung der Haupt-Loop oder das Beenden eingeleitet werden.
time	[Doky]	Mit dem time-Package können Zeiten genutzt werden. So kann das Programm z.B. pausiert werden.

4.2 Model-View-Controller in VIFCON

Für die Erstellung von VIFCON werden zwei grundlegende Methoden verwendet. Diese sind bei dem Programm das IKZ-Programm Multilog und das Muster „Model View Controller“. Das Muster sorgt dabei für eine Trennung zwischen Anwendung, Repräsentation und Benutzermodifikationen. Das Muster wird meist bei graphischen Anwendungen eingesetzt. (Quelle: [Lau02] S. 29)

Bei Multilog [Mul] handelt es sich um ein Daten-Logging-Programm, welches vom IKZ erstellt wurde. In diesem Programm werden Geräte-Daten aufgenommen, gespeichert und dargestellt. Auch dieses Programm basiert auf dem MVC-Muster. Teile von Multilog sind dabei auch in VIFCON eingeflossen bzw. VIFCON hat sich aus Multilog entwickelt. Der große Unterschied zwischen den beiden Programmen ist der Teil der Steuerung. VIFCON liest zwar auch die Werte der Geräte aus, speichert diese und zeigt sie in einem Plot an, aber sendet auch Werte an die Geräte um deren Verhalten zu verändern. Weiterhin ist auch das Design der GUI völlig neu. Die GUI wird in Kapitel 4.3 näher erläutert.

Das grundlegende Modell bzw. Muster ist das MVC, an dem sich VIFCON aufbauen lässt. Das Muster wurde so gut wie möglich eingehalten, doch besondere programmtechnische Herausforderungen sorgen für eine Abweichung vom dem Muster.

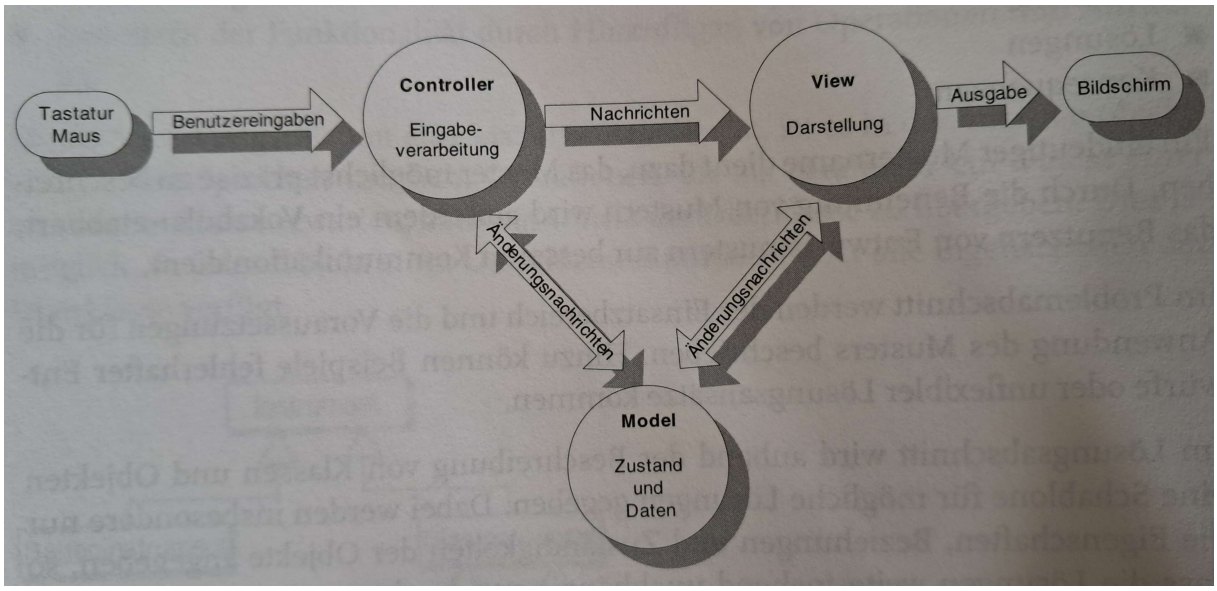


Abbildung 41: Darstellung des Entwurfsmusters MVC (Quelle: [Lau02] S. 30)

Das Modell gibt dabei die Daten und den Zustand der Anwendung an, der durch View dargestellt wird. Der Controller beschreibt die von dem Benutzer erbrachten Modifikationen. Durch dieses Muster werden diese drei Teile voneinander getrennt. Nachstehend werden die drei Teile noch etwas näher erläutert. (Quelle: [Lau02] S. 29)

In dem Entwurfsmuster ist das Modell der wichtigste Bestandteil. In diesem werden die Daten der Anwendung verarbeitet. Auch der Zustand der Anwendung ist hier wiederzufinden. Wenn sich nun die Daten oder der Zustand ändern, so muss sich auch die Ansicht (View) ändern. Das Modell hat in dem Fall keinen Bezug zu der Benutzerschnittstelle. Auch Aktionen des Benutzers, die Daten verändern, werden hier nicht bemerkt. Insgesamt sollte es nur ein einziges Modell in einer Anwendung geben. (Quelle: [Lau02] S. 30)

Der nächste Teil ist der Punkt View oder auch Ansicht genannt. Dieser Teil des Entwurfsmodells spiegelt die Darstellung des Zustandes und der Daten wieder. Hierbei kann es im Gegenzug zum Modell mehrere Objekte von View in einer Anwendung geben. (Quelle: [Lau02] S. 30)

Beim Controller (Verwalter) handelt es sich um einen Zugang. Der Zugang ermöglicht es, auf die Daten und den Zustand zuzugreifen und sie zu modifizieren. Somit kennt der Controller die Aktionen, die zu einer Veränderung des Zustandes und der Daten führen können. Diese Veränderungen werden von dem Modell verarbeitet und transformiert, sodass diese wiederum in der Ansicht genutzt werden können. (Quelle: [Lau02] S. 30-31)

Wie gesagt wurde versucht VIFCON nach diesem Muster zu entwerfen. Die grundlegendere Struktur stammt jedoch von Multilog, welches sich auch nach dem Muster gerichtet hat. In VIFCON wären alle GUI-Elemente zum Punkt View zu sehen, der VIFCON-Controller wäre das Modell und der Controller wird durch die Geräte-Schnittstellen dargestellt.

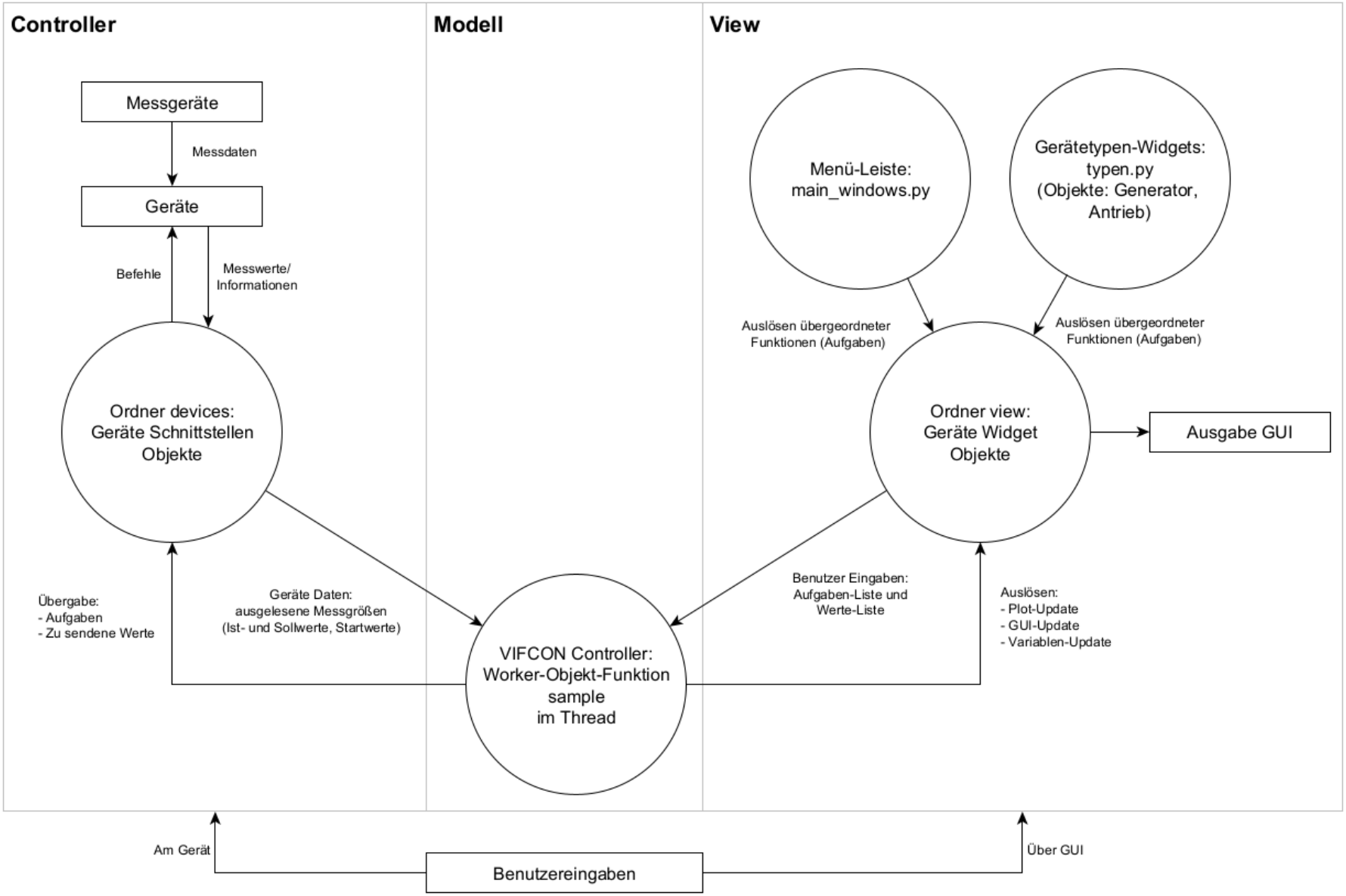


Abbildung 42: Darstellung des Entwurfsmuster MVC an VIFCON (Quelle: eigene Darstellung)

Die Abbildung 42 zeigt die grundlegende Kommunikation bei VIFCON im Programm. Der Aufbau von VIFCON wird in Kapitel 5.1 näher erläutert. Um die Abbildung zu verstehen muss etwas vorgegriffen werden. Nach dem Start von VIFCON wird ein Controller-Objekt erstellt. In diesem Objekt werden alle wichtigen Prozesse von VIFCON gestartet, so auch der Reaktionstimer, die Threads und die Signal-Erstellung. Das Modell wird somit von dem Klasse *Sampler* abgebildet. Von dieser Klasse wird für jedes Gerät eine Instanz (Worker-Objekt) erstellt, die dann die Kommunikation zwischen View und Controller ermöglicht. Dieses Worker-Objekt ist weiterhin mit dem Thread verbunden. Neben dem Worker-Objekt gibt es noch den Reaktionstimer. Dieser Timer wird durch **PyQt** (*QTimer*) erstellt. In VIFCON sorgt der Timer dafür, dass das Senden von Schreib-Befehlen so oft wie möglich erfolgen kann. Immer wenn der Reaktionstimer auslöst, wird durch ein Signal die Funktion *sample* der Worker aufgerufen. In dieser Funktion werden die Informationen vom Gerät und den Geräte-Widgets abgefragt und jeweils an den anderen weitergeleitet. So bildet der VIFCON-Controller den Kern dieser Anwendung. In der Abbildung 42 können auch Programm- und Ordnernamen gesehen werden. Der Inhalt von VIFCON kann in Anhang A gefunden werden.

Bei View werden die GUI-Elemente eingeschlossen. An drei Stellen werden Benutzereingaben über die GUI akzeptiert. Einerseits gibt das Hauptfenster eine Menüleiste, über die bestimmte Aktionen ausgelöst werden können. Weitere Aktionen können über die Geräte-Typen-Widgets ausgelöst werden. Bei diesen Anwendungen handelt es sich um Aktionen, die mehrere Geräte ansprechen können. Die Geräte-Widgets, die in ihrem Gerätetyp-Widget liegen, lösen nur spezielle Aktionen für ihr Gerät aus.

Diese Aktionen und auch Sollwerte werden von dem Modell ausgelesen und an den Controller, hier die Geräte-Schnittstellen, weitergeleitet. Andersherum werden vom Controller die Messdaten über den Controller an die GUI geleitet.

Im Unterschied zur Abbildung 41 gibt es keine direkte Verbindung zwischen Controller und View. Auch kann der Benutzer über die GUI oder das Gerät auf die Anwendung zugreifen. Ein Teil des Controllers liegt somit auch in dem Punkt View von VIFCON.

Mit diesem Muster, so wie es in Abbildung 42 dargestellt ist, wird bereits eine Modularität erreicht. Dadurch dass alles über den VIFCON-Controller läuft, können Geräte sehr leicht hinzugefügt werden. Dabei greifen alle Instanzen der Sampler-Klasse auf die selbe Programmierung zu. In dem Sinne müssen also die speziellen Funktionen wie z.B. *read* und *write* in allen Geräten (beim Beispiel Schnittstellen-Objekte) gleich heißen. Der Inhalt dieser Funktionen kann dann besonders beim Lesen und Schreiben von Werten von Gerät zu Gerät anders sein. Weitere Informationen zum Aufbau des Programms, können in Kapitel 5.1 gefunden werden.

4.3 Graphische Benutzeroberfläche (GUI)

Bei der GUI wurden die Python-Bibliotheken **PyQt5** und **pyqtgraph** verwendet. Auch die GUI an sich soll modular sein. Dies wird durch Splitter erreicht. Die Abbildung 43 zeigt zwar eine ältere Version der VIFCON-GUI, aber ein Vorteil der Splitter lässt sich in der Abbildung gut sehen. Auf der rechten Seite werden die Antriebe dargestellt. Unter dem Plot befinden sich eigentlich 4 Geräte-Widgets. Durch den Splitter lassen sich jedoch Bereiche überklappen.

Neben dem verdecken von Teilen auf der GUI, wird durch Abbildung 44 auch der zweite Vorteil der Splitter sichtbar. Durch die Splitter kann sich der Bildschirm, also die GUI, so zurechtgelegt werden, dass der Benutzer sie gut bedienen kann. Die GUI aus Abbildung 44 wurde im Sinne eines Test-Programms, das nicht Teil dieser Arbeit ist, erstellt. Durch diese beiden Vorteile kann die GUI im laufenden Programm verändert werden.

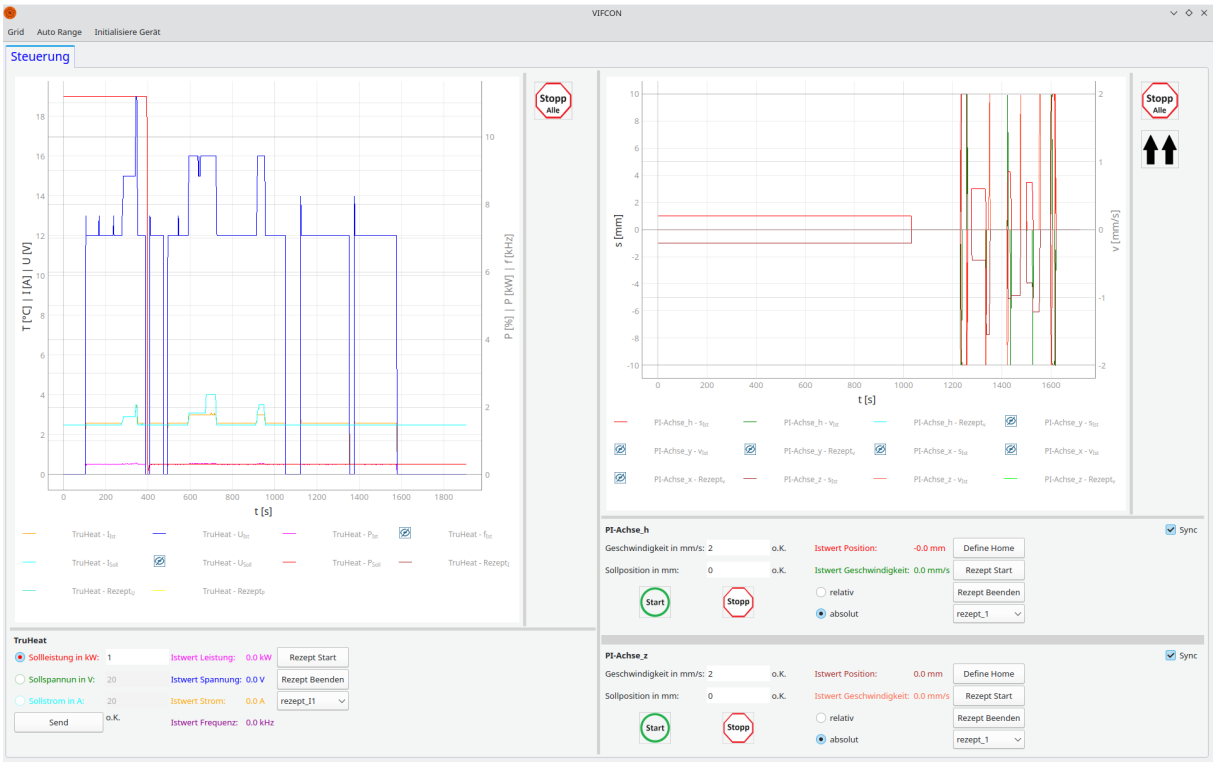


Abbildung 43: GUI - Splitter 1 (Quelle: eigene Darstellung)

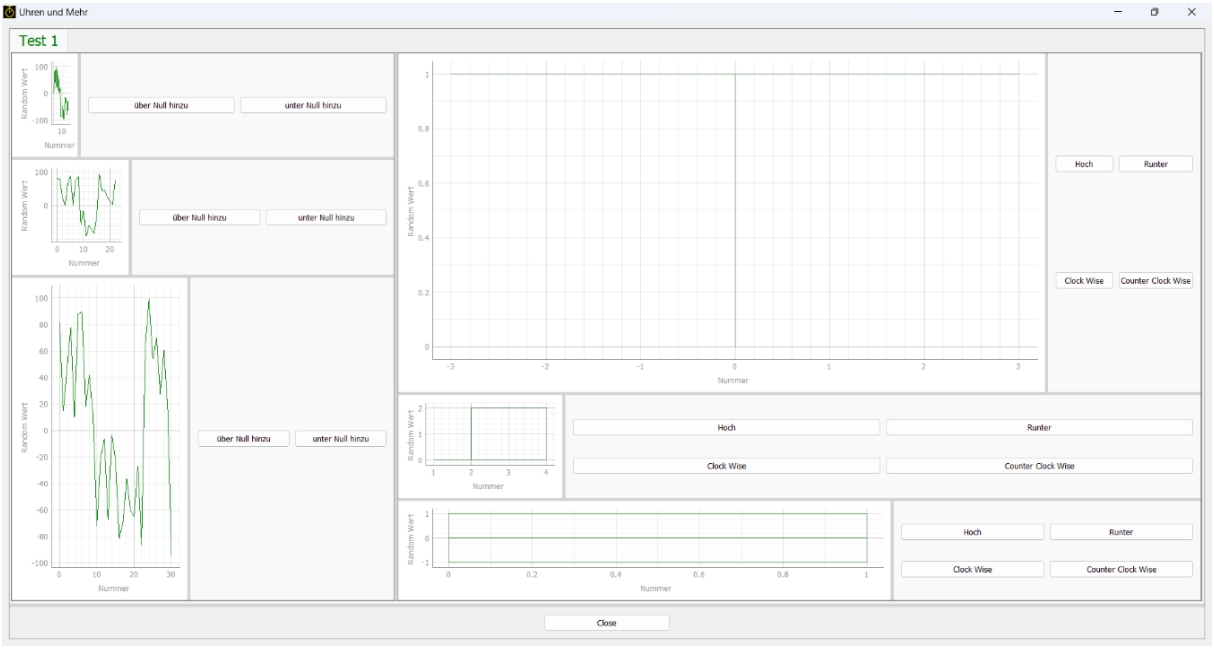


Abbildung 44: GUI - Splitter 2 (Quelle: eigene Darstellung)

Wie erwähnt wird **PyQt5** verwendet. Das GUI-Toolkit wurde von Haavard Nord und Eirik Eng entworfen. Zunächst gab es dieses Toolkit nur für C++. Erst durch Phil Thompson gibt es dieses GUI-Toolkit auch für Python. Dieses Toolkit umfasst mehr als nur Klassen für die Anwendung mit GUIs. Neben diesen Klassen gibt es auch Klassen für Multithreading und Bildbearbeitung. Besonders das erste wird bei VIFCON stark verwendet und wird deswegen in Kapitel 4.4.1 aufgegriffen. (Quelle: [Lau02] S. 191-192)

Bei Qt handelt es sich um eine Klassenbibliothek, mit einer ausgeprägten Eltern-Kind-Relation. Somit umfasst Qt sehr stark die objektorientierte Programmierung. (Quelle: [Lau02] S. 192)

Die GUI in VIFCON (siehe Abbildung 43) hat verschiedene Schichten. Bei **PyQt** werden Widgets mit Layouts versehen. In diese Layouts können wiederum neue Widgets eingefügt werden. Somit ist die erste Schicht das Hauptfenster (Main Windows) in dem die Menü-Leiste und die Tabs definiert werden. Die zweite Schicht bilden die Tabs. In diese Tabs kommen als dritte Schicht dann die Geräte-Typ-Widgets. Im Tab Steuerung werden die Typen Generator und Antrieb zu finden sein. Auf den jeweiligen Hälften werden dann die Splitter eingefügt. Somit befinden sich immer ein Plot und die verwendeten Geräte-Widgets in den Geräte-Typ-Widgets. Auf den Geräte-Widgets finden sich sämtliche Bedienelemente wieder. Die grundsätzliche Programmierung wird in Kapitel 5.9 erläutert.

4.4 Schnittstelle

Bei den Schnittstellen gibt es in VIFCON drei Gruppen. Zum einen wäre da die Schnittstelle zu den Geräten aus Kapitel 3, des weiteren eine Bediener-Schnittstellen zur Nutzung eines Gamepads und als letztes noch eine Schnittstelle zu anderen Programmen. Diese drei werden in den folgenden Kapiteln erläutert.

4.4.1 Schnittstellen zu Geräten

Bei den Schnittstellen müssen nun drei Dinge beachtet werden. Diese sind:

1. die Art der Schnittstelle,
2. das Nicht-Einfrieren der GUI und,
3. die gleichzeitige Nutzung einer Schnittstelle.

Aus Kapitel 2.3 wurde ersichtlich was bei der Umsetzung der Schnittstelle wichtig war. Bei den meisten Geräten (PI-Achse, Eurotherm, TruHeat) wird die RS232-Schnittstelle verwendet. Python liefert für diesen Zweck die Bibliothek **serial** (bzw. pySerial). Für die Geräte der Nemo-1-Anlage wird die Ethernet-Schnittstelle verwendet. Durch die Bibliothek **pyModbusTCP** wird die gesamte Kommunikation für das Kommunikationsprotokoll Modbus umgesetzt.

In Kapitel 4.3 wurde gesagt, dass **PyQt5** benutzt wird. GUI-Anwendungen basieren meist darauf, dass ein Hauptloop abläuft. In diesem Loop reagiert das Programm dann auf Interrupts durch die Widgets. Wenn nun aber größere Prozesse ablaufen, kann die GUI einfrieren und wird somit unbrauchbar. Aus dem Grund liefert **PyQt5** auch die Möglichkeit der sogenannten Threads. Durch einen Thread wird ein Nebenloop erstellt. Dieser Nebenloop läuft simultan zum Hauptloop. Somit wird der Hauptloop nicht unterbrochen.

Als letztes muss es ermöglicht werden, Schnittstellen mehrfach zu benutzen. Besonders wichtig wird dies bei den Geräten PI-Achse (Kapitel 3.2) und Nemo-1 (Kapitel 3.4). Auch hier bietet **PyQt5** eine Lösung. Der sogenannte Mutex und ein dazugehöriger Mutex Locker ermöglichen eine gemeinsame Nutzung einer Schnittstelle. Sobald der Mutex mit der Schnittstelle verbunden wird, kann der Mutex Locker verhindern, dass die Geräte gleichzeitig auf die Schnittstelle zugreifen. Sobald eines der Geräte die Schnittstelle benutzt, blockt der Locker die anderen Geräte. Erst wenn das eine Gerät fertig ist, kann das nächste arbeiten. Zum Mutex wird in Kapitel 5.4 mehr erzählt. Neben dem Kapitel wird in Kapitel 5.2 mehr über die Loops und Threads und in Kapitel 5.3 mehr zur Umsetzung der Schnittstelle erläutert.

4.4.2 Bediener-Schnittstellen (Gamepad)

Die 4 Achsen der Nemo-1-Anlage (Kapitel 3.4) sollen auch über einen Controller bedient werden. Der genutzte Controller kann in Abbildung 45 gesehen werden.

Abbildung 45: Gaming-Controller (Quelle: eigene Darstellung)

Die Umsetzung und Verknüpfung erfolgt mit der Bibliothek **PyGame**. Für VIFCON ist nun wichtig welche Knöpfe wie ausgelöst werden können. Für diesen Zweck wurde die Quelle [Pyg] genutzt. Auf dieser Internetseite befindet sich ein Programm, welches einerseits das Gamepad ausliest und andererseits dann auch als Vorlage für **gamepad.py** diente. In VIFCON müssen dann alle Reaktionen für jeden Button abgefangen werden. Wenn das Programm von [Pyg] mit dem gezeigten und angeschlossenen Gamepad ausgeführt wird, dann wird die Abbildung 46 erzeugt. In der gezeigten GUI können dann Informationen wie die Anzahl von Axes (links - Gamepad), Buttons (Knöpfe) und Hats (Bediensticks - sind hier nicht vorhanden) oder auch der Name des Gamepads gesehen werden. Um die Zugehörigkeit der Buttons zu den gezeigten Nummern herauszufinden, müssen die Buttons betätigt werden. Der jeweilige Button ändert dann seinen Wert in der GUI. Dasselbe gilt z.B. für die Achsen. VIFCON speichert diese Daten dann in der Log-Datei als Information.

Für die Nemo-Achsen wurde es so umgesetzt, dass die Knöpfe Y (T-) und A (T+) für die Rotation und X (V+) und B (V-) für die Hub-Bewegung genutzt werden. Mit Select werden die Achsen gestoppt. Bei der PI-Achse ist das derzeitige Konzept, dass die Knöpfe L und R für den linken und rechten Knopf des PI-Widgets funktionieren. Auch hier ist Select Stopp. IN VIFCON werden somit bestimmte Geräte-Widget-Objekt-Methoden durch die Buttons angesprochen. Das Konzept für die PI-Achse wird aufgrund der drei Bewegungsachsen noch weiterentwickelt. Dies soll aber nicht Teil dieser Arbeit sein. Weiteres zum Gamepad wird in Kapitel 5.6 gezeigt.

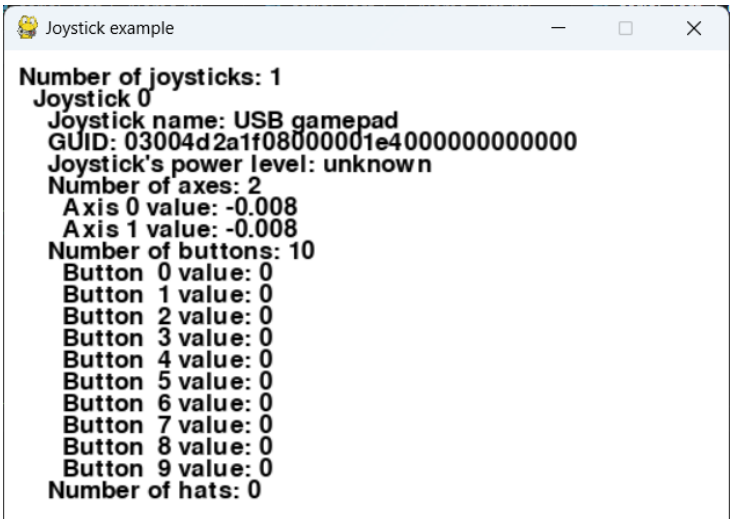


Abbildung 46: Gaming-Controller - Informationen (Quelle: eigene Darstellung)

4.4.3 Schnittstellen zu anderen Software-Tools (Multilog)

In Kapitel 4.2 wird erwähnt, dass Multilog als Vorlage für VIFCON genutzt wurde. Sowohl VIFCON als auch Multilog sollen ein Feature zur Kommunikation erhalten. Diese Kommunikation erfolgt durch das TCP/IP-Protokoll. Die Multilog-Seite wurde dabei von Felix Oesterle [Pere] am IKZ entworfen. In Zusammenarbeit mit Felix Oesterle wurde zudem die Funktion der Kommunikation getestet. Wie schon erwähnt wird die Kommunikation über TCP und IP ermöglicht. In Python wird für diesen Zweck die **socket**-Bibliothek genutzt. Neben dieser wird auch die **json**-Bibliothek für die Datenverarbeitung genutzt.

Bei dieser Kommunikation wird Multilog der Client (Master) und VIFCON der Server (Slave) sein. Multilog sendet Trigger (bzw. Befehle) an VIFCON, die von diesem ausgewertet werden. Als Antwort sendet VIFCON ein Dictionary mit Messdaten an Multilog zurück, welche dann von Multilog geloggt und im Plot angezeigt werden. Durch diese Verbindung liegen alle Messdaten an derselben Stelle (Multilog). In Kapitel 4.5 werden verschiedene Experimente erwähnt. Bei den Experimenten an der Nemo-1-Anlage wird auch Multilog verwendet. Weitere Details können in Kapitel 5.7 gefunden werden.

4.5 Experimente

Um die Steuerung zu testen, werden verschiedene Experimente geplant. Die Experimente werden an verschiedenen Modellanlagen vorgenommen, da dies auch ein Ziel der Steuerung ist. In Kapitel 1.2 wurde erläutert, dass der Sinn der Steuerung der ist, dass diese so konfigurierbar sein soll, dass diese mit verschiedensten Anlagen nutzbar ist. Bei den Anlagen werden:

1. die Demo-FZ und,
2. die Nemo-1-Anlage verwendet.

Somit umfassen diese Anlagen die beiden wesentlichen Züchtungsprinzipien. Bei der Nemo-1-Anlage kann neben dem CZ-Verfahren auch z.B. eine Magnetfeldmessung durchgeführt werden, so wie es in der eigenen Bachelorarbeit [Fun22] der Fall war.

Die Experimente sollen dabei helfen, bestimmte Dinge herauszufinden. Diese wären:

1. ein grundlegender Test der GUI und der verbundenen Programmierung (Steuerung),
2. eine Rückmeldung eines Nutzer für Optimierungen,
3. die Nutzung der GUI und der Config-Datei durch einen Bediener,
4. ein fehlerfreies Arbeiten mit den angeschlossenen Geräten und ihren Schnittstellen und,
5. die Nutzbarkeit bei einer tatsächlichen Züchtung.

Insgesamt sind drei Experimente geplant. Diese wären:

1. ein Test der ersten Geräte an der Demo-FZ,
2. ein Test der Modbus-Schnittstelle an der Nemo-1-Anlage und,
3. eine Kristallzüchtung an der Nemo-1-Anlage.

Bei Experiment 1 steht sozusagen der erste Stand der Steuerung im Vordergrund. Das Programm beinhaltet dabei nur Elemente mit RS232-Schnittstelle. Zu dem Zeitpunkt sind nur die Geräte Eurotherm (Kapitel 3.1), PI-Achse (3.2) und der TruHeat (3.3) eingebaut. Bei dem Experiment sollen die letzteren beiden angeschaut werden. Somit sollen Probleme ermittelt werden. Um etwas vorwegzunehmen, muss gesagt werden, dass hier z.B. bei dem TruHeat Dinge wie das Interface sowie der RS232-User Watchdog ermittelt worden sind. Mehr dazu kann in Kapitel 6.1.2.4 gelesen werden. Mit der Rückmeldung dieses Bedieners soll die GUI und die Steuerung für diese Gruppe optimiert werden. Weiterhin sollen die oben genannten Punkte überprüft werden.

Bei dem zweiten Experiment wird die Nemo-1-Anlage genutzt. Diese ist in Abbildung 35a zu sehen und gehörte zu den Anlagen, die zu Beginn als Stand der Technik am IKZ angesehen wurde. In dem Experiment soll die Modbus Schnittstelle getestet werden. Für diesen Zweck wird eine Heilung von Thermoelementen bzw. ein Heiztest durchgeführt.

Der abschließende Test mit VIFCON wird eine Züchtung eines Kristalls sein. Auch dieser wird an der Nemo-1-Anlage vorgenommen. In diesem Experiment wird ein Zinn-Kristall gezüchtet. Für diesen Zweck sollen die Hauptschritte aus Kapitel 2.1.1 mit VIFCON realisiert werden. Damit sind die Aufheizung, das Ankeimen, die Züchtung, die Abtrennung und das Abkühlen gemeint. Das Ziel dieses Experimentes soll dabei die komplette Funktionsfähigkeit von VIFCON für eine echte Kristallzüchtung sein.

Alle Experimente können in Kapitel 6 erläutert gefunden werden. Dabei wird der genaue Aufbau und Ablauf der Experimente genannt und darauf folgend dann das Experiment ausgewertet.

4.6 Installation auf Raspberry Pi

Ein Ziel dieser Arbeit ist die Kompatibilität von VIFCON und Raspberry Pi. Um die Steuerung auf einem Raspberry Pi zu nutzen, muss dieser zum einen ein aktuelles Betriebssystem und Python installiert haben. Die Entwicklung und auch die Tests von VIFCON wurden an einem Windows-Rechner und den verschiedenen Linux-Rechnern der Modellexperimente-Gruppe durchgeführt. Am Ende soll die VIFCON-Installation auf dem Raspberry Pi auch mit der Anlage funktionieren. Für die Umsetzung der Installation von VIFCON auf Raspberry werden folgende Punkte behandelt:

1. Was ist ein Raspberry Pi?
2. Welches Betriebssystem wird auf diesem genutzt?
3. Welche Konfigurationen werden getroffen?
4. Wie können die Python-Packages aus Tabelle 19 und 20 auf dem Raspberry Pi installiert werden?
5. Gibt es Schwierigkeiten bei der Inbetriebnahme von VIFCON und Installation des Betriebssystems und den Python-Packages?

Bevor die VIFCON spezifischen Punkte genannt werden, soll erst mal geklärt werden was ein Raspberry Pi ist. Bei Raspberry Pi handelt es sich um ein Einplatinencomputer, welcher von der britischen Raspberry Pi Foundation kreiert wurde. Meist befindet sich das Betriebssystem des Raspberry Pi auf einer SD-Karte, von der aus das Hochfahren des Computers geregelt wird. Auch wird zumeist als Betriebssystem Linux verwendet. Der Name Raspberry Pi und auch das Logo kommt von der Himbeere, was Raspberry im deutschen heißt. Das Pi steht hierbei für „Python interpreter“. (Quelle: [Rpia])

Abbildung 47: Genutzter Raspberry Pi (Quelle: eigene Darstellung)

Die Abbildung 47 zeigt den Raspberry Pi auf dem die Steuerung VIFCON installiert und getestet wurde. Für die Zwecke dieser Arbeit wurde die Installation wie folgt durchgeführt:

1. Sicherung der Daten der Micro SD-Karte und Neuformatierung dieser.
2. Den Raspberry Pi Imager von der Seite [Rpib] für Windows herunterladen, installieren und öffnen. Die GUI dessen ist in Abbildung 48 zu sehen.
3. Die Notwendigen Punkte (siehe Abbildung 48) auswählen. Diese sind:
   (a) **Raspberry Pi Modell** (kein bestimmtes Modell),
   (b) **Betriebssystem OS** mit Raspberry Pi OS (64-Bit) und,
   (c) **SD-Karte**.
4. Einstellen des Images durch weitere Einstellungen wie z.B. Sprache.
5. Das Image schreiben und verifizieren lassen.
6. Die SD-Karte in den Raspberry Pi stecken und Raspberry Pi starten.

Bei der SD-Karte wird die eingesteckte Micro SD-Karte ausgewählt. Die drei Dinge die ausgewählt worden, sind in Abbildung 48 zu sehen. Über den Imager wird die Installation sehr vereinfacht. Somit hat der genutzte Raspberry Pi die folgenden Daten:

1. Betriebssystem: RPi OS 64-bit Version 12
2. Modell/Version: Raspberry Pi 400 (Model No: RPI-400)
3. Debian Version: 12 (bookworm)
4. Python Version: 3.11.2

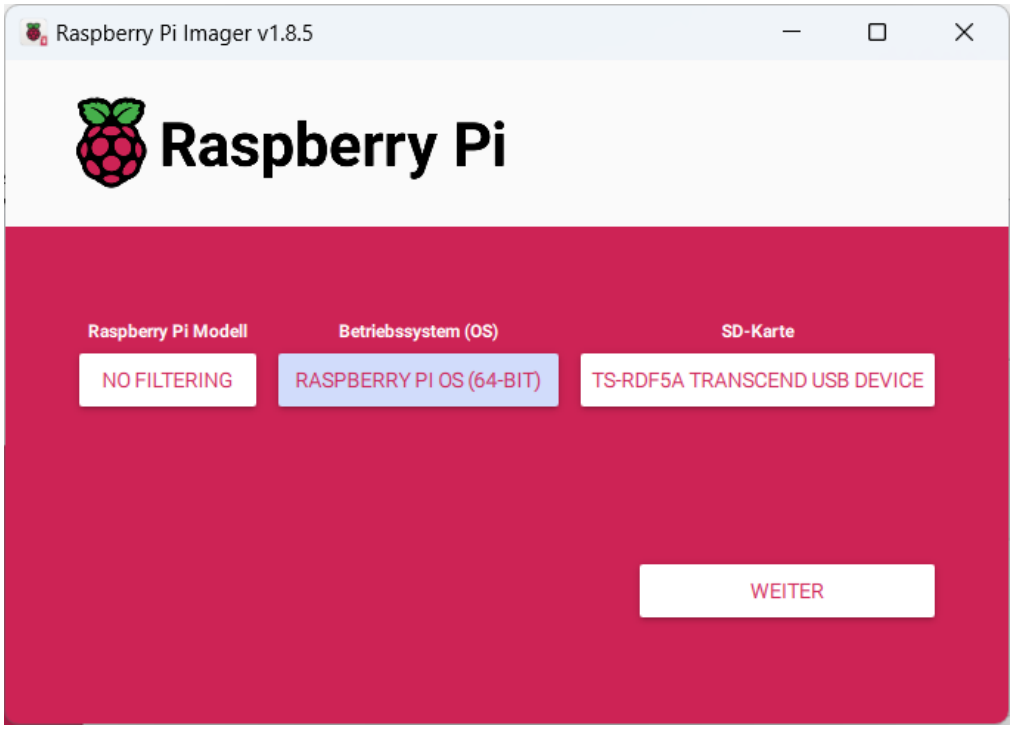


Abbildung 48: Raspberry Pi Imager (Quelle: eigene Darstellung)

Nach der Installation und dem erfolgreichen hochfahren des Raspberry Pi können nun alle weiteren Einstellungen getroffen werden. Dazu gehören:

1. die Einrichtung eines weiteren Nutzers, da ein admin Nutzer bei der Installation mit erstellt wurde,
2. die Übergabe der sudo rechte an diesen neuen Nutzer pi,
3. die Freischaltung der seriellen Ports und,
4. die Installation der Python-Bibliotheken aus Tabelle 19 und 20.

In Linux wird weitestgehend die Nutzung der Konsole verwendet. Somit werden die Einstellungen über diese getätigt. Die Erstellung eines Nutzers (Code 1) und die Freigabe von Ports (Code 2) und den sudo-Rechten (Code 1) sieht folgendermaßen aus:

```
admin@raspberrypi:~ $ sudo su
root@raspberrypi:/home/admin# useradd -m pi
root@raspberrypi:/home/admin# passwd pi
New password:
Retype new password:
passwd: password updated successfully
root@raspberrypi:/home/admin# usermod -g users pi
root@raspberrypi:/home/admin# usermod pi -a -G sudo
root@raspberrypi:/home/admin# exit
exit
```

```
pi@raspberrypi:~ $ sudo usermod -a -G dialout pi
```

Neben diesem Befehl, kann auch `sudo raspi-config` verwendet werden. Durch diesen öffnet sich dann das „Raspberry Pi Software Configuration Tool (raspi-config)“. In diesem kann z.B. die Tastatur-Einstellung oder auch die Freigabe der seriellen Ports angesehen werden. Bei der Freigabe der Ports war ein Neustart nötig, um den Befehl bzw. die Einstellung durchzuführen. Besonders wichtig hierbei sind die sudo-Rechte, da diese gebraucht werden um Python Bibliotheken zu installieren. Wie schon erwähnt haben die Raspberry Pi bereits Python installiert. Somit sind einige der Bibliotheken bereits installiert gewesen. Auf dem Raspberry Pi gibt es auch schon eine Entwicklungsumgebung für Python, nämlich Thonny. Wenn hier die Bibliothek mit import eingefügt wird und ein Programm gestartet wird, kann anhand der Fehlermeldung gesehen werden, ob eine Bibliothek da ist oder fehlt. Um die Bibliotheken im vollkommenen zu sehen, wird der Befehl `pip3 list` verwendet. In der Konsole werden so alle vorhandenen Bibliotheken angezeigt.

Von den 16 gebrauchten Bibliotheken, waren bereits 13 installiert. Diese Bibliotheken sind:

1. argparse,
2. datetime,
3. json,
4. logging,
5. os,
6. pygame (Version 2.1.2),
7. PyQt5 (Version 5.15.9),
8. pySerial (Version 3.5),
9. random,
10. shutil,
11. socket,
12. sys und,
13. time.

Interessanterweise waren neben den 10 Standard-Bibliotheken (ohne Version, nicht in pip3 list) auch drei Bibliotheken bereits installiert, die bei Windows oder anderen Linux-Rechnern erst installiert werden mussten. In den Tabellen 19 und 20 stehen auch die Versions-Nummern. Bis auf **pySerial** unterscheiden sich die Versions-Nummern bei **pygame** und **PyQt5**. In den verschiedenen Systemen wurden die Versionen 3.8.5 und 3.11.2 von Python genutzt. VIFCON soll auf 3.8.5. basieren, funktioniert aber auch unter 3.11.2 auf Windows und dem Raspberry Pi. Bei Linux (z.B. Labor-Notebook) wird 3.8.5 verwendet. VIFCON benötigt 16 Bibliotheken von Python. Fehlen tun:

1. pyModbusTCP (Version 0.2.1),
2. pyqtgraph (Version 0.13.1) und,
3. PyYaml (Version 6.0).

Bei den fehlenden Bibliotheken wird auch die Version gezeigt. Diese zeigt die Version die dann erfolgreich auf dem Raspberry Pi installiert wurde. Normalerweise wird für die Installation **pip3 install** verwendet, doch mit der neusten Version (bookworm) ist dies nicht mehr so wirklich möglich. Neben pip install gibt es noch **apt install**, welches hier auch genutzt werden wird. Um pip nutzen zu können, muss zunächst der Befehl **venv** verwendet werden. Mit diesem werden virtuelle Python-Umgebungen erstellt. Somit gibt es zwei verschiedene Wege um VIFCON auf Raspberry Pi zum laufen zu bringen. Beide Möglichkeiten sollen im Folgenden vorgestellt werden. (Quelle: [Rpic])

Wenn in dem pi-Nutzer **pip install** genutzt wird, dann erscheint in der Konsole eine Fehler-Nachricht mit dem Fehler „externally-managed-environment“. Der komplette Fehler kann in der Quelle [Rpic] gefunden werden. Mit **apt install** ist es auch ohne die Virtuellen Umgebungen möglich, alle Bibliotheken global auf dem Raspberry Pi zu installieren. Doch bieten diese noch eine Alternative dazu. Die virtuellen Umgebungen haben den Vorteil, dass sie kompakter sind, da sie nicht alle Bibliotheken, sondern nur die gebrauchten Bibliotheken beinhaltet. Außerdem können hier nun Bibliotheken von Drittanbietern sicher installiert, ohne das das Python-System beeinträchtige oder sogar im schlimmsten Fall beschädigt wird. Dies ist auch der Grund für die Fehlermeldung. Ein bereits lang existierendes Problem in Python ist der Konflikt zwischen dem Paketmanagern des Betriebssystems (z.B. apt) und den Paketverwaltungstools von Python selbst (z.B. pip). Hierbei zählen API-Inkompatibilitäten (Python Ebene) und fragen zu dem Eigentum von Dateien. Ab der Version 12 (bookworm) wird somit **pip install** nur noch in den virtuellen Umgebungen möglich sein. Folgend wird nun gezeigt wie eine solche virtuelle Umgebung erstellt wird. (Quelle: [Rpic])

```
1  pi@raspberrypi:~ $ mkdir my_project
2  pi@raspberrypi:~ $ cd my_project
3  pi@raspberrypi:~/my_project $ python -m venv env
4  pi@raspberrypi:~/my_project $ ls -la
5  total 12
6  drwxr-xr-x  3 pi users 4096 May  7 15:29 .
7  drwxr-xr-x 15 pi users 4096 May  7 15:28 ..
8  drwxr-xr-x  5 pi users 4096 May  7 15:29 env
9  pi@raspberrypi:~/my_project $ source env/bin/activate
10 (env) pi@raspberrypi:~/my_project $ pip list
11 Package    Version
12 ---------- -------
13 pip        23.0.1
14 setuptools 66.1.1
15 (env) pi@raspberrypi:~/my_project $ pip install pyModbusTCP
16 ...
17 (env) pi@raspberrypi:~/my_project $ deactivate
```

In dem Konsolen-Code kann gesehen werden wie in dem Ordner *my_project* die virtuelle Umgebung *env* erstellt, diese aktiviert und deaktiviert und die Bibliothek installiert wird.

```
1 pi@raspberrypi:~ $ sudo apt install python3-pyqtgraph
2 ...
3 pi@raspberrypi:~ $ sudo apt install python3-yaml
4 ...
```

Der Aufbau von **apt install** wird im vorrangegangenen Code gezeigt. Die benötigte Bibliothek wird dabei einfach hinter „python3-“ geschrieben, wie es am Beispiel **pyqtgraph** zu sehen ist. Durch die Installation über **apt** vereinfacht sich die Installation von großen Bibliotheken, gegenüber Pythons eigenem Verwaltungssystem von Bibliotheken. Weiterhin werden alle anderen Abhängigkeiten der zu installierten Bibliothek mit installiert und es wird ein Installationsprotokoll erstellt, sodass das zurücksetzen (oder deinstallieren) leichter abläuft. Bei den anderen beiden Bibliotheken gab es noch Schwierigkeiten. Die **pyModbusTCP**-Bibliothek konnte mit der Möglichkeit nicht installiert werden! Bei **PyYaml** war der Name anders, sodass diese mit apt installiert werden konnte. Folgend wird gezeigt wie **pyModbusTCP** installiert wurde. (Quelle: [Rpic])

```
1 pi@raspberrypi:~ $ git clone https://github.com/sourceperl/pyModbusTCP
2 pi@raspberrypi:~ $ cd pyModbusTCP
3 pi@raspberrypi:~/pyModbusTCP $ sudo python3 setup.py install
```

Bei der **pyModbusTCP**-Bibliothek musste der Weg über die GitHub-Seite gewählt werden. Durch die Erstellung eines Clones dieses Repository, konnte die gesamte Bibliothek auf den Raspberry Pi gezogen werden. Im Folgenden wird der Ordner mit cd ausgewählt und über sudo dann die Installation über **setup.py** eingeleitet. Um sind alle Bibliotheken für VIFCON installiert.

Von den beiden Möglichkeiten wurde nur die globale Variante **apt** abgeschlossen. Die Probleme die z.B. mit **pyModbusTCP** bei **apt** aufgetreten sind, gab es bei den virtuellen Umgebungen nicht. Der Grund warum die virtuellen Umgebungen am Ende nicht weiter umgesetzt wurden, war die **PyQt5**-Bibliothek. Der Versuch diese Bibliothek in den Umgebungen zu installieren führte immer zu Fehlern. Mit **apt** konnten alle Bibliotheken auf dem Raspberry Pi installiert werden.

5 VIFCON Steuerung

In diesem Kapitel finden sich nun die Überlegungen, weitere Ausführungen, sowie die Ergebnisse für die in Kapitel 4 gezeigten Methoden wieder. Unterstützt wird dieses Kapitel durch das folgende Kapitel 6, welches die Experimente mit der VIFCON Steuerung erläutert. Durch diese Experimente wurde die Steuerung weiterentwickelt.

5.1 Programmstruktur

Der Grundlegende Aufbau von VIFCON basiert auf Multilog [Mul] und dem MVC-Modell, welches in Kapitel 4.2 erläutert und anhand einer Skizze gezeigt wurde (Abbildung 42). Grundlegend wird in VIFCON nach dem Start (**vifcon_main.py**) ein *Controller*-Objekt erstellt, welches für den Aufbau der gesamten Kommunikation verantwortlich ist. In diesem Objekt werden alle Teile der GUI (Kapitel 5.9) und alle Threads erstellt. Die Erläuterung zu den Threads erfolgt in Kapitel 5.2. In VIFCON ist dies grob durch sogenannte Worker-Objekte geregelt. In VIFCON heißt dieses Objekt **Sampler** und wird für jedes Gerät als Instanz erstellt. Durch die Funktion der Signale in **PyQt** wird die Bearbeitung (Lesen und Schreiben) in diesen Geräte-Threads dann durch einen Sample-Timer aufgerufen.

Neben den Threads wird in dem Kapitel 5.2 auch von dem sogenannten Haupt-Loop geredet. Diese Schleife ist ein Funktionsprinzip der GUI. Zu diesem Thema muss zunächst klar werden, dass mit **PyQt** eine Anwendung erzeugt wird. Es ist dabei ratsam zunächst diese Anwendung mit Hilfe des Objekts *QApplication* und der *sys*-Bibliothek zu erstellen, bevor überhaupt GUI-Teile erstellt werden. Der Nachfolgende Code zeigt dies. Die Zeile 3 in dem Code startet den Haupt-Loop der GUI und sorgt für ein sauberes beenden von Python und die Freigabe von Speicherressourcen. (Quelle: [Reaa])

```
1      app = QApplication(sys.argv)
2      self.main_window.show()
3      sys.exit(app.exec())
```

Somit kann sich der erste Graph in Abbildung 49 angesehen werden. Diese Abbildung zeigt die grundsätzlichen Verbindungen in VIFCON. Somit können z.B. Start und Ende in diesem wieder gefunden werden. Eine wichtige Sache muss dazu noch gesagt werden. In der Abbildung werden nicht alle Funktionen und Verbindungen gezeigt. Es soll sich hierbei um die etwas übergeordneten Funktionen und Objekte drehen, die entweder von allen Geräten benutzt oder durch besondere Umstände aufgerufen werden. Ein solcher Umstand ist z.B. die Menü-Leiste. Im Sinne dessen sind die verschiedenen Komponenten (Programm - weiß, Funktionen - hellgelb, Objekte (Klassen) - hellgrün, Objekt-Methoden/Funktionen - lila) farblich voneinander getrennt. Durch die Pfeile soll gezeigt werden, von wo die Komponenten aufgerufen (blau) oder erstellt (dunkelgrün) werden. Besonders das *Controller*-Objekt fällt hier sehr auf. Von diesem gehen sehr viele Pfeile weg, da wie gesagt hier die GUI, die Geräte-Typen (Tabs), die Schnittstellen, die Widgets der Geräte und auch die Verbindung zu Multilog und dem Gamepad erstellt werden. Die Geräte wurden hierbei zusammengefasst. Aus dem Grund finden sich hier die Funktionen wieder, die einerseits durch die *sampler*-Funktion angesprochen werden und andererseits die Funktionen, die bei allen Geräte-Widgets identisch sind. Diese Funktionen werden z.B. durch die Methoden vom *Controller*-Objekt angesprochen. Diese Verbindungen sind unten rechts in der Abbildung zu sehen.

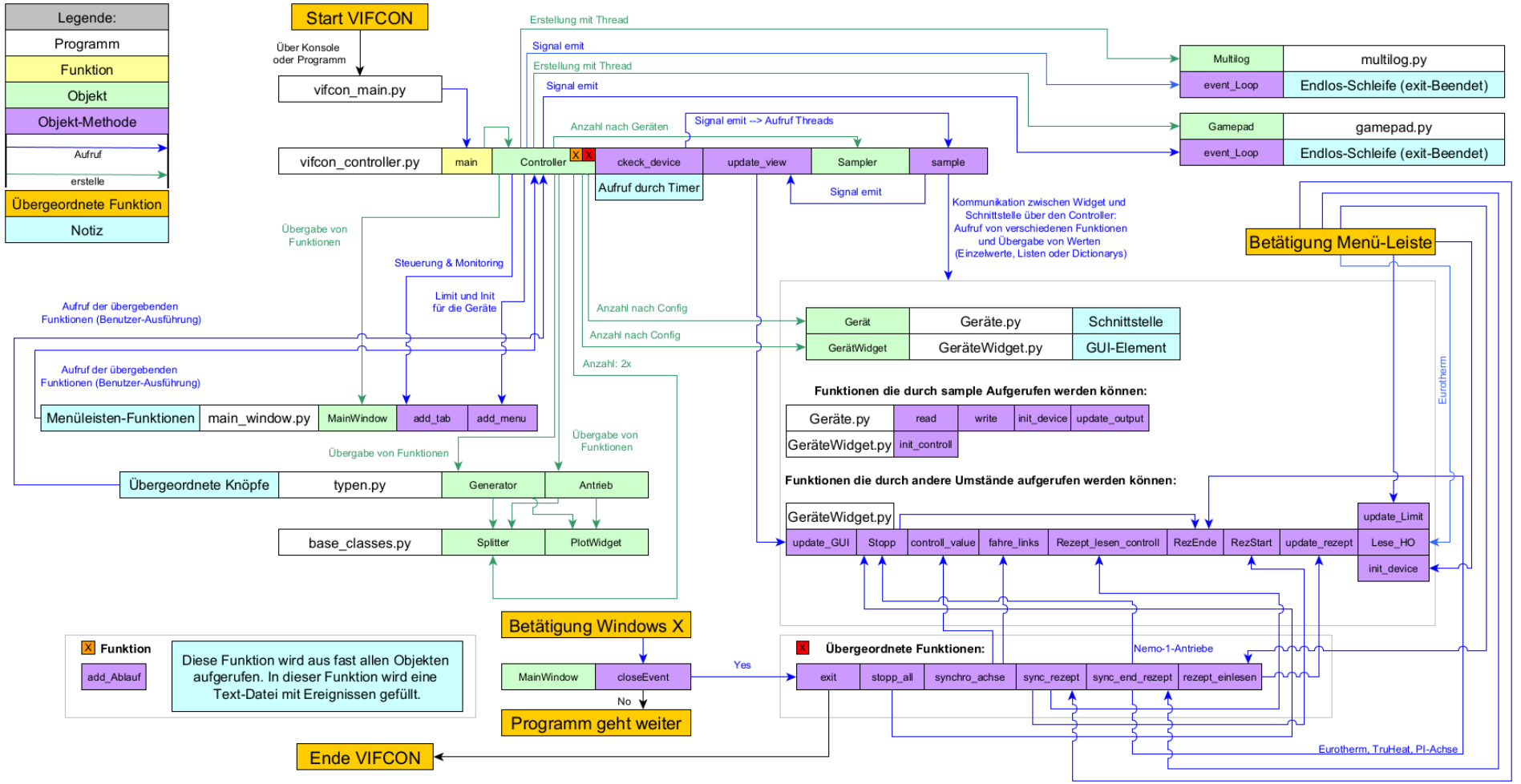


Abbildung 49: Darstellung der ungefähren Kommunikation in VIFCON 1 (Quelle: eigene Darstellung)

Mit Abbildung 50 soll nun auf die Kommunikation zwischen Geräte Widget und der Schnittstelle eingegangen werden. Die Schnittstelle bzw. das Gerät entspricht dem Controller des MVC, das Geräte-Widget entspricht View und der VIFCON-Controller dem Modell. Die Geräte können unterschiedliche interne Funktionen haben, weshalb die Abbildung 50 nur den Eurotherm-Regler zeigt. Die Programme **vifcon_controller.py** (C) und die beiden **eurotherm.py** aus den Ordnern **view** (W) und **devices** (S) beinhalten die in der Abbildung gezeigten Funktionen.

Der Grundgedanke der Kommunikation beruht auf zwei Python-Dictionaries. Diese heißen *write_task* und *write_value*. In dem Task-Dictionary stehen alle möglichen Aufgaben, die das Gerät umsetzen kann. Beim Value-Dictionary werden dann die zu sendenden Werte eingetragen. Wichtig hierbei ist, dass diese Dictionaries nicht zu oft an unterschiedlichen Stellen gesetzt werden, da durch die Threads sonst Probleme auftreten. Die Funktion *sample* wird durch die Threads parallel ausgeführt. Heißt wird ein Knopf im Haupt-Loop betätigt, wenn die beiden Dictionaries gerade von *sample* bzw. dem Thread angesehen werden, kann die *write*-Funktion dann die Aufgabe auslösen. Die Aufgaben werden so gesetzt, dass sie einerseits nur im Fehlerfall zurückgesetzt werden und nach jeder Ausführung der Aufgabe direkt auf False gesetzt werden. Im Widget werden zwar If-Anweisungen ausgeführt, besonders bei den Rezepten, aber die jeweilige Aufgabe wird nur einmal gesetzt. In Abbildung 50 kann auf der rechten Seite der Widget Teil gesehen werden. In der Abbildung wurden die drei bzw. 4, wenn der Rezept-Timer mit gezählt wird, Auslöse-Kriterien mit gezeigt. Diese sind das Geräte-Widget, der übergeordnete Knopf der Generator-Seite und eine auf Eurotherm bezogene Menü-Beschreibung. All diese Punkte lösen die Funktion (Methoden) des Eurotherm-Widget-Objektes durch Knopfdruck oder Auswahl-Änderung aus. Auf der linken Seite kann zum einen der grobe Ablauf des Programms und zum anderen daneben der Aufruf der *sampler*-Funktion gesehen werden. Durch den Timer werden diese Schritte in eine Schleife getan. Die Punkte Initialisierung, Schreiben und Lesen lösen Funktionen aus. Die Kommunikation zwischen Widget und Schnittstelle erfolgt nur durch den Controller, der dem Widget Informationen entnimmt, wie die Dictionaries, und diese an die Schnittstelle übergibt. Die Schnittstelle arbeitet die Dictionaries in der write-Funktion ab. Wenn an der Schnittstelle Werte ausgelesen werden, werden in dem Schnittstellen-Objekt die Messdaten geupdatet (Update von Dateien) und die Daten an den Controller gesendet. Dieser übermittelt die Daten über *emit* an den Hauptloop (Methode *update_view*), welcher dann die GUI aktualisiert.

Für die anderen Geräte ist diese Vorgehensweise identisch. Die einzige Ausnahme ist das Gerät Nemo-Gase, welches nur ausliest und nicht schreibt. Der Unterschied bei den anderen Geräten zu Eurotherm kommt durch andere Dictionary-Einträge und somit verbundene Funktionen. Die gemeinsamen Funktionen sind in Abbildung 49 zu finden. Aus diesen beiden Abbildungen in dem Kapitel sollte die Kommunikation und die somit verbundene Programmstruktur deutlich werden.

Zuletzt soll noch gesagt werden, dass VIFCON eine Test-Funktion besitzt. In dieser wird der Aufbau der Schnittstelle verhindert und die write-Funktion wird übersprungen. Somit funktionieren die meisten Knöpfe nicht. Auch die read-Funktion wird übersprungen und durch Zufallszahlen ersetzt. Die Test-Funktion hat den Nutzen, dass ein neuer Nutzer einen ersten Eindruck der GUI bekommt.

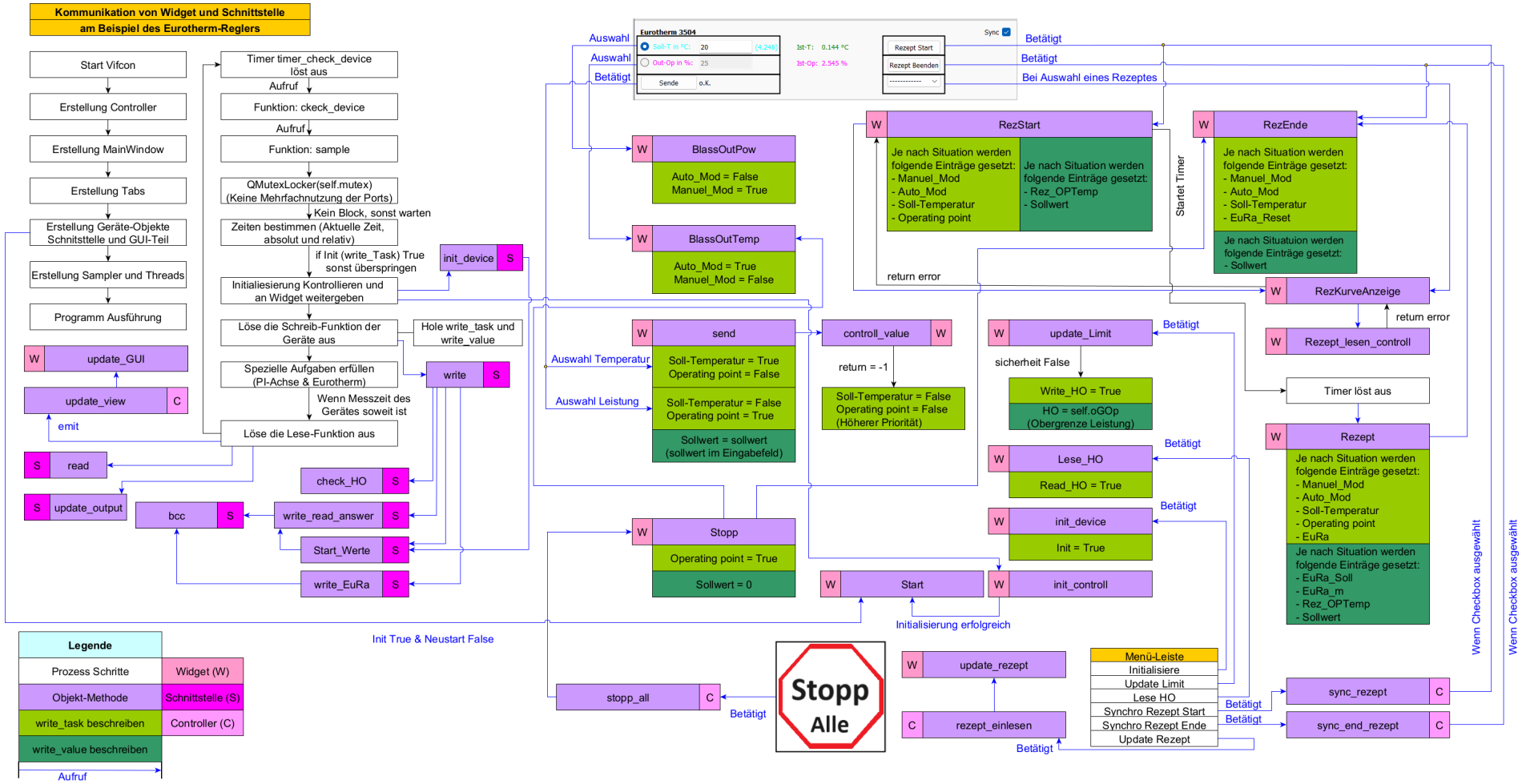


Abbildung 50: Darstellung der ungefähren Kommunikation in VIFCON 2 (Quelle: eigene Darstellung)

5.2 Threads und Loops

Nachdem VIFCON in Kapitel 5.1 erläutert wurde und der Aufbau bzw. die Struktur bekannt ist, sollen nun zwei der wesentlichsten Dinge erläutert werden. Hierbei handelt es sich um die Loops (Schleifen) und Threads. Wie schon erwähnt wird als Programm Python genutzt. Python liefert z.B. mit **tkinter** und **PyQt** Bibliotheken um GUI-Anwendungen zu erstellen. Auf diesen Oberflächen befinden sich dann verschiedenste Elemente wie Knöpfe (Buttons), Checkboxen und viele weitere Widgets. Die Aktionen auf der GUI, die durch den Benutzer getätigt werden, werden meist als Ereignis (Event) bezeichnet. Diese Ereignisse können somit verschiedenste Dinge auslösen. Zum Beispiel können Funktionen ausgeführt werden. Zum Thema GUI wird im späteren (Kapitel 5.9) mehr erläutert. Um auf diese Events zu reagieren benötigt es nun einen Loop, auch als Mainloop (Hauptschleife) oder Event-Loop (Ereignisschleife) bezeichnet. Aus dem Vorangegangen kann nun abgeleitet werden, dass diese GUI-Anwendungen ereignisgesteuert sind. Somit wird der Loop durch ein Event unterbrochen, es erfolgt ein Interrupt. Nach Abarbeitung des Events läuft der Loop weiter. Der Loop ist dabei endlos, bis die Anwendung geschlossen bzw. beendet wird. Weiterhin muss erwähnt werden, dass ein Betätigen von zwei Knöpfen in dem Hauptloop nicht beide zeitgleich ausführt. Der zuerst betätigte Knopf unterbricht den Loop und führt dessen Funktion aus. Die Funktion des zweiten Knopfes muss warten. Mit Beendigung des ersten Events startet das zweite. Die Abbildung 51 zeigt so einen Loop (Haupt-Loop) im allgemeinen. (Quelle: [Reaa])

Eine Steuerung die mit Geräten kommunizieren soll, könnte mit nur einem Loop nicht arbeiten. Wie erwähnt kann die Hauptschleife nur jeweils auf ein Ereignis gleichzeitig reagieren. Wenn z.B. die Funktion *sleep* aus der **time**-Bibliothek genutzt wird oder eine Funktion sehr viel Zeit benötigt, dann „friert“ die GUI-Anwendung ein. Die GUI würde dann so reagieren wie ein Programm, das keine Rückmeldungen mehr gibt. Die GUI wäre dann nicht mehr ansprechbar. Um das Problem zu lösen werden die sogenannten Threads verwendet. Hierbei muss gesagt werden, dass es auch einen Main-Thread (GUI Thread) gibt, in dem die Hauptschleife und die GUI ablaufen. Auch hier gibt Python bereits fertige Funktionen, Klassen und Bibliotheken. Da **PyQt5** verwendet wird, wird nicht die Python Bibliothek **threading** [Dokx], sondern die Klasse *QThread* verwendet. (Quelle: [Reab])

Mit Threads werden Programmteile so ausgelagert, dass diese gleichzeitig ausführbar werden. Somit erzeugt jeder Thread eine neue und separate Ausführung. Somit können mehrere Programmteile gleichzeitig wirken. Der Thread ist dabei eine Prozess-Komponente. Bei Prozessen können mehr als ein Thread simultan laufen. Neben diesem Umstand ist ein Prozess eine Instanz von Programmen oder Anwendungen, die in einem Computersystem ausgeführt werden. Das Problem bei den Threads ist, dass es sich um Multithreading handeln muss. Unter Multithreading ist gemeint, dass es möglich ist mehrere Threads nebeneinander laufen zu lassen. Nicht alle Systeme oder Programmiersprachen können dies umsetzen. **PyQt** ermöglicht dieses Multithreading durch das Objekt *QThread*, wodurch **PyQt** tatsächlich vorteilhaft ist. Bei **PyQt** werden der Main Thread, der schon erwähnt wurde, und der sogenannte Worker Thread verwendet. Die Worker Threads sind die Threads, die dann die Programmbearbeitung auslagern können. Neben diesem Umstand können diese mit Signalen und Slots arbeiten, die die Kommunikation mit dem Main-Thread ermöglichen. Bei den Threads handelt es sich somit wieder um eigene Event-Schleifen. (Quelle: [Reab])

Auch der Thread bzw. die Threads werden in Abbildung 51 dargestellt. In der Abbildung laufen mehrere Threads gleichzeitig neben der Hauptschleife. Bei den Threads wird noch etwas gezeigt. Da **PyQt5** verwendet wird, bezieht sich das Beispiel auch darauf. Bei der Verwendung von Threads können noch z.B. Locker verwendet werden. Dieser Umstand wird in Kapitel 5.4 näher erläutert.

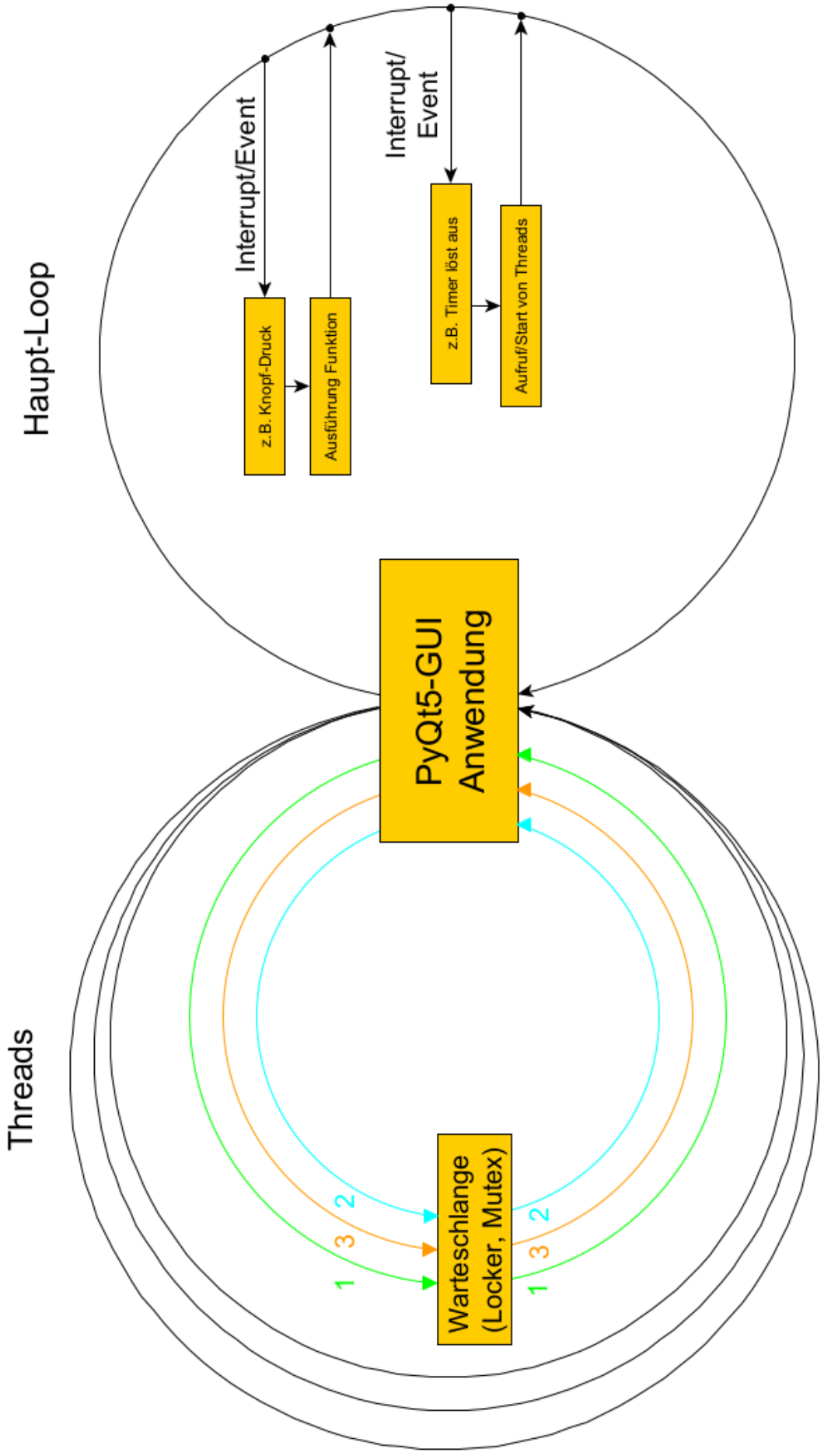


Abbildung 51: Darstellung von Loop und Thread Allgemein (Quelle: eigene Darstellung)

Nachdem der Loop und der Thread erläutert sind, muss geklärt werden, weshalb dies für VIFCON wichtig ist. VIFCON ist eine Steuerung für Anlagen. Neben dem Auslesen von Daten muss VIFCON auch Daten bzw. Befehle an die Geräte senden. Dabei muss beachtet werden, dass die Geräte diese Befehle so schnell wie möglich erhalten. So soll z.B. am besten eine Reaktionszeit von 150 ms vorliegen bzw. eingehalten werden. Durch die Threads wird es somit möglich, schnell zu reagieren. In VIFCON werden Threads für drei Programmteile gebraucht. Diese sind:

1. die Geräte,
2. das Gamepad und,
3. die Verbindung zu Multilog.

Jedes Gerät erhält einen eigenen Thread. Sollte die Schnittstelle bei Geräten identisch sein, so muss es noch einen Locker geben. Dieser wird in Kapitel 5.4 erläutert. Bei seriellen Schnittstellen kann immer nur eine Kommunikation stattfinden. Der folgende Code zeigt die Erstellung der Geräte-Threads. Auch für die anderen beiden Punkte ist die Erstellung sehr ähnlich, jedoch wird die angebundene Funktion in dem Thread nur einmal aufgerufen, da dort eine nahezu endlose while-Schleife ausgeführt wird. Bei den Geräte-Threads wird die *sample*-Funktion des *Sampler*-Objektes durch einen Timer immer wieder aufgerufen. Bei der Erstellung des Threads wird zunächst ein *QThread*-Objekt erstellt, welches der Worker Thread ist. Daraufhin wird ein Objekt (hier *Sampler*) erstellt, welches durch *moveToThread* mit dem erstellten Worker-Thread verknüpft wird. Durch die Funktion *start* wird der Thread-Loop gestartet. Neben diesem Umstand werden bestimmte Funktionen mit Signalen verbunden, die als Klassenattribute erstellt werden. Wenn die Funktion in dem Objekt liegt, das mit dem Thread verbunden ist, wird der Aufruf dieser Funktion in dem Thread durchgeführt, also neben der Hauptschleife.

```
1  class Controller(QObject):
2      # Teil hier nicht gezeigt
3      def __init__(self, config, output_dir, test_mode, neustart) -> None:
4          # Teil hier nicht gezeigt
5          self.samplers = []
6          self.threads = []
7          for device in self.devices:
8              thread = QThread()
9              dev_mutex = ...
                   self.mutexs[self.config['devices'][device]['serial-interface'] ...
                   ['port']]
10             sampler = Sampler(self.devices[device], device, ...
                   self.widgets[device], self.main_window.device_action[device], ...
                   self.start_time, test_mode, dev_mutex)
11             sampler.moveToThread(thread)
12             sampler.signal.connect(self.update_view)
13             self.signal_sample_main.connect(sampler.sample)
14             self.samplers.append(sampler)
15             self.threads.append(thread)
16         # Teil hier nicht gezeigt
```

Die Abbildung 52 zeigt nun ein spezielleres Beispiel für die Loops und Threads. Auch die jeweiligen Start-Code-Zeilen sind dort zu finden. In dem Beispiel werden nur die Nemo-1-Geräte betrachtet. In der Abbildung kann auch die Verbindung der Schleifen gesehen werden. Wenn der Sample-Timer auslöst, wird die Funktion *ckeck_ device* in der Hauptschleife ausgeführt, wodurch die Funktion *sample* in allen Geräte-Threads gestartet wird. Andersherum kann gesehen werden, dass am Ende von *sample*, die Funktion *update_ view* in der Hauptschleife aufgerufen wird und dort für ein Event sorgt. Auf die linke Seite in der Abbildung wird in Kapitel 5.4 eingegangen. Der Umstand der Verbindung kommt durch die erwähnten Signale, die wie erwähnt die Kommunikation zwischen den Loops ermöglichen.

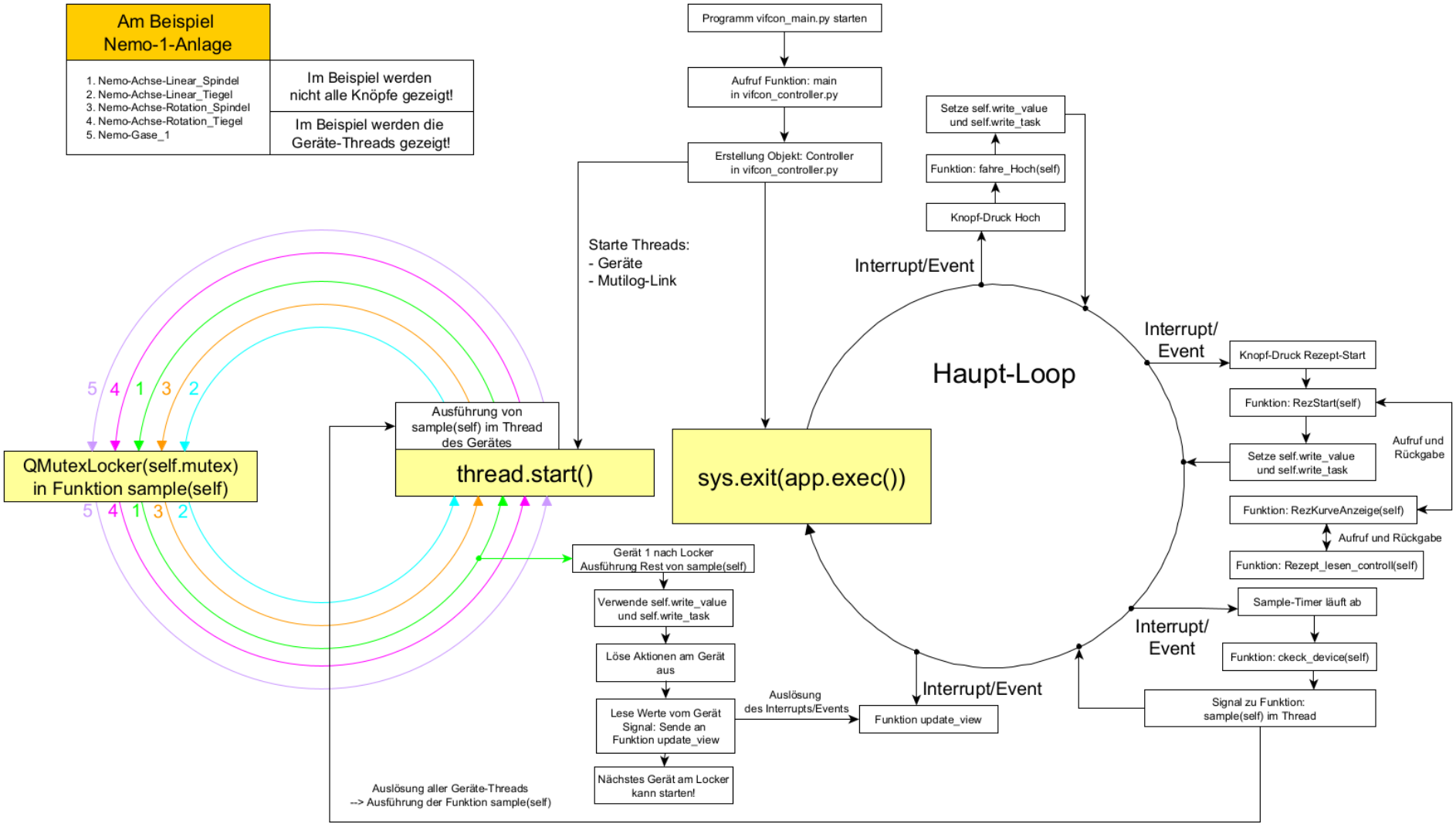


Abbildung 52: Darstellung von Loop und Thread VIFCON (Quelle: eigene Darstellung)

5.3 Geräte und Schnittstellen

In Kapitel 3 werden alle Geräte für VIFCON vorgestellt. Um noch einmal einen Rückblick auf die Geräte zu geben, werden diese folgend gezeigt. Mit VIFCON können die Geräte:

1. Eurotherm (Steuerung, Generator (Regler)) (Kapitel 3.1),
2. TruHeat (Steuerung, Generator) (Kapitel 3.3),
3. PI-Achse (Steuerung, Antriebe) (Kapitel 3.2),
4. Nemo-1-Anlage Hub (Steuerung, Antriebe) (Kapitel 3.4),
5. Nemo-1-Anlage Rotation (Steuerung, Antriebe) (Kapitel 3.4) und,
6. Nemo-1-Anlage Gase (Monitoring) (Kapitel 3.4)

in die Steuerung eingebunden werden. Neben diesen 6 Geräten, verfügt VIFCON noch über einen Anschluss eines Gamepads (Kapitel 4.4.2 und 5.6) und eine Verbindung zu Multilog (Kapitel 4.4.3 und 5.7). Das Grundlagen Kapitel 2.3 erläuterte die RS232-Schnittstelle und das Modbus-Kommunikationsprotokoll. Der Eurotherm-Regler, der TruHeat Generator und die PI-Achse mit ihrem Mercury-DC-Motorcontroller werden über die RS232-Schnittstelle betrieben. Alle Geräte die mit der Nemo-1-Anlage verbunden sind, werden über Modbus TCP und somit Ethernet betrieben. Dabei wird eine Kommunikation über die SPS (Abbildung 36) erstellt. Bei der Multilog-Verbindung mit VIFCON wird eine ähnliche Verbindung wie bei der Nemo-1-Anlage verwendet. Aus den Grundlagen ist bekannt, dass Modbus, TCP und IP Ebenen der Kommunikation sind. Bei der Multilog-Verbindung wird nun nur noch die TCP/IP-Verbindung verwendet. Bei dem Gamepad wird ein USB-Anschluss verwendet. Im Sinne der Schnittstelle muss der Anwender nichts beachten, da die **pygame**-Bibliothek von Python alles diesbezüglich regelt. Wie die Schnittstellen im Code definiert werden, kann bei Interesse in Anhang H.1 nachgelesen werden.

5.4 Mehrfachnutzung von Schnittstellen

In der VIFCON-Steuerung soll es möglich sein, mehrere Geräte über die selbe Schnittstelle zu nutzen. Für diese Zwecke müssen die Überlegungen zur Schnittstellen Definition (Anhang H.1) und zu den Threads und Loops (Kapitel5.2) angesehen und überarbeitet werden. Da serielle Schnittstellen benutzt werden, ist es nicht möglich mehr als einen Befehl gleichzeitig zu senden. Doch durch die Threads kann es passieren, dass z.B. drei PI-Achsen gleichzeitig senden wollen. Mit diesem wissen müssen nun zwei Dinge beachtet werden. Zum einen dürfen die Schnittstellen nicht öfters als Instanz von der Bibliothek **serial** oder **pyModbusTCP** erstellt werden und andererseits darf die Ausführung der Threads bzw. des Inhalts der Threads nicht gleichzeitig geschehen.

Eine Anwendung die Threads benutzt, kann auf das Problem stoßen, dass die Threads auf bestimmte Ressourcen (z.B. Instanzen, Variablen) gemeinsam zugreifen müssen. Somit müssen sich Gedanken gemacht werden wie diese geregelt wird. Hierbei gibt es die Methoden:

1. Verhinderung und,
2. Organisation des Zugriffs, welches sich in:
   (a) Atomic operations und,
   (b) Mutual exclusion (Abkürzung: Mutex)

aufteilt. Die erste Methode beschreibt dabei, dass die gemeinsame Nutzung von Ressourcen verhindert wird. Somit soll jeder Thread unabhängig von den anderen sein. Diese Methode funktioniert in VIFCON nicht, da manche Schnittstellen mit mehr als einem Gerät verwendet werden. Somit muss der Zugriff der Threads auf diese Ressource (Schnittstelle) so organisiert werden, dass diese sich nicht beeinträchtigen. Der erste Ansatz in dieser Methode wäre die Atomic operations. Diese beschreibt eine Ausführung in einem Ausführungsschritt, sodass dieser einzige Schritt nicht durch die anderen Threads unterbrochen werden kann. Dabei wird somit sichergestellt, dass nur ein Thread die Ressource nutzt. Da in VIFCON die Ressource (Schnittstelle) mehrfach genutzt wird und somit viele Teiloperationen am Stück durchgeführt werden, macht eine komplette Verriegelung des Code-Abschnittes mehr Sinn. Diese Verriegelung wird durch die Mutual exclusion geliefert. Dabei werden die Threads so organisiert, dass immer nur einer auf die Ressource zugreifen kann. Somit wird in VIFCON genau dieses Prinzip genutzt, da die Nutzung der Schnittstellen somit gesperrt werden kann. In **PyQt5** werden dabei die Objekte *QMutex* und *QMutexLocker* verwendet. Als Nachteil für die Mutal exclusion muss gesagt werden, dass solche Sperren die Leistung der Anwendung beeinträchtigen können, da Threads nun warten müssen, bis die Ressource wieder freigeschaltet wird. Dies ist bei VIFCON nicht so verheerend, da nur wenige Geräte interne Delays haben und in einem Experiment nicht alle Geräte durch den Mutex gesperrt werden. Als Alternativen bietet PyQt zu der Mutal exclusion Technik noch Objekte wie *QReadWriteLock* und *QSemaphore*. Diese werden folgend erläutert. (Quelle: [Reab])

Bei *QReadWriteLock* wird eine Blockade zwischen lesenden und schreibenden Zugriff erstellt. Somit kann dieses Objekt einen Schreibschutz auslösen, wodurch ein Thread die Ressource beschreibt und die anderen warten bis dieser fertig ist. In VIFCON gibt es getrennte Funktion für Schreiben und Lesen, doch können manche Geräte in der Schreib-Funktion auch einen Lese-Befehl ausführen. Da die Schreib-Funktion immer vor der Lese-Funktion kommt, hat sich *QReadWriteLock* nicht angeboten. Das *QMutex*-Objekt bildet eine einfachere Version von *QReadWriteLock* an. *QMutex* erzeugt nur eine Sperrung der Ressource und sorgt dafür, dass die Threads nicht gleichzeitig zugreifen können. Somit wird hier kein Unterschied zwischen Schreiben und Lesen getroffen. *QSemaphore* ist auch wieder ein Sonderfall von *QMutex*. Hierbei wird eine bestimmte Anzahl von gleichen Ressourcen geschützt. Die Funktionsweise ist folgende: Die Semaphore schützt dabei n Ressourcen. Sobald n+1 Ressourcen gesperrt werden sollen, wird die Semaphore blockiert, was den Zugriff auf die Ressourcen durch die Threads verhindert. Für VIFCON reicht die einfache Lösung durch *QMutex*, weil die Geräte immer Zugriff haben sollen, egal wie viele Geräte es sind. Auch benötigt es mehrere Mutex-Locker, was mit *QMutex* sehr leicht umzusetzen war. (Quelle: [Reab])

In Anhang H.2 wird neben der Instanziierung des *QMutex*-Objektes auch die Weiterentwicklung der Schnittstellen Definition gezeigt. Die Weiterentwicklung dieser beinhaltet eine Verhinderung der erneuten Erstellung der Schnittstellen-Instanz. Somit wird bei gleicher Schnittstelle zwischen Geräten, die Instanz für alle Geräte übernommen. Diese Instanzen bzw. die Port-Bezeichnungen werden gespeichert und bei jedem Gerät geprüft. Weiterhin wird jeder Port-Bezeichnung genau eine Instanz von *QMutex* zugewiesen. Im Folgenden wird nun auf die Realisierung des *QMutex* in VIFCON eingegangen.

Um die Umsetzung des Mutex zu zeigen werden im folgenden Bilder gezeigt. Diese Bilder wurden anhand von Logging-Nachrichten (siehe Code hier drunter) erstellt. Der folgende Code zeigt nun auch die Anwendung von *QMutexLocker*. Immer wenn die Threads aufgerufen werden, wird die *sample*-Funktion aufgerufen. Diese Funktion beinhaltet nun die zu schützenden Ressourcen. Der Mutex ist eine Instanzvariable der besagten Funktion.

```
1 def sample(self):
2     logging.debug(f"{self.device_name} - Sampler-Funktion vor Locker!")
3     with QMutexLocker(self.mutex):
4         logging.debug(f"{self.device_name} - Sampler-Funktion nach Locker! ...
              Locker {self.mutex} aktiv!")
5         # Teil hier nicht gezeigt
6     logging.debug(f"{self.device_name} - Sampler-Funktion fertig! Locker ...
          wieder freigegeben!")
```

Die drei Logging-Nachrichten in dem Code sollen hier bei die Funktion des Mutex und des dazugehörigen Mutex-Lockers beschreiben.Um es in Worte zu fassen: Wenn angenommen drei Geräte die selbe Schnittstelle haben und die Threads zeitgleich gestartet werden, so wird die erste Nachricht bei allen drei Geräten notiert (Vor-Locker). Die Reihenfolge der Bearbeitung der Geräte wird dabei nicht durch VIFCON sondern durch den Computer und dessen Prozessor bestimmt. In der Log-Datei kann nun gesehen werden, dass eins der Gerät hinter den Locker gelangt ist. Die anderen müssen nun warten. Sobald die Nachricht „Sampler-Funktion fertig!“ erscheint, wird das nächste Gerät den Code ausführen können. In dieser Zeit kann es jedoch auch schon sein, dass die Threads erneut aufgerufen werden und das erste Gerät wieder vor den Locker gelangt. Dieser Thread steht jedoch nun in der Warteschlange, hinter den noch auszuführenden Threads. Somit reihen sich die Threads ein.

In VIFCON liegt dieser Zustand der mehrfach Nutzung von Schnittstellen bisher nur bei der PI-Achse und der Nemo-1-Anlage vor. Die Funktion ist auch bei den anderen Geräten nutzbar. Dem entsprechend sollen nun ein paar Bilder gezeigt werden, die den Mutex gut zeigen. Die folgenden Abbildungen wurden dabei anhand des Zeitstempels und des Textes der Logging-Nachricht erstellt. Der Zeitwert wurde anhand dieser Zeitstempel berechnet. Wie die Nachrichten in der Log-Datei aufgebaut sind, kann dem Anhang C entnommen werden.

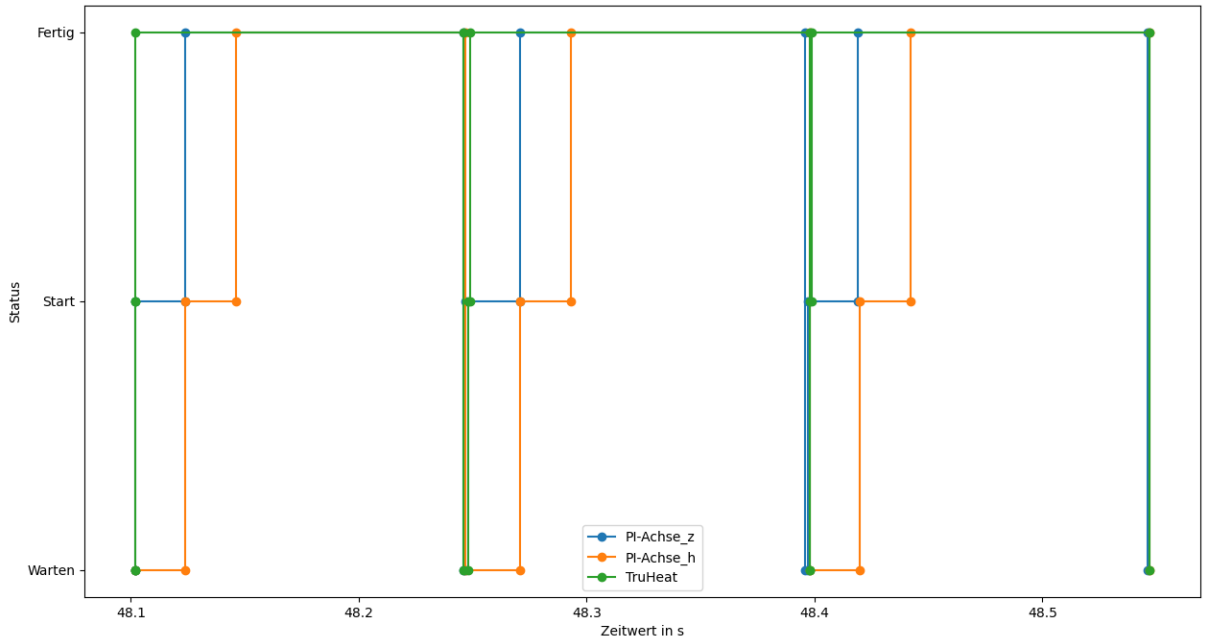


Abbildung 53: Darstellung des Mutex und des Mutex-Locker 1 (Quelle: eigene Darstellung)

Die Abbildungen 53 und 54 wurden aus den Daten des Experimentes erzeugt, welches in Kapitel 6.1 noch vorgestellt wird. Genutzt bzw. gezeigt werden in der Abbildung die Geräte PI-Achse (2x) und TruHeat-Generator. Bei den PI-Achsen wird die selbe Schnittstelle genutzt, während der TruHeat seine eigene hat. In den Abbildungen können nun die Zustandswechsel anhand der Vertikalen gesehen werden. Diese vertikalen Linien sind dabei nicht vollkommen vertikal. Wenn sich der Code oben angesehen wird, dann gibt es immer noch Zeilen zwischen den Logging-Ereignissen. Wenn z.B. nichts gelesen oder geschrieben wird in der Anwendung des Threads, so kann es sein, dass das jeweilige Gerät extrem schnell von Zustand „Warten“ zu Zustand „Fertig“ springt. Dieser Umstand ist sehr gut in Abbildung 54 zu sehen. Da der TruHeat die Schnittstelle alleine nutzt und z.B. nichts gesendet werden muss, kann dieser sehr schnell durch den Code springen. Wenn sich nun die allgemein eingestellte Auslesezeit der Geräte von 2 s (einstellbar in Config) angesehen wird, kann in der Abbildung 54 gut erkannt werden, dass alle 2 s der Zustand 101a eine längere Zeit aktiv ist. Dies kommt daher, dass die *read*-Funktion, die alle 2 s aufgerufen wird, des TruHeats beim auslesen auch Delays verwendet. Abbildung 53 zeigt nun noch den anderen Fall. Wenn mehrere Geräte die selbe Schnittstelle nutzen, so muss eins der Geräte immer warten. In dem Fall der Abbildung wird immer die PI-Achse z (blau) vor der PI-Achse h (orange) ausgeführt. Der TruHeat (grün) läuft parallel dazu.

Auch der Aufruf der Threads wird durch die Config-Datei eingestellt. Hierbei ist der Reaktionstimer gemeint. Dieser ist allgemein auf 150 ms eingestellt. Alle Geräte-Threads werden somit nach dieser Zeit aufgerufen und vom Mutex-Locker kontrolliert. Je mehr Geräte an der Schnittstelle hängen, desto eher werden sie nicht in den 150 s abgearbeitet. Dies tritt spezielle dann noch auf, wenn die Geräte Verzögerungen brauchen. Z.B. braucht die PI-Achse bei einem der Controller (siehe Kapitel 3.2.6) eine bestimmte Zeit um die Geschwindigkeit zu messen. Wenn nun 4 dieser Achsen genutzt werden, kann es sein, dass eine abgearbeitete Achse schon wieder auf die nächste Abarbeitung wartet.

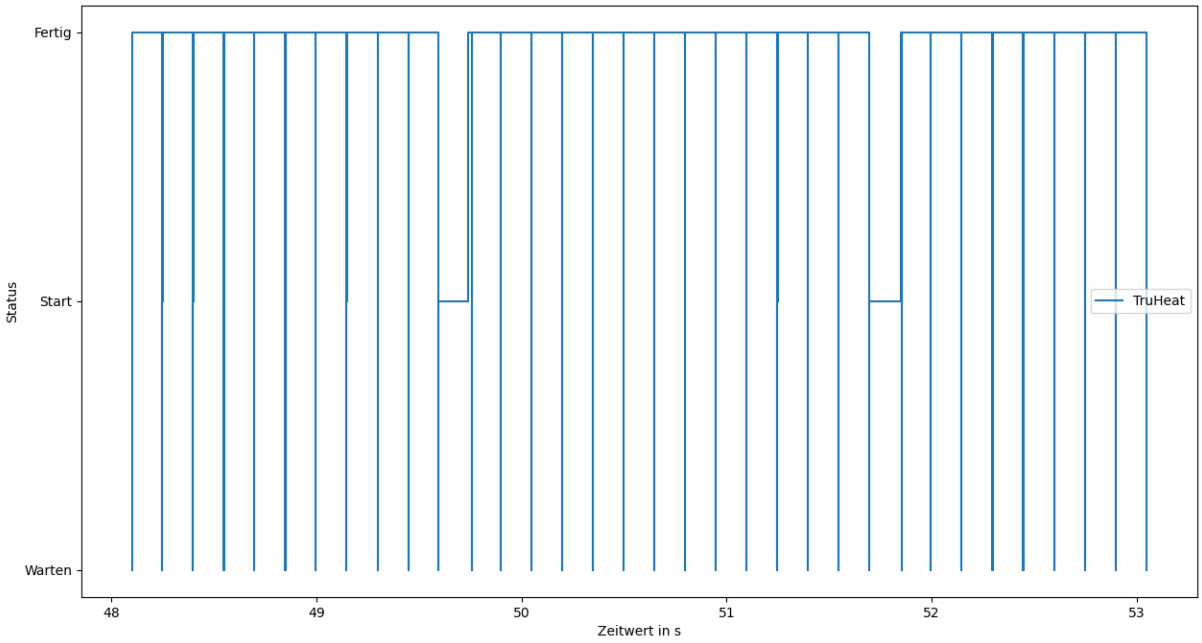


Abbildung 54: Darstellung des Mutex und des Mutex-Locker 2 (Quelle: eigene Darstellung)

Die Abbildung 55 zeigt nun die Nemo-1-Anlage und alle 5 Geräte die diese besitzt. Bei den ersten 150 ms scheint es so als würden die Threads zweimal aufgerufen werden, was auch der Fall ist. Folgend sind die ersten drei Auslösungen der Funktion *ckeck_ device* zu sehen, die die folgend gezeigte Nachricht loggen. Aber hierbei ist nun die andere Funktion des Mutex zu sehen. Die Nemo-Achse Linear Spindel (blau) kann erst wieder starten, wenn Nemo-Gase (lila) zu ende ist. Das heißt, dass die Geräte die bereits Warten Priorisiert werden, als die die folgend erst hinzukommen.

```
1 2024-02-14 12:57:10,172 DEBUG root - Aufruf der Threads!
2 2024-02-14 12:57:10,189 DEBUG root - Aufruf der Threads!
3 2024-02-14 12:57:10,348 DEBUG root - Aufruf der Threads!
```

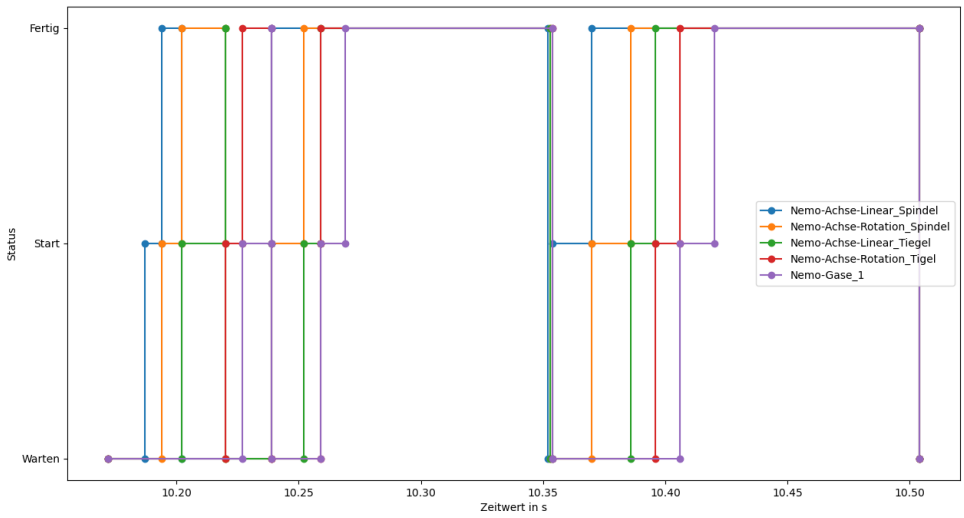


Abbildung 55: Darstellung des Mutex und des Mutex-Locker 3 (Quelle: eigene Darstellung)

5.5 Rezepte

In diesem Kapitel soll die Rezept-Funktion näher erläutert werden. Dafür wird das Konzept der Rezepte in Kapitel 5.5.1 und die einzelnen Rezeptsegmente in Kapitel 5.5.2 vorgestellt. Bei näherem Interesse zur Programmierung und den Rezept-Funktionen, kann sich Anhang F angesehen werden.

5.5.1 Konzept für Rezepten

Die Rezepte sind eine der wesentlichen Funktionen von VIFCON. Bei der Fluorid-Gruppe zeigt die GUI die einzelnen Züchtungsschritte. In Abbildung 7 kann der Schritt Züchtung gesehen werden. Auch hier wird ein Rezept abgearbeitet. Dies wurde in Kapitel 2.1.3 erläutert. Die Rezepte von VIFCON sind auf alle Geräte die in Kapitel 3 erwähnt wurden anwendbar. Nur das Monitoring Gerät Nemo-Gase hat diese Funktion nicht, da es nicht zur Steuerung gehört.

Ein Ziel dieser Arbeit ist die Übernahme der Kristallzüchtung durch die Steuerung. Am Ende soll der Anwender nun noch die Funktion und die Zucht Überwachen. Der Anwender konfiguriert das Programm und bereitet die Züchtung vor. Sobald der Schritt der Zucht kommt, startet der Anwender ein Feature von VIFCON durch einen Knopf und der Kristall wird gezüchtet. Diese Funktion liefern die Rezepte. Durch bestimmte Funktionen, welche in Anhang F.1 erläutert werden, lassen sich die Rezepte für alle Geräte synchron starten. Somit können z.B. der Eurotherm-Regler und alle 4 Achsen der Nemo-1-Anlage auf einmal betrieben werden. Um ein Rezept zu erzeugen muss dieses entweder direkt in die Config-Datei unter dem Schlüssel „rezepte“ geschrieben werden oder diese werden in eine Datei ausgelagert. Folgend wird ein Beispiel aus der Config gezeigt.

```
1     rezepte:
2       rezept_Syn:
3         n1: 1 ; -2 ; s
4         n2: 4 ; -1 ; s
5         n3: 1 ; -0.5 ; s
6         n4: 1 ; 1 ; s
7         n5: 5 ; 2 ; s
8       rezept_2:
9         dat: rezept_test_3.yml
```

Der Aufbau im Sinne der Rezepte wird in Kapitel 5.5.2 erläutert. Der Aufbau in der Config wird hier kurz erläutert. Eine größere Erläuterung der Config-Datei erfolgt im Anhang B. VIFCON ruft über den Schlüssel „rezepte“ das in diesem liegende Dictionary aus. Wenn z.B. „rezept_Syn“ (ein selbst gewählter und änderbarer Schlüssel) gewählt wird, wird das in diesem liegende Dictionary als Rezept eingelesen, im Plot angezeigt und mit Start ausgeführt. Durch das Auslagern wird die Config-Datei übersichtlicher. Um ein Rezept auszulagern, muss als Rezept ein Schritt programmiert werden der als Schlüssel „dat“ beinhaltet. Der Wert zu diesem Schlüssel ist dann die Yaml-Datei, als String.

5.5.2 Aufbau und Segmente

In dem vorherigen Kapitel konnte ein Beispiel für die Programmierung der Rezepte in der Config-Datei gesehen werden. Dabei wird auch ein vollständiges Rezept gezeigt. Jeder Rezept-Schritt ist ein einzelnes Dictionary, was durch einen Schlüssel (hier z.B. n1, n2, etc.) eingeleitet wird. Bei diesen Schritten ist nur wichtig, dass sie nicht identisch sind oder „dat“ heißen. Sonst können sie benannt werden wie sie wollen. Der Wert in jedem der Dictionaries ist ein String. Wichtig dabei ist, dass die einzelnen Komponenten durch ein Semikolon getrennt sind und die Segmentarten klein geschrieben sind. Bei den Zahlen kann sowohl Komma als auch Punkt genutzt werden. Bei den Geräten bildet nur der Eurotherm-Regler eine Ausnahme, da dieser 3 Segmentarten mehr hat. Somit hat VIFCON die Segmentarten:

1. s - Sprung,
2. r - Rampe (Stufen),
3. er - Eurotherm Rampe (Geräte eigene, Linear) (Kapitel 3.1.6.3),
4. op - Ausgangsleistungssprung im Eurotherm-Temperaturrezept und,
5. opr - Ausgangsleistungsrampe im Eurotherm-Temperaturrezept.

Die Segmente er, op und opr existieren nur beim Eurotherm-Regler. Alle anderen Geräte verwenden s und r als Segmente. Wichtig hierbei ist noch zu wissen, dass bei TruHeat und Eurotherm verschiedene Größen angesteuert werden können. So kann z.B. bei Eurotherm ein Temperatur-Rezept oder ein Leistungsrezept gestartet werden. Bei dem reinen Leistungsrezept werden op und opr nicht verwendet, sondern s und r. Wenn in einem Temperatur-Rezept auch Leistungsänderungen im Manuellen-Modus gebraucht werden, gibt es dafür op und opr. Folgend werden diese Segmente mit Beispielen erläutert.

Sprung (s):

Bei einem Sprung wird der Sollwert schlagartig auf einen anderen geändert. Zum Beispiel kann somit der Sollwert von 20°C auf 100°C geändert werden. In VIFCON wird dies wie folgt programmiert:

```
1     n1: 20 ; 100 ; s
```

In dem Rezept wird der Sollwert auf 100 gesetzt und 20 s gehalten. Somit gibt der erste Wert die Zeit für das Segment an, der zweite den Sollwert und der dritte die Segmentart.

Rampe (r):

Die Rampe ist eine eigen erstellte Rampe, die aus vielen Schritten besteht. Mit diesem Segment wurde die erste Verbesserung der Rezepte gesetzt. Zuvor musste eine Rampe durch hunderte einzelne Sprünge programmiert werden, wodurch die Dateien oder die Config-Datei noch länger wurde. Mit der Rampe wird dies durch eine Zeile ersetzt. Die Rampe hat folgendes Aussehen:

```
1     n1: 1200 ; 200 ; r ; 3
```

In dem Beispiel dauert die Rampe 1200 s und steigt bis 200 an. Die ersten drei Teile des String geben somit die Segmentzeit, den Zielsollwert und die Segmentart an. Nun kommt noch ein vierter Wert dazu. Dieser Schritt gibt den Zeitabstand für die Sprünge an. Im Beispiel wird nun angenommen das die Rampe bei 500 startet. Somit ergibt sich folgende Rechnung in VIFCON:

$$\text{Rampenbereich} = 200 - 500 = -300 \tag{5.1}$$

$$\text{Rampenschritt} - \text{Anzahl} = \frac{200\,s}{3} = 400 \tag{5.2}$$

$$\text{Sollwert} - \text{Sprung} - \text{Höhe} = \frac{\text{Rampenbereich}}{\text{Rampenschritt} - \text{Anzahl}} = \frac{-300}{400} = -0,75 \tag{5.3}$$

Diese drei Rechnungen werden auch im Code durchgeführt. Somit ändert sich alle 3 s, der Wert des Sollwertes um -0,75, was ein sinken des Wertes zeigt. Der letzte Wert in dem String verursacht somit die Steigung der Rampe.

Eurotherm-Rampe (er):

Die Nutzung der Eurotherm-Rampe wurde durch die Experimente aus Kapitel 6 gefördert. In diesen wird gezeigt, dass der PID-Regler des Eurotherm mit den Sprüngen der Rampe r nicht gut klarkommt. Somit sollte die eigene Funktion über die Schnittstelle ausführbar sein. Somit wurde das Segment er erstellt. Dieses sieht wie folgt aus.

```
1      n1: 3600 ; 500 ; er ; 0.133
```

Der Aufbau ist dem vom Segment r sehr ähnlich. Der erste Wert gibt wieder die Segmentzeit an, der zweite den Zielsollwert, der dritte die Segmentart und der vierte Wert gibt diesmal die tatsächliche Steigung an. Über die Schnittstelle werden hierbei nur der Zielsollwert und die Steigung gesendet. In VIFCON ist diese Funktion so programmiert, dass nur eine Rampe durchführbar ist. Im Original könnte ein komplettes Rezept für Eurotherm einprogrammiert werden. Aus dem Grund ist das zweite Eurotherm-Rezept Segment so programmiert das, dass Eurotherm-Rezept endet. Mit dem Sprung zu einem anderen VIFCON-Segment oder dem Rezeptende in VIFCON wird der aktuelle Sollwert gesendet und das Eurotherm-Rezept (intern) zurückgesetzt und somit beendet. Das Problem mit dieser Rampe ist, dass der Sollwert ohne Änderung durch VIFCON, den Wert vor dem Eurotherm internen Rezept annimmt. In dem Beispiel wird in 3600 s ein Temperaturwert von 500°C mit einer Steigung von 0,133°C/s angefahren.

Eurotherm: Ausgangsleistungssprung im Temperaturrezept (op):

Diese und auch das Segment opr sind notwendige Erweiterungen gewesen. Bei einem Rezept muss es möglich sein, sowohl Leistung als auch Temperatur zu regeln. Diese Funktion sollte auch in den Rezepten vorhanden sein. Mit dem Segment op wird während eines Temperatur-Rezeptes, ein Modus Wechsel ausgeführt. Die Kontrolle der Ausgangsleistung ist nur in dem Manuellen Modus von Eurotherm möglich. Auch hier soll dies anhand eines Beispiels gezeigt werden:

```
1      n1: 600 ; 500 ; op ; 20
2      n2: 600 ; 500 ; op ; IST
```

In dem Beispiel wird das Segment für 600°C ausgeführt. Der zweite Wert löst einen Sollwertsprung auf 500°C aus, während der dritte wieder die Segmentart angibt. Der vierte Wert gibt hier den Leistungssprung an. Der Leistungswert springt also auf 20 %. Als alternative kann auch der String „IST“ hier angegeben werden. Wenn dies der Fall ist, so wird nur auf den manuellen Modus gewechselt. Durch diesen Umstand wird der aktuelle OP-Wert (Leistungswert) festgehalten und ändert sich nicht mehr. Somit wird die Temperaturregelung ausgestellt und die Regelung der Leistung läuft über den eingestellten Wert.

Eurotherm: Ausgangsleistungsrampe im Temperaturrezept (opr):

Das letzte Segment ist die Leistungsrampe im Temperaturrezept. Wie der Leistungssprung, wird bei Auswahl in den Manuellen-Modus gewechselt. Die Rampe die hier ausgeführt wird, ist eine Rampe von Typ r. Somit wird hier die selbe Rechnung wie dort durchgeführt. Auch hierfür ein Beispiel:

```
1      n1: 600 ; 200 ; opr ; 5 ; 1 ; 20
2      n2: 600 ; 200 ; opr ; 5 ; 1
```

Wie bei allen anderen Segmentarten, gibt der erste Wert die Segmentzeit an, der zweite sorgt für einen Temperatursollwertsprung, der dritte zeigt die Segmentart, der vierte nennt die Zielausgangsleistung, der fünfte gibt den Zeitabstand für den Sollwertsprung und der sechste den Startwert an. Der letzte Wert ist dabei sehr wichtig, da ein Modus-Wechsel durch z.B. Oszillation des Leistungswertes keinen festen Wert vorgeben kann. Aus dem Grund muss der Anwender hier einen Wert angeben. Wenn dieser fehlt, so wird die Rampe alternativ von Null gestartet. Wie erwähnt sollen die Segmente op und opr die Nutzung der Leistungssteuerung in einem Temperaturrezept möglich machen. Ein Anwendungsfall kann im dritten Experiment (Kapitel 6.3) gesehen werden. In diesem wird die Tiegel-Rotation aktiviert, weshalb die Temperaturregelung abgestellt werden muss, da der Temperatursensor für die Regelung in diesem Tiegel steckt, Somit kann die Ausgangsleistung konstant gehalten werden und die Heizleistung des Generators läuft wie zuvor weiter.

5.6 Joystick/Gamepad

Das Gamepad ist ein kleines Extra von VIFCON. Neben dem in folgenden Kapitel 5.7 beschriebenen, stellen diese beiden Teile ein abschaltbares Feature von VIFCON da. Über das Gamepad wurde bereits in Kapitel 4.4.2 erwähnt. Das Gamepad wird mit der Python-Bibliothek **pygame** betrieben. Die Aufgabe des Gamepads ist die Ansteuerung der Antriebe. Somit sollen die Nemo-Achsen und auch die PI-Achse darüber bewegbar sein. Das Konzept für die PI-Achse ist dabei ein Punkt der nicht mehr Teil dieser Arbeit ist und eine zukünftige Erweiterung von VIFCON sein wird.

Abbildung 56: Ansicht des Gamepads (Front, von oben) (Quelle: eigene Darstellung)

Die Nemo-1-Anlage verfügt über zwei Rotations-Achsen und zwei Hub-Achsen. Das Gamepad wurde in Abbildung 45 gezeigt und ist auch in Abbildung 56 zu sehen. Bei der Nemo-Achse werden die vier Knöpfe auf der rechten Seite des Gamepads genutzt und der Select-Knopf. Anders als bei der PI-Achse gibt es bei den Nemo-Antrieben nur 4 Bewegungsrichtungen. Diese sind Auf und Ab und CW und CCW. Für die PI-Achse wurden derzeitig nur der L-, R- und Select-Knopf verwendet. Die Zuordnung der Knöpfe zu den Achsen und deren Funktionen, kann in Tabelle 21 gesehen werden. Die Erklärung der zweiten Spalte der Tabelle kann Kapitel 4.4.2 entnommen werden. Auch spezielle Daten zu dem Gamepad sind dort vorhanden.

Tabelle 21: Zuordnung Gamepad-Knopf zum VIFCON-Gerät (Quelle: eigene Darstellung)

Knopf Gamepad	**pygame event.button**	**Gerät**	**Funktion**
Y	3	Nemo-Achse-Rotation	*fahre_ cw*
X	0	Nemo-Achse-Linear	*fahre_ Hoch*
A	1	Nemo-Achse-Rotation	*fahre_ ccw*
B	2	Nemo-Achse-Linear	*fahre_ Runter*
SELECT	8	Nemo-Achsen, PI-Achse	*Stopp*
START	9	/	/
L	4	PI-Achse	*fahre_ links*
R	5	PI-Achse	*fahre_ rechts*

Aus dem Kapitel 5.2 wurde deutlich, dass das Gamepad einen eigenen Thread bekommt. Dort wurde auch die Erstellung der Threads gezeigt. Aus dem Grund werden im Folgenden nur die 4 wichtigsten Code-Zeilen für das Gamepad gezeigt. Dabei ist das Signal (Zeile 3), die Thread-Instanz (Zeile 6), die Gamepad-Instanz (Zeile 8) und die Funktion die am Signal hängt (Zeile 10). Die beiden Instanzen werden durch *moveToThread* verbunden. Somit wird die Funktion *event_ Loop* in dem Thread ausgeführt, da dies Teil des verbundenen Objektes ist.

```
1  class Controller(QObject):
2      # Teil hier nicht gezeigt
3      signal_gamepad = pyqtSignal()
4      def __init__(self, config, output_dir, test_mode, neustart) -> None:
5          # Teil hier nicht gezeigt
6          self.PadThread = QThread()
7          # Teil hier nicht gezeigt
8          self.gamepad = Gamepad(self.sprache, self.PadAchsenList, self.add_Ablauf)
9          # Teil hier nicht gezeigt
10         self.signal_gamepad.connect(self.gamepad.event_Loop)
11         # Teil hier nicht gezeigt
```

Der Unterschied zu den Geräte-Threads ist der, dass *event_ Loop* eine endlose while-Schleife beinhaltet die eine *break* Bedingung (*self.done* - boolesche) hat. Die Endlosschleife und der Thread werden gebraucht, damit VIFCON jederzeit auf das Gamepad reagieren kann. Weiterhin dürfen hier nur die Funktionen der Geräte-Widget-Objekte aufgerufen werden. Dies liegt daran, dass in diesen Funktionen alles weitere ausgeführt wird, was für die Bewegung wichtig ist. In der while-Schleife wird eine For-Schleife ausgeführt. Folgend wird ein Auszug der while-Schleife gezeigt, der den Start der For-Schleife und zwei If-Anweisungen beinhaltet. Zeilen für das Logging und füllen der Ablaufdatei wurden entfernt.

```
def event_Loop(self):
    while not self.done:
        for event in pygame.event.get():
            # Teil hier nicht gezeigt
            if event.type == pygame.QUIT:
                pygame.quit()

            if event.type == pygame.JOYBUTTONDOWN:
                if event.button == 0:
                    # Teil hier nicht gezeigt
                    for achse in self.Achsen_list:
                        if achse.gamepad.isChecked() and 'Nemo-Achse-Linear' ...
                            in achse.device_name:
                            achse.fahre_Hoch()
            # Teil hier nicht gezeigt
```

In der For-Schleife werden die Events des Gamepads durchgegangen. Danach folgen If-Anweisungen die den Typ des Events prüfen. Das Event *pygame.JOYBUTTONDOWN* ist für die Gamepad-Knöpfe relevant. Im Folgenden wird durch die If-Anweisung herausgefunden um welchen Knopf es sich handelt. Hierbei wird nicht nach dem Buchstaben oder der Bezeichnung gesucht, sondern nach den Zahlen der zweiten Spalte der Tabelle 21. Nun folgt noch eine For-Schleife, die ein weiteres Feature darstellt. In VIFCON wurden Checkboxen verwendet, die durch Betätigung eine Funktion erhalten. Zum Beispiel kann bei Nemo-Achsen-Rotation die Beachtung des Limits deaktiviert oder auch die Synchron Fähigkeit aktiviert werden. Für das Gamepad gibt es auch eine. Bei der For-Schleife werden nun alle Achsen durchgegangen, die auf der Antriebs-Seite definiert worden sind. Mit der markierten Checkbox und dem richtigen Namensteil wird dann die jeweilige verbundene Funktion in dem Geräte-Widget-Objekt ausgelöst. Die Checkbox „GPad“, lässt den Anwender somit die Achsen auswählen. Somit können auch wieder Antriebe Synchron gestartet werden. Sind z.B. die Rotations-Achsen für Spindel und Tiegel bei der Nemo-1-Anlage ausgewählt, kann der Anwender beide Achsen durch das Gamepad starten und Beenden, aber auch z.B. nur eine starten oder beenden, in dem die Checkbox gesetzt oder nicht gesetzt wird.

Für die Nemo-Achsen wurde sich somit ein Konzept ausgedacht für die Nutzung des Gamepads. Bei der PI-Achse können die Richtungen durch die Config verändert werden bzw. die Knöpfe sehen anders aus. Bisher ist das Konzept bei der PI-Achse nur, dass L den linken Knopf und R den rechten Knopf betätigt. Sonst sind die Punkte wie bei der Nemo-1-Anlage.

5.7 Verbindung zu Multilog

Wie in Kapitel 5.6 erwähnt, ist die Verbindung zu Multilog auch ein Extra Feature von VIFCON. In dieser Arbeit wurde Mulitlog bereits einige mal erwähnt. Zusammenfassend ist Multilog [Mul] ein reines Logging Programm, dass andere Geräte ausliest und die Daten in Messordnern speichert. Um beide Programme nun zu verbinden, soll eine Schnittstelle entworfen werden, die die Kommunikation zwischen VIFCON und Multilog ermöglicht. Entschieden wurde sich für die TCP/IP-Kommunikation. Bei dieser ist Multilog der Client und VIFCON der Server. Der Grund für die Verbindung ist folgender. Alle Daten eines Experimentes sollen an einem Ort liegen. Wie gesagt ist Multilog eine reine Logging-Umgebung, die z.B. auch Werte für Wärmebildkamera, Thermoelemente und viele andere Geräte aufnehmen kann. VIFCON übernimmt bei z.B. einer Kristallzüchtung die Geräte die gesteuert werden müssen. Multilog wiederum übernimmt die reinen Messgeräte. Um nun alles an einem Ort zu haben, soll VIFCON seine Messdaten an Multilog übergeben.

Für die Umsetzung werden die Bibliotheken **socket** und **json** verwendet. Grundsätzlich ist die Verbindung zu Multilog wie beim Gamepad aufgebaut. Somit sind folgende Code-Zeilen für den Thread wichtig.

```
1  class Controller(QObject):
2      # Teil hier nicht gezeigt
3      signal_Multilog = pyqtSignal()
4      # Teil hier nicht gezeigt
5      def __init__(self, config, output_dir, test_mode, neustart) -> None:
6          # Teil hier nicht gezeigt
7          self.LinkMultilogThread = QThread()
8          self.MultiLink = Multilog(self.sprache, self.port_List, ...
               self.add_Ablauf, self.widgets, self.trigger)
9          # Teil hier nicht gezeigt
10         self.signal_Multilog.connect(self.MultiLink.event_Loop)
11         # Teil hier nicht gezeigt
```

Auch hier gibt es ein Signal (1) mit dem dann die Funktion *event_ Loop* durch *emit* aufgerufen wird (4) und zwei Instanziierungen für den Thread (2) und Multilog (3). Auch hier gibt es dann einen nahezu endlosen Loop, der durch die Funktion *ende* und die Variable *self.done* beendet werden kann. Auch hier benötigt es eine while-Schleife und einen Thread, da VIFCON jederzeit schauen muss ob Multilog eine Anfrage sendet. Um die Verbindung zu starten muss dies in der Config-Datei freigegeben werden. Neben diesem Umstand muss auch in jedem Gerät der Config-Datei ein Port und ein sogenannter Trigger gesetzt werden. Diese Trigger und die Ports werden bei Erstellung der Geräte-Objekte gesammelt und an **multilog.py** bzw. dem Objekt (*LinkMultilogThread*) übergeben. Mit der Funktion *create_ socket* wird dann eine Verbindung erstellt. Hierbei ist wichtig das VIFCON vor Multilog gestartet wird, da VIFCON sonst keine Verbindung aufbauen kann.

Die Kommunikation sieht nun wie folgt aus. VIFCON wartet auf ein Triggerwort und löst bei Erhalt dann die Funktion *sendData* aus. In dieser Funktion können nun drei Dinge passieren. Wenn der Trigger existiert holt sich die Funktion die Daten direkt aus dem Geräte-Widget-Objekt und formatiert sie für das Senden an Multilog. Sollte der Trigger falsch sein, entfernt VIFCON die Verbindung aus der dazugehörigen Liste, weil VIFCON sonst in der nächsten Schleife an der Verbindung hängen bleibt. Sollte jedoch ein Leerer String erhalten werden, bricht VIFCON die gesamte Kommunikation ab, indem es die Funktion *ende* aufruft, aber der andere Programmablauf wird nicht beeinträchtigt.

In der Abbildung 57 werden nun die Messdaten von Multilog und VIFCON verglichen. Die Abbildung zeigt dabei die Temperaturwerte. Wenn sich die beiden Istwerte angesehen werden, können nur kleine Unterschiede gesehen werden. Diese Unterschiede kommen durch die Abfragen von Multilog und das Sampling von VIFCON. Wenn das Sampling, also die Messwertaufnahme aktiv ist und zumeist mit 2 s angenommen wird, nimmt VIFCON nur in diesen Zeitintervallen Werte auf. Die Abfrage von Multilog läuft, wie erwähnt, parallel dazu. Somit kann es sein, das Multilog einen Wert nicht mitbekommt oder sogar den Wert ein weiteres mal aufnimmt.

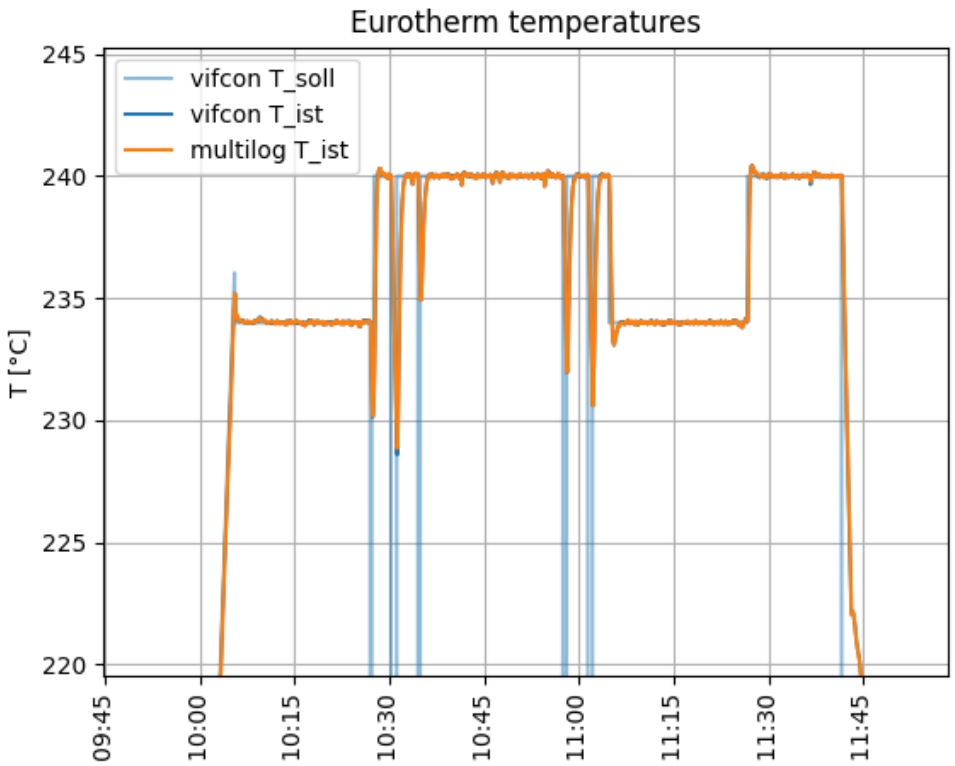


Abbildung 57: Vergleich Multilog und VIFCON Messdaten (Quelle: [Perf])

5.8 Sicherheitskonzept

Wenn sich Anlagen und Geräte angesehen werden, so ist es immer ratsam auch zu schauen ob als richtig und sicher funktioniert. Bei der Nutzung einer Steuerung können durch den Nutzer auch Fehler entstehen. Auch VIFCON soll sicher funktionieren, weshalb Sicherheitskonzepte entworfen werden müssen. So kann dies von der GUI oder auch von den Geräten ausgehen. Die folgenden Kapitel zeigen diese Sicherheitskonzepte.

5.8.1 Grenzen/Limits

Dieser Punkt ist wichtig, da somit erreicht werden kann, dass der Nutzer bestimmte Bereiche nicht überschreitet. In VIFCON wird dies überwiegend durch die Config-Datei ausgeführt. Für jede Größe können ein Minimum- und Maximumwert angegeben werden. Diese Werte werden bei den Eingabefeldern der GUI und den Rezepten beachtet. Wenn ein Wert gesendet werden soll, wird erstmals überprüft ob dieser Wert überhaupt in dem gültigen Wertebereich liegt. Bei der PI-Achse und den Nemo-Achsen wird z.B. der Weg berechnet und daran dann geschaut ob dieser überhaupt erreichbar ist. Damit der Nutzer die Config-Grenzen sehen kann, kann dieser die Kurven der Grenzen im Plot über die Config-Datei einschalten. In Abbildung 58 werden diese Kurven für die Ausgangsleistung des Eurotherms gezeigt.

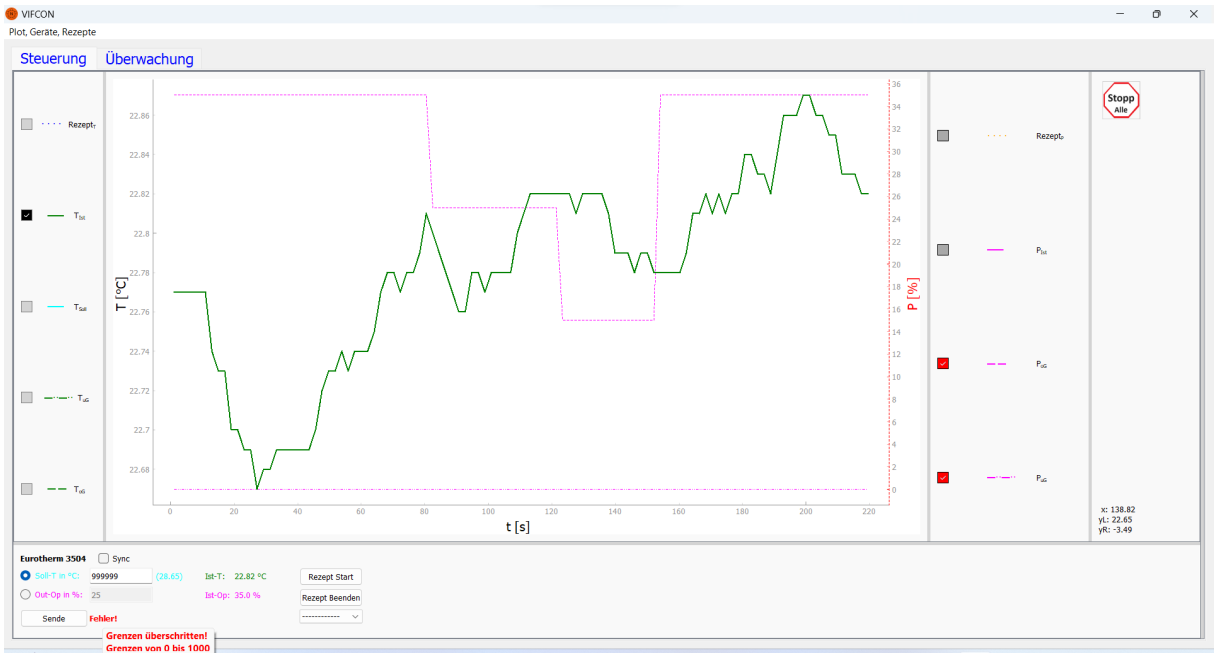


Abbildung 58: Limit-Kurven und GUI-Sende-Fehler (Quelle: eigene Darstellung)

Neben diesem Fakt haben auch Geräte eigene Limits. Einige davon werden durch VIFCON ausgelesen. Bei Eurotherm gibt es diesbezüglich einen Schlüssel in der Config-Datei, der „sicherheit“ heißt. Wenn der Wert dabei auf True steht, so wird der Limit-Wert von dem Gerät selbst festgelegt. Der Wert der hier angeschaut wird ist die Ausgangsleistung oder auch OP genannt. Der Befehl für die maximale Ausgangsleistung ist HO. Mit der Einstellung kann dieser HO-Wert nur am Regler eingestellt werden. Um VIFCON diesen Wert zu übermitteln, muss der Nutzer entweder in den Manuellen Modus umschalten oder den Menü-Befehl zum Auslesen des HO-Wertes nutzen. Mit dieser Funktion wird der OP-Max Wert in VIFCON aktualisiert. Im automatischen Modus passt Eurotherm die Ausgangsleistung durch den Ist-Sollwert-Vergleich und den dazugehörigen PID-Regler an. Im manuellen Modus wiederum kann nur der Nutzer den Leistungswert anpassen. Beim Umschalten des Modus in den Manuellen Modus, soll VIFCON die Obergrenze der Leistung kennen, weshalb der Wert an der Stelle auch ausgelesen wird. VIFCON kann den Leistungswert OP nur in diesem Modus an Eurotherm senden. Ist „sicherheit“ aber False, so hat VIFCON die Rechte zum ändern des Wertes. Der HO-Wert wird nun zum Start von VIFCON und beim Limit-Update durch das Menü aktualisiert. Die Abbildung 58 zeigt die Anpassung des Limits, anhand der Kurve. Dadurch das VIFCON beim Senden eines Wertes bereits eine Überprüfung vornimmt, sind die Geräte eigenen Limits eine zusätzliche Überwachung, falls z.B. ein Config-Limit falsch gesetzt wird.

5.8.2 Ausgabe und Speichern von Fehlern, Warnungen und Informationen

Ein weiter Punkt ist die Veranschaulichung von Problemen. Der Nutzer kann z.B. Rezepte falsch schreiben oder auch bei der Eingabe in die Eingabefelder Fehler verursachen. VIFCON schaut sich alle gesendeten Werte an, bevor diese überhaupt in die beiden Dictionaries *write_task* und *write_value* geschrieben werden. Somit können keine falschen Werte gesendet werden. Solche Fehler müssen aber angezeigt werden. In VIFCON werden Fehler wie:

1. ein leeres Eingabefeld (Keine Eingabe!) beim senden (GUI, Logging, Ablauf-Datei),
2. eine Überschreitung der Limits des gesendeten Befehls (GUI, Logging, Ablauf-Datei),
3. eine fehlerhafte Eingabe im Eingabefeld (GUI, Logging, Ablauf-Datei),
4. Rezeptfehler (GUI, Logging) wie
   (a) Überschreitung der Limits,
   (b) fehlerhaftes Rezept und,
   (c) Rezept Start oder neu einlesen der Config-Datei bei laufenden Rezept,
5. ein Senden bei nicht Initialisierung (GUI, Logging, Ablauf-Datei) und,
6. einem fehlerhaften Senden (Logging, Ablauf-Datei) angezeigt.

Wie die Fehlermeldung für die GUI bearbeitet wird, kann in Anhang E.1 nachgelesen werden. Beim Logging können bestimmte Nachrichten auch in der Konsole ausgegeben werden. Weiterhin gibt es auch Pop-Up-Fenster die bei bestimmten Punkten angezeigt werden. Z.B. im Fall der Eurotherm Sicherheit. Hierbei wird der Nutzer darüber informiert, dass der HO-Wert nur am Eurotherm geändert werden kann. Diese Fenster werden in Kapitel 5.9.4 erläutert und in Anhang E.7 gezeigt. Während eines Experimentes kann der Nutzer nicht immer in die Log-Datei sehen, weshalb die direkten Fehlermeldungen in der GUI, die Pop-Up-Fenster und auch die Konsolen-Ausgabe die Anwendung sicherer machen, da diese schneller gesichtet werden können.

5.8.3 Try-Except

Um einen Absturz zu verhindern wird im Code z.B. das Try-Except verwendet. Dieser Anwendung findet sich bei einigen Punkten aus Kapitel 5.8.2 wieder, speziell beim Senden von Werten. Die Funktion *controll_value* (aus Eurotherm-Widget-Objekt) zeigt eine Anwendung für die Nutzung von Try-Except. Wenn z.B. Buchstaben gesendet werden, dann würde die Umwandlung zu Float (Zeile 6) ein Fehler generieren. Durch Except wird dieser Fehler nicht zum Absturz führen, sondern geloggt und angezeigt.

```
1  def controll_value(self, value, oG, uG):
2      if value == '':
3          self.Fehler_Output(1, self.err_1_str[self.sprache], ...
               self.Text_19_str[self.sprache])
4      else:
5          try:
6              value = float(value)
7              if value < uG or value > oG:
8                  self.Fehler_Output(1, f'{self.err_2_str[self.sprache]} {uG} ...
                       {self.err_3_str[self.sprache]} {oG}', ...
                       self.Text_20_str[self.sprache])
9              else:
10                 self.Fehler_Output(0, ...
                       error_Message_Ablauf=f'{self.Text_21_str[self.sprache]} ...
                       {value}.')
11                 return value
12         except Exception as e:
13             self.Fehler_Output(1, self.err_5_str[self.sprache], ...
                   self.Text_22_str[self.sprache])
14             logger.exception(f"{self.device_name} - ...
                   {self.Log_Text_38_str[self.sprache]}")
15     return -1
```

Nicht nur bei Fehlern wird Try-Except genutzt, sondern auch um bestimmte Zustände zu prüfen. Neben dem Fall der Umwandlung wird Try-Except noch bei:

1. dem Auslesen der ausgelagerten Rezept-Datei,
2. der Anzeige der Rezept-Kurve,
3. dem Berechnen von Werten (z.B. Verriegelungstimer PI-Achse),
4. dem Auslesen von Messdaten aus einem Gerät,
5. dem Senden von Daten an ein Gerät,
6. dem Versuch der Initialisierung,
7. dem Aufbau der Schnittstelle,
8. der Übernahme von Config-Daten (z.B. Konsolen Logging Level),
9. der Erstellung der Init- und Limit-Knöpfe im Menü für die Geräte,
10. die Erstellung der Gamepad-Instanz,
11. der Aktualisierung des Plots und der GUI und,
12. der Verbindung zu einem Port (Multilog-Link) angewendet.

5.8.4 Überwachen von Messdaten

Auch die Überwachung von Daten erfüllt ein Sicherheitskonzept. Durch den Plot kann der Nutzer z.B. die Grenzen und auch alle Istwerte sehen. Wenn sich z.B. ein Wert seltsam benimmt, kann der Nutzer darauf reagieren und Schritte einleiten. Neben diesem Umstand besitzt VIFCON auch einen Tab Monitoring indem die Drücke für die Gase der Nemo-1-Anlage angezeigt werden. Auch hier kann der Nutzer immer nachsehen, ob alles stimmt, ohne an die Anlage zu gehen. Ein Beispiel dafür ist in Kapitel 6.2 in der Abbildung 75 zu sehen.

5.8.5 Sicherheitskonzepte der Geräte

Wie schon bei den Limits erwähnt, beinhalten auch die Geräte bestimmte Limits und Sicherheitskonzepte. So hat z.B. der TruHeat Generator einen sogenannten Watchdog. Der TruHeat Generator wurde in Kapitel 3.3 vorgestellt. Um den Generator einschalten zu können und das Senden der Sollwerte zu gewährleisten müssen:

1. das aktuelle Interface und,
2. der Watchdog,

beachtet werden. Zu Beginn war die Überlegung, dass das Anschalten des Generators sicherheitshalber über das Bedienungspanel des Generator erfolgt. Sobald jedoch am Generator das Interface auf RS232 bzw. auf User 4 (IKZ-Profil mit RS232) gestellt wird, wird der Einschaltknopf am Panel verriegelt. Somit ist es nicht mehr möglich diesen anzuschalten. Aus dem Grund wurden die Befehle für das Ein- und Ausschalten in VIFCON eingefügt. Diese Knöpfe sind in Abbildung 71 zu sehen und wurden nach dem Panel (Abbildung 26) entworfen.

Das Kapitel 3.3.1 zeigte, dass der Generator nur dann angehen kann, wenn keine Fehler- bzw. Störmeldungen anliegen. Wenn ein Fehler durch den Watchdog ausgelöst wird, wird das Einschalten des Generators verriegelt und es müssen alle Fehler am Generator zurückgesetzt werden. Dafür muss das Interface auf Panel geändert werden, da die Reset Funktion bei RS232 nicht auf den Panel funktioniert.

Einige der Informationen für den Watchdog sind in Kapitel 3.3.6 zu finden. In der Dokumentation [Dokb] können folgende zwei Beschreibungen des Watchdogs gefunden werden:

1. „Legt die Zeitspanne fest, welche das Netzteil zwischen zwei gesendeten Befehlen vom Host toleriert, ohne eine Fehlermeldung zu generieren.“ Zitat: [Dokb] S. 155
2. „Stellen Sie die Zeitspanne ein, die der Generator zwischen zwei gesendeten Befehlen vom Host toleriert, ohne eine Fehlermeldung zu erzeugen.“ Zitat: [Dokb] S. 106

Die Abbildung 59 zeigt eine schematische Darstellung der beiden Zitate. Wenn z.B. der RS232 User-Watchdog auf 20 ms gesetzt wird, so muss nach Befehl 1, der Befehl 2 innerhalb der 20 ms am Generator ankommen. Sobald dies nicht passiert, wird die Störmeldung ausgelöst.

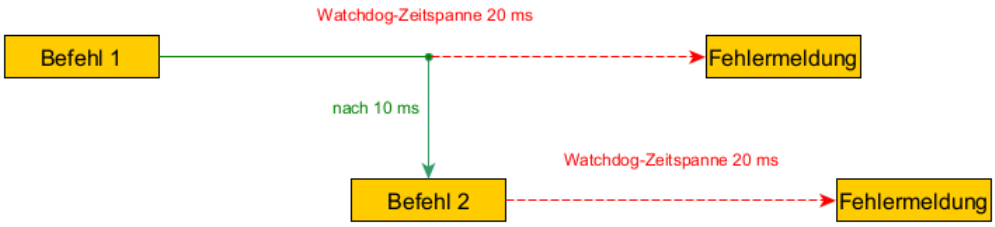


Abbildung 59: Darstellung des Watchdogs (Quelle: eigene Darstellung)

Um den Watchdog weiter zu nutzen wurde eine Zeit in der Config-Datei eingetragen, die zum Start des Programms oder zur Initialisierung an den Generator gesendet wird. Bei der Initialisierung ist folgendes gemeint. Das Programm (VIFCON) kann so gestartet werden, dass Geräte nicht ansprechbar sind. Sobald diese angeschlossen wurden und ein Initialisierungsknopf (Menüleiste) betätigt wird, wird das Gerät ansprechbar. Beim Test mit dem Watchdog konnte gesehen werden, das der Watchdog erst dann startet, wenn der erste Befehl am Generator ankommt. Nun ist zu beachten das die Watchdog-Zeit nicht kleiner ist als die Lese-Zeit. Im Regelfall ist diese Auslesezeit bei 2 s gesetzt. Somit werden Zyklisch Befehle gesendet, wodurch der Watchdog nicht auslöst. Wenn die Messung durch die Config-Datei abgeschaltet wird, dann gibt es keine zyklischen Lese-Befehle und der Watchdog löst aus. In dem Fall sollte der Watchdog durch das setzen auf Null deaktiviert werden. In VIFCON kann der Watchdog recht wichtig sein, sollte das Programm abstürzen oder der Generator nicht ausgeschaltet werden. Mit der Auslösung des Watchdogs, wird keine Leistung mehr ausgegeben. Der TruHeat besitzt noch einen Fußtaster, als Extra-Schutz. Der Generator-Leistungsbetrieb geht erst an, wenn Fußtaster und Ein-Schalter betätigt worden sind.

5.8.6 Sicherer Endzustand

Zu guter Letzt müssen noch sichere Endzustände angesehen werden. In VIFCON gibt es verschiedene Stopp-Knöpfe. Zum einen haben einige Geräte einende eigenen und zum anderen haben die Geräte-Typen einen übergeordneten, der alle Geräte des Typs abschaltet. In der Config-Datei kann nun bei jedem Gerät gesagt werden, ob ein sicherer Endzustand erfolgen soll. Jedes Gerät in VIFCON hat dafür in der Config-Datei einen Schlüssel **ende** bekommen. Dieser Schlüssel erhält einen Booleschen Wert. Bei True wird zum Ende die Funktion *Stopp* der Geräte-Widget-Objekte ausgelöst. Um das Programm zu Beenden wird das „X“ in der Windows-Leiste betätigt. Neben dem sicheren Endzustand werden hier auch die Threads beendet und die Schnittstellen geschlossen. Die Funktion die durch das Programmende ausgelöst wird heißt *exit* und ist im Programm **vifcon_controller.py** zu finden. Der folgende Code zeigt den relevanten Teil für die sicheren Endzustände. In dem Code wird jedes Geräte-Widget-Objekt durchgegangen und geschaut ob der Schlüssel *ende* auf True oder False steht. Wenn es True ist, dann wird die Funktion *Stopp* ausgeführt. darauf folgend werden die Threads noch einmal gestartet (*ckeck_device*) und kurz gewartet, damit sie ausgeführt werden können.

```
1 def exit(self):
2     # Teil hier nicht gezeigt
3     for device in self.widgets:
4         if not 'Nemo-Gase' in device:
5             if self.config['devices'][device]['ende']:
6                 self.widgets[device].Stopp()
7     self.ckeck_device()
8     time.sleep(1)
9     # Teil hier nicht gezeigt
```

Die Tabelle 22 zeigt dabei, was in der ausgelösten Funktion *Stopp* passiert. Manche Beschreibungen wie die Änderung der GUI finden nur bei dem Stopp-Knöpfen eine Anwendung. Für den Endzustand sind die Werte Änderungen relevant.

Tabelle 22: Sicherer Endzustand der Geräte (Quelle: eigene Darstellung)

Geräte-Bezeichnung	**Sicherer Endzustand/*Stopp*-Funktion**
Eurotherm	1. Rezept beenden (Aufruf Funktion *RezEnde*) 2. Wechsel in Manuellen Modus (Aufruf Funktion *BlassOutTemp*) 2.1. Dictionary write_task - Wert des Schlüssel Auto_Mod = False 2.2. Dictionary write_task - Wert des Schlüssel Manuel_Mod = True 3. GUI abändern 3.1. Radio-Button Leistung aktivieren 3.2. Text Eingabefeld Leistung auf Null setzen 4. Sende Null an OP 4.1 Dictionary write_task - Wert des Schlüssel Operating point = True 4.2. Dictionary write_value - Wert des Schlüssel Sollwert = 0
TruHeat	1. Rezept beenden (Aufruf Funktion *RezEnde*) 2. Generator ausschalten (Aufruf Funktion *THAus*) 2.1. Dictionary write_task - Wert des Schlüssel Aus = True 2.2. Dictionary write_task - Wert des Schlüssel Ein = False 3. Sende Null an alle Größen 3.1. Dictionary write_value - Wert des Schlüssel Sollwert = 0 3.2. Dictionary write_task - Wert des Schlüssel Soll-Leistung = True 3.3. Dictionary write_task - Wert des Schlüssel Soll-Spannung = True 3.4. Dictionary write_task - Wert des Schlüssel Soll-Strom = True
PI-Achse	1. Rezept beenden (Aufruf Funktion *RezEnde*) 2. Stoppe Achse 2.1. Dictionary write_task - Wert des Schlüssel Stopp = True 3. Bewegungsknöpfe entriegeln bzw. wieder freischalten
Nemo-Achse Hub	1. Rezept beenden (Aufruf Funktion *RezEnde*) 2. Stoppe Achse 2.1. Dictionary write_task - Wert des Schlüssel Stopp = True
Nemo-Achse Rotation	1. Rezept beenden (Aufruf Funktion *RezEnde*) 2. Stoppe Achse 2.1. Dictionary write_task - Wert des Schlüssel Stopp = True
Nemo-Gase	Nemo-Gase besitzt keine Funktion *Stopp*. Da bei der Kommunikation nur Werte ausgelesen werden, gibt es hier keinen sicheren Endzustand.

5.9 Graphische Benutzeroberfläche (GUI)

In Kapitel 2.1 wurden Anlagen des IKZ vorgestellt. Alle Steuerungen haben dabei eine GUI vorzuweisen, die auch in dem Kapitel gezeigt wurden. Die GUI macht es dem Nutzer möglich, die Anlage zu steuern und zu überwachen. Bei der Nemo-1-Anlage (Abbildung 35) ist die derzeitige Steuerung (Abbildung 4) Teil des Schaltschrankes (Abbildung 35b). Da die Nemo-1-Anlage eine der Anlagen ist, die von der Modellexperimente-Gruppe genutzt wird, soll VIFCON auch diese steuern können. Somit können über VIFCON sowohl alle von der Anlage genutzten, als auch separate Geräte wie z.B. Eurotherm (andere in Kapitel 3) an einem Ort gesehen und gesteuert werden.

Daraus ergibt sich, dass die GUI eines der wichtigsten Teile der Steuerung ist. Mit ihr bekommt der Nutzer eine Kontrolle und Übersicht über die Anlage. Durch die Zusammenfassung verschiedener Geräte in einer GUI hat der Nutzer Zugriff auf diese, ohne sich zu dem Gerät bewegen zu müssen. Um eine GUI zu erstellen muss sich für ein geeignetes Programm entschiedene werden. Da die Aufgabe eine Umsetzung mit Python fordert, müssen sich nun GUI-Toolkits angesehen werden. In Kapitel 4.3 wurde das angewendete GUI-Toolkit bereits erwähnt. Neben der gewählten Bibliothek **PyQt5** gibt es noch Bibliotheken wie **tkinter** [Doke] und **niceGUI** [Dokt]. In der Bachelorarbeit wurde das dazugehörige Programm mit tkinter programmiert. Da **PyQt** sehr stark auf der objektorientierten Programmierung basiert, ist tkinter als Einstieg besser geeignet. Der Vorteil von **PyQt** ist jedoch die sehr umfangreiche Bibliothek, die selbst Threads bereits mit einschließt. Somit wird das Multithreading, welches in Kapitel 5.2 erwähnt wurde, leichter ermöglicht. Mit **niceGUI** können zwar Oberflächen gestaltet werden, die über das Internet aufgerufen werden können, aber auch dieses verfügt nicht über diese große Bibliothek. Auch bei der Darstellung der Plots war die Entscheidung recht einfach, da **pyqtgraph** mit der **PyQt** Bibliothek kompatibel ist. Als Alternative liefert Python z.B. die **matplotlib**-Bibliothek [Dokh]. Durch die Verbindung zu **PyQt** liefert **pyqtgraph** eine leichtere Umsetzung des Plots.

Die Abbildungen 60 und 61 präsentieren die GUI der VIFCON-Steuerung. Dabei wird die minimal-Form der GUI gezeigt. Die GUI benötigt eine Mindestauflösung des Bildschirms von 1240x900 Pixel. Abbildung 60 zeigt den Tab Steuerung, der die Plots mit Legende und die Widgets der Geräte beinhaltet. Dabei wird für jedes vorhandene Gerät in der Config ein Widget erstellt. Bei den angezeigten Werten handelt es sich um Zufallswerte, da die GUI über den Test-Modus erstellt wurde, um alle Gerät zu zeigen. In Abbildung 61 wird der Tab Überwachung (Monitoring) dargestellt. Hier wurde das Widget des Nemo-Gase Gerätes eingefügt. Auf die Widgets wird hier nicht weiter eingegangen. Bei Interesse gibt es in Anhang E.10 eine kleine Erläuterung zu den Widgets und eine größere und separate Darstellung.

Abbildung 60: VIFCON GUI - Tab Steuerung (Quelle: eigene Darstellung)

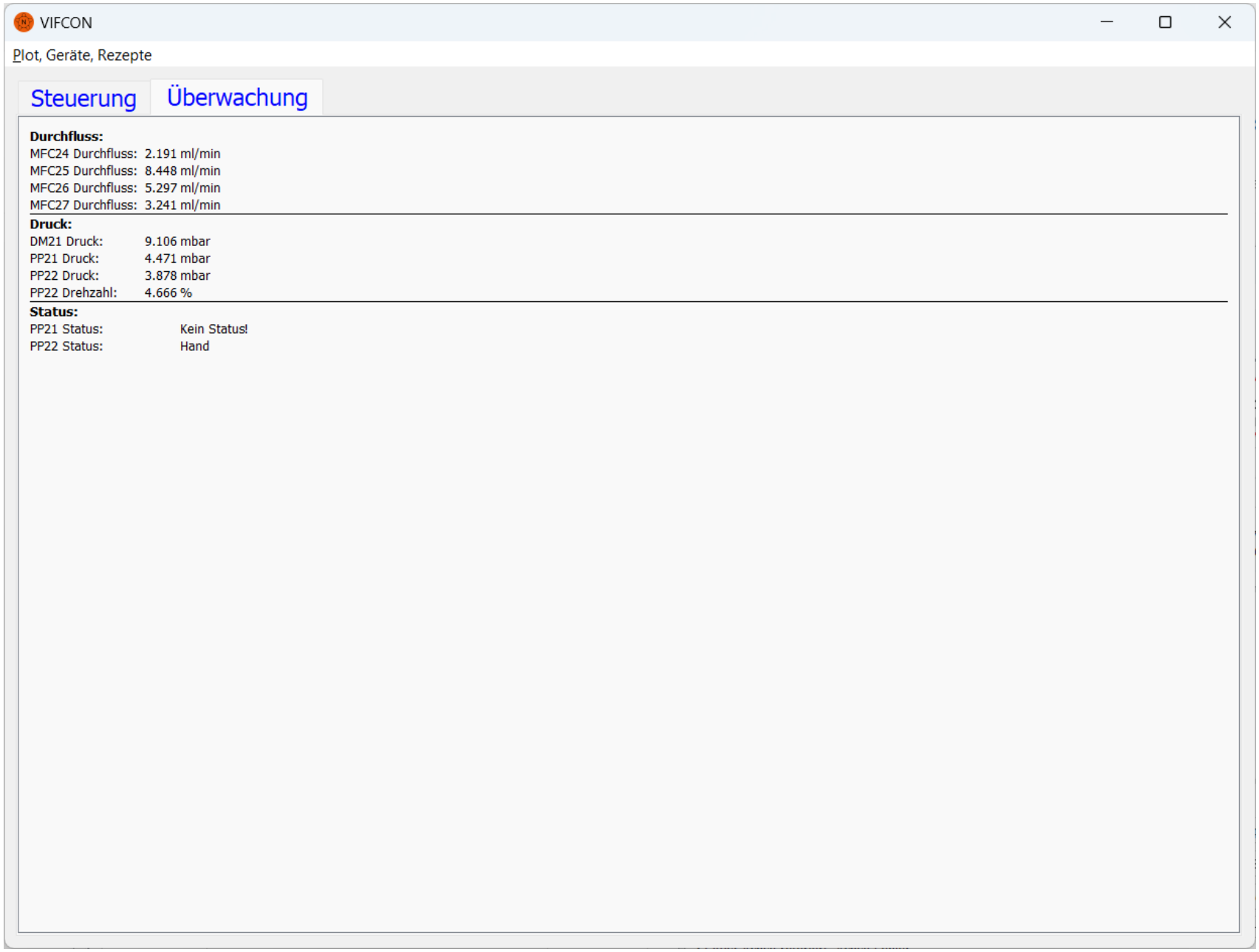


Abbildung 61: VIFCON GUI - Tab Monitoring (Quelle: eigene Darstellung)

Bevor es zu den Features der GUI kommt, soll noch etwas zu den Programmen gesagt werden. In Anhang A finden sich alle Programme, Bilder und sonstige Dateien, die mit dieser Arbeit abgegeben wurden. Im Ordner **view** befinden sich alle Programme die zur Erstellung der GUI gebraucht werden. Neben diesem Ordner wird noch der Ordner **icon** gebraucht, der die Icons für die Knöpfe (Buttons) und z.B. die Checkboxen der Legende beinhaltet. Die Icons werden in Anhang E.9 näher beschrieben und gezeigt.

In den folgenden Kapiteln werden bestimmte (nicht alle) Features der GUI erläutert bzw. wird gezeigt, warum und weshalb diese so umgesetzt worden sind. Die Features der GUI sind:

1. die Funktion des Verdecken der Widgets über Splitter-Objekte (Kapitel 5.9.1),
2. die Platzierung der Widgets (einzelne und Gruppen) auf der GUI (Kapitel 5.9.2),
3. eine Tab-Bar (erwähnt in Kapitel 5.9.2),
4. eine Menü-Leiste (Kapitel 5.9.3),
5. die Anzeige von Pop-Up-Fenstern (Kapitel 5.9.4),
6. ein Plot (Kapitel 5.9.5),
7. eine Legende die an drei verschiedene Stellen gesetzt werden kann (Kapitel 5.9.5.2),
8. eine über die Skalierungsfaktor veränderbare y-Achse (Achsenbeschriftung) (Kapitel 5.9.5.4),
9. eine Anzeige der Achsenwerte durch die Maus (Kapitel 5.9.5.5),
10. die Nutzung von Knöpfen (am Gerät, für Gerätetyp, Typen übergreifend) und,
11. die Gestaltung der GUI durch die Config-Datei (Anhang B).

Der letzte Punkt meint z.B. die Änderung der Plot-Legende oder auch die Auswahl der anzuzeigenden Kurven und Geräte-Widgets. Der wichtigste Teil in dem Punkt ist die Auswahl der Sprache. VIFCON kann derzeitig in Englisch und Deutsch angezeigt werden, wodurch diese auch international nutzbar wird. Die Programmierung dieser Sprache wird in Anhang E.6 näher erläutert. Kurz gesagt basiert dies auf Python-Listen.

Die folgenden Erklärungen und Darstellungen beziehen sich auf den abgegebenen Stand des Programms. Das Programm an sich wird nach Abgabe noch weiter entwickelt und später auf GitHub veröffentlicht, damit andere Forschende diese Steuerung auch nutzen können. Die in Kapitel 6 gezeigten GUIs zeigen weiterhin noch ältere Versionen der GUI, da die Experimente zur Weiterentwicklung von VIFCON dienen. Auf Code wird in diesem Kapitel weitestgehend verzichtet. Einige der wichtigsten Features werden im Anhang E im Sinne der Programmierung, also mit Code, näher beschrieben.

5.9.1 Splitter

Wenn sich die verschiedenen GUIs im Kapitel 2.1.3 angesehen werden, kann gesehen werden, dass in allen Tabs verwendet werden. VIFCON soll durch die Splitter nun noch eine zusätzliche Freiheit in den einzelnen Tabs erhalten. Durch die Nutzung der Splitter kann der Nutzer sich die Seite so einrichten wie er es möchte. Zum Beispiel könnte der Plot einfach verdeckt werden oder nur die Generator-Seite sichtbar gemacht werden. Die Splitter sind gut in Abbildung 60 und 44 dargestellt. Die Splitter bilden somit eine zusätzliche Möglichkeit im Vergleich zu den Tabs. Anstelle der Splitter hätte jedes Geräte-Widget auch als Tab dargestellt werden können. In Multilog [Mul] wird dies so getan. Da Multilog eine reines Logging-Programm ist, kann dies auch so umgesetzt werden. VIFCON ist aber ein Programm, welches auch Geräte-Funktionen auslöst wie Bewegung und Werteänderung. Aus dem Grund ist es hier von Vorteil, dass alle Geräte in der GUI zeitgleich zu sehen sind.

In Kapitel 2.1.3 wird auch die GUI der Anlage aus der Fluorid-Gruppe gezeigt. Diese Steuerung basiert auf vielen Tabs. Dies kann an Abbildung 7 gesehen werden. Die Abbildung 6 zeigt das Hauptbild der Fluorid-Gruppen-GUI. Diese Seite ähnelt der GUI von VIFCON (Abbildung 60). Von dort aus kann Einfluss auf die Anlage genommen werden. Im Unterschied zu VIFCON ist die GUI starr und nicht flexibel anzuordnen. Die Nutzung von Tabs ermöglicht eine breite Aufteilung von Steuerelemente der Geräte, jedoch auch eine stärkere Nicht-Nutzung des gesamten Platzes auf der Seite des Tabs. Weiterhin muss der Nutzer immer hin und her klicken, um bestimmte Dinge anzuklicken. Wenn alles auf eine Seite gesetzt wird, so kann der Nutzer schnell auf Dinge reagieren. Mit den Splittern kann der Nutzer bestimmte Widgets wegschieben bzw. sogar nicht sichtbar machen. Natürlich bringt dies zwar auch das Problem des Findens mit sich, was aber durch eine gute Planung und Konfiguration behoben werden kann. Z.B. werden in Experiment 1 (Kapitel 6.1) vier PI-Achsen genutzt. Diese 4 Achsen nehmen recht viel Platz ein auf der einen Seite. Zwei der Achsen wurden jedoch nur zur Platzierung gebraucht. Durch die Splitter konnten diese dann verdeckt werden, wodurch die GUI weniger gequetscht wurde.

Zusammenfassend muss gesagt werden, dass sowohl Tabs als auch die Splitter geeignet sind für die Steuerung. Die Tabs sind nützlich, wenn man sehr viel und Separates, wie z.B. Prozessschritte der Züchtung, zeigen will. Aufgrund der gewünschten Flexibilität des Programms durch die Config-Datei bieten sich die Splitter bei VIFCON an. Jedes Gerät bekommt ein eigenes Widget und alle gewollten Kurven werden in dem Plot über den Widgets angezeigt. Durch die Legende und die Config-Datei kann auch die Übersichtlichkeit im Plot verändert werden. Die Splitter geben in dem Fall dem Nutzer einen besseren Überblick über die genutzten Geräte und die Möglichkeit der Anpassung der GUI im laufenden Betrieb. Die VIFCON GUI hat als Mindestauflösung eine Größe von 1240x900 Pixel. Diese Mindestgröße kann mit drei Geräten je Tab-Hälfte eingehalten werden. Bei mehr als drei Geräten sollte die Splitter-Funktion genutzt werden, um die Geräte-Widgets oder Plots innerhalb der Mindestauflösung anzuzeigen.

5.9.2 Platzierung von Widget

Bei **PyQt5** folgt die Platzierung der verschiedenen Widgets einem Bausteinsystem. Durch das Bausteinsystem wird ein Vorteil für die GUI-Anwendung ersichtlich. Die GUI kann gut übersichtlich entworfen werden und folgt einem einfachen Prinzip. Dieses System kann bei Interesse in Anhang E.8 näher erläutert gefunden werden. In VIFCON werden nun bestimmte Objekte der **PyQt5**-Bibliothek verwendet. Diese speziellen Widgets, die als Bedienelemente auf der GUI zu finden sind, waren:

1. QCheckBox (Checkbox),
2. QLabel (Texte, Zeichen, Strings),
3. QPushButton (Knöpfe),
4. QComboBox (Auswahl Drop-Down),
5. QLineEdit (Eingabefeld) und,
6. QRadioButton (Optionsfeld).

Neben den verschiedenen genannten Widgets ist auch noch zum einen das *QWidget* und zum anderen die verschiedenen Layouts wie:

1. QGridLayout,
2. QHBoxLayout und,
3. QVBoxLayout

wichtig. In dem folgenden Code-Beispiel kann die Nutzung von *QWidget* gesehen werden. Dieses Objekt bildet meist die Ebene die dann mit den genannten Bedien-Widgets gefüllt wird. Das *QWidget* kann dann wieder an andere Widgets oder höhere Ebenen (Main Windows) gehangen werden. Mit den verschiedenen Layouts werden dabei die Widgets so platziert wie es gewollt wird. Die beiden Box-Layouts setzen dabei die Widgets nebeneinander bzw. übereinander in das Widget. Das Grid-Layout kann nun wie eine Tabelle angesehen werden, wo die verschiedenen Widgets in Zeile und Reihe eingeordnet werden. Das Erläuterte soll nun durch ein Beispiel noch etwas näher vertieft werden. Der folgende Code zeigt, wie einige der Knöpfe symmetrischer angeordnet wurden. In dem Beispiel sind dies die Bewegungsknöpfe der PI-Achse. Für den Zweck wurde ein neues Widget mit einem *QHBoxLayout* als Layout erzeugt. Mit der Methode *setSpacing* konnte der Abstand zwischen den Widgets (Knöpfe) definiert werden. Am Ende wurde das Widget mit den vier Knöpfen in das übergeordnete Geräte-Widget mit dem *QGridLayout* eingefügt. Hierfür müssen mehrere Spalten verwendet werden, was durch die Angabe des ColumSpan (Code-Zeile 14: Row, Colum, RowSpan, ColumSpan) erreicht wird. Neben diesen Knöpfen wurden auch die Rezept-Knöpfe (auch Define Home) zusammengefügt. Da diese aber vertikal angeordnet sind, wird hier als Layout das *QVBoxLayout* verwendet.

```
1  class PIAchseWidget(QWidget):
2      def __init__(self, sprache, Frame_Anzeige, widget, line_color, config, ...
           config_dat, start_werte, neustart, add_Ablauf_function, piAchse, ...
           gamepad_aktiv, typ = 'Antrieb', parent=None):
3          # Teil hier nicht gezeigt
4          self.btn_group_move = QWidget()
5          self.btn_move_layout = QHBoxLayout()
6          self.btn_group_move.setLayout(self.btn_move_layout)
7          self.btn_move_layout.setSpacing(20)
8
9          self.btn_move_layout.addWidget(self.btn_left)
10         self.btn_move_layout.addWidget(self.btn_mitte)
11         self.btn_move_layout.addWidget(self.btn_right)
12         self.btn_move_layout.addWidget(self.btn_group_RB)
13         # Teil hier nicht gezeigt
14         self.layer_layout.addWidget(self.btn_group_move, 3, 0, 1, 4, ...
               alignment=Qt.AlignLeft)
15         # Teil hier nicht gezeigt
```

Anhand des Beispiels kann auch der große Umfang der Bibliothek gesehen werden. Neben der Platzierung können auch Größen wie Höhe, Breite und Abstand beeinflusst werden. Zum Beispiel können die Abstände in alle Richtungen durch *setContentsMargins(0,0,0,0)* auf Null gesetzt werden. Auch um die Größe von Eingabefeldern anzupassen bzw. diese fest einzustellen, gibt es Methoden wie *setFixedWidth* und *setFixedHeight*. Um die Größen der Widgets zu sehen, kann die folgende Funktion genutzt werden:

```
self.layer_widget.setStyleSheet("border: 1px solid black;")
```

Mit *setStyleSheet* werden Widget-Eigenschaften gesetzt. In dem Fall wird ein Rahmen für das Widget sichtbar gemacht. Der Effekt dieser Zeile ist in Abbildung 62 zu sehen. Durch den Rahmen werden die Bereiche der Widgets sichtbar. Auch Punkte wie das Spacing zwischen zwei Widgets werden dadurch sichtbar. Das Platzieren kann recht mühselig sein, da die Widgets manchmal einen festen Abstand haben oder **PyQt** die Platzierung anders setzt als es angenommen wird. Diese Funktion lässt sich über die Config-Datei ein- und ausschalten.

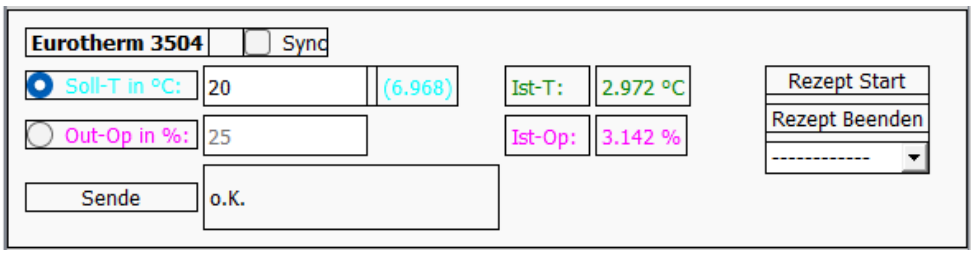


Abbildung 62: VIFCON GUI - Rahmen sichtbar (Quelle: eigene Darstellung)

Ein weiterer Punkt, der bei der Platzierung beachtet werden muss, ist der, dass die Widgets der Geräte-Widgets fest an ihrem Platz bleiben und die anderen Teile nicht verschieben. Aus dem Grund musste die Funktion *setColumnMinimumWidth* benutzt werden. Dadurch bekommen die angegebenen Spalten eine feste Minimumbreite. Diese Breite muss so gesetzt werden, dass z.B. die Fehlertexte sowie auch die Werte in diese Spalte passen, ohne die Minimumbreite zu überschreiten. Dieses Feature soll die GUI benutzerfreundlicher machen, da der Nutzer nicht von den beweglichen Teilen abgelenkt werden soll.

5.9.3 Menü-Leiste

Neben den Plots, den Geräte-Widgets und der Tab-Bar kann **PyQt** auch eine Menü-Leiste erstellen. Diese Menü-Leiste kann dann verschiedene Unter-Punkte enthalten, die über eine Drop-Down Funktion angezeigt werden. In VIFCON ist das Menü so gegliedert:

1. Menü-Leiste: Plot, Geräte, Rezepte
   (a) Plot
      i. Gitter
         A. Generator - Toggle Gitter An/Aus
         B. Antrieb - Toggle Gitter An/Aus
      ii. Auto Achse
         A. Generator - Passe Achse an
         B. Antrieb - Passe Achse an
   (b) Geräte
      i. Initialisiere
         A. „Gerätename“ initialisieren
      ii. Lese Limits erneut
         A. für „Gerätename“
      iii. Eurotherm - HO lesen
   (c) Rezepte
      i. Synchro Start
         A. Starte Synchron Modus
      ii. Synchro beenden
         A. Beende Synchron Modus
      iii. Neu Einlesen
         A. Alle Geräte

In VIFCON selbst, sehe das Menü wie in Abbildung 63 aus. Die Abbildung zeigt dabei den Geräte-Pfad zur Initialisierung. In der Liste steht „Gerätename“ dafür das alle ausgewählten Geräte, wie es in Abbildung 63 gezeigt, dort stehen.

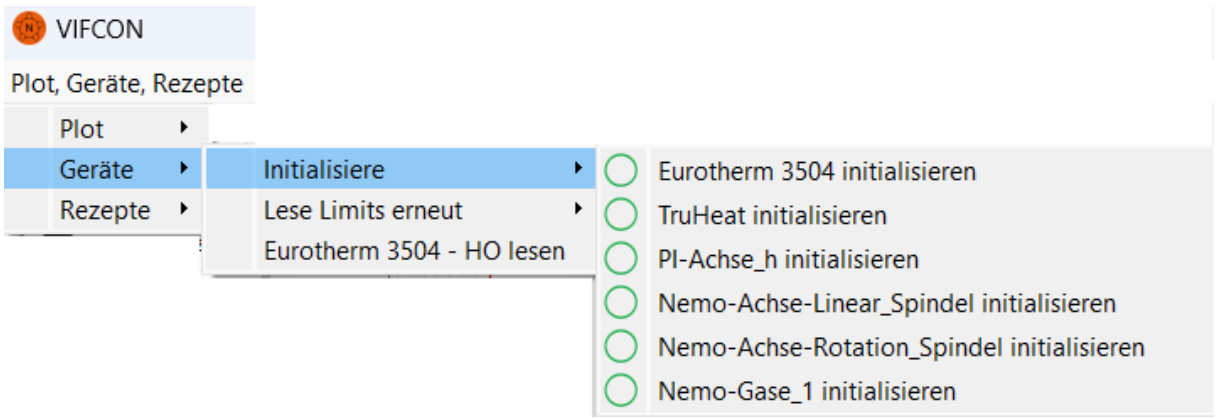


Abbildung 63: Darstellung der Menü-Funktion (Quelle: eigene Darstellung)

In VIFCON gibt es somit drei Ebenen der Ausführung. Es können Funktionen speziell für das Gerät, speziell für den Geräte-Typ (Generator, Antrieb) oder für alles angewendet werden. Neben diesem Fakt muss eine Checkbox beachtet werden, damit die synchronen Funktionen funktionieren. Als Alternative hätten diese Funktionen auch in den Tab Steuerung integriert werden können, doch das hätte wieder Platz gekostet. Da es sich um übergeordnete Funktionen bzw. auch um Sicherheitsaspekte (Eurotherm HO) handelt, ist es besser, wenn diese in der Menü-Leiste zu finden sind. Somit kann der Anwender diese bei Bedarf nutzen, ohne diese immer sehen zu müssen.

5.9.4 Pop-Up-Fenster

In VIFCON können durch **PyQt** mit dem *QMessageBox*-Objekt Pop-Up-Fenster entworfen werden. In VIFCON gibt es zwei verschiedene Pop-Up-Fenster. Diese sind:

1. das für die Sicherheitsabfrage zum Programm-Ende und,
2. das Pop-Up-Fenster, welches dem Nutzer Warnungen oder Informationen liefert.

Die genaue Programmierung und das Aussehen dieser Fenster kann in Anhang E.7 gesehen werden. Folgend soll noch der Nutzen dieser Pop-Up-Fenster genannt werden. Das erste Fenster ist das zum Beenden. Wenn der Nutzer von VIFCON auf das X in der oberen rechten Ecke (Windows-Leiste) der VIFCON-GUI drückt, soll das Programm beendet werden. Dieses X kann auch versehentlich betätigt werden, z.B. durch einen falschen Klick an der Stelle. Damit dies nicht geschieht, öffnet ein Pop-Up-Fenster, das den Nutzer fragt, ob dieser tatsächlich das Programm beenden will. Somit erfüllt dies eine Art Sicherheit. Die anderen Pop-Up-Fenster werden durch bestimmte Konfigurationen oder Knopf-Betätigungen ausgelöst. Der Nutzer erhält eine Information oder Warnung und kann das Fenster durch Betätigung von Okay beenden. Alle diese Fenster sollen den Nutzer somit über bevorstehende Aktionen vorwarnen und haben daher einen Sinn für die Sicherheit.

5.9.5 Plot

Ein zentrales Feature der VIFCON GUI ist der Plot. Der Plot beinhaltet drei Achsen (2x y-Achse, eine x-Achse) und eine Legende. Durch die einzelnen Geräte-Widgets (Programme/Klassen) wird der Plot mit Kurven gefüllt, die durch die Config-Datei eingestellt werden können. Zur Plot-Programmierung wird mehr in Anhang E.3 gesagt.

Der Plot ist bei den Geräte-Typ-Widgets (Hälften des Tabs Steuerung) immer über den Geräte-Widgets zu finden. In der Zeile des Plots sind weiterhin noch die Legende, der Cursor und die Typen übergeordneten Knöpfe wieder zu finden. Der Plot beinhaltet kurz gesagt Achsen (Kapitel 5.9.5.1), eine Legende (Kapitel 5.9.5.2) und verschiedene Kurven (Kapitel 5.9.5.3). Als weitere Feature gibt es noch die Skalierungsfaktoren (Kapitel 5.9.5.4) und einen Cursor (Kapitel 5.9.5.5). Der Plot ist ein Live-Plot, welcher nach einer Messzeit die Daten selbst aktualisiert und eine Änderung über die Zeit darstellt.

5.9.5.1 Achsen

Ein Teil des Plots ist die Anzeige der gemessenen Werte von den Geräten. Dazu werden Kurven in einen Plot eingetragen. Diese Plots bestehen üblicherweise aus einem Koordinatensystem mit x-Achse (Ordinate) und y-Achse (Abszisse). Jedes Gerät bringt Größen mit sich, die in einem Plot gezeigt werden. Zum Beispiel bringt der Eurotherm-Regler die Temperatur in °C und die Ausgangsleistung in % mit sich, während der TruHeat Generator die Leistung in kW, den Strom in A, die Spannung in V und die Frequenz in kHz zum Plot hinzufügt. Somit gibt es 6 Größen, die in dem Plot untergebracht werden müssen. Um nicht alles auf einer Seite zu haben, wurde eine zweite y-Achse erstellt, sodass die Größen aufgeteilt werden können. In Abbildung 60 kann die Aufteilung gesehen werden. Hierbei wurde sich so entschieden, dass die Leistung auf der rechten Seite angezeigt wird und die anderen Haupt-Größen des Gerätes auf der linken Seite sind. Die Frequenz wird nur ausgelesen und wurde noch auf die rechte Seite gesetzt. Bei der Generator-Seite ist die Aufteilung kniffliger als bei den Antrieben. Dort wurde es so gesetzt, dass die Positions-Größe auf der linken Seite ist und die Geschwindigkeit rechts ist.

Auf der Antriebs-Seite in Abbildung 60 kann ein Problem mit den y-Achsen gesehen werden. Es ist nicht alles zu sehen in der Achsenbeschriftung der einen y-Achse. Dieses Problem kommt durch die Größe des Fensters. Neben diesem Umstand kann es auch sein, dass die Größen von ihrem Wertebereich nicht zueinander passen. Dies ist z.B. beim TruHeat aufgefallen. Die Frequenz ist deutlich größer als die Leistung. Beide Probleme werden durch die Skalierungsfaktoren der Config-Datei behoben. Die Erläuterung folgt dazu in Kapitel 5.9.5.4. Kurzgefasst kann ein Label durch einen Skalierungsfaktor von Null aus der Achsenbeschriftung entfernt werden. Dies ist nützlich, da z.B. die PI-Achse und die Nemo-Achsen nicht zusammen genutzt werden. Auch bei Eurotherm und TruHeat ist dies wahrscheinlich, da ein Eurotherm einen TruHeat steuert und der Generator dann nicht über VIFCON direkt gesteuert wird.

Ein wichtiger Aspekt ist das Gitter, dass durch die Achsen erstellt wird. Das Problem hierbei ist, dass die Wertebereiche der y-Achsen nicht identisch sind, wodurch die Gitter einen Offset haben. Um trotzdem die Gitter zu erkennen, wurden diese unterschiedlich eingefärbt. Die linke Achse ist schwarz, die rechte rot. Bei der Side-Legende wurden diese Farben auch auf die Checkboxen angewendet. Neben der Unterscheidung kann das Gitter auch unsichtbar gemacht werden, was in Kapitel 5.9.3 erläutert wurde.

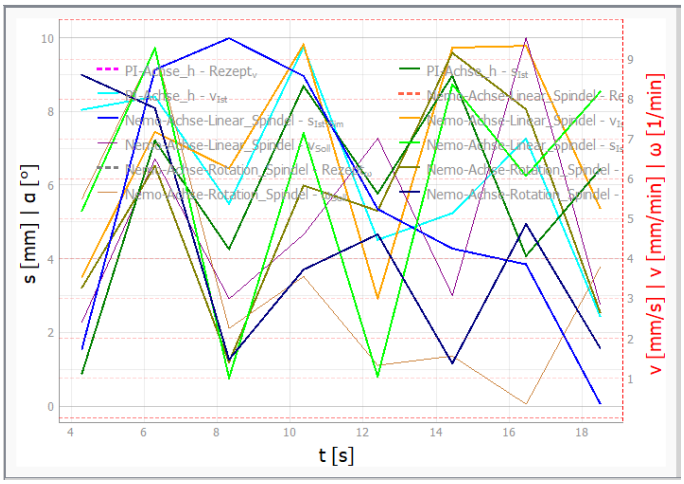
Zusammenfassend ist zu sagen, dass es zwei y-Achsen für die Messgrößen und eine Zeit-Achse (x) gibt. Aus den Achsen entstehen dann zwei Gitter, die farblich unterschiedlich sind, damit sie unterschieden werden können. Weiterhin können die y-Label durch die Config-Datei angepasst werden, wenn die Skalierungsfaktoren bearbeitet werden. Wie die Codierung zu dem Plot und den Achsen ist, kann bei Interesse in Anhang E.3 angesehen werden.

5.9.5.2 Legende

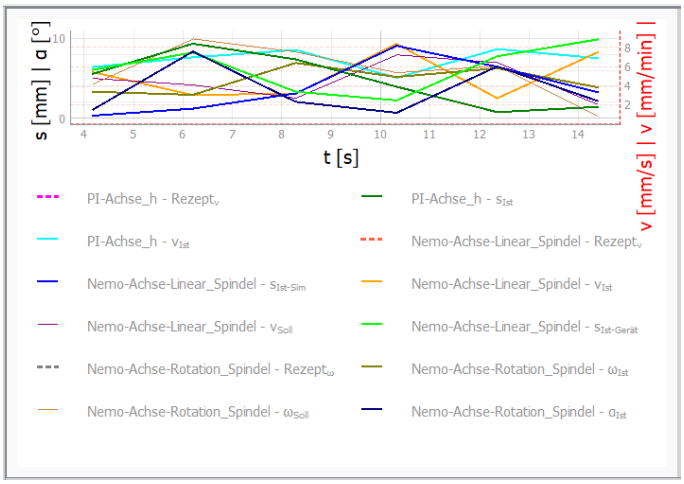
Das nächste Plot-Feature ist die Legende. Üblicherweise wird die Legende mit einem Aufruf einer Funktion erstellt, wodurch die Legende in dem Plot angezeigt wird. Auch der Plot besteht aus Elementen, die aneinander geheftet werden. Doch auch hier gibt es Probleme. Einerseits verdecken die Kurven die Legende und anderseits nimmt die Legende extrem viel Platz ein. Für diesen Zweck wurden zwei weitere Legenden-Typen definiert. Diese Typen sind unter und neben dem Plot. Alle drei Legenden-Positionen sind in Abbildung 64 zu sehen. Die Abbildung 64a zeigt die Position in dem Plot. Der Plot wird dabei recht groß angezeigt, jedoch sind die Label aufgrund der Größe und der Masse an Kurven nicht erkennbar. Die Erweiterung dafür ist Abbildung 64b. Die Legende wird nicht im Plot angezeigt, sondern darunter. Dadurch werden die Kurven nicht mehr verdeckt und die Legende ist lesbar, jedoch wird der Plot sehr gequetscht. Die letzte Variante aus Abbildung 64c vereint die Vorteile der anderen beiden. Der Plot ist groß genug und erkennbar und die Label der Legende verursachen keine Probleme. Die Varianten In und Out sind Teil der **pyqtgraph**-Programmierung, während die Side-Variante auf der Funktion *setVisible* beruht. Die Label und die Checkboxen wurden mit **PyQt**-Objekten erstellt und durch *clicked.connect* an eine Funktion gehängt, die die Kurvensichtbarkeit toggelt. Hier wird nun auch die Funktion *setToolTip* genutzt, wodurch die Label stark verkürzt werden konnten. Dieser ToolTip (Anzeige Gerätename) ist in Abbildung 64c unten links zu sehen und wird sichtbar, wenn über Checkbox oder Label gefahren wird.

Die beste Option für die Legende liefert somit die Side-Variante. Durch die separate Programmierung von **pyqtgraph** können hier auch die Splitter verwendet werden, wodurch die Legende auch „ausgeblendet“ werden kann. Dieser Umstand heißt nicht, dass es die anderen beiden Varianten nicht mehr gibt. Über die Config-Datei (Anhang B) kann die Variante gewählt werden. Neben diesem Umstand können auch die Kurven bei den Geräten ausgewählt werden. Wenn z.B. nur wenige Kurven angezeigt werden, können sich die anderen beiden eher lohnen. Bei diesen beiden (In und Out) kann noch eine Anzahl angegeben werden. Damit legt der Nutzer fest, wie viele Legenden-Einträge in einer Zeile stehen. Eine Erweiterung, die im späteren noch getroffen werden kann, ist die Zuweisung zu den Achsen, die bei Side schon umgesetzt wurde. Bei der Side-Variante gibt es noch eine flexible Umsetzung. In der Config gibt es noch die Funktion „position“, wenn dort „rl“ angegeben wird, sieht dies wie in Abbildung 64c aus. Bei „r“ und „l“ wird die Legende entweder nur **r**echts oder **l**inks angefügt. Durch diesen Umstand wird ein Splitter Widget eingespart und es wird der Vorteil der Side-Variante behalten. Diese beiden Varianten können sich in Abbildung 90 im Anhang E.4 angesehen werden. In dieser Abbildung kann noch ein Punkt gesehen werden. Die Reihenfolge der Legenden-Label hängt von der Kurven-Erstellung ab. Jedes Gerät hat in der Config einen Schlüssel „legend“, der einen String als Wert beinhaltet. Dieser String legt die Reihenfolge der Label in der Legende fest. Weiterhin sind die Geräte somit immer als Block in der Legende zu finden, was nur durch die Achsen-Zuweisung (nur bei Side rl) getrennt wird. Auch hierzu findet sich im Anhang E.4 die Programmierung wieder.

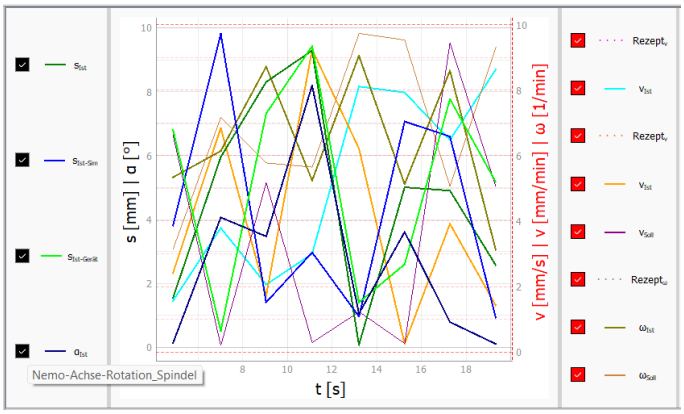
Somit verfügt VIFCON über drei verschiedene Varianten der Darstellung der Legende. Die Side-Variante hat sich aufgrund ihres geringen Platzbedarfes und keiner Quetschung des Plots bewährt. Wie erwähnt wurden die anderen beiden Varianten weiter im Programm gelassen, damit jeder Nutzer sich die GUI so anpassen kann wie dieser es möchte.



a) Legendenoption In



b) Legendenoption Out



c) Legendenoption Side

Abbildung 64: Position der Legende (Quelle: eigene Darstellung)

5.9.5.3 Erstellung einer Kurve

Ein weiterer wichtiger Punkt sind die Kurven. Alle Kurven werden in den einzelnen Geräte-Objekt-Widgets (z.B. **\view\eurotherm.py** - class *EurothermWidget*) erstellt. In Kapitel 5.9.5.2 wurde bereits erwähnt, dass die Reihenfolge der Erstellung durch die Config-Datei erfolgt. Um die Programmierung etwas leichter zu machen, werden alle Kurven, die möglich sind, fest einprogrammiert. Fehlt die Kurve in der Config, wird sie nicht erstellt. Hierfür wird ein Dictionary erstellt, welches in Anhang E.3 neben der Programmierung anhand eines Beispieles gezzeigt wird. Bei der Side-Variante der Legende wurde in Kapitel 5.9.5.2 erwähnt, dass die Variante „rl“ von der angegebenen Achse abhängt. Dies wird durch ein Element in dem Dictionary ermöglicht. Dieser muss jedoch auch bei der Erstellung der Kurven beachtet werden. Die Erstellung für die rechte und linke Achse ist unterschiedlich, dazu mehr im Anhang.

Insgesamt gibt es in VIFCON 4 Arten von Kurven. Für diese 4 Arten muss nun ein Schema entworfen werden. Dazu gehört die Linienfarbe und der Linientyp der Kurven. **PyQt** liefert mit der *Qt*-Klasse (Inhalt von *QtCore*) fünf feste Linientypen. Diese sind Solidline, Dashline, Dotline, Dash-Dot-Line und Dash-Dot-Dot-Line. Die Kurvenarten und die dazugehörigen Linientypen, werden in Tabelle 23 gezeigt.

Tabelle 23: Kurven-Tabelle (Quelle: eigene Darstellung)

Kurvenart	**Linie**	**Besonderes**
Istwert	Solidline	Liniendicke 2
Sollwert	Solidline	/
Rezept	Dotline	Nur bei Rezept Auswahl sichtbar!
Grenzen	Obere: Dashline, Untere: Dash-Dot-Dot-Line	Farbe des Istwertes

In der Tabelle 23 wird in der Spalte Besonderheiten drei Dinge erwähnt. Die Istwerte geben den aktuellen Wert der Geräte zurück. Damit diese immer in dem Wirrwarr der Kurven gefunden werden können, wurde dort die Dicke erhöht. Bei den anderen Kurven ist dies nicht so wichtig, da dort die Linientypen geändert worden sind. Die Rezept- und Grenzkurven sind somit erkennbar. In VIFCON sind 23 Farben definiert. Hierbei bekommen die Rezepte, Sollwerte und Istwerte ihre eigenen Farben. Bei den Grenzkurven wird dieselbe Farbe wie bei den Istwerten genutzt, damit diese zugeordnet werden können. Auf der GUI sind die Text-Label für die Messgrößen auch farblich markiert, sodass eine Zuordnung zu den Kurven gewährleistet ist. Dies ist in Abbildung 60 zu sehen. Die Rezept-Kurven werden nur bei Rezept-Auswahl und Rezept-Start angezeigt.

5.9.5.4 Skalierungsfaktor

Das erste Feature, welches hier erwähnt werden soll, ist der Skalierungsfaktor. Diese Optimierung wurde aufgrund des Experimentes aus Kapitel 6.1 getroffen. Bei den Testen mit TruHeat (Kapitel 3.3) ist aufgefallen, dass die 4 Messgrößen (Spannung, Strom, Leistung und Frequenz) stark verschiedene Wertebereiche haben. Wenn sich die Abbildung 71 angeschaut wird, kann auf der rechten Seite gesehen werden, dass die y-Achse rechts die Temperatur, den Strom und die Spannung enthält. Zum Beispiel ging bei dem Experiment die Spannung maximal bis zu 1200 V und der Strom maximal bis zu 35 A. Weiterhin geben TruHeat und auch Eurotherm (Kapitel 3.1) die Leistung in unterschiedlichen Einheiten wieder. Durch die Anzahl von 2 y-Achsen gelingt es zwar, eine Verteilung der Größen zu erreichen, aber können diese sich, wie gezeigt, von ihrem Wertebereich stark unterscheiden. Aus dem Grund müssen Skalierungsfaktoren eingearbeitet werden.

Die Lösung wird durch die Config-Datei ermöglicht. Zur Config wird im Anhang B mehr gesagt. Die Skalierungsfaktoren liegen in der Config unter dem Schlüssel „skalFak“. In diesem Dictionary finden sich nun alle Größen, die genutzt werden können, wieder. Dort können nun die Faktoren angegeben werden. Wenn dort eine Null angegeben wird, so wird die Kurve aus der Achsenbeschriftung der jeweiligen y-Achse entfernt. Als letztes soll noch die Anpassung der y-Label gezeigt werden. Dafür wird dies anhand von Eurotherm und TruHeat gezeigt. Somit entstehen folgende Label bei den genannten Faktoren:

1. Faktor = 1: T[°C] | I[A] | U[V]
2. Faktor ≠ 1: T[°C]x0.1 | I[A] | U[V]
3. Faktor = 0: I[A] | U[V]
4. Alle Faktoren = 0: Keine Einträge!

Die ersten drei beziehen sich auf die Größe Temperatur des Eurotherm-Reglers. Beim letzten werden alle Faktoren einer Achse auf Null gesetzt. Die Reihenfolge dieser Label-Teile ist fest programmiert. Die Null löscht somit Teile des Strings. Bei Interesse kann mehr zu den Skalierungsfaktoren in Anhang E.2 gelesen werden.

5.9.5.5 Anzeige der Cursor-Werte

Der Cursor ist eine weitere Erweiterung des Plots. Wie die Side-Legende ist der Cursor kein direkt im Plot existierendes Widget. Dieser wurde in das Widget der übergeordneten Geräte-Typen-Knöpfe eingebettet. Dabei sollte dieser in diesem Widget unten sein. Der Cursor hat die Aufgabe, die Koordinaten der Maus in dem Plot anzuzeigen. Somit kann der Nutzer die Werte eines Punktes herausfinden. Wichtig ist hierbei zu wissen, dass der Cursor nur bei Bewegung updatet und nicht, wenn sich der Plot aktualisiert. Die Programmierung des Cursors kann bei Interesse in Anhang E.5 nach gelesen werden.

5.10 Inbetriebnahme von VIFCON auf Raspberry Pi

Die Installation von dem Raspberry Pi Betriebssystem und den python-Bibliotheken wurde in Kapitel 4.6 erläutert. Die Python-Bibliotheken die für VIFCON gebraucht wurden, konnten installiert werden. Das Aussehen von VIFCON auf dem Raspberry Pi kann in Abbildung 65 gesehen werden. Die anderen Abbildungen zur GUI zeigen die Darstellungen auf den anderen Systemen. Zum Beispiel zeigt Abbildung 60 die GUI in Windows, Abbildung 70 die GUI auf dem Labor-Notebook (Linux) und Abbildung 73 die GUI auf dem Nemo-1-Rechner (Linux). Alle diese GUIs haben kleinere Unterschiede an sich, doch bei dem Raspberry Pi gab es ein paar größere Probleme, die im weiteren genannt werden.

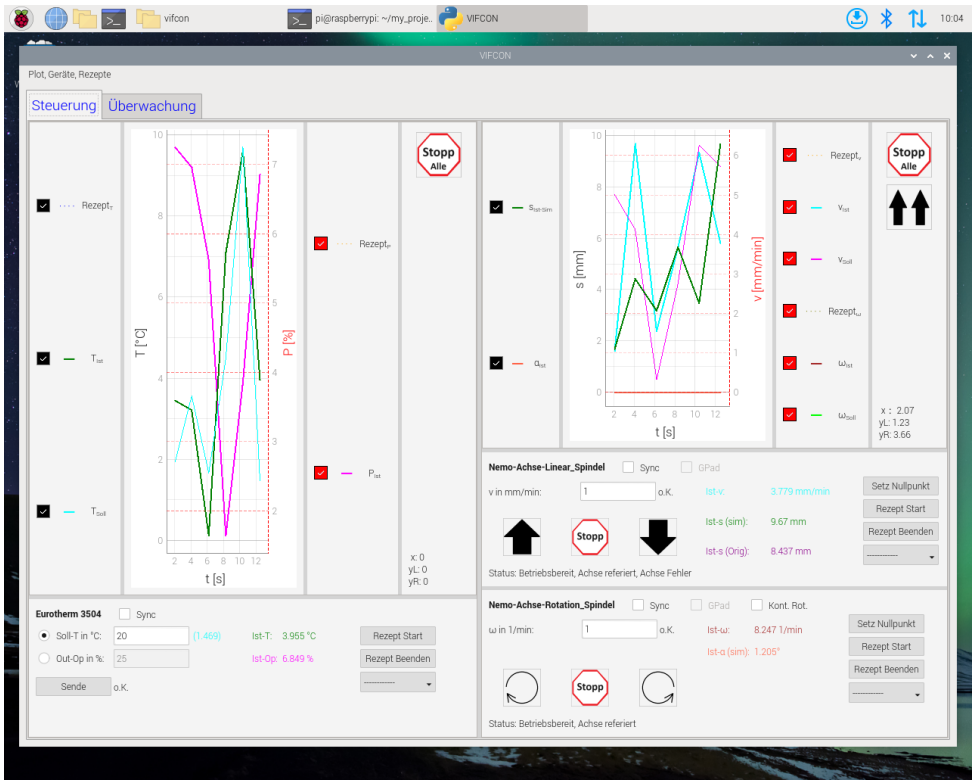


Abbildung 65: Aussehen der GUI von VIFCON auf Raspberry Pi (Quelle: eigene Darstellung)

Ein großes Problem war bei den ersten Test mit Raspberry Pi die Größe der GUI. Die VIFCON GUI hatte ursprünglich eine Einstellung von 1400x900 Pixel, doch waren es tatsächlich 1540x900 Pixel. Dieser Unterschied kam durch die sehr breiten Geräte-Widgets. Als Schlussfolgerung wurde daraus gezogen, dass VIFCON noch etwas schmaler werden muss. VIFCON muss auf den verschiedensten Bildschirmen anzeigbar sein. Nach dem einige Punkte verbessert wurden, hat die GUI nun eine Mindestgröße von 1240x900 Pixel, die von den Anwendern von VIFCON im Hinterkopf behalten werden muss. Im Vergleich zu den anderen beiden Betriebssystem, verschwinden einige Features der GUI bei Raspberry Pi. Diese sind:

1. die Schriftgröße (wurde direkt am Raspberry angepasst),
2. die Farben der Label bei den Radio-Buttons und,
3. die Icons im Menü und der Windows-Leiste.

6 VIFCON Tests in Experimenten

Um die Funktion und die Benutzerfreundlichkeit von VIFCON zu testen, wurden verschiedene Experimente durchgeführt. Dabei wurden die verschiedenen Modellanlagen der Modellexperimente-Gruppe am IKZ genutzt.

In den folgenden Kapiteln sollen diese Experimente beschrieben und deren Ergebnisse diskutiert werden. In Kapitel 5 wurden bestimmte Umsetzungen erwähnt. Diese beschreiben den letzten Stand von VIFCON vor Abgabe dieser Arbeit. Bei den folgenden Experimenten wird nun der Grund für mancher dieser VIFCON-Teile erläutert, da diese während der Experimente als Optimierungspunkte gefunden wurden. Somit dienen die Experimente auch der Weiterentwicklung von VIFCON. Nach Beendigung der Master-Arbeit soll dem IKZ eine Steuerung bleiben, die diese weiterhin verwenden können.

6.1 Experiment: Demo FZ

Das Experiment wurde unter keinen besonderen Vorkehrungen durchgeführt, sondern dient dem ersten Test von VIFCON. Das Experiment wurde von Iason Tsiapkinis [Perf] durchgeführt, der die Steuerung im weiteren nutzen soll. Somit soll geschaut werden ob das Programm in seinem vorliegenden Stand funktioniert und wo Optimierungen getroffen werden müssen.

Der Stand von VIFCON ist folgender:

1. VIFCON besitzt nur den Tab Steuerung, in dem die Geräte:
   a. Eurotherm (Kapitel 3.1),
   b. PI-Achse mit Mercurry Controller C862 und C863 (Kapitel 3.2) und,
   c. TruHeat HF Generator (Kapitel 3.3)

   eingefügt werden können. Somit besitzt es 2 Geräte des Typen „Generator“ und ein Gerät vom Typ „Antrieb“.
2. Die Geräte werden über Threads bearbeitet und können auch die selbe Schnittstelle haben, da die Funktion der *QMutex* und *QMutexLocker* eingebaut sind.
3. Bei dem TruHeat-Generator sind zu dem Zeitpunkt nur die Lesefunktion eingearbeitet. Das Schreiben von Werten wird derzeitig noch nicht unterstützt.
4. Das Programm verfügt noch über keine Aufnahme von Messdaten.

Im Laufe dieses Experimentes wurden die Punkte 3 und 4 eingearbeitet. Bei dem Experiment wird die Positionierung der Achsen, die Kontrolle des Generators und das Aussehen der GUI in den Mittelpunkt gestellt.

6.1.1 Aufbau und Ablauf

Neben der in Abbildung 35 gezeigten Nemo-1-Anlage, die das CZ-Verfahren zur Züchtung verwendet, gibt es in dem Labor auch noch eine Anlage für das FZ-Verfahren. Diese Anlage wird Demo-FZ genannt. Das Experiment bzw. der erste VIFCON Test wird an dieser durchgeführt. Im Folgenden sollen Aufbau und Durchführung des Experimentes erläutert werden.

6.1.1.1 Aufbau des Experimentes

In Abbildung 66 wird der Aufbau in einer Skizze gezeigt. Das VIFCON-Programm wird dabei über den Laborrechner (Laptop/Mess-Notebook) gestartet und ausgeführt. Der Rechner (Laptop, Raspberry PI) hat dabei ein Linux-Betriebssystem, was in dem Labor der Modellexperimente-Gruppe hauptsächlich verwendet wird. Aus der Skizze kann weiterhin abgelesen werden, dass zwei RS232-Schnittstellen verwendet wurden.

Aus Abbildung 66 können die genutzten Geräte abgelesen werden. Genutzt wurden:

1. das Mess-Notebook mit Linux-Betriebssystem,
2. ein weiterer größerer Bildschirm (Anzeige von VIFCON),
3. 2x Mercury DC-Motor Controller C-862 (schwarz) (Abbildung 23),
4. 2x Mercury Motor Controller C-863 (grau) (Abbildung 21),
5. 4x PI-Achse (Abbildung 20),
6. ein RS232-USB Adapter,
7. ein selbst gebauter Adapter für die Nutzung des TruHeat Generators (Abbildung 32) und,
8. der TruHeat HF 5010 (Abbildung 24).

Aus Kapitel 3 ist bekannt, dass die Geräte PI-Achse (Tabelle 7) und TruHeat (Abbildung 31) eine spezielle RS232-Verkreuzung haben. Während die PI-Achse ein passendes Kabel liefert, musste für TruHeat selbst eins hergestellt werden. Weiterhin ist noch ein Induktor an den Außenkreis des Generator angeschlossen. Dabei wird das Kühlwasser und die Leistungsversorgung über den Außenkreis an den Induktor geleitet. Dieser Punkt wurde in Kapitel 3.3.1 gezeigt und erläutert. Der Außenkreis ist in Abbildung 28 zu sehen.

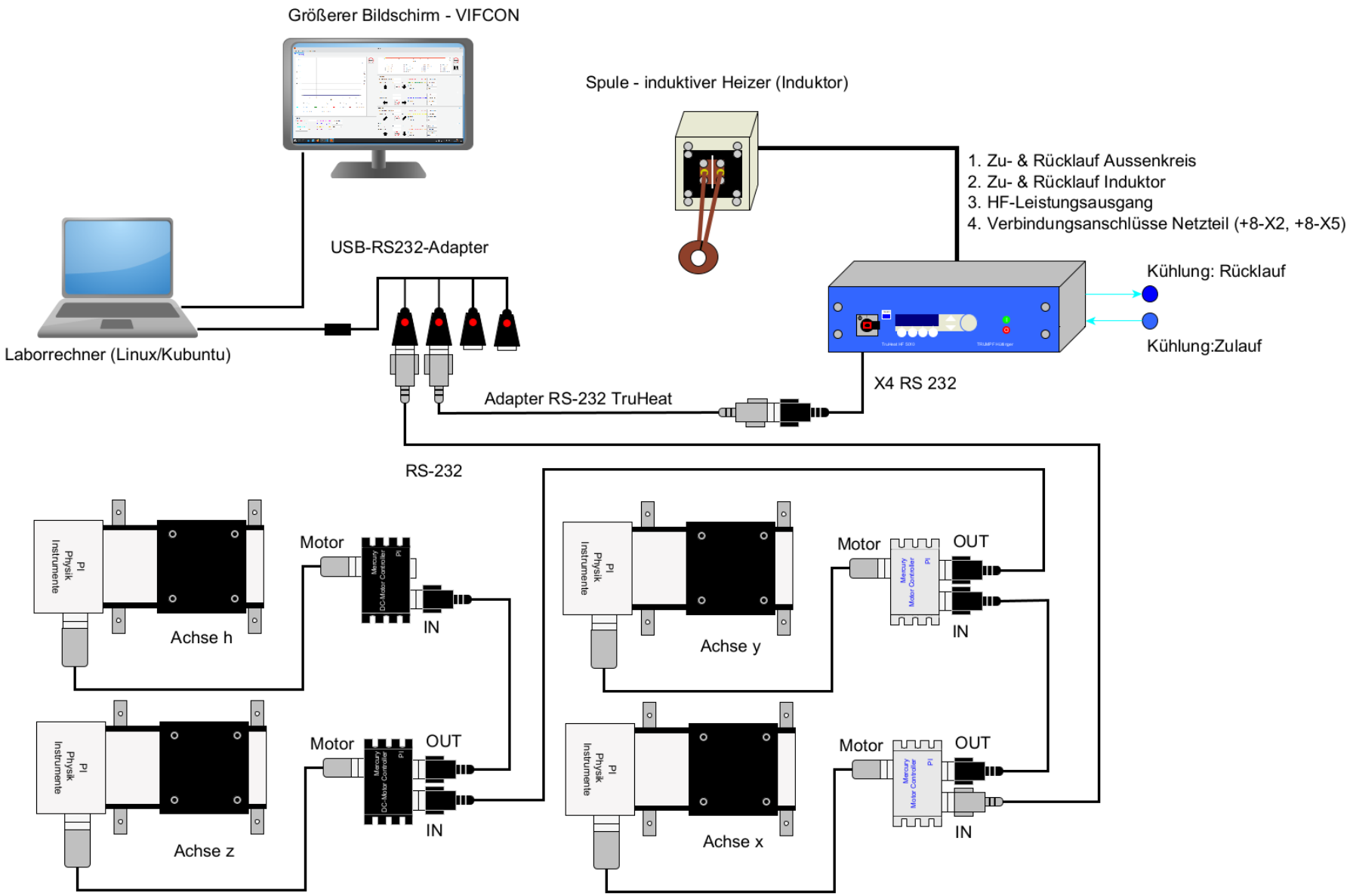


Abbildung 66: Skizze vom Aufbau des Experimentes (Quelle: eigene Darstellung)

In Abbildung 66 werden die vier PI-Achsen gezeigt. Die Anordnung der Achsen und die Bedeutung der Richtungen x, y, z und h können in Abbildung 67 gesehen werden. Die Achse z kann nach dem Aufbau durch die x- und y-Achse verschoben bzw. positioniert werden.

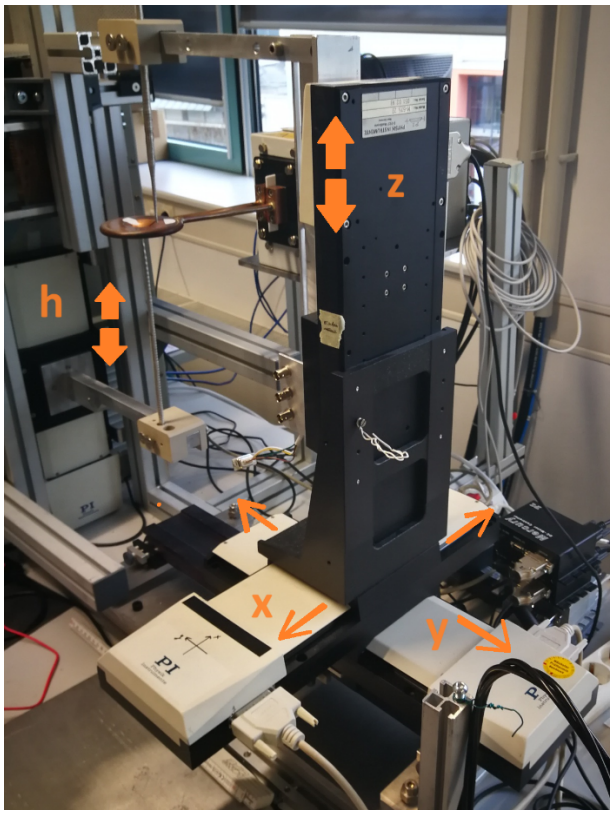


Abbildung 67: Richtungsangabe an den Achsen (Quelle: eigene Darstellung)

Die Achsen z und h haben dabei die selbe Richtung, sind aber entgegengesetzt aufgebaut. Dies bedeutet die Position des Logos (PI) ist bei z oben und bei h unten (siehe auch Abbildung 80). Neben diesem Umstand gibt es noch einen flachen Induktor (in Abbildung 67 zu sehen), der für das Aufschmelzen des Vorratsstabs und die Tropfenbildung (Keim, Vorratsstab) genutzt wird. Der Induktor bildet den Heizer für dieses Verfahren und ist wie erwähnt mit dem Generator verbunden. Für das Experiment wurden zwei fertige Kristalle als Vorratsstab und Keim verwendet, um das Positionieren und das Bewegen der Achse zu testen. Diese wurden in die an die Achsen h und z montierten Halterungen eingespannt.

6.1.1.2 Vorbereitung und Durchführung des Experimentes

Im Experiment wurden die Achsen x und y nur zur Positionierung der besagten Kristalle verwendet. In einem richtigen FZ-Versuch würde die Achse z den Vorratsstab bzw. das zu schmelzende Material und die Achse h den Impfkeim halten. Nach der Positionierung wurden die Achsen x und y nicht weiter verwendet. Durch die Splitter-Funktion in der GUI können diese nun einfach verdeckt werden oder durch einen Neustart des Programms aus der Config-Datei genommen werden. Die Startposition und die Bewegung sind in Abbildung 68 als Skizze gezeigt. Die Skizze zeigt eine Ansicht von vorne. Der Induktor ist dabei der Braune Strich am Induktor. Die Reale Endposition der Positionierung wird in Abbildung 69 gezeigt. Diese Position bildet den Start der weiteren Tests.

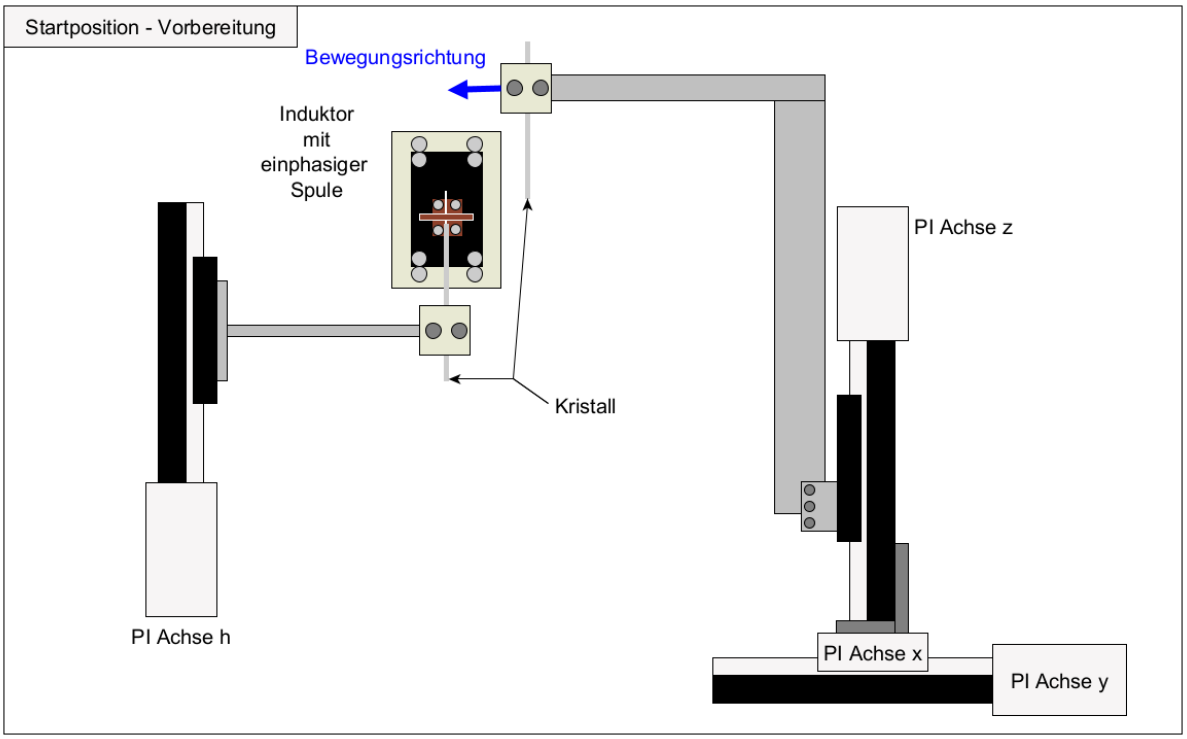


Abbildung 68: Skizze der Positionierung der Achsen (Quelle: eigene Darstellung)

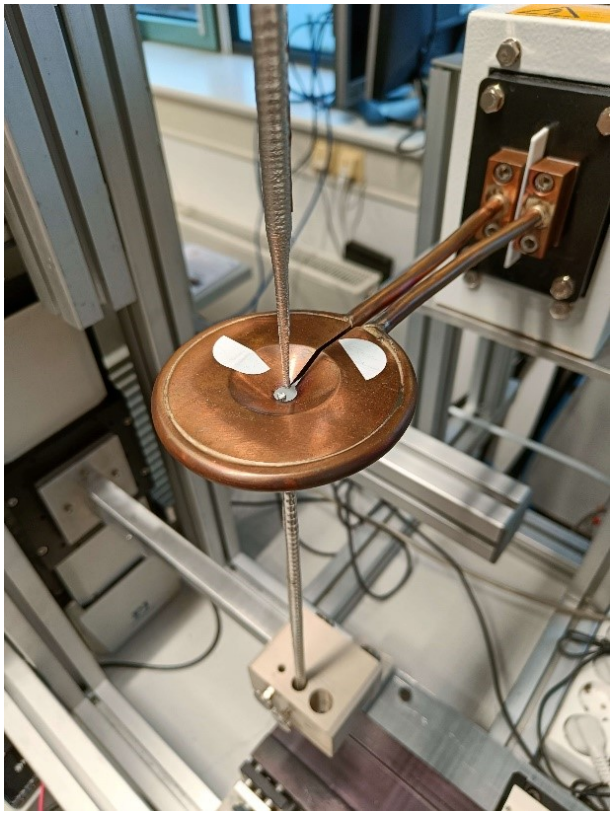


Abbildung 69: Position nach Positionierung - Experiment Startposition (Quelle: [Perf])

Wie schon erwähnt ist dies der erste größere Test von VIFCON. Mit dem Experiment sollen die beiden Geräte-Typen und auch die einzelnen Funktionen wie Knöpfe und Rezepte getestet werden. Das Experiment umfasst somit konkret:

1. die Positionierung mit Hilfe der Achsen h, z, x und y,
2. das gleichzeitige Fahren der Achsen h und z (Züchtungs-Achsen),
3. die Funktion der Elemente (Widgets) auf der GUI,
4. die Nutzung des TruHeat Generators und,
5. die Nutzung der Rezept bei mehreren Geräten gleichzeitig.

Außerdem sollen Schwierigkeiten mit der GUI und den Geräten aufgedeckt werden, die im späteren dann zu Optimierungen oder neuen Ideen führen. Diese Punkte, sowie die Auswertung des Experimentes werden in Kapitel 6.1.2 angesprochen.

6.1.2 Auswertung

Nachdem der Aufbau des Experimentes und das Experiment selbst in Kapitel 6.1.1 erläutert wurden, sollen in diesem Kapitel nun die Auffälligkeiten und ihre Verbesserungen gezeigt werden.

6.1.2.1 Optimierungen an Plot und GUI

Für eine gute Steuerung ist die GUI ein wesentlicher Faktor. Aus dem Grund muss auch die Funktionstauglichkeit, sowie die Benutzerfreundlichkeit dieser in einem solchen Experiment getestet werden. Die Abbildung 70 zeigt dabei die GUI während des Experimentes, während die Abbildung 71 den Stand der optimierten GUI zeigt. Die Optimierungen wurden in Absprache mit der Modellexperimente-Gruppe des IKZ (Benutzer/Tester) so wie gezeigt durchgeführt. Mit den anderen Experimenten wird somit die GUI, sowie die Brauchbarkeit immer weiter verbessert. Die Abbildungen die folgend zu sehen sind unterscheiden sich leicht im Design (Hintergrundfarbe, Splitter-Aussehen, etc.). Dies liegt daran, dass die Abbildungen an verschiedenen Betriebssystemen und Rechnern aufgenommen wurden. Die Abbildung 70 wurde an einem Linux-Rechner und die Abbildung 71 an einem Windows-Rechner aufgenommen. Somit ergibt sich noch ein wichtiger Punkt für VIFCON. Durch die verschiedenen Experimente wird VIFCON auch an verschiedenen Rechnern verwendet und VIFCON muss dabei immer problemlos funktionieren.

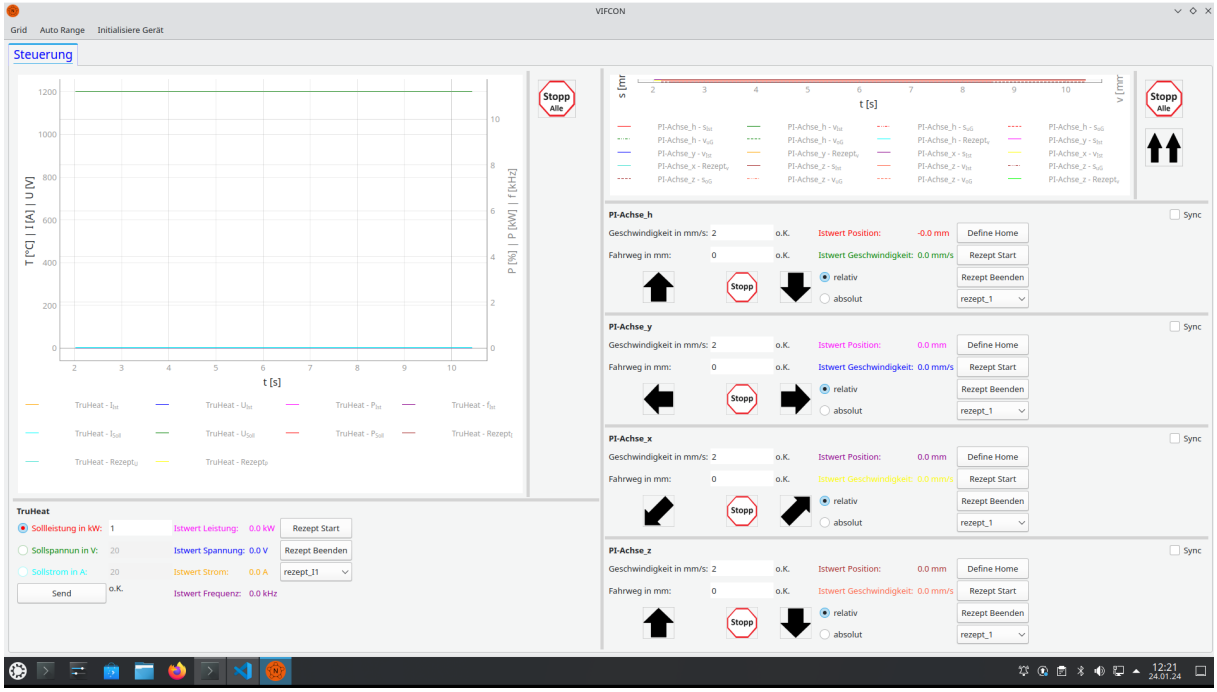


Abbildung 70: GUI zum Zeitpunkt des Experimentes (Quelle: eigene Darstellung)

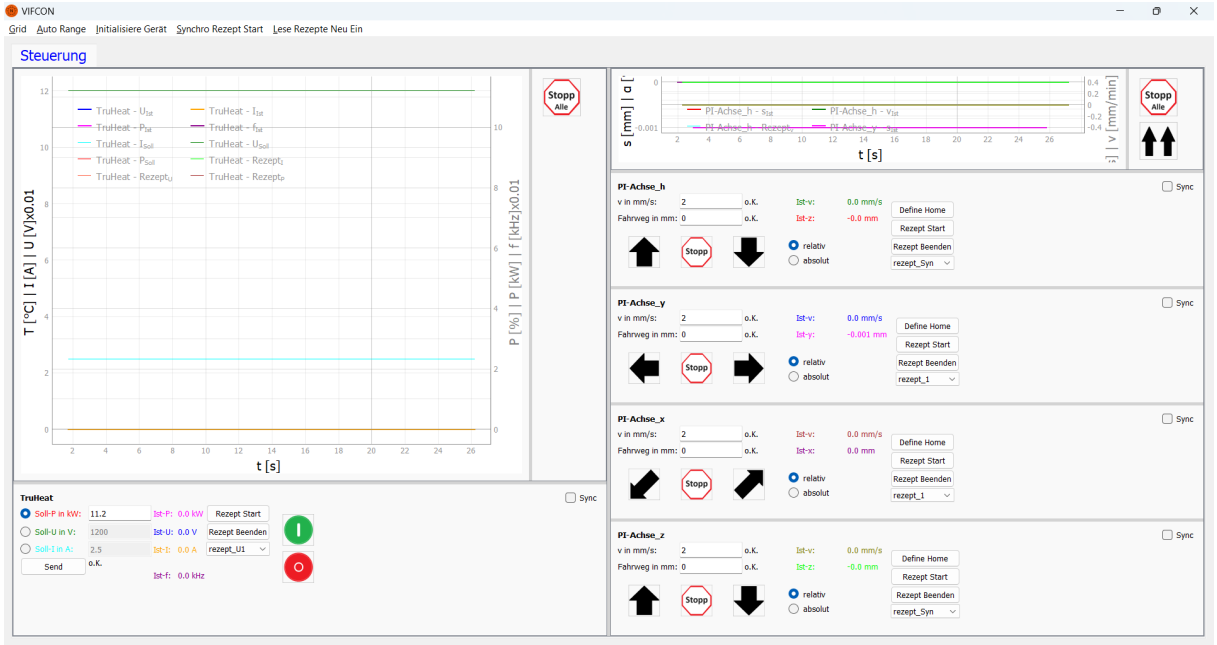


Abbildung 71: GUI Optimiert (Quelle: eigene Darstellung)

Folgende Punkte sind bei der GUI aufgefallen:

1. Die Messgrößen benötigen einen Skalierungsfaktor, da manche Messgrößen viel größer sind als die anderen.
2. Istwerte sollen im Plot hervorgehoben werden.
3. Verbesserung bzw. Optimierung des Geräte-Widget Platzes.

Die Punkte 1 und 2 wurden im Kapitel 5.9 aufgegriffen. Somit bleibt nur noch Punkt 3. Bei diesem Punkt geht es nun um die Geräte-Widgets. In Abbildung 70 kann gesehen werden, dass die Bezeichnungen von Messgrößen und anderem viel zu lang ist. Hierbei wurden bestimmte Größen als ihr Formelzeichen dargestellt. So wurde z.B. Sollleistung zu Soll-P und Istwert Position zu Ist-z (z, x und y sind Richtungen für die Position).

Neben diesem Umstand sollen die Knöpfe auch besser und schöner platziert werden. Dies kann sehr gut an den Bewegungsknöpfen der PI-Achse gesehen werden. In Abbildung 70 haben diese noch verschiedene Abstände zueinander. Die Verbesserung ist in Abbildung 71 zu finden. Für diesen Zweck wurden bestimmte Widgets im Geräte-Widget mit einander verbunden. Auch dazu ist mehr in Kapitel 5.9 zu finden.

Im Sinne des Designs der GUI muss die Legende unbedingt noch weiterentwickelt werden. In beiden Abbildungen kann gesehen werden, dass diese nur schwer zu deuten ist. Die endgültige Legende wird in Kapitel 5.9.5.2 erläutert und wurde aufgrund vom Experiment des Kapitels 6.2 erarbeitet.

6.1.2.2 Neue Funktionen

Bei den neuen Funktionen (Programmerweiterung, Knöpfe, etc.) sind zwei wesentliche Punkte zu betrachten. Zum einem muss die Funktion der Rezepte erweitert werden und zum anderen wird die Speicherung von Dateien und Daten angesehen.

Bei den Rezepten sind zwei neue Funktionen hinzugekommen. Diese Funktionen sind das Synchrone Starten der Rezepte, welche es bei den Antrieben bereits gibt und das neue Einlesen dieser. Alles zu den Rezepten kann in Kapitel 5.5 nachgelesen werden. In diesem wurde der Aufbau der Rezepte und auch alle Funktion erläutert. Diese beiden neuen Menü-Funktionen sind in Abbildung 71 wieder zu finden.

Die neuen Funktionen für Datenspeicherung sind nicht als Knopf oder Menü-Leisten Element eingearbeitet worden, sondern existieren nur im Programm und können durch die Config-Datei verändert werden. Für die Experimente ist es wichtig, dass diese auch Wiederholbar und Reproduzierbar sind. Aus dem Grund sollten bestimmte Dinge gespeichert werden. Dementsprechend wurde die Config-Datei so erweitert, dass der Nutzer durch einen einfachen Booleschen Wert (True, False) entscheiden kann, ob die Log-Datei und Config-Datei, bei Beendigung des Programms in den aktuellen Messdatenordner kopiert werden. Somit kann der Nutzer die Dateien wieder ansehen und auswerten. Beachtet müssen hier die Funktionen, die die Config-Datei erneut auslesen. Dies passiert bei den Knöpfen Define Home und Rezept Update. Am Ende des Programms wird die aktuelle Datei zu diesem Zeitpunkt gespeichert. Die Änderungen sind hier nicht zu erkennen! Weiterhin kann an der Stelle noch gesagt werden, ob die beiden Plots gespeichert werden sollen. Hierbei ist es wichtig zu beachten, dass die Plots so gespeichert werden wie sie im Zustand des Beendens aussehen. Wenn die Kurven durch die Legende auskommentiert sind oder der Plot verdeckt ist, wird nicht alles oder der Inhalt in einer Komprimierten Form gespeichert. Wenn die Option der außenstehenden Legende gewählt wurde, wird diese separat gespeichert.

Auch die Messdatenaufnahme wird dringend für den Zweck der Auswertung des Experimentes benötigt. Für die Darstellung der Daten in der Datei, wurde die selbe Darstellung wie in Multilog [Mul] genutzt. Somit wurden die Funktionen dies bezüglich aus diesem entnommen und an die speziellen Geräte angepasst.

6.1.2.3 Anpassungen für PI-Achse

Auch für die Geräte mussten Dinge angepasst werden. In Abbildung 67 können die vier genutzten PI-Achsen gesehen werden. Auch die Skizze in Abbildung 66 zeigt die Achsen. Die Modelle des Mercurry-Controllers waren C-862 (Achse h und z) und C-863 (Achse x und y). Bei der z-Achse traten während der Experimente Probleme auf. Das Problem kann folgendermaßen beschrieben werden. Die z-Achse wurde synchron mit h gestartet. Bevor die Achse ihr Ziel erreichte, wurden alle Achsen auf einmal gestoppt. Danach werden die beiden Achsen wieder auf Null gefahren. Hierbei fährt die z-Achse über die Null hinaus. Anhand von Messwerten ist dies wie folgt:

1. Achsen fahren synchron von 0 mm zur Absoluten Position z = -10 mm und h = 10 mm.
2. Stopp wird betätigt und die Achsen halten bei den Positionen z = -3,942 mm und h = 3,455 mm.
3. Achsen sollen nun synchron zur Nullposition fahren.
4. Die Achsen erreichen folgende Positionen: z = -6,053 mm und h = 0 mm.
5. Die Achsen sollen erneut auf Null fahren, was diese auch tun.

Um das Problem zu verstehen muss sich die Funktionsweise der Positionsbestimmung der Achse angesehen werden. Mit dem Senden eines MR-Befehls (Fahre die angegebene Position) wird der Motor-Servo-Loop gestartet und die Achse bewegt sich. Neben diesem Umstand wird intern eine Zielposition berechnet. Diese Zielposition kann mit dem Befehl TT ausgelesen werden. Genau dieser Befehl wurde in das Programm eingearbeitet. Da er nur Informationen bringen soll, kann das Auslesen über die Config-Datei Ein- und Ausgeschaltet werden.

Durch die Stopp-Befehle AB1 und AB wird der Servo-Motor-Loop beendet und die Achse steht wieder. Weiterhin wird die Aktuelle Position als Zielpositon gespeichert. Wenn z.B. nur MF (Motor Off) verwendet wird, dann hält die Achse an, doch ändert nicht die Zielposition intern. Wenn danach der Motor angeschaltet wird (ohne Änderung der Zielposition), bewegt sich die Achse zu ihrem programmierten Ziel.

Durch das Auslesen von TT konnte herausgefunden werden, dass die z-Achse den falschen Zielwert im speicher hatte. Hierfür ein Beispiel: Beim Stopp hatte die z-Achse eine Position von -4,522 mm erreicht. Die Zielposition war zu dem Zeitpunkt aber 4,01 mm. Als dann das Fahren zu Null gestartet wurde, berechnet VIFCON anhand der aktuellen Position einen Fahrweg von 4,522 mm, wodurch die Achse dann um 0,5 mm an der Null vorbei fährt.

Im oberen Beispiel wurde die Zielposition gar nicht angepasst. Im Controller lag TT bei -10 mm. Als dieser von ca. -4 mm zu Null fahren sollte, hat der Controller den neuen Fahrweg, die 4 mm, von der Zielposition abgezogen, wodurch diese zu -6 mm wurde. Die Achse fuhr somit in die falsche Richtung. Dieser Fehler kam durch den Stopp-Befehl. Der Unterschied zwischen AB1 und AB ist die Art der Bremsung. Bei AB1 wird sanft gebremst, während AB ein sofortigen harten Stopp bewirkt. Als Lösung wird nun nur noch der AB Befehl verwendet. Das benannte Problem trat nur bei der z-Achse auf. Die h-Achse funktionierte mit demselben Modell Controller problemlos.

Aus dem oben genannten Beispiel kann nun noch etwas abgelesen werden. In dem Beispiel sollten die Achsen synchron laufen. Die Geschwindigkeit und die Startposition waren identisch. Trotz allem war die z-Achse um 0,5 mm schneller als die h-Achse. Diese Verzögerung hat folgende Hintergründe. Über eine For-Schleife werden die einzelnen Bewegungsbefehle nach einander gesetzt. Nach dem dies geschehen ist und die Threads aufgerufen wurden, muss eine der Achsen bis zum Start warten (*QMutex*), da die Achsen über die selbe Schnittstelle betrieben werden und immer nur eine angesprochen werden kann. Wenn nun noch die Lese-Befehle ausgeführt worden sind, verzögert sich der Start immer mehr. So mehr Achsen genutzt werden, desto länger wird auch der Abstand zwischen der ersten und letzten Achse. Wie die Reihenfolge der Ausführung der Achsen ist, wird durch den Computer bestimmt. Um die Zeit etwas zu verbessern, wurde die Auslese-Zeit für die Geschwindigkeit beim C-862 auf 25 ms gesetzt. Somit ist die Verzögerung sehr klein. In dem Beispiel war dieser Wert bei 100 ms. Das C-862 Modell benötigt eine Verzögerung um die Geschwindigkeit richtig zu bestimmen. Beim C-863 wird dies nicht gebraucht. Hier gibt es nur eine Default Verzögerung von 10 ms. Das Ziel hierbei ist es einen Unterschied von kleiner als 0,5 mm, idealer weise 0,2 mm zu bekommen. Dies konnte jedoch noch nicht erreicht werden.

6.1.2.4 Anpassungen für TruHeat Generator

Das Senden und Schreiben der Werte funktionierte Problemlos. Die Funktion des Schreibens wurde während des Experimentes noch implementiert. Während des Experimentes sollte der Generator auch eingeschaltet werden. Dieses Unterfangen wurde durch den Watchdog (siehe Kapitel 5.8.5) gebremst. Mit den Erkenntnissen aus dem Experiment wurde dieser in VIFCON eingearbeitet.

6.1.3 Schlussfolgerungen

Der Erste Stand von VIFCON hat größtenteils positive Erwartungen erfüllt. Nachdem sich der Tester an die GUI gewöhnt hatte, konnten verschiedene Ausführungen an der DEMO-FZ ausgeführt werden. Weiterhin konnten so Probleme mit den Geräten aufgedeckt, Optimierungen für die GUI und Erweiterungen für VIFCON gefunden werden.

Besonders die Nutzung des TruHeat Generators ist in diesem Experiment hervorgetreten. So mussten ganz besonders die Interface-Einstellungen und ihre dazugehörigen Funktionen beachtet werden. Auch die Nutzung des Watchdogs und der verbundene Reset der Fehlermeldungen traten zum Vorschein. Für den Watchdog wurde eine Lösung gefunden, jedoch für den Reset noch nicht. Wenn zu Beginn der Nutzung der Reset eingestellt wurde und dann wieder manuell auf RS232 umgeschaltet wird, dann beginnt der Watchdog erst mit dem ersten Befehl durch VIFCON, was im Falle des TruHeats das setzen des Watchdogs ist.

Ein wichtiger Punkt der aus dem Experiment hervorging ist die Nutzung der Rezepte. Wie in Kapitel 6.1.2.2 gezeigt ist es für den späteren nutzen der Steuerung wichtig, dass die Rezepte gleichzeitig starten. Durch diese Funktion kann der Nutzer mehrere Geräte so laufen lassen wie dieser möchte und muss nur noch auf das Ergebnisse warten. Zum Zeitpunkt des Experimentes war dies nur durch die von einander entfernten Rezept-Start-Knöpfe möglich. Durch die Umsetzung mit Menü-Leiste und Check-Box wird ein synchrones starten gegeben.

Auch wenn es Probleme mit der z-Achse und dem TruHeat gab, kann das Experiment als Erfolg gewertet werden. Die Positionierung und die Knöpfe auf der GUI waren für das Experiment gut nutzbar.

6.2 Experiment: Nemo-1-Anlage

In diesem Experiment wird die Nemo-1-Anlage im Vordergrund stehen. Das Kapitel 3.4 beschreibt dabei diese Anlage und die Geräte die über den Modbus angesprochen werden können.Somit wird in diesem Experiment die zweite Art der Kommunikation zusätzlich betrachtet, welche über LAN und Switches mit dem Computer verbunden wird. In Kapitel 6.1 wurde nur die Kommunikation über RS232 angesehen. Der Versuch der hier vorgenommen wurde ist die Heilung von Thermoelementen oder um es Allgemeiner zu fassen wurde ein Heiztest durchgeführt. Das Experiment an sich wird von dem Zweitprüfer dieser Arbeit und Leiter der Gruppe Modellexperimente Dr. Kaspars Dadzis [Perc] durchgeführt. Im Folgenden werden verschiedene Abbildungen (Plots) gezeigt, die auch Rezepte genutzt haben. Die Rezepte zu den Abbildungen sind in Anhang G.1 zu finden.

Neben dem Stand vom Experiment aus Kapitel 6.1 und dessen Optimierungen, wurde VIFCON inzwischen um Folgendes erweitert:

1. VIFCON hat einen Tab Monitoring bekommen.
2. Die Geräte:
    a. Nemo-Gase (Gerät für das Monitoring),
    b. Nemo-Achse-Linear (Gerät der Steuerung) und,
    c. Nemo-Achse-Rotation (Gerät der Steuerung) wurden hinzugefügt (alle in Kapitel 3.4).
3. Steuerung der Anlage Nemo-1 über Modbus und dessen Register.
4. Weiterentwicklung der Init-Funktion. Startwerte werden nun bei Init (Config-Init auf False) und Programm-Start (Config-Init auf True) vom Gerät ausgelesen.
5. Nutzung und Erstellung einer Rampe im Rezept, durch das Programm und der Config-Datei.

6.2.1 Aufbau und Ablauf

Der Grundlegende Aufbau kann in Abbildung 35a gesehen werden. Auf eine Skizze zum Aufbau wird hier verzichtet, da eine Ausführliche Skizze in Kapitel 6.3 folgen wird. In diesem Experiment werden in VIFCON die Geräte:

1. Nemo-Gase,
2. Nemo-Achse-Linear (Spindel),
3. Nemo-Achse-Rotation (Spindel) und,
4. Eurotherm genutzt.

Über den Eurotherm-Regler wird hierbei ein Generator gesteuert. Bei den Achsen handelt sich um die in Abbildung 39 und 38 gezeigten Antriebe, wobei nur die Spindel genutzt wird. Neben diesen Geräten werden auch verschiedenste Messgeräte genutzt, die über Multilog (Programm des IKZ, [Mul]) ausgelesen werden. Für die Heilung der Thermoelemente wird als Heizer ein Induktor (Spule) genutzt. In diesem steht ein Grafit-Block, in dem die Thermoelemente stecken. Dies kann in Abbildung 72 gesehen werden. Der große Unterschied wird dabei der Rezipient sein. Aus dem Grund wird nur dieser in der folgenden Abbildung gezeigt.

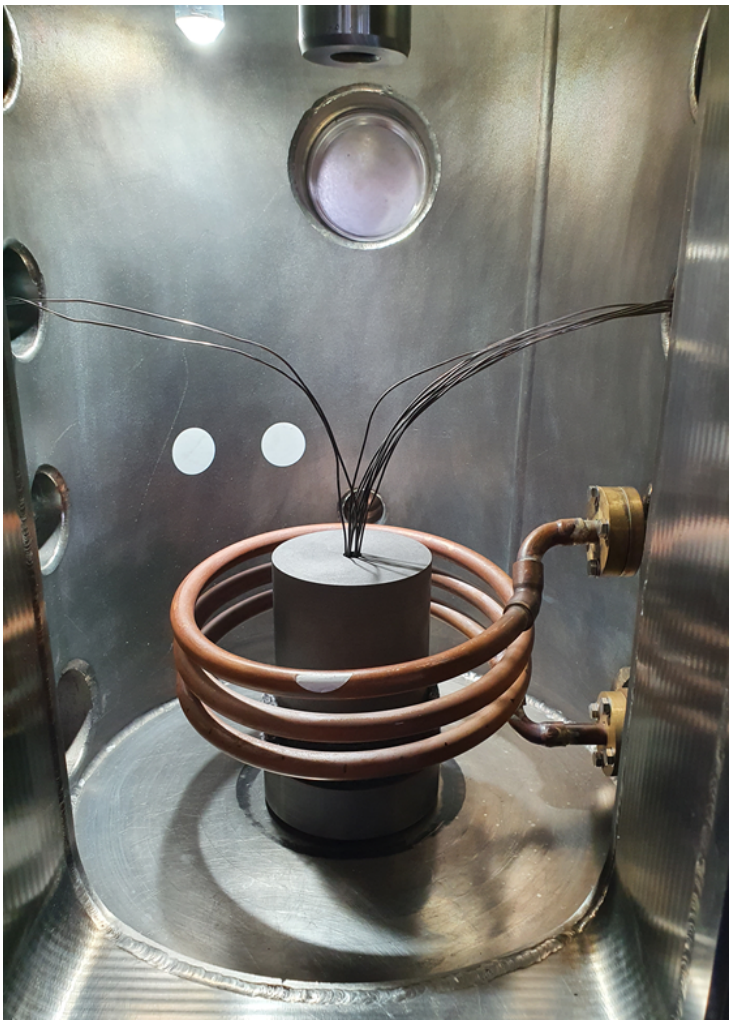


Abbildung 72: Aufbau im Züchtungsofen (Quelle: [Perc])

Neben dem Test der Achsen ging es hauptsächlich um einen Heiztest mit VIFCON. Bei Grundlegende Ablauf des Experimentes war folgender:

1. Zunächst wurde die Temperatur stufenweise bis 800°C erhöht. Also in einer Rampe. (n2)

2. Die Temperatur wurde danach etwas gehalten. (n3)

3. Als letztes wurde die Temperatur Linear (Rampe) runter gesetzt und somit das Abkühlen gestartet. (n4)

Diese Punkte können in dem genutzten Rezept gesehen werden. Zu dem Zeitpunkt benötigte es noch einen Sprung zu Beginn eines Rezeptes. Mit der Optimierung wurde dies so geändert, dass eine Rampe beim aktuellen Istwert starten kann. Die Funktion der Rampe r wurde kurz vor dem Experiment noch hinzugefügt.

```
1 n1: 1     ; 100 ; s
2 n2: 600   ; 800 ; r ; 5
3 n3: 600   ; 800 ; s
4 n4: 43200 ; 20  ; r ; 5
```

6.2.2 Auswertung

In den folgenden Kapiteln, sollen die Auffälligkeiten des Experimentes ausgewertet werden. Neben dieser Auswertung sollen auch die Optimierungen und Verbesserungen erwähnt werden. Das Kapitel 5 zeigt dabei das Endprodukt von VIFCON, welches durch die Experimente immer weiter entwickelt werden konnte.

6.2.2.1 Optimierungen Allgemein

Neben der GUI und der Geräte, gibt es in VIFCON auch Dateien wie die Config- und Log-Datei. Die Config-Datei wird im Anhang B näher erläutert. Kurz gefasst handelt es sich bei dieser Datei um viele verschachtelte Dictionaries. Um ein Gerät nicht zu nutzen, musste dieses auskommentiert werden. Dieser Umstand kann zu Fehlern führen. Aus dem Grund hat jedes Gerät nun einen Schlüssel „Skip“, der eine boolesche Variable als Wert besitzt.

Die andere Datei ist die Log-Datei, die näher im Anhang C erläutert wird. Aus dem Experiment ging hervor, dass Punkte wie Rezept und Änderung der Config-Datei in die Dateien aufgenommen werden. Auch die dritte Datei (Ablauf-Datei, Anhang D) wird damit versehen. Neben diesem Umstand werden nun auch bestimmt Logging Nachrichten in die Konsole geschrieben. Dieser Umstand wird in Anhang C näher erläutert.

6.2.2.2 Optimierungen an Plot und GUI

Wie auch schon bei Experiment 1 (Kapitel 6.1) gibt es auch hier Verbesserungen für Plot und GUI. Wie erwähnt wurden bestimmte Features dazu in Kapitel 5.9 erläutert. Mit Experiment 2 wurden sich die folgenden Punkte angesehen. Diese waren:

1. die Platzierung der Legende (Erstellung der Option: Side - Legende an den Seiten der Achse, als Extra Widget mit Checkboxen),
2. die Erstellung einer Plot-Cursor Anzeige,
3. die Optimierung des Plot-Gitters und der Widget-Platzierung und,
4. die Option von Sprachen.

Die Abbildungen 73 (vorher) und 74 (nachher) zeigen die GUI vor dem Experiment und nach der Verbesserung. Bei der Widget-Platzierung war das Problem, dass die *QLabel*, besonders die Fehlermeldungen und Werte, die Knöpfe teilweise verschoben haben. Bei einer guten Steuerung sollen die einzelnen Widgets am besten kompakt und an Ort und Stelle bleiben. Die neue Legenden Option ist in der optimierten Abbildung zu sehen.

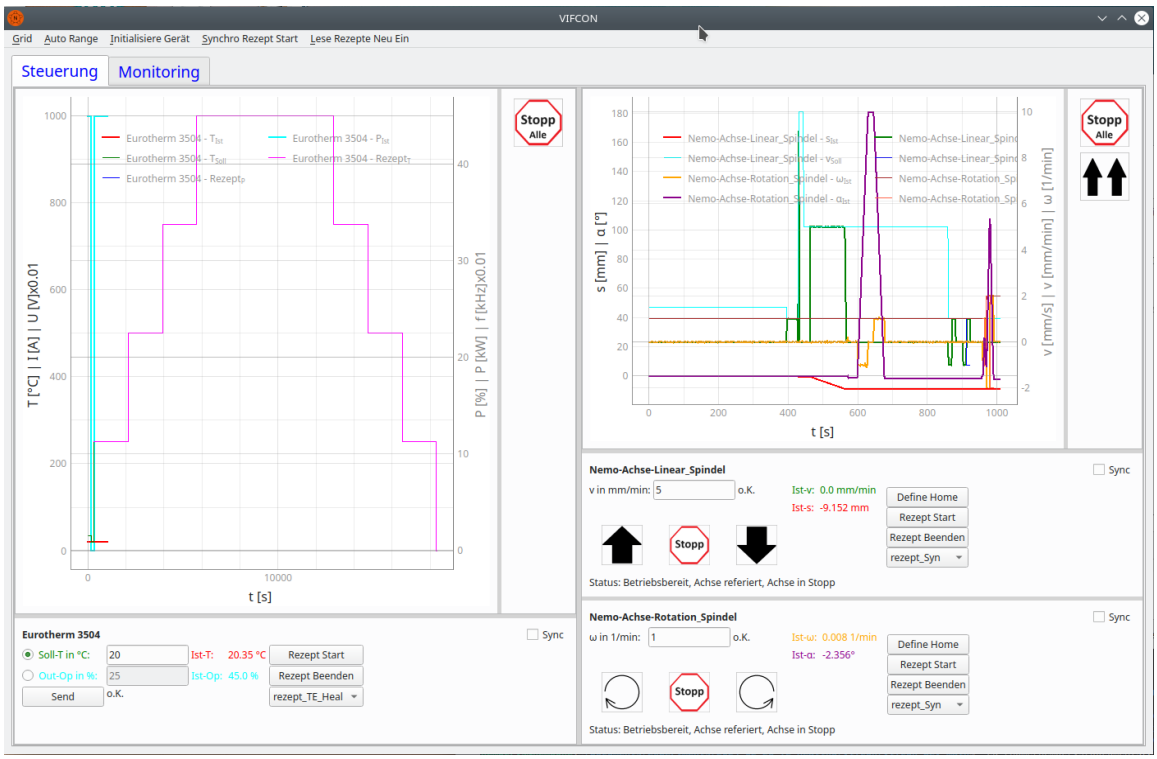


Abbildung 73: GUI Screenshot vor Optimierung (Quelle: [Perc])

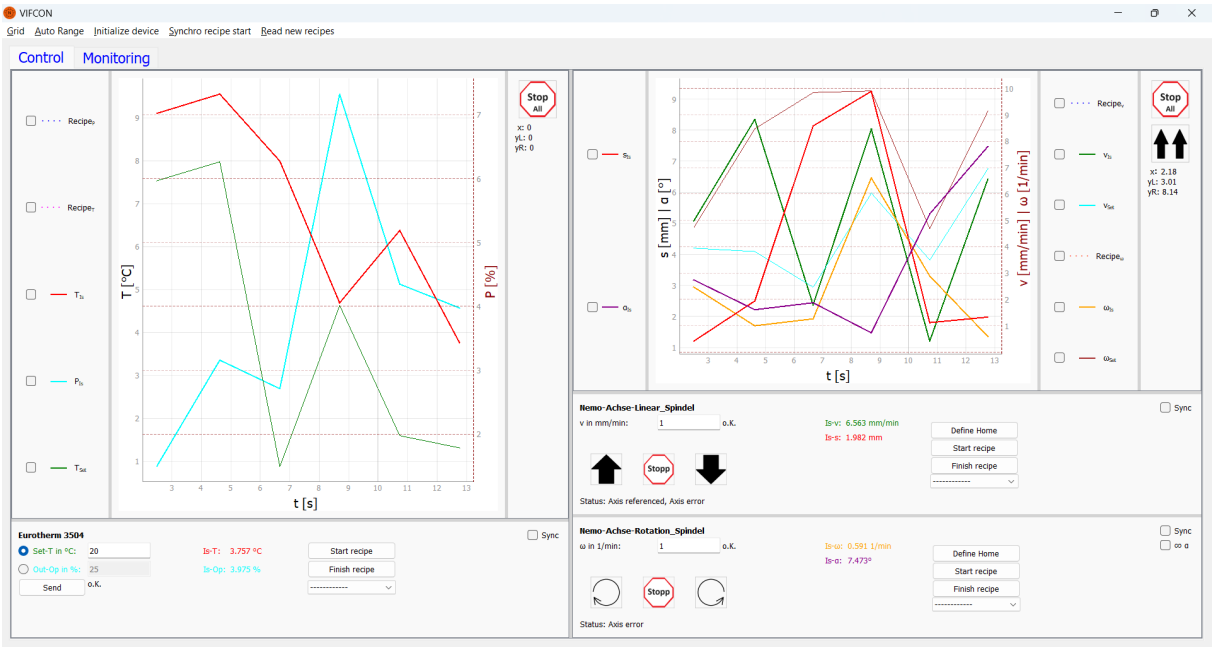


Abbildung 74: GUI Screenshot nach Optimierung (Quelle: eigene Darstellung)

Um die Steuerung z.B. auch für den internationalen Gebrauch zu nutzen, wurde der Config die Möglichkeit der Sprach-Auswahl hinzugefügt. Möglich sind derzeitig Englisch und Deutsch. Die Englische Version wird auch in Abbildung 74 gezeigt. Neben diesen Umstand wurden auch die Ablauf-Datei und die Log-Datei an die Sprache angepasst.

6.2.2.3 Anpassung Rezepte

Auch die Rezepte wurden anhand des Experimentes weiterentwickelt. Am Anfang von Kapitel 6.2 wurde der aktuelle Stand von VIFCON beschrieben. Im letzten Punkt wurde erwähnt, dass VIFCON nun eine eigene Rampen-Erzeugung hat. Vor dem Experiment gab es einen Probelauf mit VIFCON. In diesem ist die Notwendigkeit der Rampen aufgefallen. Im Probelauf mussten die Rampen noch selbst durch viele Sprünge erstellt werden. Im Experiment erstellt VIFCON eine Rampe nun durch ein Rezept-Segment. Neben dieser Anpassung wurden folgende weitere Verbesserungen eingebaut. Alle Verbesserungen können in Kapitel 5.5 gefunden und nachgelesen werden. Diese Verbesserungen sind:

1. eine Verriegelung des Rezept-Einlese Funktion,
2. die Anzeige des Rezeptes im Plot,
3. eine Möglichkeit der Rampen und,
4. eine Möglichkeit das Rezept in einer separaten Datei zu legen.

6.2.2.4 Anpassungen für Eurotherm

Der Eurotherm-Regler (Kapitel 3.1) wird verwendet um den Heizer der Nemo-1-Anlage über einen Generator zu betreiben. Durch eine Temperaturmessung wird ein PID-Regler betrieben, welcher dann die Leistung am Ausgang regelt. Die Abbildung 75 zeigt dabei drei Ausschnitte aus der VIFCON-GUI. In allen drei Plots, der Generatorseite, sind extreme Schwankungen zu sehen. In dem Experiment war eine maximale Ausgangsleistung von 45 % (HO) eingestellt, welche immer wieder erreicht wird. Die Ausgangsleistung schwankt somit zwischen Null und Maximum. Die Abbildung 78 in Kapitel 6.2.3 zeigt das Ergebnis der letzten Messung des Experimentes. Auch dort sind Schwankungen zu erkennen. Im Unterschied zu Abbildung 75 wurden die PID-Parameter dort zum Test angepasst. Es wurden mehrere Versuche durchgeführt. Dies kann auch daran gesehen werden, dass in Abbildung 75 die Temperatur der drei Abbildungen immer eine andere ist. Links wird bis 400, in der Mitte bis 200 und Rechts bis 800°C aufgeheizt werden. Somit ist das Rechte ein Teil des in Abbildung 78 gezeigten Plots.

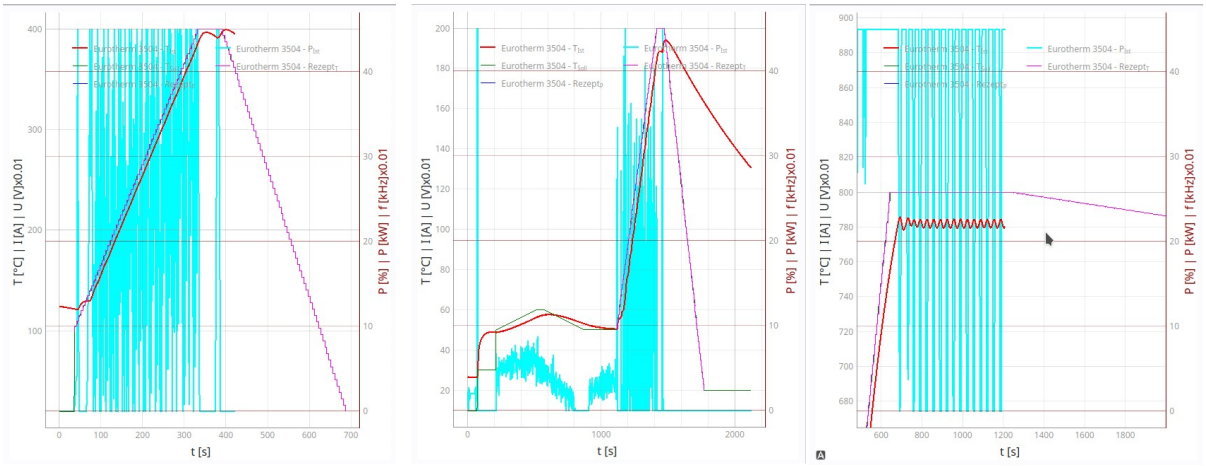


Abbildung 75: drei Screenshots aus dem Experiment - PID Problem (Quelle: [Perc])

Der Grund für diese Schwankungen kann verschiedene Gründe haben. In der eigenen Bachelorarbeit [Fun22] wurde bei dem älteren Eurotherm Modell auch schon so ein Verhalten festgestellt. Aus dem Grund wurde damals ein Auto Tune bzw. eine Selbstoptimierung am Eurotherm durchgeführt. Dies kann in der Bachelorarbeit auf den Seiten 28 bis 32 nachgelesen werden. In dem Fall wurde das nicht getan. Somit kann einer der Gründe der uneingestellte PID-Regler sein. Nach der Log-Datei waren die Parameter folgende: P: 5.8, I: 595 s und D: 99 s. Das die PID-Parameter einer der Gründe für die Schwankung ist, kann in Abbildung 78 gesehen werden. In der Abkühlrampe wurden durch den Ausführenden des Experimentes die Parameter geändert. Am Ende wurden die Parameter P = 4, I = 20 s und D = 4 s eingestellt, bei einem Leistungsmaximum von 35 %. Wie gesehen werden, ist das extreme Schwanken nun deutlich reduziert. Oben wurde bereits von der Selbstoptimierung erzählt. Diese Verfahren für die Selbstoptimierung, wie z.B. das von Ziegler-Nichols [Reg], geben nur die Startwerte für den PID-Regler zurück. Mit diesen Werten muss nun weiter probiert werden, bis die Regelung wie gewünscht läuft.

Eine weitere Fehlerquelle könnte der D-Anteil sein, da dieser empfindlich auf Schwankungen reagiert. Da die Temperatur eine recht schwankende Größe ist, könnte dieser hier auch für die Schwankung sorgen. In der Bachelorarbeit [Fun22] wurde sich damals entschieden nur ein PI-Regler zu nutzen.

Ein weitere Grund kann die Rampe selbst sein. Die Rezept-Rampe ist so programmiert worden, dass diese aus vielen Sollwertsprüngen besteht. Alles über die Rezepte kann in Kapitel 5.5 nachgelesen werden. Bei der Regelung ist der Sprung die höchste Belastung für das System. Aus dem Grund werden auch Sprünge bei der Optimierung von Reglern genutzt. Wenn der Regler nämlich die höchste Belastung aushält und sein Zielwert erreicht kann der Regler als gut angesehen werden. Aus Abbildung 75 können nun verschiedene Dinge abgelesen werden. Bei dem Linken Plot sind während der gesamten Rampe immer wieder die Sprünge auf das Maximum. Weiterhin befindet sich der Istwert immer unter dem Sollwert. Bei jedem Sprung wird der Regler belastet, wodurch dieser nun Maximale Leistung ausgibt, um den Istwert anzupassen. In der Mittleren Darstellung kann zu Beginn eine etwas kleiner schwankende Leistungskurve gesehen werden. Nachdem überschreiten des Sollwertes durch den Istwert, fällt die Leistung langsam auf Null ab. Mit dem erreichen der Rampe beginnt das, was im linken Bild zu sehen ist. Ein ähnliches ist bei dem Sprung zu Beginn des mittleren Plots zu sehen. Im Rechten Bild ist noch etwas zu sehen. Die Temperatur von 800°C konnte nicht mit einer Ausgangsleistung von 45 % erreicht werden. In dem Fall probiert der Regler die Temperatur zu erreichen. Immer wenn sie steigt, sinkt die Leistung schnell auf Null und wenn sie sinkt steigt die Leistung schnell zum Maximum.

Als Schlussfolgerung aus dem Problem des Schwankenden Leistungsausgangs ist folgender. Zum einen muss der PID-Regler in Bezug auf seine Regelparameter verbessert werden. Dies muss dann auch an der Anlage direkt geschehen. Zum anderen sollten die Sollwertsprünge vermieden werden. Für diese Zwecke liefert Eurotherm eine eigene Rampen-Funktion, die im Folgenden in VIFCON implementiert wurde. Wie diese Rampen funktionieren, kann in Kapitel 3.1 nachgelesen werden. Um das kurz zusammenzufassen, kann zu diesen Rampen gesagt werden dass diese einen Zielwert mit einer angegebenen Steigung (°C/s) anfahren. Dabei wird der Sollwert kontinuierlich Linear angehoben bis das Ziel erreicht ist.

6.2.2.5 Anpassungen für Nemo-1 Achsen

Auch bei der Nemo-1-Anlage gab es ein paar Probleme mit dem Antrieb. In Abbildung 76 soll dies dargestellt werden. Die Rezept-Kurve ist gespiegelt zu der Geschwindigkeitskurve. Das Problem an der Stelle ist, dass dies bei Tiegel und Spindel unterschiedlich ist. Beim Tiegel ist dies nämlich nicht so. Aus dem Grund wurden die Spindel-Istgeschwindigkeiten gespiegelt. Somit passen auch die Geschwindigkeitswerte, zu der positiven Position.

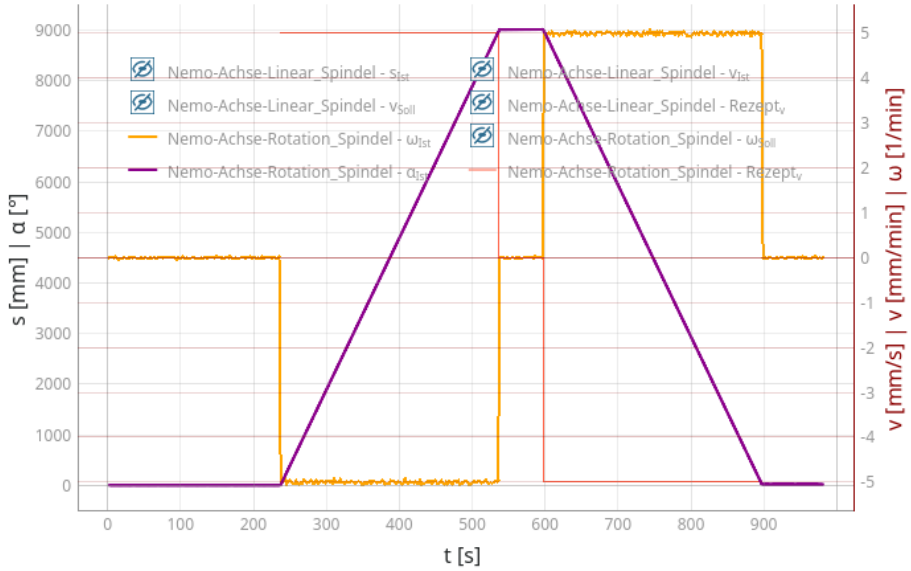


Abbildung 76: Problem mit der Nemo-1 Geschwindigkeitsdarstellung (Quelle: [Perc])

Für die Rotationen wurde sich lediglich für die Erweiterung eines endlosen Modus entschieden. Somit soll die Achse nicht durch die Limits gebremst werden und solange laufen bis Stopp gedrückt wird.

6.2.3 Schlussfolgerungen

Wie in Kapitel 6.2.1 erwähnt, wird nun das Ergebnis des Experimentes gezeigt. Die Abbildungen 77 und 78 zeigen dabei die Messkurven der letzten Messreihe. Abbildung 77 zeigt dabei die von Multilog aufgenommenen Temperaturkurven der Thermoelemente, während Abbildung 78 die Messkurven vom Eurotherm an VIFCON zeigt. In Abbildung 75 wurden noch andere Messreihen gezeigt.

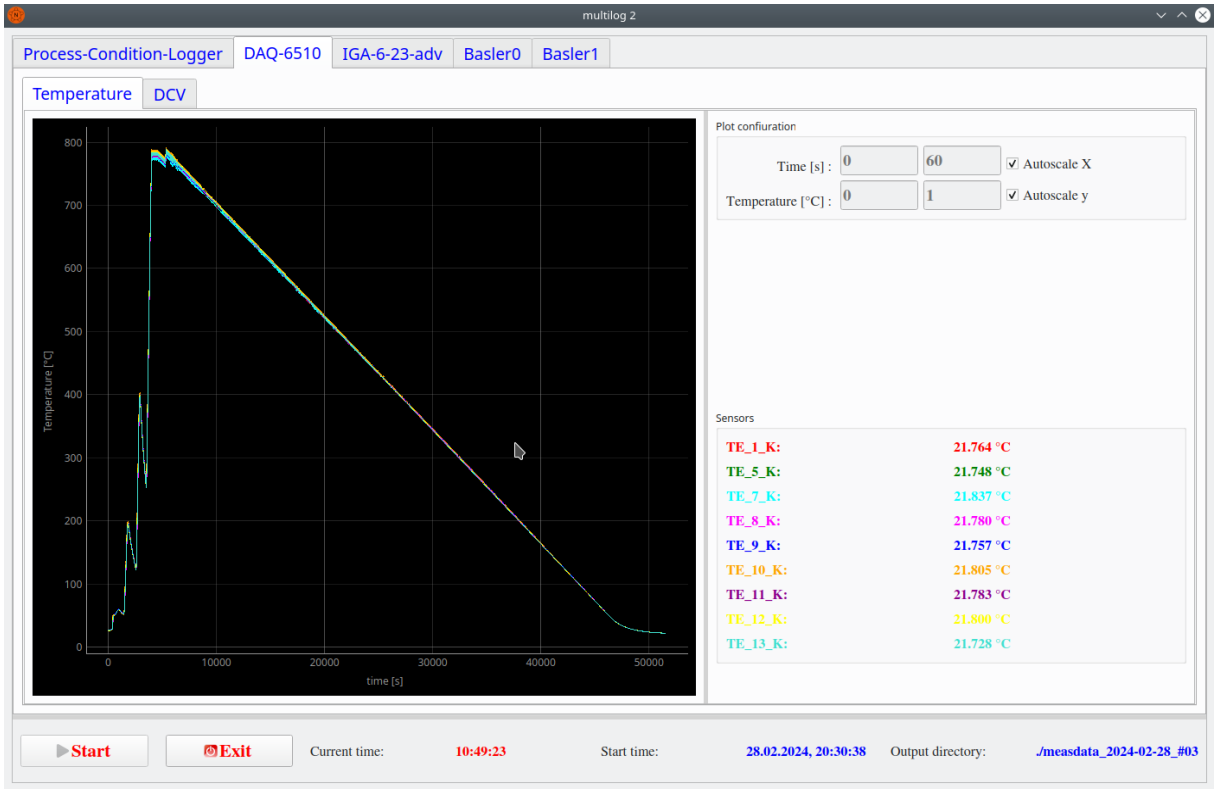


Abbildung 77: Temperaturmessung der Thermoelemente über das Keithley Multimeter DAQ-6510 (Quelle: [Perc])

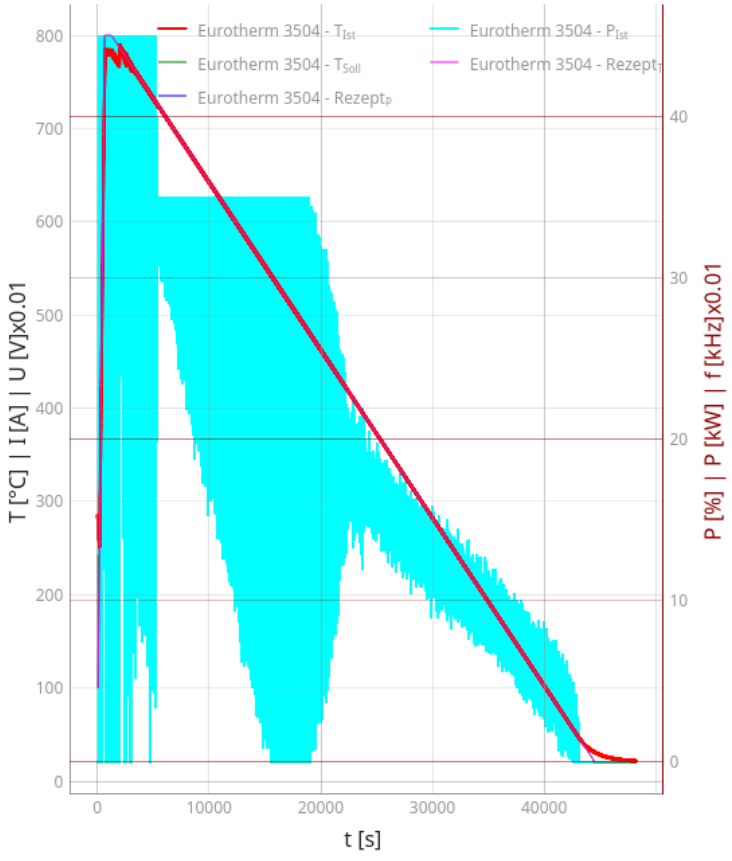


Abbildung 78: VIFCON Plot - Tab Steuerung - Generatorseite (Quelle: [Perc])

Zunächst muss gesagt werden das VIFCON die Aufgabe des Heiztests mit den Thermoelemente erfüllt hat. Über die GUI konnten die verschiedenen Geräte angesteuert werden. Die Heilung der Thermoelemente konnte erfolgen.

Neben den kleinen Problemen mit den Achsen, muss besonders der Eurotherm-Regler noch erweitert werden. Das Problem des schwankenden Leistungsausganges und des mit diesem verbunden PID-Reglers wurde in Kapitel 6.2.2.4 angesprochen. Als Lösung muss der PID-Regler optimiert werden, als auch von der eigenen Rampe weggegangen werden. Da Eurotherm eine eigene Rampe zu Verfügung stellt, ist davon auszugehen, das diese bessere Ergebnisse liefert und keine zu starken Sollwertsprünge liefert. Die Auswirkungen der neuen Rampe wird dann im Experiment aus Kapitel 6.3 relevant, dort eine Kristallzüchtung durchgeführt wird.

6.3 Experiment: Kristallzüchtung an Nemo-1

Das dritte Experiment bildet das wesentlichste. Bei den Experimenten aus Kapitel 6.1 und 6.2 wurde VIFCON auf die Probe gestellt. Mit diesen wurden Probleme gefunden und Geräte ausgetestet. Das im folgende beschreiben Experiment wird eine Kristallzüchtung sein. Auch hier wird die Nemo-1-Anlage verwendet. Somit bildet das Experiment den wesentlichen Test von VIFCON, da die Züchtung eines Kristalls das Ziel von Kirstallzüchtungsanlagen ist und VIFCON eine solche steuern soll.

Das Experiment wurde von Dr. Kaspars Dadzis [Perc] und Vincent Funke durchgeführt. Der Stand von VIFCON beinhaltet die Optimierungen aus Kapitel 6.2. Es wurden keine weiteren Geräte hinzugefügt. Die Rezepte, wie sie in der Config stehen, werden in Anhang G.2 gezeigt.

6.3.1 Aufbau und Ablauf

In dem Kapitel sollen zwei Dinge geklärt werden. Diese sind der Aufbau (Kapitel 6.3.1.1) und die Vorbereitung und Durchführung (Kapitel 6.3.1.2) des Experimentes. Wie erwähnt ist das Thema des Experimentes eine vollständige Kristallzüchtung mit dem CZ-Verfahren.

6.3.1.1 Aufbau des Experimentes

Wie bei Experiment 2 (Kapitel 6.2) wird neben VIFCON auch Multilog verwendet. Somit werden die folgenden Geräte für die beiden Programme verwendet.

Mit VIFCON werden die Geräte:

1. Eurotherm (Abbildung 17) mit angeschlossenen Pt100-6 (Widerstandsthermometer) im Tiegel,
2. Nemo-1 Antrieb Hub Spindel (Abbildung 38a),
3. Nemo-1 Antrieb Rotation Spindel (Abbildung 38b),
4. Nemo-1 Antrieb Rotation Tiegel (Abbildung 39b) und,
5. Nemo-1 Monitoring Gase (Abbildung 40)

verwendet. Bei Multilog wurden die Geräte:

1. Thermoelemente des Typs 1J (Tiegel) und 3K (Schmelze),
2. Pyrometer (IGA 6/23-Laser/ IGA-6-23-adv),
3. 2x Kameras von Basler (Basler0 (acA2440-20gc), Basler1 (acA2500-20gc)),
4. Wärmebildkamera (Optris-IP-640) und,
5. DAQ 6510 Multimeter von Keithley

verwendet. Die folgenden beiden Abbildungen 79 und 80 zeigen eine graphische Darstellung des Experimentes. In Abbildung 79 werden dabei die Verbindungen zwischen den Geräten dargestellt und in Abbildung 80 der Tiegel und der Rezipient. Z.B. kann in Abbildung 79 gesehen werden, dass die Thermoelemente über eine Temperatur Referenz Unit zum Multimeter geführt werden. Weiterhin wird die Strommessung durch eine Rogowskispule und das Multimeter durchgeführt. In der Darstellung kann auch das Eurotherm gefunden werden. Dieses wurde in eine Box integriert, sodass der Ausgang des Eurotherm umgeschaltet und ein Hardware Limit gesetzt werden kann.

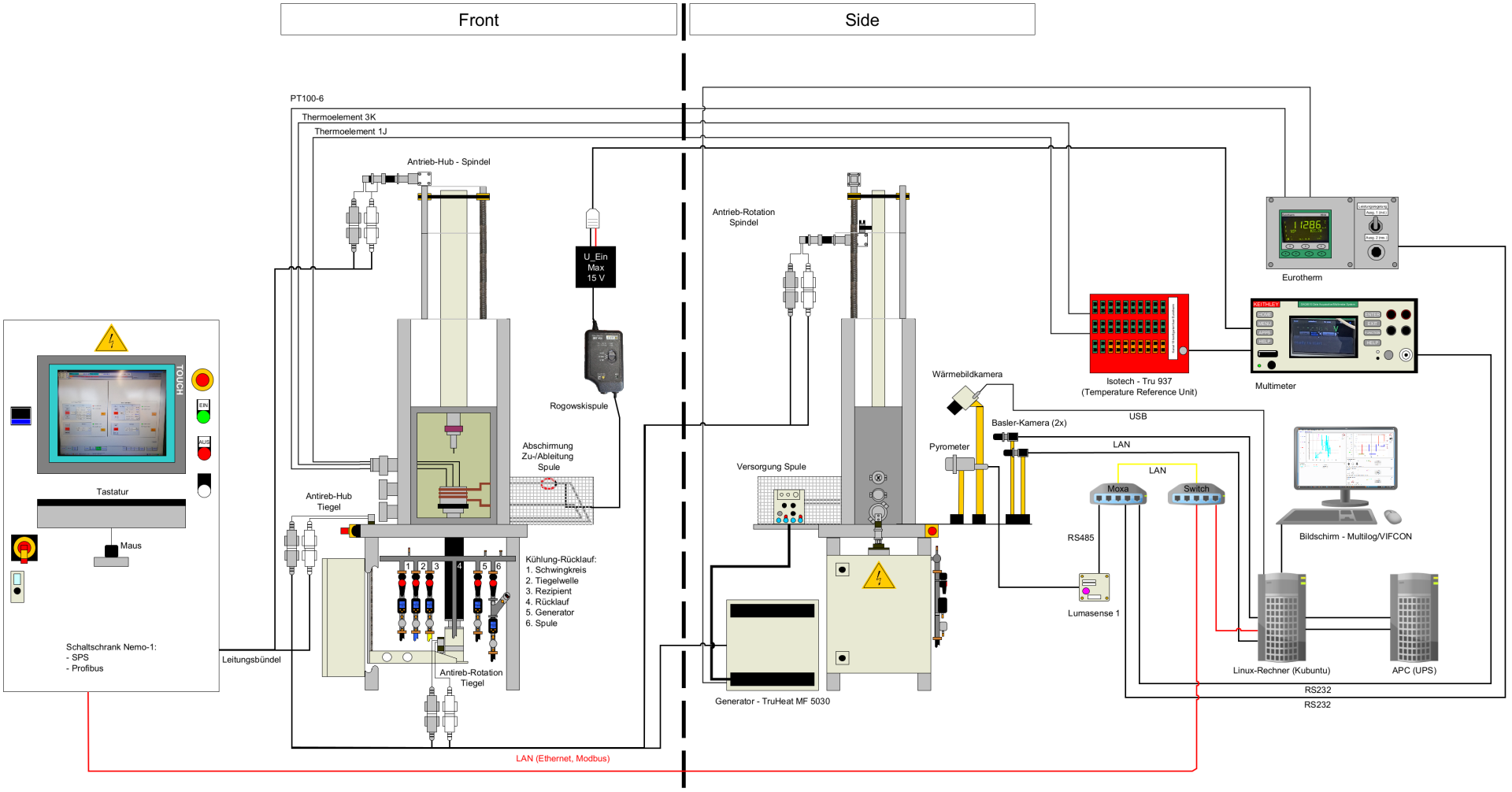


Abbildung 79: Graphische Darstellung des Experimentes (Quelle: eigene Darstellung)

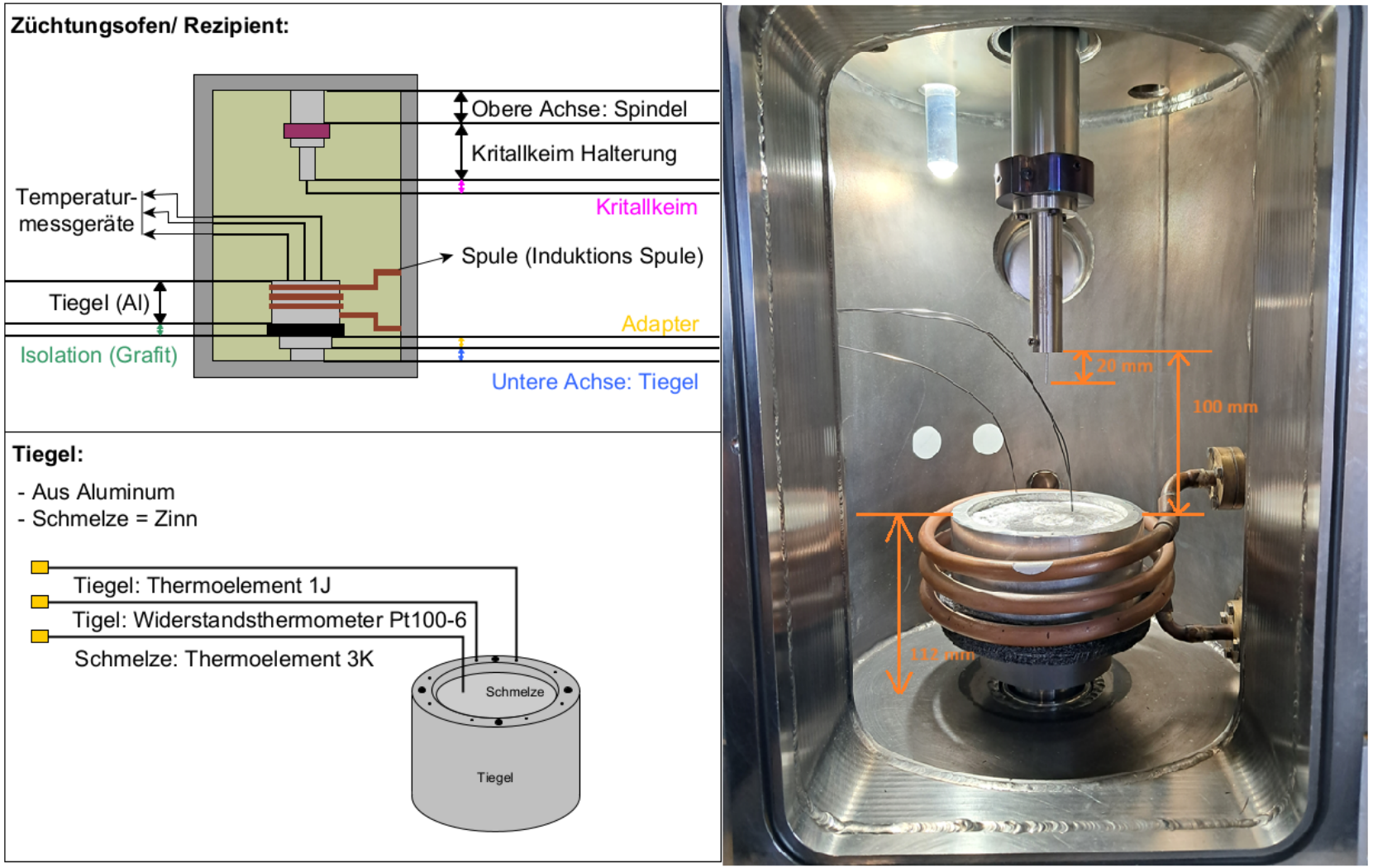


Abbildung 80: Graphische Darstellung des Tiegels und des Züchtungsofens (Quelle: eigene Darstellung)

Nach dem der Aufbau der Anlage bekannt ist, sollen nun noch weitere Daten zu dem Experiment genannt werden:

1. Max. Rotationsrate von Kristall = 10 $\frac{1}{\mathsf{min}}$
2. Max. Hubgeschwindigkeit Spindel = 5 $\frac{\mathsf{mm}}{\mathsf{min}}$, Limit Hubgeschwindigkeit Spindel = 7 $\frac{\mathsf{mm}}{\mathsf{min}}$
3. Gesamtzeit des Experimentes ≈ 3 h
4. Kristalllänge = 120 mm
5. Länge des Impfkristalls = 30 mm, Sichtbare Länge des Impfkristalls = 20 mm
6. Emissivität = 0,95 % (Pyrometer, Wärmebildkamera)
7. PID-Parameter Eurotherm:
   - P-Glied = 4
   - I-Glied = 20 s
   - D-Glied = 4 s
8. Maximale Eurotherm Ausgangsleistung (HO) = 25 %
9. Schmelztemperatur von Zinn = 231,9°C

In Abbildung 79 wird auch wieder ein TruHeat-Generator gezeigt. Diesmal ist dieser aber ein TruHeat HF 5030. In der Skizze wurden die einzelnen Anschlüsse nicht verbunden, sondern nur die Rückseite dargestellt. Der Außenkreis ist mit dem Generator und der Kühlung verbunden.

6.3.1.2 Vorbereitung des Experimentes

Neben der Vorbereitung des Aufbaus des Experimentes (Kapitel 6.3.1.1) mussten auch andere Punkte beachtet werden. Diese Punkte sind:

1. die Planung der Rezepte bzw. den Ablauf des Experimentes,
2. die Beachtung eines Vorfaktors bei der Nemo-1 Achse für den Spindel Hub und,
3. eine Erweiterung der Rezepte für Eurotherm.

Das erste was erwähnt werden soll ist der Vorfaktor. Bei der Vorbereitung des Experimentes und der Geschwindigkeitsrezepte des Experimentes, wurde festgestellt das der Antrieb nicht an die Position gefahren ist, an den sie sollte. Zum Beispiel sollte die Achse um 90 mm bewegt werden, bewegte sich aber nur um 70 mm. VIFCON berechnet intern den Weg selbst aus, da Nemo-1 diesen Wert nicht richtig ausgegeben kann! In Abbildung 81 wird das Messgerät zur Bestimmung des Vorfaktors gezeigt. In dem Messgerät befindet sich ein bewegbarer Stift. Das eine Ende dieses Stiftes ist oben an der Keim-Halterung platziert. Wenn die Achse nun hoch fährt, rutscht der Stift nach oben und der Wert auf der Anzeige wird kleiner. Durch die Differenz des Start- und Endwertes kann der tatsächliche Fahrweg bestimmt werden.

Abbildung 81: Bestimmung des Vorfaktors (Quelle: eigene Darstellung)

Die Tabelle 24 zeigt die 5 Messungen und den berechneten Vorfaktor (Formel 6.1) für jede einzelne Messung. Um den Vorfaktor zu bestimmen wurde dann noch der Arithmetische Mittelwert aller Vorfaktoren genommen. Anhand dieses Versuches kann auch die Berechnung der Position durch VIFCON als erfolgreich angesehen werden, da der zufahrende Weg kaum von der VIFCON Anzeige abweicht.

Tabelle 24: Fahrweg-Bestimmung für den Vorfaktor (Quelle: eigene Darstellung)

Messgerät Start $[mm]$	**Messgerät Ende** $[mm]$	**Messgerät Diff.** $[mm]$	**VIFCON Anzeige** $[mm]$	**Diff. Messgerät zu VIFCON** $[mm]$	**v** $[mm/min]$	**t Rezept** $[s]$	**Fahrweg berechnet** $[mm]$	**Vorfaktor**
13,279	5,313	7,966	9,996	2,03	2	300	10	0,796918768
13,01	9,034	3,976	4,957	0,981	5	60	5	0,802098043
13,051	9,827	3,224	4,001	0,777	1	240	4	0,80579855
13,066	8,272	4,794	5,95	1,156	3	120	6	0,805714286
13,207	5,161	8,046	9,99	1,944	2	300	10	0,805405405

$$Vorfaktor = \frac{Messgerät\ Diff.}{VIFCON\ Anzeige} \tag{6.1}$$

$$\text{z.B. } Vorfaktor = \frac{7,966\ \text{mm}}{9,996\ \text{mm}} = 0,796918768 \tag{6.2}$$

Nach der Tabelle 24 beträgt der mittlere Vorfaktor 0,80318701. Dieser Wert wird nun in VIFCON mit dem Geschwindigkeitswerten verrechnet. Wenn der Wert gesendet wird, wird dieser durch den Vorfaktor geteilt, damit die Geschwindigkeit größer wird, da die Achse nicht schnell genug fährt. Beim Lesen des Wertes wird dieser mit dem Vorfaktor multipliziert um die eigentliche Geschwindigkeit anzuzeigen. Da das Geschwindigkeitslimit 7 mm/min ist und im Experiment mit 5 mm/min gefahren werden soll, wird an der Anlage durch den Vorfaktor eine Geschwindigkeit von ca. 6,25 mm/min anliegen, was noch im Rahmen ist.

Da die endgültige Rezept Gestaltung in Kapitel 5.5 erläutert wird, soll hier nur die Neuerung erwähnt werden. Für das Experiment wurden die Eurotherm eigenen Rampen in die Rezepte aufgenommen. Dazu wurde in Kapitel 6.2.2.4 mehr gesagt. Neben dieser Änderung wurde versucht auch ein Leistungsrezept mit in das Temperaturrezept einzubinden.

Das letzte bei der Vorbereitung ist die Planung der Züchtung. In Abbildung 3 wurde der Allgemeine Ablauf der Züchtung gezeigt. Die Abbildung 82 zeigt den Ablauf (die Rezepte) für dieses Experiment. Die Schritte Vorbereitung und Aufräumen werden in der Abbildung nicht dargestellt. Auch das Spülen findet hier nicht statt, da die Tür des Rezipienten offen bleibt und die Atmosphäre somit Sauerstoff ist. Somit zeigt Abbildung 82 die Schritte:

1. Aufheizen,
2. Ankeimen,
3. Züchten,
4. Abziehen und,
5. Abkühlen.

In VIFCON wird für die Umsetzung die Synchrone Rezept Start Funktion verwendet. Somit beinhaltet das Experiment 7 Rezepte bei 4 Geräten in 2 Phasen. In der ersten Phase werden 3 Rezepte (Eurotherm - Temperatur, Nemo-1-Antrieb Hub Spindel - Geschwindigkeit, Nemo-1-Antrieb Rotation Spindel - Winkelgeschwindigkeit) parallel laufen und in der zweiten Phase dann 4 (Eurotherm - Temperatur, Nemo-1-Antrieb Hub Spindel - Geschwindigkeit, Nemo-1-Antrieb Rotation Spindel - Winkelgeschwindigkeit, Nemo-1-Antrieb Rotation Tiegel - Winkelgeschwindigkeit).

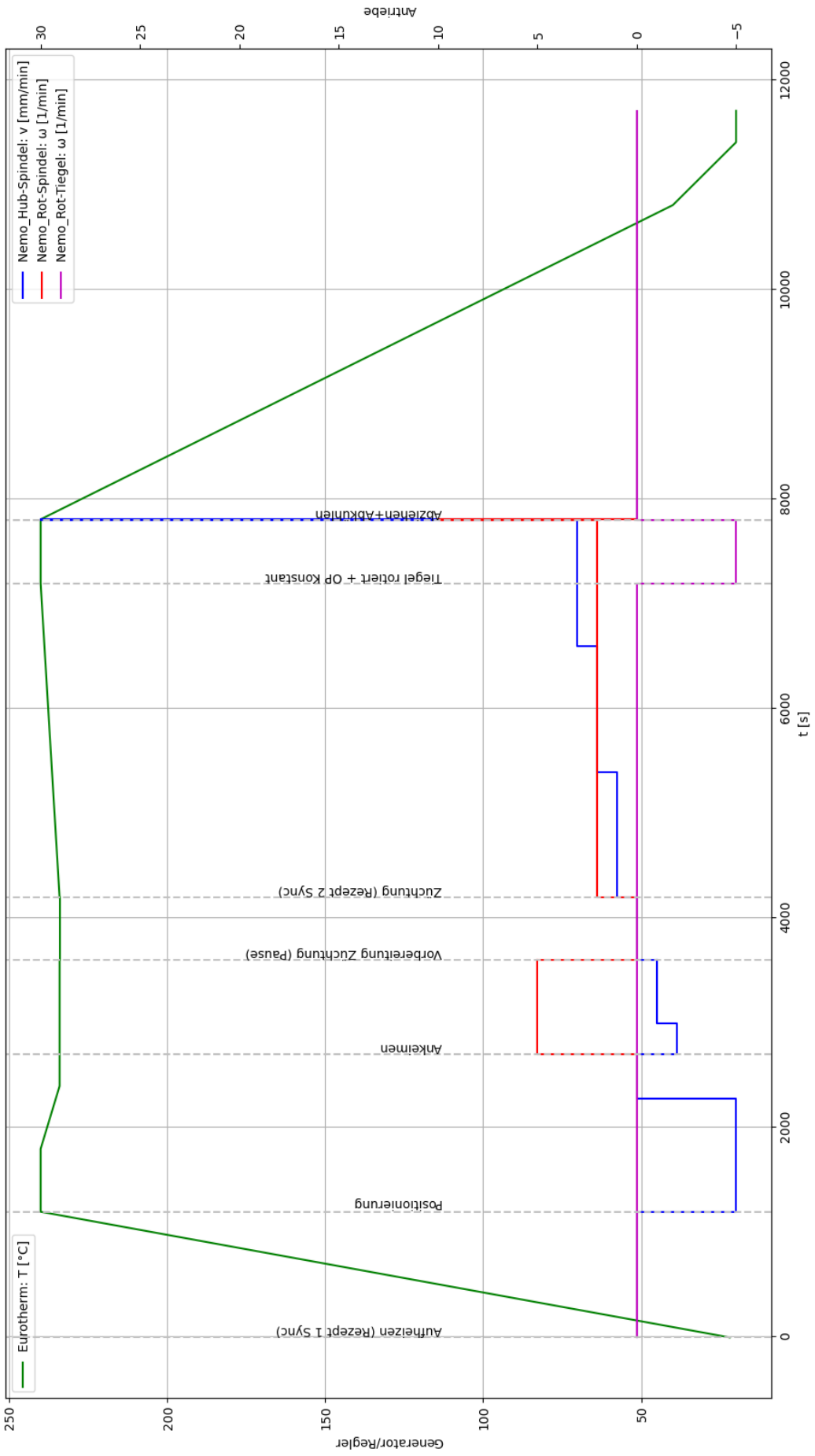


Abbildung 82: Rezept-Planung für Experiment 3 (Quelle: eigene Darstellung)

6.3.2 Auswertung

In dem Kapitel wird das Experiment ausgewertet. In Kapitel 6.3.2.1 werden die Kurven der Messung und auch der geschaffene Kristall gezeigt. Neben diesen Punkt werden auch die durchgeführten Optimierungen und Probleme genannt.

6.3.2.1 Ablauf und Ergebnis des Experimentes

Dieses Kapitel wird das Ergebnis und die Probleme mit dem Experiment beschreiben. Das Experiment wurde in zwei Abschnitte unterteilt. Die Abbildungen 83, 84, 85 und 86 zeigen dabei das Ende der jeweiligen Messreihe. Das Experiment wurde in die Abschnitte „Aufheizen und Ankeimen“ und „Zucht, Abziehen und Abkühlen“ unterteilt. Diese Unterteilung wurde geplant, um die Zucht vorzubereiten und z.B. z.B. Oxid von der Oberfläche zu entfernen. Wenn sich nun die Rezept-Kurven in den gezeigten Abbildungen angesehen werden, dann können die Teile in Abbildung 82 wiedergefunden werden. Im Folgenden werden die genannten Abbildungen erläutert.

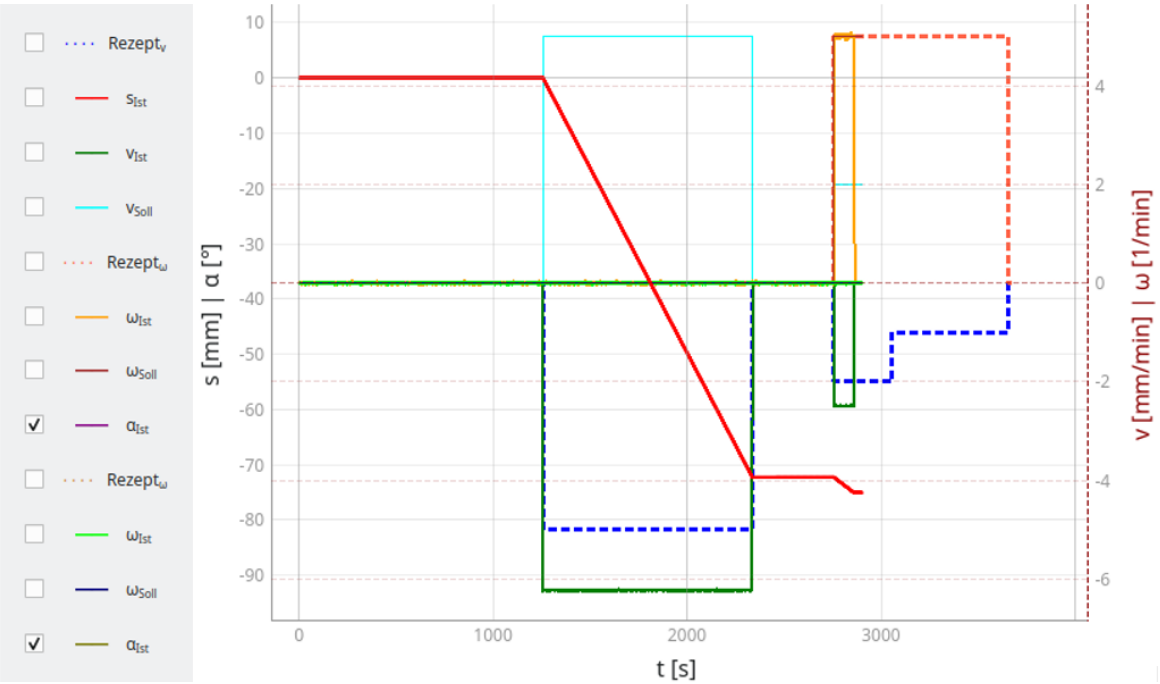


Abbildung 83: Phase 1 (Aufheizen und Ankeimen) - Antriebe (Quelle: eigene Darstellung)

In Abbildung 83 können die Kurven der drei genutzten Antriebe gesehen werden. Der Prozess wurde gestoppt, um die Position des Keimes anzupassen.

Als nächstes muss noch ein weiteres Problem genannt werden, welches aber für die zweite Phase behoben wurde. Bei der Programmierung des Vorfaktors bzw. das einbinden in den Code, wurde dieser beim Spindel-Hub falsch platziert. In der Abbildung 83 kann dies an dem Weg (s_{Ist} - rot) und der Istgeschwindigkeit (v_{Ist} - dunkel grün) gesehen werden. Eigentlich sollte der Vorfaktor auf die Istgeschwindigkeit angewendet werden, doch wurde auf den Weg angewendet. Somit wird der berechnete Weg um den Vorfaktor kleiner dargestellt. Bei der Geschwindigkeit wird der berichtigte Geschwindigkeitswert ausgelesen.

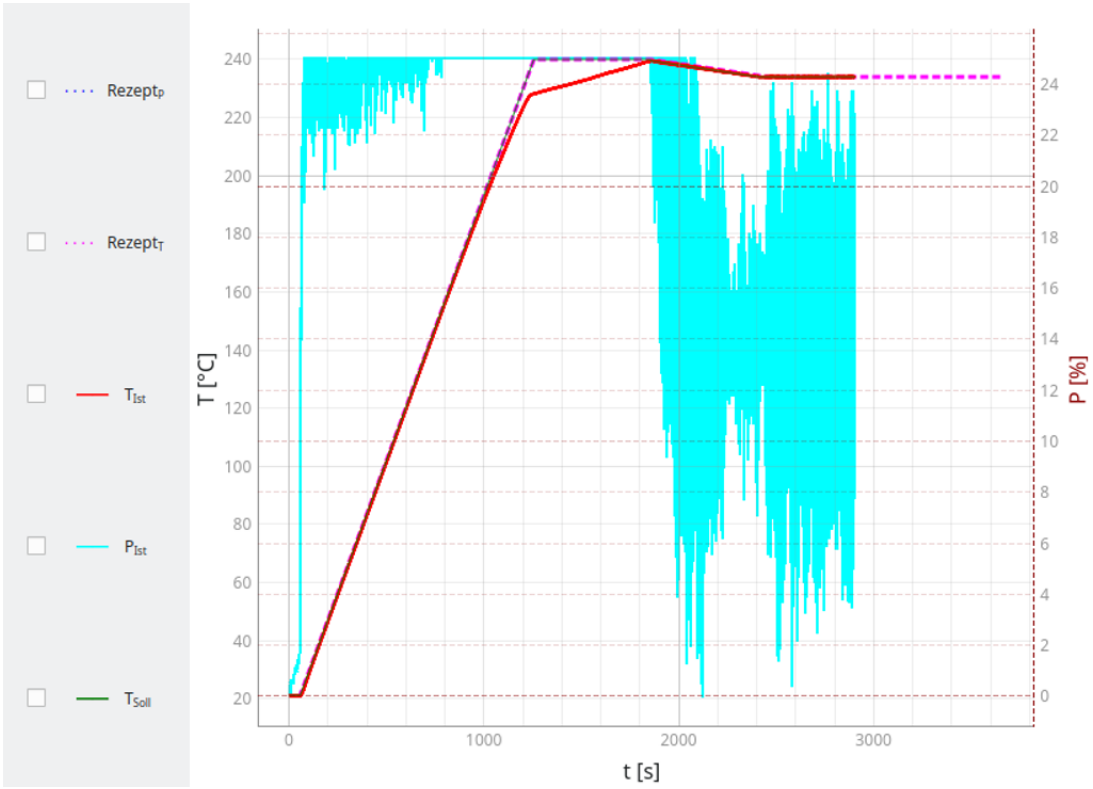


Abbildung 84: Phase 1 (Aufheizen und Ankeimen) - Generator (Quelle: eigene Darstellung)

In Abbildung 84 wird die Generatorseite der Phase 1 gezeigt. Da bei den Generatoren nur der Eurotherm-Regler ausgewählt wurde, werden auch nur dessen Größen angezeigt. In dem Prozess werden zwei Eurotherm-Rampen durchgeführt. Zu diesen Rampen kann in Kapitel 3.1 etwas nachgelesen werden. Der Grund für die Nutzung dieser Rampen wurde in Kapitel 6.2.2.4 (Experiment 2) erwähnt. In dem früheren Experiment wurden eigen gebaute Rampen benutzt, die an Eurotherm immer wieder Sollwertsprünge erzeugen. Der PID-Regler verursachte dort starke Schwankungen der Ausgangsleistung. Auch in dem aktuellen Experiment können relativ starke Schwankungen gesehen werden. Besonders bei der zweiten Rampe, wo die Leistung ca. zwischen 3 und 22 % schwankt. Abgesehen von diesen Schwankungen hat die Nutzung der Eurotherm-Rampen funktionierte gut. Der Knick nach Rampe 1 kommt daher, dass zum Zeitpunkt des Rampenendes, die Isttemperatur noch nicht erreicht wurde. Im Rezept ist nach dieser Rampe ein Plato gekommen, was durch einen Sollwertsprung in VIFCON erzeugt wird. Das Plato war jedoch lang, genug um die zweite Rampe am richtigen Wert zu starten. Bisher sieht es so aus, als könnten die Eurotherm-Rampen nur am aktuellen Istwert starten, was in Abbildung 86 noch besser zu sehen ist.

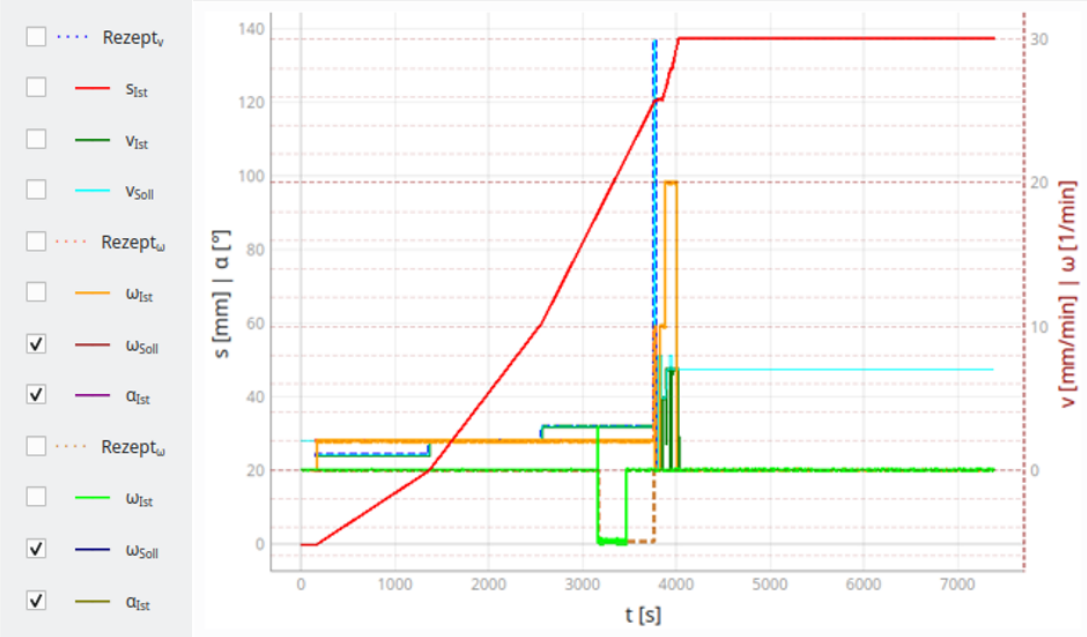


Abbildung 85: Phase 2 (Zucht, Abziehen und Abkühlen) - Antriebe (Quelle: eigene Darstellung)

In Phase 2 werden nun die Punkte Züchtung, Abziehen und Abkühlen erledigt. Das Problem mit dem Vorfaktor wurde hierfür auch behoben. Ein erstes Problem was aus Abbildung 85 erkannt werden kann ist die aktuelle Position. Da VIFCON den Weg selbst berechnet, wird der Weg bei Start des Programms auf Null gesetzt. Dies bildet ein Problem, da z.B. die Grenzen nun nicht mehr so sind, wie sie bedacht worden. Somit bräuchte das Programm eine Möglichkeit den Startweg in der Config und die Limits ohne Define Home (Null setzen) zu setzen.

Während der Züchtung hat das Programm so gearbeitet wie gewollt. Vor dem Abziehen erreichte die Achse eine Position von ca. 120 mm, was der gewünschte Fahrweg war. Die beiden größeren Probleme in dem Prozess waren das letzte drittel der Züchtung und das Abziehen. Dieser Teil ist in Abbildung 86 noch besser zu sehen. In dem Teil wurde sich dazu entschieden, den Tiegel rotieren zu lassen. Wen sich Abbildung 80 angesehen wird, kann gesehen werden, dass im Tiegel Temperaturmessgeräte gesteckt haben. Diese wurden zum Ende des zweiten Teil der Züchtung rausgezogen. Da eins der drei Messgeräte, aber die Istwertmessung des Temperaturwertes für die Regelung der Leistung war, musste das Eurotherm manuell auf einen Wert gestellt werden. Im Programm gab es zwar eine Idee für die Umsetzung, diese hätte aber nicht geklappt, da zum einen das manuelle ziehen nicht zeitgleich mit dem drehen gestartet hätte und zum anderen die Schwankung des Ausgangsleistungswertes Probleme machte. Wenn in den Manuellen Modus von Eurotherm gewechselt wird, wird der aktuelle Wert der Leistung gehalten. Neben diesem Problem war nun auch die Temperaturüberwachung und die dranhängende Regelung nicht mehr richtig gegeben. Aus dem Grund ist die Schmelze abgekühlt und der Durchmesser des Kristalls (Abbildung 88) ist gewachsen. Die Messwerte der beiden Thermoelemente können in Abbildung 87 gesehen werden. In dieser können auch die in Kapitel 6.3.1.1 genannten Multilog-Geräte gefunden werden (also deren Messdaten). Der Schmelzpunkt von Zinn liegt bei ca. 232°C. In Abbildung 87 kann gesehen werden, dass zu dem Zeitpunkt des Wiedereinsteckens, die Temperatur nicht von 240°C startet, sondern bei ca. 230°C liegt. Somit ist die Schmelze abgekühlt, was den Durchmesser größer werden ließ. Die Tiegel-Rotation wurde am Ende manuell angehalten und die Messgeräte vor dem Abziehen und Abkühlen wieder eingesteckt.

Am Ende von Kapitel 6.3.1.1 wurden verschiedene Werte für das Experiment gezeigt. Einer dieser Werte war die Grenze der Hub-Geschwindigkeit, die bei ca. 7 mm/min liegt. Das Abziehen wurde in Kapitel 2.1.1 so beschrieben, dass in dem Punkt eine hohe Geschwindigkeit für kurze Zeit ausgeführt wird. Die geplanten 30 mm/min schafft die Anlage jedoch nicht. Die Höchstgeschwindigkeit hat die Abtrennung nicht geschafft, wodurch der Kristall, nun wieder schmaler, weiter wuchs. Am Ende musste der Kristall manuell abgetrennt und entnommen werden.

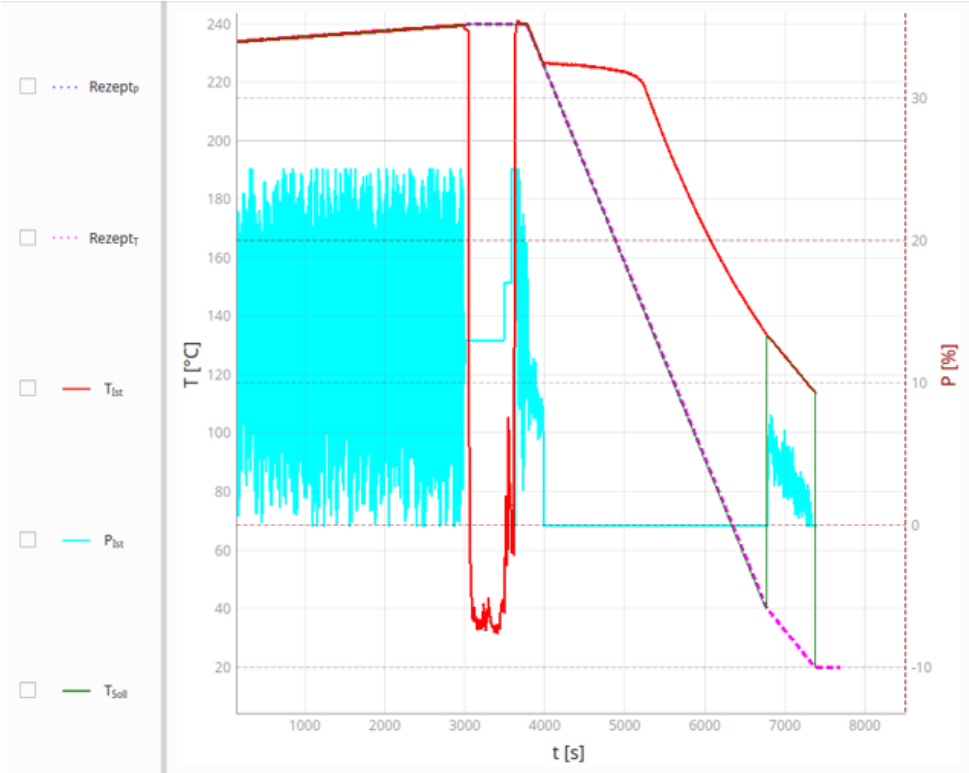


Abbildung 86: Phase 2 (Zucht, Abziehen und Abkühlen) - Generator (Quelle: eigene Darstellung)

Die Abbildung 86 zeigt die Kurven des Eurotherms in Phase 2. Wie erwähnt kann hier das Rausziehen des Pt100 gut gesehen werden, da die Temperatur an der Stelle stark absinkt und nun die Atmosphären Temperatur im Rezipienten misst. Um die Regelung aufrecht zu erhalten, muss an der Stelle eine Manuelle Ausgangsleistung gesendet bzw. eingestellt werden. Aus diesem Punkt wurde klar, dass die Umsetzung der Rezepte noch weiterentwickelt werden muss, da es möglich sein muss Leistung und Temperatur gleichzeitig zu steuern. Dies ist Stand der Technik am IKZ. Hier ist nun auch nochmal deutlich zu erkennen das der PID-Regler keine gute Arbeit leistet. Die Temperatur wird zwar gut gehalten, aber das Schwanken ist zu stark.

Beim Abkühlen wird nun auch eine Eurotherm-Rampe verwendet. Zu Beginn dieser sind Soll- und Istwert noch gut bei einander. Doch dann gibt es ein Plato in der Isttemperatur. Dieses Plato wird durch die Aggregatzustandsänderung von Flüssig zu Fest erzeugt. Bei der Erstarrung des Zinns wird nun Energie frei, die das Plato erzeugt. Durch diesen Umstand kommt es zum Wechsel der zweiten Abkühlungsrampe nun zu dem erwähnten Problem mit dem Eurotherm-Rampen. Da der Istwert der Temperatur am Ende der Rampe nicht den Zielwert erreicht hat, beginnt die zweite Rampe nun an einer falschen Stelle. Anhand der Leistungskurve, kann auch gesehen werden, dass der Generator wieder heißt. Die Abkühlrampe war nicht gut gewählt, da diese zu steil war. Ohne aktive Kühlung konnte die Rampe nicht erreicht werden. Am Ende wird noch ein Sprung ausgeführt, wodurch die Solltemperatur bei 20°C bleibt. Die Abkühlung der Temperatur läuft sonst gut ab.

Abbildung 87: Phase 2 (Zucht, Abziehen und Abkühlen) - Multilog Thermoelemente (Quelle: eigene Darstellung)

Das Endergebnis des Experiments kann in Abbildung 88 gesehen werden. Der Kristall hat am Ende eine Länge von ca. 150 mm. Der Zeitpunkt der Tiegel-Rotation kann an dem Anstieg des Durchmessers des Kristalls ab ca. 90 mm gesehen werden. Nach dem die Regelung wieder funktionierte, wuchs der Kristall mit einem kleineren Durchmesser weiter. Am Ende wird der Kristall wieder dicker, da zu dem Zeitpunkt die Abkühlung begonnen hatte und die Schmelze somit wieder kühler wurde.

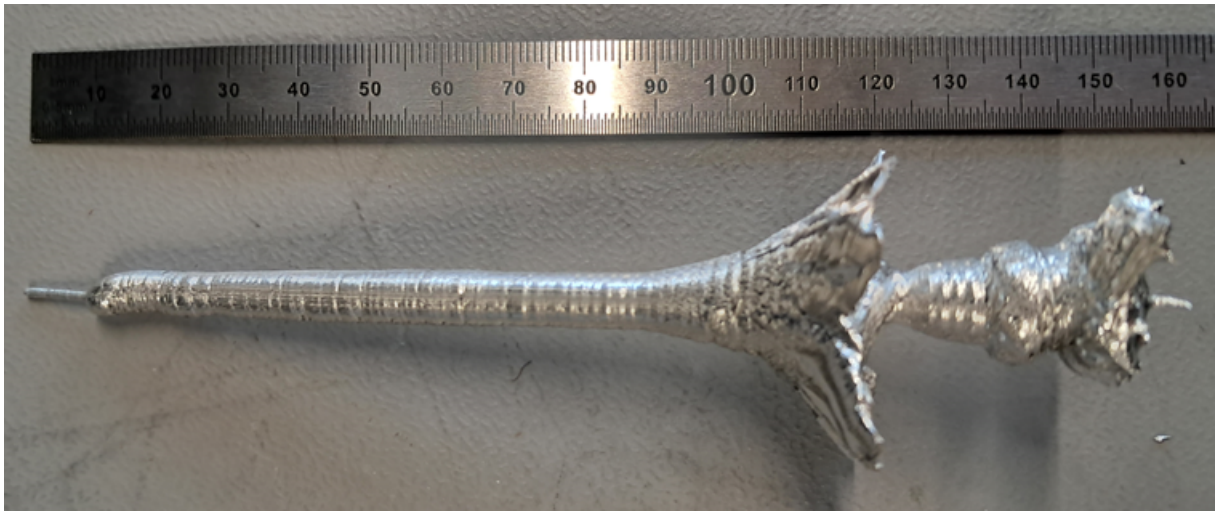


Abbildung 88: gezüchteter Kristall (Quelle: eigene Darstellung)

In Kapitel 6.3.1.2 werden ein paar Ziele dieses Experiment genannt. Diese sollen nun beantwortet werden. Mit VIFCON lässt sich ein Kristall züchten, wie es Abbildung 88 zeigt. Auch die Nutzung des Vorfaktors konnte geprüft werden. Die Achse fuhr an die Stellen, an denen sie liegen sollte. Wenn die Denkfehler mal vernachlässigt, können die Rezepte durch die Synchron-Funktion für die Prozessschritte genutzt werden. Nur die Umsetzung des Eurotherm Gerätes muss noch überdacht werden, da die Leistungskontrolle gebraucht wird, um einen Kristall ohne Durchmesserverbreiterung zu züchten. Somit konnten die Rezepte auch mit allen Geräten genutzt werden. Einige der Probleme wurden bereits erwähnt. Diese sollen nun kurz aufgezählt/zusammengefasst werden. Somit bildeten:

1. die Verbindung von Eurotherm-Temperatur und Eurotherm-Leistung Rezepte,
2. das Umstellen in den Manuellen Modus bei Schwankenden Leiustungswerten,
3. der nicht optimierte PID-Regler und,
4. der Start der Eurotherm-Rampen vom aktuellen Istwert

Probleme. In den folgenden Kapiteln wird nun noch spezieller auf die gefundenen Probleme und deren Optimierung eingegangen.

6.3.2.2 Neue Funktionen

In folge des Experimentes sind auch wieder neue Funktionen für VIFCON hinzugekommen. Alle drei Neuerungen sind teil der Menü-Leiste. Somit wurden die Funktionen:

1. Geräte-Grenzen (Limits) neu einlesen und in VIFCON anpassen,
2. ein synchrones Rezept-Stoppen und,
3. das auslesen von HO und eventuelle aktualisieren der Eurotherm-Ausgangsleistungs-Obergrenze

hinzugefügt. Bei dem ersten Punkt wird nun auch wieder die Config-Datei ausgelesen und in der Log-Datei gespeichert. Die anderen beiden Punkte wurden in Kapitel 5.5 (Rezept) und 5.8 (Sicherheitskonzept) näher erläutert.

6.3.2.3 Anpassung Allgemein

Um VIFCON noch benutzerfreundlicher zu machen, soll die Config-Datei weiterentwickelt werden. So soll z.B. die Init-Funktion besser beschrieben werden. Dies bedeutet das die Datei um mehr Kommentare erweitert wird. Auch die Rezept-Funktion muss näher erläutert werden.

6.3.2.4 Anpassung GUI

Im Unterschied zu den anderen beiden Experimenten, musste die GUI nach diesem Experiment nur minimal angepasst werden. So müssen einige Bezeichnungen abgeändert und die Widgets etwas unplatziert werden. Weiterhin soll der Toggle-Status bei den Checkboxen der Side-Legend umgedreht werden. Somit zeigt ein Häkchen die Kurve an. Als letztes sollten die drei Menü-Punkte zu einem zusammengefasst werden. Somit besteht das Menü in der Menü-Leiste nun aus ca. 3 Ebenen. Die GUI dieses Experimentes ist in Abbildung 70 zu sehen. Die verbesserte GUI ist in Kapitel 5.9 in Abbildung 60 und 61 zu sehen und zeigt die für diese Arbeit endgültige Version der GUI. Neben der Anpassung der GUI wurden auch Pop-Up-Fenster eingearbeitet. Diese werden in Anhang E.7 gezeigt und im Sinne der Programmierung erläutert. Angesprochen wurden diese bereits in Kapitel 5.9.4.

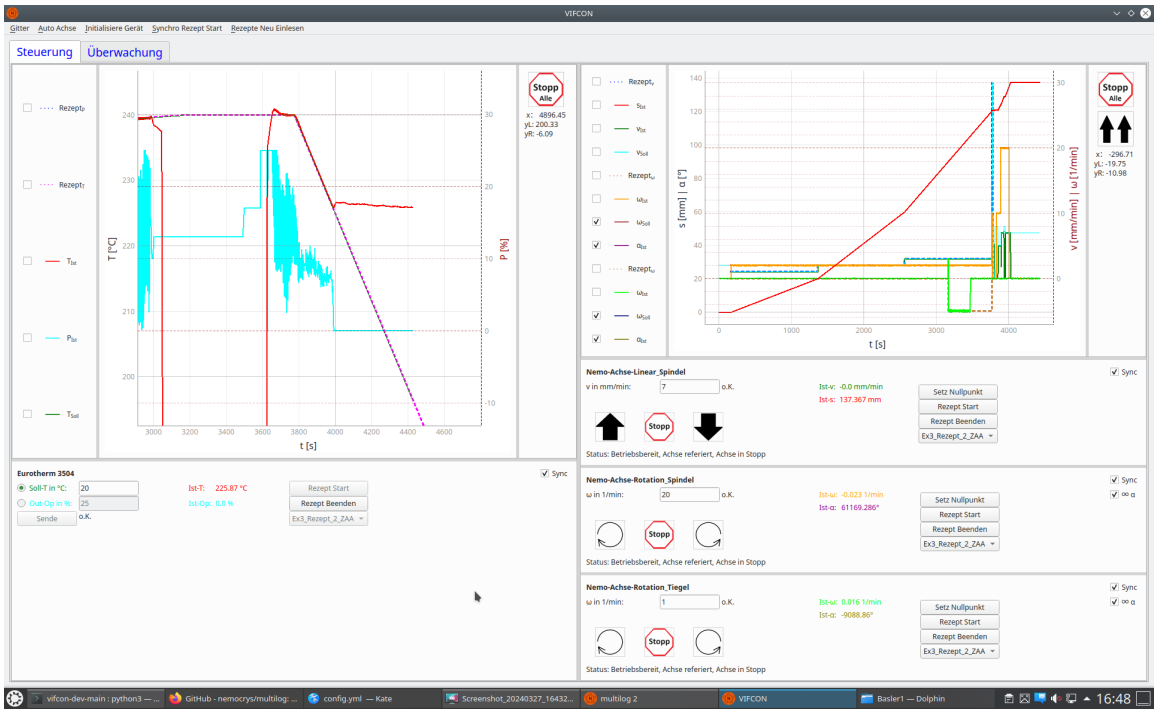


Abbildung 89: GUI während des Experimentes (Quelle: eigene Darstellung)

6.3.2.5 Anpassung Eurotherm

Sowohl in diesem Experiment, als auch in dem Experiment in Kapitel 6.2 wurde der Eurotherm-Regler zur Ansteuerung eines Generators genutzt. In dem Fall war dies ein TruHeat. In den Beschreibungen der Experimente bzw. den Ergebnissen war immer eine starke Oszillation des Leistungseinganges zu sehen. Um die Rezepte zu verbessern, wurde der „op“ Schritt entworfen. Dieser löst einen Leistungssprung im Manuellen Modus des Eurotherms aus. Die Abbildung 86 zeigt den Zeitpunkt an dem dieser Schritt ausgelöst werden sollte. In dem Schritt wurde „IST“ angegeben. Durch diese Einstellung wird nur in den Manuellen Modus gewechselt, wodurch sich der Wert nicht mehr ändert und die aktuelle Ausgangsleistung anliegende bleibt. Im Manuellen Modus wird die Ausgangsleistung durch den Benutzer bestimmt und nicht mehr durch den PID-Regler. Das Problem hierbei ist die Schwankung bzw. die Oszillation der Ausgangsleistung, die durch den PID-Regler kommt. Der Nutzer kann noch immer an dem Eurotherm-Regler direkt Änderungen treffen, obwohl VIFCON aktiv arbeitet. Neben dem Rezept-Segment „op“ wurde der Eurotherm-Regler auch noch um das Segment „opr“ erweitert. In diesem wird kein Sprung, sondern eine Leistungsrampe in dem Temperaturrezept ausgelöst.

Bei Eurotherm wurde nun noch nach weiteren Funktionen gesucht. Zum Beispiel wird die Rampensteigung durch die Zeit beeinflusst. An Eurotherm kann direkt diese Einheit (z.B. Sekunde, Minute, Stunde) ausgewählt werden. Auch ob eine Rampe (Eurotherm eigene) bei Istwert oder Sollwert startet, kann direkt in der Konfiguration eingestellt werden. Leider gibt es für beide keine Mnemonik-Befehle. Um über VIFCON doch eine gewisse Anpassbarkeit zu bekommen, wurde der Config-Datei der Schlüssel „ramp_start_value“ hinzugefügt. Dadurch wird dem Programm gesagt, ob die erste Rampe bei dem Sollwert (SOLL) oder dem Istwert (IST) starten soll.

Zu guter Letzt wurde das Sicherheitskonzept für den Eurotherm erweitert. In Kapitel 6.3.2.2 wurde dies bereits erwähnt. Die Menü-Leiste besitzt nun eine Funktion, wodurch der HO-Wert (maximale Ausgangsleistungs Grenze) ausgelesen wird. Das Sicherheitskonzept wurde in 5.8 erläutert.

6.3.2.6 Anpassung Nemo-1

Bei der Nemo-1-Anlage wurde sich dazu entschieden, dass auf der GUI der simulierte bzw. selbst berechnete Wert auch so gekennzeichnet wird. In Tabelle 14 konnte gesehen werden, dass der tatsächliche Weg der Hub-Bewegung der Achsen ausgelesen wird. Aufgrund eines Fehlers in der Anlage wird dieser nicht richtig angezeigt bei der Spindel. Um diesen Wert trotzdem zu sehen, kann dieser im Plot angezeigt werden. Zusätzlich ist dieser im Widget des Nemo-Antriebs Hub zu sehen.

Auch der Vorfaktor konnte mit dem Experiment verifiziert werden. Durch die weitere Zusammenarbeit mit AUTEAM, wurde dieser jedoch behoben, sodass dieser an sich nicht mehr gebraucht wird. Die Funktion wurde in VIFCON beibehalten. Neben diesem Umstand wurde die Config-Datei so geändert, dass eine Startposition für Hub und Rotation angegeben werden kann. Diese wird zu Beginn in die Simulierten Daten eingelesen und beachtet. Dabei wird sichergestellt das die angegebenen Limits nicht überschritten werden.

6.3.3 Schlussfolgerungen

Das Experiment kann als Erfolg gewertet werden, da mit VIFCON ein Kristall gezüchtet werden konnte. Die aufgetretenen Probleme waren nicht alle durch VIFCON verursacht worden. Zum einem hat die Anlage manches nicht hergegeben und zum anderen wurden Dinge falsch durchgeführt oder nicht beachtet. Das Problem mit der Anlage war einerseits der Vorfaktor, der in der ersten Hälfte auch noch in VIFCON an die falsche Stelle getan wurde und andererseits das Limit in der Geschwindigkeit. Durch dieses war das Abziehen nicht gut umsetzbar. Das Abziehen erfolgt nämlich meist mit einem abrupten, kurzen und sehr schnellen Ziehen des Kristalls, wie es in Abbildung 82 zu sehen ist. Neben diesem Fakt hat auch der Eurotherm durch seinen Regler Probleme bei der Ausgangsleistung verursacht. Obwohl sich die Temperatur recht gut an die Rezept-Kurve anpasst, ist die Oszillation trotz allen zu groß.

Neben den Problemen mit der Anlage wurde auch Punkte bei der Durchführung nicht gut bedacht. Zum einen gab der Punkt für die Rotation und die dazugehörige Änderung auf einen festen OP-Wert Probleme. Um die Rotation möglich zu machen, mussten die Temperaturmessgeräte im Tiegel entfernt werden, wodurch die Temperaturmessung und die dazugehörige Regelung einstürzten. Um den Schritt zu verbessern, wäre es ratsam gewesen, erst den Modus Wechsel zu vollziehen und dann die Rotation des Tiegels einzuleiten, anstelle beides gleichzeitig zu machen. Bei der Abtrennung des Kristalls muss sich auch erst etwas besseres einfallen lassen. Als letztes gab es noch den Punkt der Abkühlrampe, die zu steil war.

7 Zusammenfassung

Mit VIFCON wurde eine funktionsfähige und benutzerfreundliche Steuerung mit GUI geliefert, die dem IKZ, speziell der Modellexperimente-Gruppe, mit der weiteren Durchführung von Kristallzüchtungen helfen soll. Die Umsetzung der Ziele dieser Arbeit wurde erfüllt. VIFCON liefert eine Steuerung, die die Züchtung eines Kristalls nach einem vorgegeben Rezept automatisch durchführt und benutzerfreundlich gestaltet ist. Die Steuerung ist modular erweiterbar und überschaubar. Durch die Splitter- und Tab-Funktionen lässt sich VIFCON an verschiedene Experimente anpassen und ist somit konfigurierbar. Durch die **PyQt5**-Bibliothek ist es gelungen, viele Wünsche an die Steuerung (GUI) umzusetzen. Somit liefert diese Bibliothek eine sehr umfangreiche Fülle von GUI-Tools und auch Punkte wie die Threads und den Mutex für die parallele Nutzung von Ressourcen (Schnittstellen). Somit war es möglich, die Geräte-Bearbeitung in separate Schleifen (Threads) auszulagern und sicherzustellen, dass die Steuerung nicht einfriert und dass Geräte über dieselbe Schnittstelle betreibbar sind.

Weiterhin wurde VIFCON an drei verschiedenen Betriebssystemen getestet. So funktionierte VIFCON an dem Entwicklungsrechner (Windows), dem Nemo-1-Anlagen-Rechner (Linux - Kubuntu), dem Labor-Notebook (Linux - Kubuntu) und zuletzt auch an dem Raspberry PI 400 (Linux basierend auf Debian). Besonders die Kompatibilität mit dem Raspberry Pi war für die Arbeit bedeutend. Anders als ein Rechner kann der Raspberry Pi nun nur mit VIFCON und Multilog betrieben werden, sodass ein eigenständiges und transportierbares System entsteht. Schlussfolgernd kann gesagt werden, dass alle Ziele dieser Arbeit erfüllt wurden.

Im Großen und Ganzen konnte sich VIFCON in den drei Experimenten bewähren und kann somit auch im Folgenden für die Kristallzüchtung verwendet werden. VIFCON zeigt durch die Experimente, dass verschiedenste Anlagen gesteuert werden können. Auch die Nutzung für verschiedene Aufgaben konnte gezeigt werden. Das große Ziel dieses Experimentes war, dass VIFCON einen Kristall züchtet, was gelungen ist.

Ausblick:

VIFCON soll zu einem späteren Zeitpunkt eine freie Software werden, die über GitHub veröffentlicht wird. Das Ende dieser Arbeit bedeutet nicht, dass VIFCON nicht weiter bearbeitet wird. Für VIFCON können noch Optimierungen umgesetzt werden wie:

1. der Anschluss an die Nemo-2-Anlage mit einer ähnlichen Schnittstelle wie bei der Nemo-1-Anlage mit Modbus,
2. die Weiterentwicklung der Nutzung des Gamepads (z.B. PI-Achsen Konzept),
3. die Reduzierung von Verzögerungen bei PI-Achsen und Synchro-Funktionen und,
4. die aktive Regelung des Prozesses wie z.B. die Anpassung der Ziehrate für konstante Durchmesser.

Weiterhin kann VIFCON verschiedene Experimente steuern, wenn die verwendeten Geräte entweder schon vorhanden sind oder im Späteren noch eingearbeitet werden.

Literaturverzeichnis

[Baua] *Baud*. mikrocontroller.net. URL: https://www.mikrocontroller.net/articles/Baud (besucht am 21. 02. 2024).

[Baub] *Baudrate*. IBM Corporation. URL: https://www.ibm.com/docs/de/aix/7.3?topic=parameters-baud-rate (besucht am 13. 03. 2024).

[Bauc] *Baudrate verstehen: Ein umfassender Leitfaden*. Riverdi Sp. z o.o. URL: https://riverdi.com/de/blog/baudrate-verstehen-ein-umfassender-leitfaden (besucht am 24. 03. 2024).

[BT20] Borchers-Tigasson, Prof. Dr. Steffen. *Grundlagen der Automation. 3.3 SPS*. Hochschule für Technik und Wirtschaft Berlin (HTW), 2020.

[Doka] *argparse — Parser for command-line options, arguments and sub-commands*. docs.python.org. URL: https://docs.python.org/3/library/argparse.html (besucht am 14. 11. 2023).

[Dokb] *Betriebsanleitung TruHeat HF Serie 1000/3000/5000*. Fa. © TRUMPF (TRUMPF Hüttinger GmbH + Co. KG), März 2018.

[Dokc] *datetime — Basic date and time types*. docs.python.org. URL: https://docs.python.org/3/library/datetime.html (besucht am 02. 02. 2024).

[Dokd] *Eurotherm. 900 SERIES. 900 Series Digital Communications Handbook*. Fa. Eurotherm Regler GmbH, Jan. 1992.

[Doke] *Graphical User Interfaces with Tk*. docs.python.org. URL: https://docs.python.org/3/library/tk.html (besucht am 06. 05. 2024).

[Dokf] *json — JSON encoder and decoder*. docs.python.org. URL: https://docs.python.org/3/library/json.html (besucht am 24. 04. 2024).

[Dokg] *logging — Logging facility for Python*. docs.python.org. URL: https://docs.python.org/3/library/logging.html (besucht am 14. 11. 2023).

[Dokh] *Matplotlib: Visualization with Python*. The Matplotlib development team. URL: https://matplotlib.org/ (besucht am 06. 05. 2024).

[Doki] *Modelle 3508 und 3504 Prozessregler. Bedienungsanleitung*. Ausgabe 19. Schneider Electric, Mai 2017.

[Dokj] *MS 74E User Manuel. Mercury II C-862 Single-Axis DC-Motor Controller*. Release 8.10. Fa. ©Physik Instrumente (PI) GmbH & Co. KG, Dez. 2002.

[Dokk] *MS173E User Manuel. C-863 MercuryTM. DC Motor Controller*. Release 1.2.7. Fa. ©Physik Instrumente (PI) GmbH & Co. KG, Apr. 2008.

[Dokl] *MS176E Software Manual. Native Commands for MercuryTM Class Controllers*. Release 1.0.1. Fa. ©Physik Instrumente (PI) GmbH & Co. KG, Dez. 2007.

[Dokm] *News*. www.pygame.org. URL: https://www.pygame.org/news (besucht am 03. 05. 2024).

[Dokn] *os — Miscellaneous operating system interfaces*. docs.python.org. URL: https://docs.python.org/3/library/os.html (besucht am 02. 02. 2024).

[Doko] *PyQtGraph*. PyQtGraph developers. URL: https://pyqtgraph.readthedocs.io/en/latest/ (besucht am 02. 02. 2024).

[Dokp] *pySerial*. Chris Liechti. URL: https://pyserial.readthedocs.io/en/latest/pyserial.html (besucht am 02. 02. 2024).

[Dokq] *PyYAML Documentation*. pyyaml.org. URL: https://pyyaml.org/wiki/PyYAMLDocumentation (besucht am 14. 11. 2023).

[Dokr] *Qt for Python.* The Qt Company Ltd. URL: https://doc.qt.io/qtforpython-6/ (besucht am 14. 11. 2023).

[Doks] *random — Generate pseudo-random numbers.* docs.python.org. URL: https://docs.python.org/3/library/random.html (besucht am 02. 02. 2024).

[Dokt] *Reference, Demos and more. NiceGUI Documentation.* NiceGUI. URL: https://nicegui.io/documentation (besucht am 06. 05. 2024).

[Doku] *shutil — High-level file operations.* docs.python.org. URL: https://docs.python.org/3/library/shutil.html (besucht am 02. 02. 2024).

[Dokv] *socket — Low-level networking interface.* docs.python.org. URL: https://docs.python.org/3/library/socket.html (besucht am 24. 04. 2024).

[Dokw] *sys — System-specific parameters and functions.* docs.python.org. URL: https://docs.python.org/3/library/sys.html (besucht am 02. 02. 2024).

[Dokx] *threading — Thread-based parallelism.* docs.python.org. URL: https://docs.python.org/3/library/threading.html (besucht am 24. 04. 2024).

[Doky] *time — Time access and conversions.* docs.python.org. URL: https://docs.python.org/3/library/time.html (besucht am 02. 02. 2024).

[Dokz] *TruHeat HF Serie 1000/3000/5000. Technische Daten.* Fa. © TRUMPF (TRUMPF Hüttinger GmbH + Co. KG). URL: https://www.trumpf.com/de_INT/produkte/leistungselektronik/generatoren-zur-industriellen-erwaermung/truheat-hf/truheat-hf-serie-1000-3000-5000/ (besucht am 30. 05. 2024).

[Dokaa] *Universalregler/Programmregler Serie 902 - 904. Bedienungsanleitung + Kurzanleitung im Anhang.* Ausgabe 03/98 Iss. 9.1. Eurotherm Regler GmbH, 1996.

[Dokab] *Welcome to MinimalModbus' documentation!* Jonas Berg. URL: https://minimalmodbus.readthedocs.io/en/stable/ (besucht am 13. 05. 2024).

[Dokac] *Welcome to pyModbusTCP's documentation.* Loïc Lefebvre. URL: https://pymodbustcp.readthedocs.io/en/latest/ (besucht am 09. 02. 2024).

[Dokad] *Welcome to PyModbus's documentation!* URL: https://pymodbus.readthedocs.io/en/latest/index.html (besucht am 13. 05. 2024).

[Fora] *How to show cursor coordinate in pyqtgraph embedded in pyqt5?* Stack Exchange Inc. URL: https://stackoverflow.com/questions/65323578/how-to-show-cursor-coordinate-in-pyqtgraph-embedded-in-pyqt5 (besucht am 31. 05. 2024).

[Forb] *How to use logging.getLogger(__name__) in multiple modules.* Stack Exchange Inc. URL: https://stackoverflow.com/questions/50714316/how-to-use-logging-getlogger-name-in-multiple-modules (besucht am 31. 05. 2024).

[Forc] *logger configuration to log to file and print to stdout.* Stack Exchange Inc. URL: https://stackoverflow.com/questions/13733552/logger-configuration-to-log-to-file-and-print-to-stdout (besucht am 31. 05. 2024).

[Ford] *pyqtgraph: Add Items from seperate ViewBox to Legend.* Stack Exchange Inc. URL: https://stackoverflow.com/questions/71175650/pyqtgraph-add-items-from-seperate-viewbox-to-legend (besucht am 31. 05. 2024).

[Fore] *python logging specific level only.* Stack Exchange Inc. URL: https://stackoverflow.com/questions/8162419/python-logging-specific-level-only (besucht am 31. 05. 2024).

[Fun22] Funke, Vincent. *Resistive und induktive Heizung in einer Kristallzüchtungsanlage: Automatisierung mit Python und Vermessung der elektromagnetischen Parameter.* Hochschule für Teschnik und Wirtschaft (HTW) und Leibniz Institut für Kristallzüchtung (IKZ), März 2022.

[Fur03] Furrer, Frank J. *Industrieautomation mit Ethernet-TCP/IP und Web-Technologie*. 3., neubearbeitete und erweiterte Auflage. Hüthig GmbH & Co. KG Heidelberg, 2003. ISBN: 3-7785-2860-2.

[Hera] *Kegelradgetriebe KU/I, Bauart K, 4:1*. MÄDLER GmbH. URL: https://www.maedler.de/product/1643/1331/1337/1344/kegelradgetriebe-kui-bauart-k-41 (besucht am 28. 05. 2024).

[Herb] *Rotative Servomotoren für hochpräzise Positionieraufgaben*. JAT – Jenaer Antriebstechnik GmbH. URL: https://www.jat-gmbh.de/produkte/servoantriebe/rotative-servomotoren (besucht am 28. 05. 2024).

[Iee] *IEEE Hauptseite*. IEEE. URL: https://www.ieee.org/ (besucht am 26. 02. 2024).

[Kai97] Kainka, Burkhard. *Messen, Steuern, Regeln über die RS 232-Schnittstelle. Meßdatenerfassung und Prozeßsteuerung mit dem PC*. 7. Auflage. Franzis-Verlag GmbH, 1997. ISBN: 3-7723-6058-0.

[Kria] *Einkristall*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/Einkristall (besucht am 11. 04. 2024).

[Krib] *Kristall*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/Kristall (besucht am 11. 04. 2024).

[Kric] *Polykristall*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/Polykristall (besucht am 11. 04. 2024).

[Lau02] Lauer, Michael. *Python und GUI-Toolkits*. 1. Auflage. mitp-Verlag, 2002.

[Moda] *Modbus*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/Modbus (besucht am 16. 02. 2024).

[Modb] *Modbus Protocol Description*. modbus tools. URL: https://www.modbustools.com/modbus.html (besucht am 26. 02. 2024).

[Mul] *multilog*. IKZ: Nemocrys (Modellexperimente). URL: https://github.com/nemocrys/multilog (besucht am 14. 11. 2023).

[Pera] Persönliche Mitteilung Adrian Wagner (IKZ).

[Perb] Persönliche Mitteilung Bernd Eberhardt (Fa. AUTEAM Industrie-Elektronik GmbH).

[Perc] Persönliche Mitteilung Dr. Kaspars Dadzis (IKZ).

[Perd] Persönliche Mitteilung Dr. Sepehr Foroushani (IKZ).

[Pere] Persönliche Mitteilung Felix Oesterle (IKZ).

[Perf] Persönliche Mitteilung Iason Tsiapkinis (IKZ).

[Perg] Persönliche Mitteilung Max Scheffler (IKZ).

[Perh] Persönliche Mitteilung Stefan Pueschel (IKZ).

[Piaa] *Daisy Chain*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/Daisy_Chain (besucht am 22. 02. 2024).

[Piab] *MP113D M-5x1 Versteller.Benutzerhandbuch*. Version: 1.0.0. Fa. Physik Instrumente (PI) GmbH & Co. KG, Sep. 2013.

[Piac] *Position & Bewegung News. M-500er Klasse setzt (auf) neue Maßstäbe!* Ausgabe 26. Fa. Physik Instrumente (PI) GmbH & Co. KG, 1999. URL: https://www.pi-usa.us/fileadmin/user_upload/pi_us/files/newsletters/PI_PuB26_crp.pdf (besucht am 22. 02. 2024).

[Pyg] *pygame.joystick*. www.pygame.org. URL: https://www.pygame.org/docs/ref/joystick.html (besucht am 15. 05. 2024).

[Reaa] *Python and PyQt: Building a GUI Desktop Calculator*. Real Python. URL: https://realpython.com/python-pyqt-gui-calculator/ (besucht am 24. 04. 2024).

[Reab] *Use PyQt's QThread to Prevent Freezing GUIs*. Real Python. URL: https://realpython.com/python-pyqt-qthread/ (besucht am 24. 04. 2024).

[Reg] *Faustformelverfahren (Automatisierungstechnik)*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/Faustformelverfahren_(Automatisierungstechnik) (besucht am 02. 06. 2024).

[Rpia] *Raspberry Pi*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/Raspberry_Pi (besucht am 16. 05. 2024).

[Rpib] *Raspberry Pi*. Raspberry Pi. URL: https://www.raspberrypi.com/software/ (besucht am 17. 05. 2024).

[Rpic] *Raspberry Pi*. Raspberry Pi Documentation. URL: https://www.raspberrypi.com/documentation/computers/os.html (besucht am 17. 05. 2024).

[Scha] *Kommunikationsprotokoll*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/Kommunikationsprotokoll (besucht am 07. 02. 2024).

[Schb] *Paritätsbit*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/Parit%C3%A4tsbit (besucht am 07. 02. 2024).

[Schc] *Serielle Schnittstelle*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/Serielle_Schnittstelle (besucht am 07. 02. 2024).

[Sch20] Schäfer, Prof. Dr.-Ing. Stephan. *Industrielle Automation an der HTW Berlin (Vorlesungsskript)*. HTW Modul: M5 Industrielle Kommunikation, Okt. 2020.

[Spsa] *CODESYS*. CODESYS Group. URL: https://de.codesys.com/ (besucht am 07. 02. 2024).

[Spsb] *EN 61131*. de.wikipedia.org. URL: https://de.wikipedia.org/wiki/EN_61131 (besucht am 06. 02. 2024).

[Spsc] *WAGO*. WAGO GmbH & Co. KG. URL: https://www.wago.com/de/ (besucht am 07. 02. 2024).

[Sta] *The Python Standard Library*. docs.python.org. URL: https://docs.python.org/3/library/index.html (besucht am 02. 02. 2024).

[Thi96] Thieser, Michael. *PC-Schnittstellen. Datensammlung zu parallelen und seriellen Schnittstellen, Bauanleitung für eine Multischnittstellenkarte, Programmierung*. 3., neu bearbeitete Auflage. Franzis-Verlag GmbH, 1996. ISBN: 3-7723-8092-1.

[Wil88] Wilke K.-TH. und Bohm, Joachim. *Kristallzüchtung*. VEB Deutscher Verlag der Wissenschaften, 1988. ISBN: 3-326-00092-8.

[wis10a] wissenmedia GmbH, Herausgeber. *Brockhaus. A-Z Wissen. In 12 Bänden. Band 6. KONI-MECK*. Bibliographisches Institut & F.A. Brockhaus AG, 2010. ISBN: 978-3-7653-9385-3.

[wis10b] wissenmedia GmbH, Herausgeber. *Brockhaus. A-Z Wissen. In 12 Bänden. Band 8. PACK-ROTD*. Bibliographisches Institut & F.A. Brockhaus AG, 2010. ISBN: 978-3-7653-9385-3.

[wis10c] wissenmedia GmbH, Herausgeber. *Brockhaus. A-Z Wissen. In 12 Bänden. Band 9. ROTE-SPORT*. Bibliographisches Institut & F.A. Brockhaus AG, 2010. ISBN: 978-3-7653-9385-3.

[Xor] *XOR Python Byte Strings*. nitratine.net. URL: https://nitratine.net/blog/post/xor-python-byte-strings/ (besucht am 02. 02. 2024).

A Anhang: Übersicht zu abgegebenen Programmen

Neben dieser Arbeit werden auch die erstellten Programme der VIFCON Steuerung mit abgegeben. Diese liegen in dem Zip-Ordner **VIFCON_Master_HTW_ET_VF.zip**.

Der Ordner enthält neben den Programmen auch die Icons für die GUI sowie Templates für die Log-, Ablauf- und die Config-Datei.

Ordner **VIFCON_Master_HTW_ET_VF.zip**:

1. Ordner: Template
   (a) Ablauf_temp.txt
   (b) config_temp.yml
   (c) vifcon_temp.log
2. Ordner: vifcon
   (a) Ordner: devices
      i. eurotherm.py
      ii. gamepad.py
      iii. multilog.py
      iv. nemoAchseLin.py
      v. nemoAchseRot.py
      vi. nemoGase.py
      vii. piAchse.py
      viii. truHeat.py
   (b) Ordner: icons
      i. checked.png
      ii. checked2.png
      iii. grid.png
      iv. nemocrys.png
      v. p_ccw.png
      vi. p_cw.png
      vii. p_hoch.png
      viii. p_Init_nicht_Okay.png
      ix. p_Init_Okay.png
      x. p_links.png
      xi. p_nichts.png
      xii. p_raus.png
      xiii. p_rechts.png
      xiv. p_rein.png
      xv. p_runter.png
      xvi. p_start.png
      xvii. p_stopp.png
      xviii. p_stopp_all.png
      xix. p_stopp_all_en.png
      xx. p_stopp_En.png

xxi. p_synchro.png
xxii. p_TH_Aus.png
xxiii. p_TH_Ein.png
xxiv. unchecked.png

(c) Ordner: rezepte
   i. Beispiel.png
   ii. Note.txt

(d) Ordner: view
   i. base_classes.py
   ii. eurotherm.py
   iii. main_window.py
   iv. nemoAchseLin.py
   v. nemoAchseRot.py
   vi. nemoGase.py
   vii. piAchse.py
   viii. truHeat.py
   ix. typen.py

(e) __init__.py

(f) _version.py

(g) vifcon_controller.py

3. vifcon_main.py

B Anhang: VIFCON Config-Datei

Die Config-Datei für die VIFCON-Steuerung ist eine Datei mit der Endung **yml**. Diese Endung zeigt dem Nutzer, dass es sich um eine YAML-Datei handelt. Diese Dateien können mit der Python-Bibliothek **PyYAML** (siehe Tabelle 19 und 20) ausgelesen werden. Die Struktur dieser Datei kann folgendermaßen aussehen:

```
time:
  dt-main: 150
config_save: False
log_save: True
plot_save: True
logging:
  level: 20
  filename: vifcon.log
  format: '%(asctime)s %(levelname)s %(name)s - %(message)s'
  filemode: w
  encoding: 'utf-8'
```

Durch das einrücken der einzelnen Punkte (z.B. level, filename, etc.) wird es dem darüber liegenden Punkt (im Beispiel logging) zugeordnet. In Python wird diese Datei wie folgt eingelesen:

```
with open(config, encoding="utf-8") as f:
    self.config = yaml.safe_load(f)
```

Durch das *with open* wird die Datei geöffnet. Im Beispiel liegt der Dateipfad in der Variable *config*. Durch die Methode *safe_load* der **yaml**-Bibliothek (import yaml) wird die YAML-Datei ausgelesen.

Würde die ausgelesen Datei in die Konsole ausgegeben, so kann gesehen werden, dass die YAML-Datei eine Vielzahl von Dictionaries bzw. eine Verschachtelung von Dictionaries ist. Der Grundlegende Aufbau eines Elements im Dictionary ist folgender:

```
dictionary_variable = {Schlüssel:Wert}
```

Um an den Wert eines bestimmten Dictionary-Schlüssels zu kommen muss dieses wie folgt ausgerufen werden:

```
self.config['time']['dt-main']
```

In einem *print*-Befehl würde nun 150 ausgegeben werden. Die Interpretation ist wie folgt. Durch die erste Eckige Klammer wird der Wert des Schlüssels „time“ ausgelesen, welches ein Dictionary ist. Durch die zweite Eckige Klammer wird der Wert des Schlüssels „dt-main“ ausgelesen, was ein Integer ist. Diese Art Code ist im gesamten Code verteilt und ruft Teile in Gruppen (z.B. Schnittstelle) oder einzeln ab.

Die Config-Datei ist eine sehr wichtige Komponente der VIFCON-Steuerung, da mit dieser eine weitere Modularität gegeben wird. Somit kann der Nutzer entscheiden welche Geräte dieser in der GUI sehen will, welche Kurven im Plot erstellt werden sollen und weitere Punkte festlegen.

Im Folgenden sollen nun die einzelnen Haupt-Schlüssel der Config-Datei gezeigt werden. In Tabelle 25 werden diese beschreiben. Die vollständige Config-Datei ist als Template in der Datei **config_temp.yml**, mit Kommentaren zu jedem Wert, zu finden.

Tabelle 25: Haupt-Schlüssel der Config-Datei (Quelle: eigene Darstellung)

Schlüssel	Beschreibung
time	Gibt die Reaktionszeit in Millisekunde wieder. Mit der Zeit werden die Threads aufgerufen!
Function_Skip	Mit der Einstellung können die Extra-Funktionen Multilog-Kommunikation und Gamepad aktiviert werden. Ein True (1) schaltet diese ein.
save	Durch das setzen von drei booleschen Variablen können Dateien im Messordner gespeichert werden. Diese Dateien sind die Plots mit Legende, die Config-Datei und die Log-Datei. Bei den letzten beiden wird eine Kopie erstellt.
language	Mit dem Schlüssel wird die Sprache der GUI eingestellt. Derzeitig gibt es Englisch (EN) oder Deutsch (DE).
GUI_Frame	Schaltet den Rahmen der Geräte-Widgets ein (siehe Abbildung 62).
logging	Hier wird die Log-Datei definiert. Mehr dazu in Anhang C.
consol_logging	Durch die Einstellungen hier werden bestimmte Log-Nachrichten in der Konsole ausgegeben. Mehr dazu in Anhang C.
legend	Hier wird entschieden welche der Legenden gewollt wird. Die Möglichkeiten sind in Abbildung 64 und 90 zu sehen.
skalFak	Hier werden die Skalierungsfaktoren für die derzeitig 11 vorhanden Größen definiert. Bei Null wird die Größe aus dem Label der y-Achse entfernt.
devices	Unter diesem Schlüssel werden alle gewollten Geräte definiert. Die einzelnen Geräte besitzen verschiedene Unter-dictionaries.

Somit regelt die Config-Datei GUI-Einstellungen, Funktionsfreigaben, Speicherdaten, Schnittstellen und das Logging. Die Tabelle 25 zeigt nur die Hauptschlüssel, also die erste Ebene der Konfiguration. Alle tieferen Ebenen können in dem Template gefunden werden. Einige der Punkte wurden in dem Kapitel 5 gezeigt. Zum Beispiel die Auswirkung der Legenden- und Skalierungsfaktoren-Konfiguration. Die Logging-Konfigurationen werden in Anhang C aufgegriffen.

Im Folgenden wird der Schlüssel *devices* vertieft. Wie in der Tabelle zu lesen war, werden hier alle Geräte konfiguriert. Dabei werden Teile für die GUI und die Schnittstelle festgelegte. Da die Geräte unterschiedlich sind, sind die Inhalte auch unterschiedlich. Bei den Geräten muss folgendes beachtet werden. Dies ist einerseits der Schlüssel der Geräte (String) und andererseits die Nutzung mehrere gleicher Geräte. Um das Gerät im Programm zu erkennen, muss der Schlüssel einen bestimmten String beinhalten. Diese sind für (Gerät - **String**):

1. Eurotherm - **Eurotherm**,
2. TruHeat-Generator - **TruHeat**,
3. PI-Achse - **PI-Achse**,
4. Antrieb Nemo-1-Anlage Hub - **Nemo-Achse-Linear**,
5. Antrieb Nemo-1-Anlage Rotation - **Nemo-Achse-Rotation** und,
6. Nemo-1-Anlage Gase Werte - **Nemo-Gase**.

Diese Bezeichnungen erscheinen in der ersten Zeile der Geräte-Widgets. Bei der Mehrfachnutzung eines Gerätes muss der String erweitert werden, da Dictionaries jeden Schlüssel nur einmal beinhalten dürfen, da Python sonst durcheinander kommt und den Schlüssel den Wert nicht zu ordnen kann. So muss z.B. bei der PI-Achse z.B. „PI-Achse_h“ und „PI-Achse_z“ definiert werden. Bei der zweiten Ebene haben die Geräte folgende Punkte gemeinsam. Diese sind:

1. skip - Gerät nicht erstellen bei False (0),
2. typ - Einordnung in die GUI (Antrieb, Generator, Monitoring),
3. start - Starteinstellungen (z.B. init oder readTime),
4. ende - End-Bedingung (Ausführung der Stopp-Funktionen der Geräte),
5. serial-interface - Schnittstellen Definition,
6. multilog - Trigger und Port für die Multilog-Verbindung,
7. limits - interne Grenzwerte für die Größen des Gerätes,
8. GUI - Auswahl der Kurven für den Plot,
9. defaults - Eingabefeld-Anfangswerte und,
10. rezepte - Angabe der Rezepte (siehe Kapitel 5.5).

Bei dieser zweiten Ebene gibt es weiterhin noch:

1. mercury_model (PI-Achse) - Angabe des Mercury-Modells für die Achse,
2. read_TT_log (PI-Achse) - Auslesen und Loggen der aktuellen Zielposition,
3. parameter (PI-Achse, Nemo-1-Anlage - Antriebe) - Extra Parameter für das Gerät (z.B. Adressauswahlcode) und,
4. register (Nemo-1-Anlage - Antriebe, Gase) - Modbusregister Nummer (siehe Tabelle 16, 14 und 15).

Somit sind alle Schlüssel der zweiten Ebene genannt worden. Bei dem Gerät Nemo-Gase, sind dabei nicht alle vorhanden, da hier nur gelesen wird. Somit hat dieses Gerät nur skip, typ, start, serial-interface, multilog, register und parameter. Folgend soll nun noch auf die dritte Ebene eingegangen werden. Den Start macht der Schlüssel „start“. Wie schon erwähnt, werden unter diesem Schlüssel Start-Ereignisse ausgelöst. Alle Geräte besitzen dabei die beiden Schlüssel „readTime“ und „init“. Bei dem ersten handelt es sich um die Auslesezeit. In der *sample*-Funktion wird dabei geschaut ob die Zeitdifferenz, zwischen einer Zeit und der aktuellen Zeit den Wert von „readTime“ überschreitet. Wenn dies der Fall ist, wird die *read*-Funktion ausgelöst. Die Zeit wird in Sekunden angegeben. Unter „init“ wird ein boolescher Wert angegeben. Bei True wird das Gerät direkt initialisiert, also als vorhanden und angesteckt angesehen. Bei False nimmt VIFCON an, dass zwar die Schnittstelle existiert, aber noch kein Gerät angesteckt ist. Solange die Initialisierung False ist, wird im Menü ein rotes Kreuz (Abbildung 99b) angezeigt. Mit der Betätigung des Menü-Knopfes wird versucht das Gerät zu initialisieren. Bei Erfolg beginnt die Messung und die Abbildung 99a ist zu sehen. Zur Messung muss noch gesagt werden, dass eine Angabe von 0 s unter „readTime“ die Messung deaktiviert. Die Tabelle 26 zeigt die anderen spezielleren Schlüssel für die einzelnen Geräte.

Nun soll auch der Schlüssel „ende“ kurz erläutert werden. Der Sinn, dieser Erweiterung der Config-Datei, ist der, dass somit eine Beenden von VIFCON auch bei den Geräten ankommt. Bei setzen auf True weiß VIFCON, dass es die Stopp-Funktion dieses Gerätes auslösen muss. Zum Beispiel wäre dies bei den Achsen wichtig, da diese dann stoppen würden. Eine Auflistung der Ereignisse wurde in Tabelle 22 gezeigt.

Eine weitere Sache in der Config ist der Umstand, dass in VIFCON die Methode *upper* benutzt wird. Diese Methode gehört zu den Strings und ermöglicht es, dass alle Buchstaben in dem String groß geschrieben werden. Hierbei muss nun gut auf die Kommentare in VIFCON aufgepasst werden. An einigen Stellen benötigt es auch klein Buchstaben. Die Stellen an dem *upper* genutzt wird, sind gekennzeichnet. Ein Beispiel für die Nutzung von klein Buchstaben ist der Schlüssel „GUI“, speziell „legend“. In dem Schlüssel „legend“ wird ein String definiert, der von VIFCON in eine Liste umgewandelt wird. Jede vorhandene Kurve der 11 Größen (Temperatur (T), Strom (I), Spannung (U), Leistung in kW (P), Leistung in % (P), Geschwindigkeit in mm/s (v), Geschwindigkeit in mm/min (v), Winkelgeschwindigkeit (v), Frequenz (f), Weg (s) und Winkel (w)) kann mit einem Rezept (Rez), einer Obergrenze (oG), einer Untergrenze (uG), einem Istwert (IW) oder einem Sollwert (SW) in Zusammenhang gebracht werden. Die Möglichen Kurven sind als Kommentar gegeben. Wenn z.B. eine Obergrenze für den Strom im Plot gezeigt werden soll, so muss in dem Wert des Schlüssels „legend“ der Teilstring **oGI** stehen. Wie am Beispiel gesehen werden kann, kommt immer zuerst die Art und dann die Größe. Die 11 Größen werden nur bei den Skalierungsfaktoren unterschieden, da sie hier im Plot Gerät spezifisch auftreten.

Tabelle 26: Ebene 3 - Schlüssel **start** (Quelle: eigene Darstellung)

Schlüssel	Geräte	Beschreibung
sicherheit	Eurotherm	Über den Schlüssel lässt sich eine Sicherheitsfunktion einschalten. Bei True kann der HO-Wert (Maximale Ausgangsleistung) nur am Gerät geändert werden. Bei False wird der OP-Max-Wert der Config-Datei als HO gesetzt (Start, Limit Update).
start_modus	Eurotherm, TruHeat	Mit dem Schlüssel lässt sich der Modus bei Start des Gerätes festlegen (intern, Radio-Button). Eurotherm: Hier gibt es den manuellen (Manuell) und automatischen (Auto) Modus. TruHeat: Hier kann P (Leistung), I (Strom) und U (Spannung) ausgewählt werden. Hierbei wird der Start-Radio-Button gesetzt.
ramp_start_value	Eurotherm	Hier kann entweder IST oder SOLL angegeben werden. Durch die Einstellung wird das Rezept geändert. Bei IST wird die erste Rampe (wenn erstes Segment) beim Istwert und bei SOLL beim Sollwert starten.
ad	TruHeat	Adresse des Generators (Binär)
watchdog_Time	TruHeat	Hier wird eine Zeit in Millisekunden angegeben. Der Watchdog ist ein internes Sicherheitssystem, der bei Auslösung den Generator abstellt.
send_Delay	TruHeat	Eine Zeit die in Millisekunden angegeben wird. Diese Zeit sorgt für ein Delay zwischen den zu sendenden Befehlen.
mode	PI-Achse	Mit dem Schlüssel wird der Knopf-Entriegelungs-Modus eingestellt. 0 = Keine Verriegelung 1 = Entriegelung durch Timer 2 = Entriegelung bei erreichen von 0 mm/s
invert	Nemo-Achse-Linear, Nemo-Achse-Rotation	Wenn True werden die Werte der Geschwindigkeit invertiert! Bei der Spindel würden Rezept und Reale Geschwindigkeit sich unterscheiden!
start_weg	Nemo-Achse-Linear	Startweg für die interne Weg-Berechnung.
start_winkel	Nemo-Achse-Rotation	Startwinkel für die interne Berechnung.

C Anhang: VIFCON Log-Datei

Mit dem Logging kann der Nutzer von VIFCON verschiedene Informationen sammeln. Mit der Config-Datei (Anhang B), der Ablauf-Datei (Anhang D) und den Messdateien bilden diese Dateien eine Wiedergabe des durchgeführten Experimentes. In der Log-Datei werden Debug-, Info-, Warnungs- und Error-Nachrichten gespeichert. In Anhang B wurde gezeigt, dass diese Log-Datei auch durch die Config-Datei konfiguriert werden kann. Die beiden Schlüssel (Ebene 1) waren „logging“ und „consol_logging“. Mit dem Schlüssel „logging“ wird das Logging freigeschaltet. In der Config-Datei sieht dies wie folgt aus:

```
1 logging:
2   level: 20
3   filename: vifcon.log
4   format: '%(asctime)s %(levelname)s %(name)s - %(message)s'
5   filemode: w
6   encoding: 'utf-8'
```

Hierbei müssen folgende Sachen beachtet werden. Bei „level“ werden die angezeigten Nachrichten ausgewählt. Dabei gibt es 10 (debug), 20 (info), 30 (warning) und 40 (error). Wenn das Debug-Level angegeben wird, werden alle höheren Level auch angezeigt. Wird jedoch das Info-Level, wie im Beispiel, angegeben, dann werden keine Debug-Nachrichten mehr geloggt. Somit werden immer nur das angegebene Level und höhere Level geloggt. Mit „filename“ wird der Dateiname mit der Endung log angegeben. Wenn dies frei bleibt, so werden die Nachrichten in der Konsole ausgegeben. In „format“ wird das Nachrichten-Format angegeben. Das gezeigte Format sieht wie folgt aus:

```
1 2024-05-22 09:39:50,117 INFO vifcon.view.nemoGase - Nemo-Gase_1 - Erstelle ...
      Widget.
```

Beim „filemode“ kann nun entschieden werden ob die Datei immer überschrieben wird (w) oder immer erweitert wird (a). Zu guter Letzt wird noch das „encoding“ gesetzt. Hierbei muss darauf geachtet werden, dass bei Linux dies auskommentiert werden muss, da es sonst zu Fehlermeldung kommt.

Um das Logging in VIFCON zu nutzen benötigt es die Python-Bibliothek **logging**. Das Erstellen und Vorbereiten der Log-Datei geschieht in dem *Controller*-Objekt im Programm **vifcon_controller.py** und sieht wie folgt aus:

```
1  class Controller(QObject):
2      # Teil hier nicht gezeigt
3      def __init__(self, config, output_dir, test_mode, neustart) -> None:
4          # Teil hier nicht gezeigt
5          logging.basicConfig(**self.config["logging"])
6          logging.info(self.Log_Text_6_str[self.sprache])
7          logging.info(f"{self.Log_Text_7_str[self.sprache]} {self.config}")
8          if self.test_mode:
9              logging.info(self.Log_Text_8_str[self.sprache])
10         # Teil hier nicht gezeigt
```

In Zeile 5 werden alle Config-Daten aus dem Schlüssel „logging“ an die Logging-Konfiguration übergeben. Die darauf folgenden Zeilen zeigen wie etwas geloggt wird. Hierbei werden die Methoden *info*, *debug*, *warning*, *error* und *exception* genutzt. Die letzte Methode wird bei Try-Except-Anwendungen verwendet, um den Fehler mit zu speichern.

Nun gibt es jedoch ein kleines Problem, da der Nutzer nicht immer in diese Datei schauen kann. Um zum einen die Datei zu haben und zum anderen die Warnungen oder andere Nachrichten im Verlauf des Experimentes zu sehen, müssen diese in der Konsole oder durch Pop-Up-Fenster ausgegeben werden. Die Pop-Up-Fenster sind ein kleiner Zusatz und wurden nicht mit dem Logging verbunden. Zu diesen folgt in Anhang E.7 etwas mehr. Weiterhin wurden diese in Kapitel 5.9.4 erläutert. Um auf das Logging zurückzukommen, wird für die Anzeige in der Konsole nun der Schlüssel „consol_logging“ verwendet. Dieser sieht in der Config-Datei wie folgt aus:

```
1 consol_Logging:
2   level: 30
3   print: 1
4   format: '%(asctime)s %(levelname)s %(name)s - %(message)s'
```

Wie beim Schlüssel „logging“ wird hier ein Level und ein Format angegeben. Der Unterschied ist nun der Unter-Schlüssel „print“. Die Funktion des Konsolen-Loggings ist folgende. Durch die Angabe des Levels und von „print“ weiß VIFCON, welche Nachrichten diese in die Konsole schreiben soll. Mit „print“ werden drei Optionen gegeben: Schreibe nur das angegebene Level (1), Schreibe auch alle kleineren Level (2) und Schreibe auch alle größeren Level in die Konsole. Bei diesen Einstellungen muss der Schlüssel „logging“ mit beachtet werden. Wenn z.B. bei „logging“ 30 als Level gewählt wird und bei „consol_logging“ 20, dann wird nichts in der Konsole ausgegeben. Das selbe gilt auch für die „print“-Einstellungen. Wenn 20 bei „logging“ ausgewählt ist und die Einstellung 2 mit Level 30 bei „consol_logging“ gewählt wurde, dann werden nur Info- und Warnungsnachrichten ausgegeben. Der Schlüssel „logging“ wird vorrangig behandelt und setzt das Logging-Level. Kein anderes Level außer Größer kann dann geloggt werden!

Bevor das Konsolen-Logging weiter erläutert wird, muss gesagt werden, wie das Logging überall freigeschaltet wird. Für diese Zwecke wird ein Logger in der Variable *logger* erstellt. Diese Variable wird außerhalb der Klassen definiert und befindet sich in den Programmen immer unter dem Import der Bibliotheken. Folgend ist dies zu sehen:

```
1 vifcon_controller.py:       logger = logging.getLogger()
2 andere Programme:           logger = logging.getLogger(__name__)
```

Die Erstellung von *logger* ist unterschiedlich. Um das Konsolen-Logging freizuschalten muss der Logger in **vifcon_controller.py** als übergeordneter Logger gelten, was durch den Code gewährleistet wird. Somit bekommen alle anderen Programme (außer **vifcon_main.py**) die Zeile 2 zugewiesen. Auch hier wird diese Variable außerhalb der Klassen und Funktionen deklariert. Der Grundgedanke hierfür stammt aus der Quelle [Forb].

Um das Konsolen-Logging nun zu gewährleisten müssen ein neuer Streamhandler und ein Filter erzeugt werden. Die Grundidee für den Streamhandler wurde der Quelle [Forc] und beim Filter aus Quelle [Fore] entnommen. Der neue Streamhandler wird wie folgt erstellt:

```
1  class Controller(QObject):
2      # Teil hier nicht gezeigt
3      def __init__(self, config, output_dir, test_mode, neustart) -> None:
4          # Teil hier nicht gezeigt
5          ### Erzeugung eines weiteren Handlers für das Logging:
6          consoleHandler = logging.StreamHandler()
7          consoleHandler.setLevel(logging.DEBUG)
8          consolFormat = logging.Formatter(self.config['consol_Logging']['format'])
9          consoleHandler.setFormatter(consolFormat)
10         ### Füge an bestehenden Handler:
11         logger.addHandler(consoleHandler)
12         # Teil hier nicht gezeigt
```

Wie gesehen werden kann wird hier der Aufruf *StreamHandler* genutzt. Im Folgenden wird diesem Streamhandler ein Level und ein Format zugewiesen. Um diesen neuen Handler nun zu nutzen, muss dieser noch an den Haupt-Streamhandler, in dem Fall *logger*, angeheftet werden. Um nun nur bestimmte Log-Nachrichten anzeigen zu können, wird ein Filter gebraucht. Der Filter wird wie folgt erstellt:

```
1  class Controller(QObject):
2      # Teil hier nicht gezeigt
3      def __init__(self, config, output_dir, test_mode, neustart) -> None:
4          # Teil hier nicht gezeigt
5          ### Filter für den neuen Handler:
6          ak_Level_Consol_Log = self.config['consol_Logging']['level']
7          level_Log = {10: logging.DEBUG, 20: logging.INFO, 30: ...
               logging.WARNING, 40: logging.ERROR}
8          ak_Anzeige_Level = self.config['consol_Logging']['print']
9          try:
10             consoleHandler.addFilter(MyFilter(level_Log[ak_Level_Consol_Log], ...
                   ak_Anzeige_Level))
11         except:
12             consoleHandler.addFilter(MyFilter(logging.WARNING, 1))
13             logger.warning(self.Log_Text_9_str[self.sprache])
14         # Teil hier nicht gezeigt
```

In dem Code-Fragment wird nun eine Instanz der Klasse *MyFilter* erzeugt. Diese bekommt das Level und die Print-Einstellung zugewiesen. Durch *addFilter* wird der erstellte Filter an den Handler gehangen. Im Falle eines Erstellungsfehlers, wird der Filter so erstellt, dass nur Warnungs-Nachrichten geprintet werden. Die Klasse *MyFilter* besitzt die Methoden *__init__* und *filter*. In der Init-Funktion werden nur die übergebenen Variablen für das Objekt gespeichert. Die Funktion *filter* wird folgend gezeigt:

```
1  class MyFilter(object):
2      # Teil hier nicht gezeigt
3      def filter(self, logRecord):
4          if self.operator == 1:
5              return logRecord.levelno == self.__level
6          elif self.operator == 2:
7              return logRecord.levelno <= self.__level
8          elif self.operator == 3:
9              return logRecord.levelno >= self.__level
```

Durch die Verbindung mit dem Haupt-Logger wird die Funktion *filter* immer dann aufgerufen, wenn eine Nachricht geloggt wird. In dem Fall wird geschaut, welcher Operator (Schlüssel „print“) genutzt werden soll und das Level der Log-Nachricht (*logRecord*) mit dem gewollten Level verglichen. Wenn alles stimmt, wird die Nachricht in der Konsole ausgegeben. Hierbei kommt auch das Beispiel mit dem Logging-Levels zu tragen. Durch das Logging werden nur die Nachrichten ausgelöst, die das passende Level haben. Das bedeutet, dass die Funktion *filter* nur beim regulären Logging-Level aufgerufen wird.

Mit diesen Einstellungen wird ein sehr umfangreiches und bemerkbares Logging möglich. Der Nutzer hat nach dem Versuch eine Datei, in der alle Vorkommnisse stehen und hat während des Versuches die Möglichkeit, die wichtigsten Log-Nachrichten zu sehen. Eine wichtige Erweiterung in der Log-Datei war auch die Speicherung der Rezepte und der Änderung der Config-Datei. Am Ende eines Experimentes können die Config- und Log-Dateien in den Messordner kopiert werden. Dabei wird immer die aktuelle Version der Config-Datei bei Beendigung von VIFCON kopiert. Während eines Experimentes kann diese Datei, sowie auch die Rezepte und Rezept-Dateien, verändert werden, wodurch Informationen verloren gehen können. Aus dem Grund werden diese Werte und Daten als Information in der Log-Datei gespeichert. Die Log-Datei ist somit sehr umfangreich. Ein Beispiel ist in dem beiliegenden Ordner **Template** in der Datei **vifcon_temp.log** zu finden.

D Anhang: VIFCON Ablauf-Datei

Die Ablauf-Datei ist eine spezielle Text-Datei. Sie ist der Log-Datei aus Anhang C ähnlich, speichert aber andere Informationen. Der Sinn dieser Datei ist es, Informationen zu Befehlssendungen und Knopf-Betätigungen zu speichern. Die Datei selbst wird immer im Messordner gespeichert/erstellt, wenn der Test-Modus nicht aktiv ist. Auf Seite 191 werden verschiedene Code-Teile gezeigt. Der erste zeigt die Erstellung der Datei.

Zur Erstellung wird *with open* verwendet. Wie in Zeile 8 (Code-Teil 1 Seite 191) gesehen werden kann, ist der Datei-Name fest einprogrammiert. Die Funktion *add_Ablauf* wird an alle Objekte übergeben. Sie füllt die Datei und liefert durch den leichten Aufruf eine leichte Erweiterung um Nachrichten in der Datei. Die genannte Funktion ist im zweiten Code-Teil zu sehen und ihr Aufruf im dritten.

Die Funktion *add_Ablauf* wird in den Objekten zu *add_Text_To_Ablauf_Datei*. Bei Aufruf wird die Datei erweitert (a) und nicht überschrieben (w). Im Ordner **Template** gibt es eine Beispiel Datei mit dem Namen **Ablauf_temp.txt**. Der vierte Code-Teil zeigt ein Beispiel für die Nachrichten in der Ablaufdatei.

Datei-Erstellung:

```
class Controller(QObject):
    # Teil hier nicht gezeigt
    def __init__(self, config, output_dir, test_mode, neustart) -> None:
        # Teil hier nicht gezeigt
        if not self.test_mode:
            # Teil hier nicht gezeigt
            self.txtDat_btn = f'{self.directory}/Ablauf.txt'
            with open(self.txtDat_btn,'w', encoding="utf-8") as f:
                f.write(f'{self.Text_2_str[self.sprache]}\n')
                f.write(f'{self.Text_3_str[self.sprache]}\n\n')
            # Teil hier nicht gezeigt
```

Funktion:

```
def add_Ablauf(self, text):
    if not self.test_mode:
        timestamp = ...
            datetime.datetime.now(datetime.timezone.utc).astimezone().isoformat(timespec='milliseconds').replace('T', ' ')
        with open(self.txtDat_btn,'a', encoding="utf-8") as f:
            f.write(f'{timestamp} - {text}\n')
```

Aufruf:

```
self.add_Text_To_Ablauf_Datei(f'{self.device_name} - {error_Message_Ablauf}')
```

Beispiel Inhalt:

```
Ablauf der Messung:
Werteingaben und Knopfberührungen

2024-05-02 16:24:13.203+02:00 - Nemo-Achse-Linear_Spindel - Knopf betätigt - Hoch!
2024-05-02 16:24:13.345+02:00 - Nemo-Achse-Linear_Spindel - Geschwindigkeit erfolgreich gesendet!
```

E Anhang: Erläuterungen zur Programmierung von der VIFCON GUI

In dem Kapitel soll die GUI noch näher beschrieben werden. Dabei soll auch auf den Code eingegangen werden.

E.1 GUI-Fehlermeldungen

In den Geräte-Widget-Objekten wurde für die GUI-Fehlermeldung die Funktion *Fehler_ Output* erstellt. Diese unterscheidet sich nur minimal von Gerät zu Gerät. Der Unterschied ist der Fall, dass die Antriebe zwei Fehler-Label (*self.La_ error_ 1* und *self.La_ error_ 2*) in ihrem Widget besitzen. Durch diesen Umstand muss zwischen den beiden Fehler-Labeln der Antriebe eine Unterscheidung vorgenommen werden. Folgend wird der Code für die Funktion *Fehler_ Output* vom Eurotherm-Widget-Objekt gezeigt. Die Funktion *add_ Text_ To_ Ablauf_ Datei* erzeugt eine Zeile in der Ablauf-Datei. Diese Datei wurde näher in Anhang D erläutert. Die Auswirkung des Codes auf die GUI, kann in Abbildung 58 gesehen werden. An den Eingabefeldern taucht in dem Fall eines Fehlers, das Wort Fehler (fett und rot) auf. Durch den Tooltip kann der genaue Fehler dann angesehen werden. Die Variablen wie z.B. *err_ 13_ str* sind Teil der Spracheinstellung, welche näher in Anhang E.6 erläutert wird.

```
1  def Fehler_Output(self, Fehler, error_Message_Log_GUI = '', ...
       error_Message_Ablauf = ''):
2      if Fehler:
3          self.La_error.setText(self.err_0_str[self.sprache])
4          self.La_error.setToolTip(error_Message_Log_GUI)
5          self.La_error.setStyleSheet(f"color: red; font-weight: bold")
6          log_vorberietung = error_Message_Log_GUI.replace("\n"," ")
7          logger.error(f'{self.device_name} - {log_vorberietung}')
8      else:
9          self.La_error.setText(self.err_13_str[self.sprache])
10         self.La_error.setToolTip('')
11         self.La_error.setStyleSheet(f"color: black; font-weight: normal")
12     if not error_Message_Ablauf == '':
13         self.add_Text_To_Ablauf_Datei(f'{self.device_name} - ...
               {error_Message_Ablauf}')
```

Der Aufruf sieht wie folgt aus:

```
1  self.Fehler_Output(1, self.La_error_1, self.err_4_str[self.sprache])
2  self.Fehler_Output(0, self.La_error_1)
```

In Zeile 1 wird ein Fehler (1) an die GUI, die Log-Datei und die Ablauf-Datei gesendet. Bei der zweiten Zeile wird der Label-Text „o.K.“ an die GUI übergeben. Die Ablauf-Datei muss nicht immer mit beschreiben werden.

E.2 Skalierungsfaktoren

Über die Skalierungsfaktoren wurde in Kapitel 5.9.5.4 bereits etwas gesagt. Die Skalierungsfaktoren spielen eine große Rolle bei den Messkurven des Plots, da die Wertebereich mancher Größen stark anders sein können. Aus der Config bekommt jede der 11 Größen einen Skalierungsfaktor zugeordnet. Diese 11 Dictionaries sind der Wert vom Schlüssel „skalFak“. Dieser wird in **vifcon_controller.py** ausgelesen und den beiden Geräte-Typen Antrieb und Generator übergeben. Die Faktoren gelangen somit in die Klassen *Generator* und *Antrieb* des Programms **typen.py**. Beide Klassen sind ähnlich aufgebaut. In diesen wird die Achsenbeschriftung der y-Achsen vorbereitet und weiter an die Klasse *PlotWidget* in **base_classes.py** gegeben. In Kapitel 5.9.5.4 wurde bereits das Ergebnis dieses Codes gezeigt. Somit tauchen die Faktoren in der Achsenbeschriftung auf oder die ganze Messgröße wird aus der Beschriftung entfernt. Der letzte erwähnte Fall erfolgt bei Skalierungsfaktor Null, wodurch auch eine Warnung in der Log-Datei eingetragen wird.

```
1  class Generator(QWidget, Cursor, PopUpWindow):
2      def __init__(self, start_zeit, tab_splitter_widget, add_Ablauf_function, ...
           stopp_all_function, menu_dict, legend_ops, Faktoren, sprache, ...
           parent=None):
3          # Teil hier nicht gezeigt
4          # Faktoren bestimmen für Label:
5          label_dict = {'Temp': 'T [°C]', 'Current': 'I [A]', 'Voltage': 'U ...
               [V]', 'Op': 'P [%]', 'Pow': 'P [kW]', 'Freq': 'f [kHz]'}
6          label_list = []
7          for size in label_dict:
8              if not self.Faktor[size] == 1:
9                  if not self.Faktor[size] == 0:
10                     label_list.append(f'{label_dict[size]}x{self.Faktor[size]}')
11                 else:
12                     label_list.append('0')
13                     logging.warning(f'{self.Log_Text_185_str[self.sprache]} ...
                           {size}')
14             else:
15                 label_list.append(f'{label_dict[size]}')
16         # Teil hier nicht gezeigt
```

Das mit der Achse wird im Folgenden gezeigt. Sobald kein Eintrag mehr vorhanden ist, wird „Keine Einträge!“ im Plot angezeigt!

```
1  class Generator(QWidget, Cursor, PopUpWindow):
2      def __init__(self, start_zeit, tab_splitter_widget, add_Ablauf_function, ...
           stopp_all_function, menu_dict, legend_ops, Faktoren, sprache, ...
           parent=None):
3          # Teil hier nicht gezeigt
4          Achse_y1_str = f'{label_list[0]} | {label_list[1]} | ...
               {label_list[2]}'.replace(' | 0', '').replace('0 |','')
5          if Achse_y1_str == '0':
6              Achse_y1_str = self.Eintrag_Achse[self.sprache]
7          # Teil hier nicht gezeigt
```

Durch den Umstand, dass die Geräte-Widget-Objekte auf die Geräte-Typen-Objekte zugreifen können, besitzen somit alle Geräte-Widget-Objekte auch den Zugang zu den Skalierungsfaktoren. Bei der Initialisierung dieser Objekte wird dieser Wert übernommen und für das Objekt gespeichert. Um diese Faktoren zu nutzen wird ein Dictionary namens *skalFak_dict* erstellt. Hierbei sind die Kurven-Bezeichnung (erwähnt in Anhang B) wichtig. Anhand dieser werden die Faktoren ihren Größen (Istwert, Sollwert) zugeordnet. Folgend wird die Zuordnung am Beispiel Eurotherm gezeigt. Das Dictionary *curveDict* findet in Anhang E.3 mehr Erläuterung.

```
class EurothermWidget(QWidget):
    def __init__(self, sprache, Frame_Anzeige, widget, line_color, config, ...
        config_dat, neustart, add_Ablauf_function, eurotherm = 'Eurotherm', ...
        typ = 'Generator', parent=None):
        # Teil hier nicht gezeigt
        self.skalFak_dict = {}
        for size in self.curveDict:
            if 'WT' in size:
                self.skalFak_dict.update({size: self.skalFak['Temp']})
            if 'WOp' in size:
                self.skalFak_dict.update({size: self.skalFak['Op']})
        # Teil hier nicht gezeigt
```

Nach dem das Dictionary erstellt wurde, wird dieses erst wieder in der Funktion *update_ GUI* aufgerufen. Hierbei gibt es zwei Stellen an dem die Faktoren gebraucht werden. Der erste Fall ist die Anpassung der Messdaten. In der Funktion werden die Messgrößen in einer For-Schleife abgearbeitet. Wenn ein Messwert vorliegt, wird der zur Größe (*messung*) passende Faktor ermittelt und mit allen Messdaten aus *listDict[messung]* verrechnet. Am Ende wird die neuen Daten für die passende Kurve (*curveDict[messung]*) gesetzt.

```
def update_GUI(self, value_dict, x_value):
    # Teil hier nicht gezeigt
    for messung in value_dict:
        # Teil hier nicht gezeigt
        self.listDict[messung].append(value_dict[messung])
        if not self.curveDict[messung] == '':
            faktor = self.skalFak_dict[messung]
            y = [a * faktor for a in self.listDict[messung]]
            self.curveDict[messung].setData(x_value, y)
    # Teil hier nicht gezeigt
```

Der zweite Fall sind die Grenzen und Rezepte. Hierbei wird der passende Faktor manuell zu den Größen zugeordnet. Auch diese Kurven müssen angepasst werden.

```
Grenze:     self.grenzValueDict['oGT']  = self.oGST * self.skalFak['Temp']
Rezept:     faktor = self.skalFak['Temp']
```

Die Skalierungsfaktoren haben somit zwei Aufgaben. Zum einen sollen diese in dem y-Achse-Label zu sehen sein bzw. dieses bearbeiten und zum anderen die Plot-Kurven um skalieren. Die Skalierungsfaktoren haben keine Auswirkung auf die Messdatei.

E.3 Plot und Kurven

Insgesamt besitzt VIFCON zwei Plots, einen für die Geräte des Generator-Typs und einen für die Geräte des Antrieb-Typs. Die Erstellung der Geräte-Typen-Widgets erfolgt durch die Klassen in **typen.py**. Aufgerufen werden sie aus dem *Controller*-Objekt. Für den Plot an sich ist nur das Objekt *PlotWidget* aus **base_classes.py** relevant. Das Objekt erbt von *QWidget*. Das Grundgerüst für die Erstellung des Plots ist im Folgenden zu sehen. Je nach dem wie die Legende gesetzt werden soll, wird die Variable *graph* eine andere Instanz bekommen. Bei der Legende im und neben dem Plot wird die Instanz von *pg.PlotWidget* erstellt. Bei der Legende unter dem Plot (Out) wird eine Instanz von *pg.GraphicsLayoutWidget* erstellt. Das „pg“ kommt von dem Import der Bibliothek **pyqtgraph**, die für die Plots genutzt wird. Der obere Teil des Codes zeigt das erstellt Widget mit seinem *QGridLayout*. An dieses Layout wird dann die Instanz des Graphen gehangen. Um die Hintergrundfarbe zu ändern wird *setBackground* genutzt.

```
1  class PlotWidget(QWidget):
2      def __init__(self, menu, legend_ops, sprache, typ, label_x, label_y1, ...
           label_y2 = '', parent=None):
3          # Teil hier nicht gezeigt
4          self.plot_widget = QWidget()
5          self.plot_layout = QGridLayout()
6          self.plot_widget.setLayout(self.plot_layout)
7
8          if self.legend_pos == 'OUT':
9              self.graph = pg.GraphicsLayoutWidget()
10         else:
11             self.graph = pg.PlotWidget()
12
13         self.graph.setBackground('w')
14         self.plot_layout.addWidget(self.graph)
15         # Teil hier nicht gezeigt
```

Nach dem Code-Teil würde die Definition der Legende kommen. Diese wird jedoch erst in Anhang E.4 gezeigt. In diesem Code wird auch der Plot durch *addplot* oder *plotItem* erzeugt. Mit der nun erzeugten Variable *achse_1* kann nun weitergearbeitet werden. Diese Achse erhält durch *setlabel* eine Bezeichnung. Durch die Funktions-Variable *axis* wird dabei zwischen der linken y-(left) und der x-Achse (button) unterschieden. Mit *showGrid* wird das Gitter sichtbar gemacht und mit *enableAutoRange* werden Kurven und Achsen aneinander angepasst.

In Kapitel 5.9.5.1 wurde erwähnt, dass VIFCON zwei y-Achsen hat. Die Umsetzung der zweiten Achse wird durch eine *pg.ViewBox* ermöglicht. Diese ViewBox wird an den Plot (Variable *achse_1*) angefügt und im Folgenden bearbeitet. Hierbei wird sowohl die Farbe der Achse, als auch der Linien-Style festgelegt. Der folgende Code zeigt die Umsetzung. Die grundlegende Idee für diesen Code stammt aus der Quelle [Ford].

```
class PlotWidget(QWidget):
    def __init__(self, menu, legend_ops, sprache, typ, label_x, label_y1, ...
        label_y2 = '', parent=None):
        # Teil hier nicht gezeigt
        if label_y2 != '':
            self.achse_2 = pg.ViewBox()
            self.achse_1.showAxis('right')
            self.achse_1.scene().addItem(self.achse_2)
            self.achse_1.getAxis('right').linkToView(self.achse_2)
            self.achse_2.setXLink(self.achse_1)
            right_color = 'red'
            self.achse_1.getAxis('right').setLabel(label_y2, **{'color': ...
                right_color, 'font-size': f'{fontsize}pt'})
            self.achse_1.getAxis('right').setPen(color=right_color, ...
                style=Qt.DashLine)

            self.achse_2.enableAutoRange(axis="x")
            self.achse_2.enableAutoRange(axis="y")

            def updateViews():
                self.achse_2.setGeometry(self.achse_1.vb.sceneBoundingRect())
                self.achse_2.linkedViewChanged(self.achse_1.vb, ...
                    self.achse_2.XAxis)

            updateViews()
            self.achse_1.vb.sigResized.connect(updateViews)
        # Teil hier nicht gezeigt
```

Nachdem der Plot erstellt wurde, müssen die einzelnen Kurven angefügt werden. Der obere Code auf Seite 198 zeigt dies. Um die Kurven zu erstellen wird eine Liste durchgegangen. Diese Liste wird der Config-Datei (Schlüssel „GUI“) entnommen. Im folgend zu sehenden Code werden die Liste erzeugt und die Leerzeichen entfernt. Dies ist bei den Geräten identisch. Als Beispiel wird TruHeat genommen.

```
class TruHeatWidget(QWidget):
    def __init__(self, sprache, Frame_Anzeige, widget, line_color, config, ...
        config_dat, neustart, add_Ablauf_function, truheat = 'TruHeat', typ = ...
        'Generator', parent=None):
        # Teil hier nicht gezeigt
        self.legenden_inhalt = self.config['GUI']['legend'].split(';')
        self.legenden_inhalt = [a.strip() for a in self.legenden_inhalt]
        # Teil hier nicht gezeigt
```

Weiterhin ist das Dictionary *kurv_dict* wichtig. In diesem sind die Kurven-Kürzel als Schlüssel und als Wert eine Liste mit Achse (a1 - links, a2 - rechts), Kurven-Linien-Formatierung und Legenden-Label enthalten. Folgend wird ein Auszug aus dem Dictionary des TruHeat gezeigt.

```
1 class TruHeatWidget(QWidget):
2     def __init__(self, sprache, Frame_Anzeige, widget, line_color, config, ...
          config_dat, neustart, add_Ablauf_function, truheat = 'TruHeat', typ = ...
          'Generator', parent=None):
3         # Teil hier nicht gezeigt
4         kurv_dict = {
5             'IWI':  ['a1', pg.mkPen(self.color[5], width=2), f'{truheat} - ...
                  {I_einzel_str[self.sprache]}<sub>{istwert_str[self.sprache]} ...
                  </sub>'],
6             'SWI':  ['a1', pg.mkPen(self.color[2]), f'{truheat} - ...
                  {I_einzel_str[self.sprache]}<sub>{sollwert_str[self.sprache]} ...
                  </sub>'],
7              # Teil hier nicht gezeigt
8         }
9         # Teil hier nicht gezeigt
```

In dem oberen Code von Seite 198 wird die Kurve je nach Legenden-Einstellung erzeugt. Die Kurve wird der Variable *curve* zugeordnet und am Ende in dem Dictionary *kurven_dict* mit dem Kurven-Kürzel gespeichert. Mit der *ist_drin_list*-Liste wird eine doppelte Erstellung der Kurve verhindert. Nach Erstellung der Kurven werden diese noch für das Update an verschiedene andere Dictionaries übergeben. Hierbei bekommen die Kürzel die Kurven-Instanz in *curveDict*, die *QLabel* in *labelDict*, ihre Einheiten in *labelUnitDict* und die Daten-Listen in *listDict*. Für die Grenzen gibt es die Dictionaries *grenzListDict* und *grenzValueDict*. Nachdem nun auch noch die Skalierungsfaktoren (Anhang E.2) zugeordnet sind, werden die Kurven nur noch bei der Funktion *update_GUI* betrachtet. Auf Seite 198 und 199 können zwei Teile der Funktion gefunden werden. Der untere Code auf Seite 198 zeigt, wie die genannten Dictionaries genutzt werden. Hier werden die Label und die Kurven aktualisiert. Der Beispiel-Code dafür stammt vom TruHeat. Der Code von Seite 199 zeigt das Aktualisieren der Grenz/Limit-Kurven.

```python
class TruHeatWidget(QWidget):
    def __init__(self, sprache, Frame_Anzeige, widget, line_color, config, config_dat, neustart, add_Ablauf_function, ...
        truheat = 'TruHeat', typ = 'Generator', parent=None):
        # Teil hier nicht gezeigt
        ## Kurven erstellen:
        ist_drin_list = []
        self.kurven_dict = {}
        for legend_kurve in self.legenden_inhalt:
            if legend_kurve in kurv_dict and not legend_kurve in ist_drin_list:
                if kurv_dict[legend_kurve][0] == 'a1':
                    curve = self.typ_widget.plot.achse_1.plot([], pen=kurv_dict[legend_kurve][1], ...
                        name=kurv_dict[legend_kurve][2])
                    if self.typ_widget.legend_ops['legend_pos'].upper() == 'OUT':
                        self.typ_widget.plot.legend.addItem(curve, curve.name())
                elif kurv_dict[legend_kurve][0] == 'a2':
                    curve = pg.PlotCurveItem([], pen=kurv_dict[legend_kurve][1], name=kurv_dict[legend_kurve][2])
                    self.typ_widget.plot.achse_2.addItem(curve)
                    if self.typ_widget.legend_ops['legend_pos'].upper() == 'OUT' or ...
                        self.typ_widget.legend_ops['legend_pos'].upper() == 'IN':
                        self.typ_widget.plot.legend.addItem(curve, curve.name())
                self.kurven_dict.update({legend_kurve: curve})
                ist_drin_list.append(legend_kurve)
        # Teil hier nicht gezeigt
```

```python
def update_GUI(self, value_dict, x_value):
    # Teil hier nicht gezeigt
    for messung in value_dict:
        if not 'SW' in messung:
            self.labelDict[messung].setText(f'{value_dict[messung]} {self.labelUnitDict[messung]}')
        self.listDict[messung].append(value_dict[messung])
        if not self.curveDict[messung] == '':
            faktor = self.skalFak_dict[messung]
            y = [a * faktor for a in self.listDict[messung]]
            self.curveDict[messung].setData(x_value, y)
        # Teil hier nicht gezeigt
```

```
1 def update_GUI(self, value_dict, x_value):
2     # Teil hier nicht gezeigt
3     for kurve in self.kurven_dict:
4         if kurve in self.grenzListDict:
5             self.grenzListDict[kurve].append(float(self.grenzValueDict[kurve]))
6             self.kurven_dict[kurve].setData(x_value, self.grenzListDict[kurve])
```

E.4 Legende

Über die Legende wurde in Kapitel 5.9.5.2 geredet. Zusammenfassend gibt es drei verschiedene Möglichkeiten der Darstellung der Legende. Die Legende kann im Plot, unter dem Plot oder neben dem Plot erscheinen. Die Legende wird an zwei Stellen im Code bearbeitet. Zum einen bei der Erstellung des Plots selbst und zum anderen beim erstellen der Kurven. Diese beiden Stellen sollen im Folgenden gezeigt werden.

Wenn ein Plot erstellt wird, gibt es zumeist auch eine direkte Möglichkeit eine Legende einzufügen. Dies ist auch der Fall für die Legende Innen und Außen. In Anhang E.3 wurde die Plot-Erstellung gezeigt. Je nach dem ob die Plot-Instanz mit *pg.GraphicsLayoutWidget* (Außen) oder *pg.PlotWidget* (Innen, Seite) erstellt wird muss auch die Legende anders erstellt werden. Die Erstellung der Legende kann im Folgenden gesehen werden:

```
1  class PlotWidget(QWidget):
2      def __init__(self, menu, legend_ops, sprache, typ, label_x, label_y1, ...
           label_y2 = '', parent=None):
3          # Teil hier nicht gezeigt
4          if self.legend_pos == 'OUT':
5              self.legend = pg.LegendItem(horSpacing=25, verSpacing=-5)
6              self.graph.addItem(self.legend, row=1, col=0)
7              self.achse_1 = self.graph.addPlot(row=0, col=0)
8          elif self.legend_pos == 'IN':
9              self.legend = self.graph.addLegend()
10             self.achse_1 = self.graph.plotItem
11         else:
12             self.achse_1 = self.graph.plotItem
13
14         if not self.legend_pos == 'SIDE':
15             self.legend.setColumnCount(legend_ops['legend_anz'])
16         # Teil hier nicht gezeigt
```

Hier können nun die drei Legenden-Varianten gesehen werden. Bei der Side-Legende (Zeile 12) wird nur der Plot mit *plotItem* erstellt, da die Legende in den Geräte-Typen-Objekten selbst gebaut wird. Bei der Legende im Plot, wird der Plot auch mit *plotItem* erstellt, jedoch auch die Methode *addLegend* verwendet. Durch die Methode erzeugt die Plot-Instanz direkt eine Legende, an die die Labels nur noch angefügt werden müssen. Bei der Legende unter dem Plot wird eine Art Grid-Layout genutzt. Der Plot und die Legende müssen mit *addItem* (Legende) und *addPlot* (Plot) an die Instanz angefügt werden. Die OUT-Legende war die erste Idee für die Legende außerhalb des Plots. Später hat sich dann aber die Side-Legende, die auch außen ist, durchgesetzt. Die letzten beiden Zeilen (14,15) in dem Code, geben an, wie viele Legenden-Label in einer Zeile der Legende stehen sollen. Dies findet nur bei In und Out Anwendung.

Mit dem gezeigten Code wird somit die Variable *legend* besetzt, die später in den Geräte-Widget-Objekten aufgegriffen wird. Für die Side-Legend muss nun noch etwas spezielles erstellt werden. In den Geräte-Typen-Objekten *Generator* und *Antrieb* wird der Plot und die dazugehörige Legende in der ersten Zeile des Splitter-Widgets erstellt. Dies konnte in Abbildung 60 gesehen werden. Bei der Side-Option werden nicht zwei Widgets in dieser Zeile erstellt, sondern bis zu 4 (minimal 3). Die Varianten mit 3 Widgets sind in Abbildung 90 zu sehen. Der folgende Code erzeugt eines dieser Widgets (identisch in *Generator*- und *Antrieb*-Objekt). Hierbei wird die Klasse *Widget_ VBox* aus **base_ classes.py** aufgerufen. Die Spätere Legende, wird in ein *QVBoxLayout* gesetzt, sodass die Legenden-Label untereinander sind. Durch die Methode *setSpacing* wird der Abstand zwischen den Legenden-Labeln eingestellt. Durch das setzen dieses Wertes auf Null und dem späteren setzen von `check_Layout.setContentsMargins(0,0,0,0)` können die Label dann sehr nah beieinander liegen. Der folgend gezeigte Code wird in zwei Varianten aufgerufen, einmal

für Links und einmal für Rechts. Zwischen diesen beiden If-Anweisungen, wird die Instanz von *PlotWidget* erstellt, die den Plot und die Legende erstellt. Der obere Code stammt aus diesen Objekt.

```
1 class Generator(QWidget, Cursor, PopUpWindow):
2     def __init__(self, start_zeit, tab_splitter_widget, add_Ablauf_function, ...
          stopp_all_function, menu_dict, legend_ops, Faktoren, sprache, ...
          parent=None):
3         # Teil hier nicht gezeigt
4         if legend_ops['legend_pos'].upper() == 'SIDE' and (legend_ops['side'] ...
              == 'rl' or legend_ops['side'] == 'l'):
5             self.legend_achsen_Links_widget = Widget_VBox()
6             self.splitter_row_one.splitter.addWidget( ...
                  self.legend_achsen_Links_widget.widget)
7         # Teil hier nicht gezeigt
```

Nach der Erstellung der Legende und des Plots, müssen die Kurven eingetragen werden. Auf Seite 198 wird die Erstellung der einzelnen Kurven im oberen Code gezeigt. In diesem Code werden aber die Label für die In- und Out-Legende gesetzt. Bei der Legende muss noch ein Unterschied beachtet werden. VIFCON besitzt zwei y-Achsen, die unterschiedlich mit den Legenden-Labeln arbeiten. Bei der Legende im Plot, wird die Legende der Kurve automatisch durch die Erstellung des Plots bei der linken y-Achse erstellt. Im folgenden Code wäre das in Zeile 1 und 2 zu sehen. Durch die Variable *name* wird die Legende erweitert. Bei allen anderen Einstellungen (linke Achse Out, rechte Achse Out, rechte Achse In) wird der Code in Zeile 4 unf 5 ausgeführt.

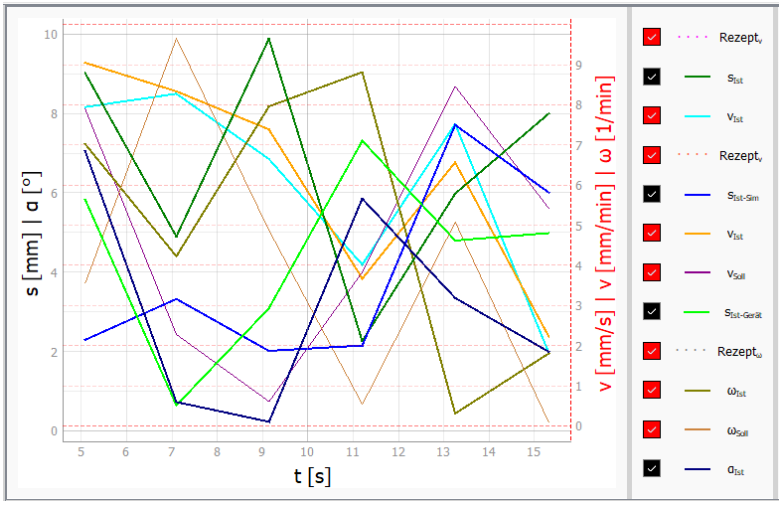
```
1 # Achse Links - IN:
2 curve = self.typ_widget.plot.achse_1.plot([], pen=kurv_dict[legend_kurve][1], ...
      name=kurv_dict[legend_kurve][2])
3
4 # Alle anderen:
5 self.typ_widget.plot.legend.addItem(curve, curve.name())
```

Somit bleibt nur noch die Side-Legende. In dem oberen Code von Seite 198 wird die Side-Legende nicht beachtet. Für diese Legende wird der obere Code auf Seite 203 ausgeführt. Hierbei wird das Dictionary *self.kurven_dict* in einer For-Schleife durchgegangen, wodurch jede Kurve die Funktion *GUI_Legend_Side* aufruft. Diese Funktion bekommt das anzuzeigende Label, den Stift (Format, Farbe) und die Achse übergeben. All diese Daten liegen im Dictionary *kurv_dict*. Eine Besonderheit hier ist, dass der Name nicht als String, sondern als Liste (*split*) übergeben wird. Durch diese Liste wird der Text des Labels gekürzt. Die Funktion *GUI_Legend_Side* gibt darauf hin das Widget (2x*QLabel*, 1x*QCheckbox*) und die Checkbox-Instanz selbst zurück. Im Folgenden wird das Widget an die richtige VBox angefügt. Somit können die Label mit „RL“ an die jeweilige Achse, mit „L“ an die linke Achse und mit „R“ an die rechte Achse gebunden werden (Position der VBox, Config-Datei Angabe). Am Ende werden die Checkbox-Widgets (Instanz von *QCheckBox*) mit der jeweiligen Instanz der Kurve in einem Dictionary gespeichert.

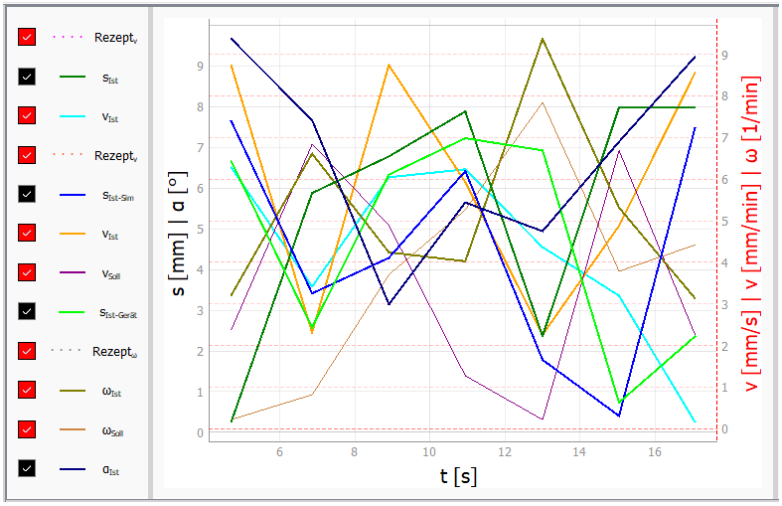
Zum Schluss müssen nun noch die Funktionen *GUI_Legend_Side* und *Side_Legend* erläutert werden. Der Aufruf der ersten Funktion erfolgt im Code von Seite 203. In dieser Funktion wird ein *QWidget* mit *QHBoxLayout* erstellt. Daraufhin werden zwei *QLabel* und eine *QCheckBox* angefertigt und an das *QWidget* angefügt. Die Bezeichnung einer Kurve besteht aus dem Geräte-Namen und der Größe (Übergabe durch Liste). Der erste Eintrag (Geräte-Name) wird den drei Objekten als Tooltip (Abbildung 64c) mit *setToolTip* zugeordnet. Der zweite Teil (Größe) wird als Text angezeigt. Zwischen der Checkbox und der Größe befindet sich die Darstellung der Linie. Hierfür wurde das Dictionary *style* (Seite 203) erstellt, welches Unicode-Zeichen für Punkt und Minus benutzt. Die Dicke der Linie wird bei der Legende nur Simuliert, indem der Strich

dick (Bold) gemacht wird, wenn der Wert dafür im Stift (pen) größer 1 ist. Die Farbe wird auch dem übergebenen Stift entnommen und mit *setStyleSheet* gesetzt. Diese Funktion wird auch für einen weiteren Punkt genutzt. Die Checkboxen sollen auch bei Links und Rechts zu den Achsen zuordenbar sein. Aus dem Grund gibt es die drei Icons **checked.png**, **checked2.png** und **unchecked.png** (siehe Abbildung 98). Das setzen des Styles mit Icons ist auch auf Seite 203 zu finden. Die erstellten Checkboxen werden mit der Funktion *Side_Legend* verbunden. Diese Funktion wird auch auf Seite 204 gezeigt. Der Aufruf der Funktion wird durch die Betätigung der Checkbox ausgelöst. Die Funktion ist mit allen Checkboxen verbunden. Um die richtige Checkbox zu erkennen wird die Methode *sender* genutzt, die jedoch das erben von *QWidget* beim Geräte-Widget-Objekt erwartet. Durch das Dictionary *kurven_side_legend* wird dann die richtige Kurve zur Checkbox gefunden. Die Funktion an sich toggelt nur die Methode *setVisible*.

Die Abbildung 90 zeigt noch die fehlenden Varianten die im Haupttext nicht gezeigt wurden.



a) Legendenoption Side Rechts



b) Legendenoption Side Links

Abbildung 90: Side-Legend Varianten (Quelle: eigene Darstellung)

```python
class PIAchseWidget(QWidget):
    def __init__(self, sprache, Frame_Anzeige, widget, line_color, config, config_dat, start_werte, neustart, ...
        add_Ablauf_function, piAchse, gamepad_aktiv, typ = 'Antrieb', parent=None):
        # Teil hier nicht gezeigt
        ## Checkboxen erstellen:
        self.kurven_side_legend         = {}

        if self.typ_widget.legend_ops['legend_pos'].upper() == 'SIDE':
            for kurve in self.kurven_dict:
                widget, side_checkbox = self.GUI_Legend_Side(kurv_dict[kurve][2].split(' - '), kurv_dict[kurve][1], ...
                    kurv_dict[kurve][0])
                if self.typ_widget.legend_ops['side'].upper() == 'RL':
                    if kurv_dict[kurve][0] == 'a1': self.typ_widget.legend_achsen_Links_widget.layout.addWidget(widget)
                    elif kurv_dict[kurve][0] == 'a2': self.typ_widget.legend_achsen_Rechts_widget.layout.addWidget(widget)
                elif self.typ_widget.legend_ops['side'].upper() == 'R':
                    self.typ_widget.legend_achsen_Rechts_widget.layout.addWidget(widget)
                elif self.typ_widget.legend_ops['side'].upper() == 'L':
                    self.typ_widget.legend_achsen_Links_widget.layout.addWidget(widget)
                self.kurven_side_legend.update({side_checkbox: kurve})
            # Teil hier nicht gezeigt
```

```python
def GUI_Legend_Side(self, text, check_pen, achse):
    style = {1: '\u2501' * 2, 2: '\u2501 \u2501', 3: '\u00B7 \u00B7 ' * 2, 4: '\u2501\u00B7' * 3, 5: '\u2501\u00B7\u00B7' ...
        * 2}
    # 1: Solid, 2: Dashed, 3: Dot, 4: Dash Dot, 5: Dash Dot Dot
    # Teil hier nicht gezeigt
```

```python
def GUI_Legend_Side(self, text, check_pen, achse):
    # Teil hier nicht gezeigt
    check_legend.setStyleSheet("QCheckBox::indicator::unchecked { background-color : darkgray; image : ...
        url('./vifcon/icons/unchecked.png'); }\n"
                               "QCheckBox::indicator::checked { background-color : red; image : ...
                                   url('./vifcon/icons/checked.png'); }\n"
                               "QCheckBox::indicator{ border : 1px solid black;}")
    # Teil hier nicht gezeigt
```

```
1 def Side_Legend(self, state):
2     checkbox = self.sender()
3     kurve = self.kurven_side_legend[checkbox]
4     if checkbox.isChecked():
5         self.curveDict[kurve].setVisible(True)
6     else:
7         self.curveDict[kurve].setVisible(False)
```

E.5 Cursors

Der Cursor ist ein extra Feature, welches in Kapitel 5.9.5.5 erläutert wurde. Der Cursor wird durch die Klasse *Cursor* im Programm **typen.py** gegeben. Hierbei ist anzumerken, dass die beiden Klassen *Generator* und *Antrieb* von dieser Klasse erben. Folgend wird die Klasse gezeigt. Mit der Methode *Add_ Widget* wird das *QLabel* vorbereitet und erstellt. Um die Koordinaten zu erhalten wird eine Verknüpfung mit der Funktion *onMouseMoved* erstellt. Hierbei wird der Plot (der Graph) so mit der Funktion verbunden, dass eine Bewegung diese auslöst. Die Funktion dazu ist *sigMouseMoved*. In der nun ausgelösten Funktion wird nur dann ein Wert ausgelesen, wenn die Maus auch wirklich in dem Plot ist. Mit *mapSceneToView* werden nun die Koordinaten ausgelesen. Da *achse_ 2* eine ViewBox ist, benötigt es hier nicht das „vb“, was auf die Viewbox des Elements zugreift. Zuletzt muss nun noch das *QLabel* mit *setText* aktualisiert werden. Die grundlegende Programmierung des Cursors wurde der Quelle [Fora] entnommen.

```
class Cursor:
    def Add_Widget(self, Zeile):
        self.achs_werte = QLabel(text=" x: 0 \nyL: 0\nyR: 0")
        self.controll_layout.addWidget(self.achs_werte, Zeile, 0, Qt.AlignBottom)
        self.plot.graph.scene().sigMouseMoved.connect(self.onMouseMoved)

    def onMouseMoved(self, evt):
        if self.plot.achse_1.vb.mapSceneToView(evt):
            point_yLx = self.plot.achse_1.vb.mapSceneToView(evt)
            point_yR = self.plot.achse_2.mapSceneToView(evt)
            self.achs_werte.setText(f' x: {round(point_yLx.x(),2)} \nyL: ...
                {round(point_yLx.y(),2)} \nyR: {round(point_yR.y(),2)}')
```

E.6 Einstellung der GUI-Sprache

Die Umsetzung der Sprache ist relativ simpel. Hierbei werden Listen genutzt. In einigen der bereits gezeigten Code-Teilen wurde die Nutzung der Sprachauswahl schon gezeigt. Die Sprache wird durch die Config-Datei gesetzt. Um die Sprache nun zu nutzen wird der folgende Code ausgeführt:

```
class Controller(QObject):
    # Teil hier nicht gezeigt
    def __init__(self, config, output_dir, test_mode, neustart) -> None:
        # Teil hier nicht gezeigt
        if self.config['language'].upper() == 'DE':
            logger.info('Sprache der GUI ist Deutsch!')
            self.sprache = 0
        elif self.config['language'].upper() == 'EN':
            logger.info('The language of the GUI is English!')
            self.sprache = 1
        else:
            logger.warning('The language is not defined. Only German and ...
                English are defined in this program. Set language to English!')
            print('Warning: The language is not defined. Only German and ...
                English are defined in this program. Set language to English!')
            self.sprache = 1
        # Teil hier nicht gezeigt
```

Je nachdem, was in der Config-Datei festgelegt wird, wird entweder Deutsch (DE, 0) oder Englisch (EN, 1) ausgewählt. Als Default-Wert wird Englisch genommen. Die Variable *sprache* bekommt hier einen Integer-Wert, da es sich wie gesagt um Listen handelt. Um an ein Element in einer Liste zu kommen, muss der Platz des Elements angegeben werden. Hierbei wird von dem Integer Null aufwärts gezählt. Die Definition der Liste kann folgend gesehen werden:

```
class Controller(QObject):
    # Teil hier nicht gezeigt
    def __init__(self, config, output_dir, test_mode, neustart) -> None:
        # Teil hier nicht gezeigt
        main_window_tab_1_str   = ['Steuerung',    'Control']
        main_window_tab_2_str   = ['Überwachung', 'Monitoring']
        # Teil hier nicht gezeigt
```

Der Aufruf eines der gezeigten Beispiele sieht dann wie folgt aus:

```
class Controller(QObject):
    # Teil hier nicht gezeigt
    def __init__(self, config, output_dir, test_mode, neustart) -> None:
        # Teil hier nicht gezeigt
        self.main_window.add_tab(self.tab_GenAnt.splitter, ...
            main_window_tab_1_str[self.sprache])
        # Teil hier nicht gezeigt
```

Der Aufruf und die Definition der Listen sind in jedem Programm zu finden. Wenn eine neue Sprache gewollt wird, muss die Liste nur erweitert und eine neue Elif-Anweisung angelegt werden.

E.7 Pop-Up-Fenster

Die Pop-Up-Fenster wurden in Kapitel 5.9.4 erläutert. In VIFCON gibt es zwei verschiedene Pop-Up-Fenster, die zum einem für das Beenden und zum anderen für Infos oder Warnungen sind. Das Fenster für das Beenden von VIFCON ist in **main_window.py** zu finden. Der Code dafür wird folgend gezeigt. Für die Fenster wird das Objekt *QMessageBox* verwendet. In **PyQt5** gibt es auch interne Events. In dem Fall bedeutet dies, dass die Funktion *closeEvent* von keiner Funktion direkt aufgerufen wird. Sobald das Beenden-Event ausgelöst wird, wird die Funktion ausgelöst. In Abbildung 91 ist das Fenster zu sehen. Wenn dort nun Ja bzw. Yes betätigt wird, wird die Funktion **exit_function** ausgeführt. Diese Funktion wird an das *MainWindow*-Objekt übergeben. Die Funktion heißt wirklich *exit* und ist im **vifcon_controller.py** zu finden.

```
def closeEvent(self, event):
    reply = QMessageBox.question(self, self.exit_1_str[self.sprache], ...
        self.exit_2_str[self.sprache],
                                     QMessageBox.Yes | QMessageBox.No , ...
                                         QMessageBox.No)

    # Reaktion auf Antwort der Box:
    if reply == QMessageBox.Yes:
        event.accept()
        self.exit_function()
    else:
        event.ignore()
```

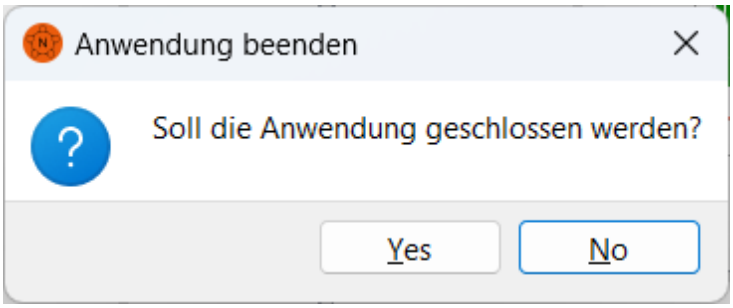


Abbildung 91: Pop-Up-Fenster - VIFCON Ende (Quelle: eigene Darstellung)

Die anderen Pop-Up-Fenster sind etwas anders. Wie bei der Ablauf-Datei, sollten die Pop-Up-Fenster schnell und einfach einfügbar sein. Aus dem Grund wurde in **typen.py** die Klasse *PopUpWindow* erstellt. Die beiden Geräte-Typen-Klassen erben von dieser Klasse. Somit kann die Funktion *Message* aus allen Geräte-Widget-Objekten aufgerufen werden. Die Funktion sieht wie folgt aus:

```
1  def Message(self, Zeile, art, width = 400):
2      artStr_dict     = {1: self.PopUp_1[self.sprache], 2 : ...
           self.PopUp_2[self.sprache], 3 : self.PopUp_3[self.sprache], 4 : ...
           self.PopUp_4[self.sprache]}
3      artIcon_dict    = {1 : QMessageBox.Question, 2 : QMessageBox.Information, ...
           3 : QMessageBox.Warning, 4 : QMessageBox.Critical}
4
5      mess = QMessageBox(self.tab)
6      mess.setIcon(artIcon_dict[art])
7      mess.setWindowTitle(artStr_dict[art])
8      breite = '{min-width: ' + str(width) + 'px;}'
9      mess.setStyleSheet(f"QLabel{breite}")
10     mess.setText(Zeile)
11     mess.setStandardButtons(QMessageBox.Ok)
12     mess.setModal(False)
13     mess.show()
```

Die Funktion *Message* erhält die Variablen *Zeile* (Text), *art* (Icon) und *width* (Breite des Fensters). Je nach dem wie die *QMessageBox* aufgerufen wird, kann das Icon in der Box verändert werden. An Abbildung 91 kann gesehen werden, das es sich um eine Frage (Question) handelt. Aus dem Grund wurden die beiden Dictionaries *artStr_dict* und *artIcon_dict* erzeugt. Somit kann der Nutzer vier verschiedene Boxen erzeugen. Die Boxen sollen auf Probleme hinweisen und den Nutzer warnen, weshalb derzeitig Warnung am meisten genutzt wird. Sehr wichtig ist die Funktion *setModal*, da diese mit Setzung auf False verhindert, dass der Haupt-Loop angehalten wird. Wenn ein Fenster aufgeht, läuft das Programm weiter. Die Erzeugung eines solchen Pop-Up-Fenster sieht wie folgt aus:

```
1  self.typ_widget.Message(self.puF_HO_str_2[self.sprache], 3, 450)
```

Die Abbildungen 92 bis 95 zeigen die vorhandenen Pop-Up-Fenster in VIFCON. Bei Abbildung 95 kann ein kleineres Problem gesehen werden. Manchmal ist das Verhalten von **PyQt** rätselhaft. Als Default-Wert gibt es als Breite 400 Pixel, jedoch wird kein automatischer Zeilenumsprung ausgeführt, was bei den anderen getan wird.

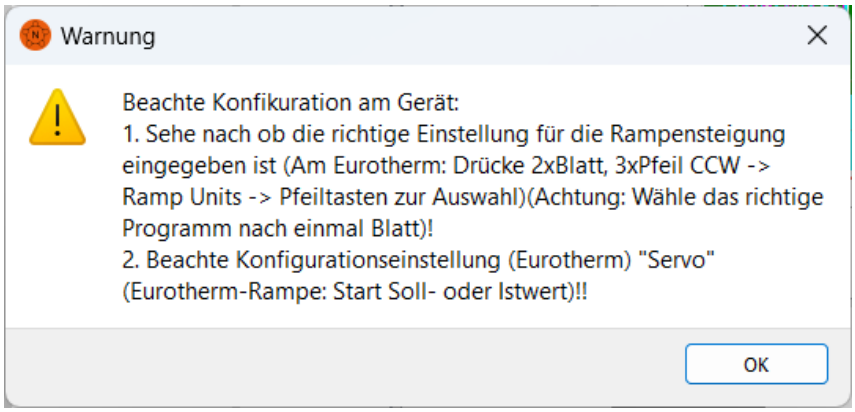


Abbildung 92: Pop-Up-Fenster - Eurotherm Rezept (Quelle: eigene Darstellung)

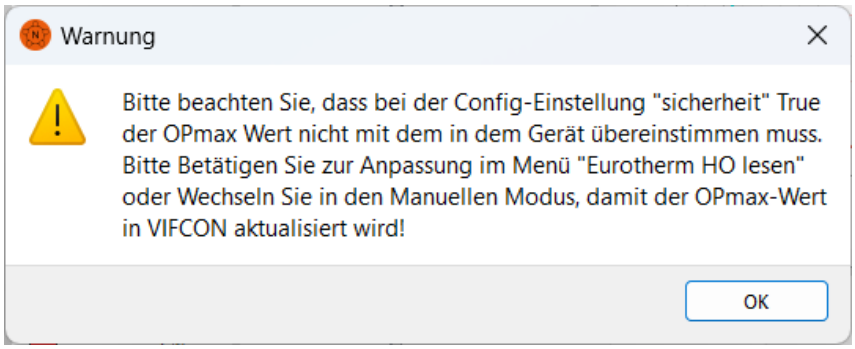


Abbildung 93: Pop-Up-Fenster - Eurotherm Sicherheit True 1 (Quelle: eigene Darstellung)

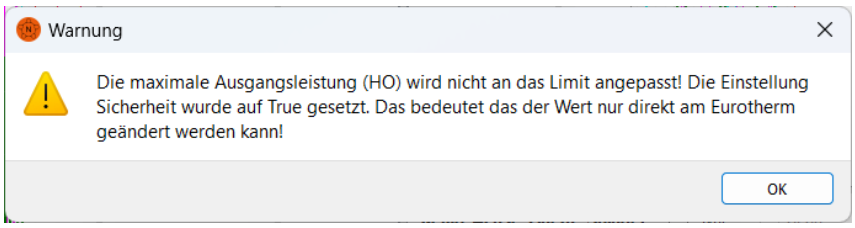


Abbildung 94: Pop-Up-Fenster - Eurotherm Sicherheit True 2 (Quelle: eigene Darstellung)

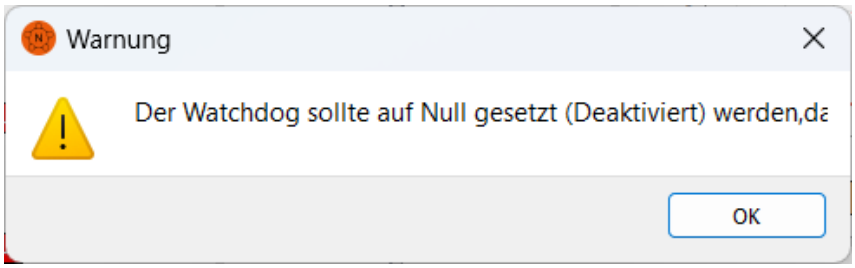


Abbildung 95: Pop-Up-Fenster - TruHeat Watchdog (Quelle: eigene Darstellung)

Als Anmerkung zu Abbildung 92 soll noch gesagt werden, dass die Begriff „Blatt“ und „Pfeil CCW“ auf die Icons der Knöpfe des Eurotherm-Reglers (Abbildung 17) hinweisen sollen!

E.8 Platzierung von Widgets

Um die Platzierung und den Aufbau der GUI mit PyQt zu zeigen, wurde die Abbildung 96 erstellt. PyQt kann sich wie ein großes Bausteinsystem vorgestellt werden. Auf die unterste Ebene wird eine neue Ebene platziert, auf die dann Widgets gelegt werden. Unter Widgets werden die einzelnen Teile verstanden, die auf der GUI platziert werden können. Die Abbildung zeigt fast alle Teile der VIFCON GUI. Bei der Menü-Leite und den Geräte-Widgets wurde es etwas kürzer gefasst. Die Pfeile sind wie folgt zu lesen: *Das* ***Central Widget*** *vom Objekt QWidget mit dem Layout QVBoxLayout wird hinzugefügt mit* ***setCentralWidget*** *zu* ***MainWindow*** *vom Objekt QMainWindow.* So werden alle Teile Stück für Stück zusammengesetzt. Das unterste Layer ist das des Main Windows. Die obersten sind die einzelnen Widgets, die an den Geräte-Widgets hängen. Bei den rosafarbenen Teilen wird eine Reihenfolge gezeigt. Bei den Splittern (*QSplitter*), werden die Teile von rechts oder von unten angefügt. Dasselbe gilt für die Layouts *QVBoxLayout* und *QHBoxLayout*. Weiterhin muss klar sein, dass durch die Config-Einstellungen auch Teile nicht immer angefügt werden müssen. Zum Beispiel ist die Legende an der Seite nur bei Auswahl da. Auch der Plot wird aus vielen Teilen zusammengefasst. Das Grundgerüst wird z.B. in **base_classes.py** erstellt, während die Kurven erst durch die Geräte-Widget Objekte erstellt werden. Auch hier kann die Auswahl der Legende andere Teile für die Achsen auslösen.

Die Platzierung funktioniert somit über ein Objekt und ein Layout. Das Objekt ist dabei meist ein *QWidget*, das mit einen *QGridLayout*, *QVBoxLayout* oder *QHBoxLayout* belegt wird. An diese Layouts werden dann alle anderen Widgets angebunden, welche auch wieder ein Layout besitzen können. *QSplitter* ist in dem Fall ein spezielles Layout, da es wie ein Objekt verwendet werden kann und einfach an ein anderes Widget angeheftet wird. Besonders sind hierbei noch *QAction* (Menü-Leiste), *QTabWidget* und *QScrollArea*. Bei den letzten beiden wird nur die Tab-Leiste eingerichtet.

In Abbildung 96 wird das Menü auch auf der linken oberen Seite gezeigt. Eine ausgeklappte Version wird in Abbildung 112 oder Kapitel 5.9.3 gezeigt. Die Menü-Leiste wird durch die Funktion *menuBar* direkt an das Main Windows angefügt. Mit *addMenu* können dann immer mehr Ebenen angefügt werden. Dabei wird diese Funktion immer auf die gewollte darüberliegende Ebene ausgeführt. Um eine Funktion an den Knopf zu binden wird das Objekt *QAction* verwendet. Dadurch erhält die oberste Ebene des Menüs eine Funktion, die bei Betätigung dieser ausgeführt wird.

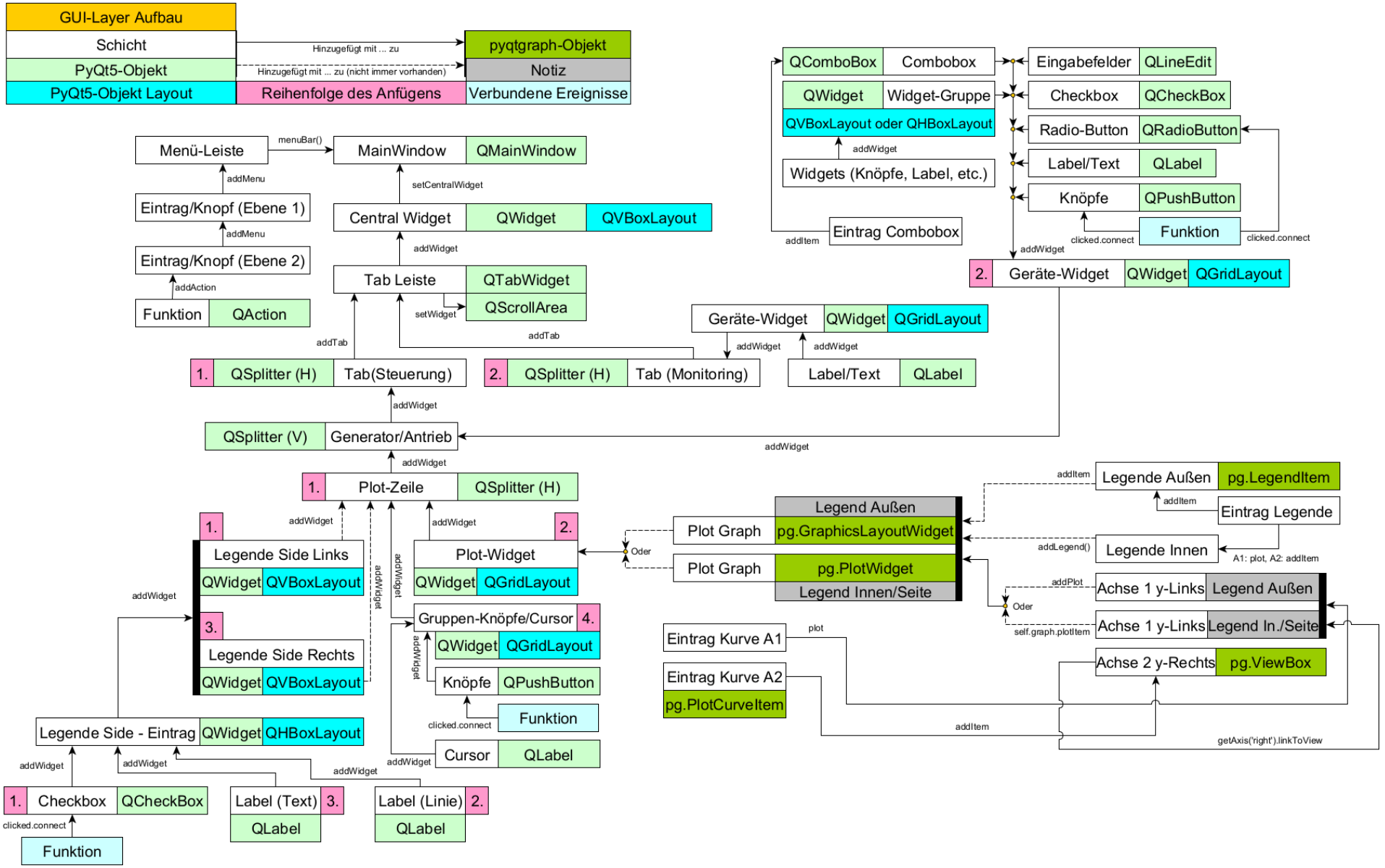


Abbildung 96: VIFCON GUI - Bausteinsystem (Quelle: eigene Darstellung)

E.9 GUI-Icons

In Kapitel A wurde die Ordner-Struktur und der Inhalt von VIFCON gezeigt. Einer dieser Ordner heißt „icons“. In diesem befinden sich die verschiedenen Icons die in der GUI von VIFCON angezeigt werden. Im Folgenden werden diese gezeigt und kurz gesagt wozu diese dienen.

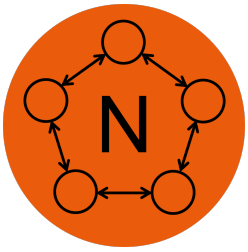


Abbildung 97: Programm-Icon - Nemocrys Logo (nemocrys.png) (Quelle: [Perc])

In den meisten Programmen wird in der oberen linken Ecke des Bildschirms, das Logo des Programms und der Name angezeigt. Auch bei Multilog [Mul] wird das in Abbildung 97 angezeigt. Das Projekt, das von der Modellexperimente-Gruppe durchgeführt wird, trägt den Namen Nemocrys. Das Logo zeigt die Abbildung. Die folgenden Abbildungen werden an verschiedenen Stellen in der GUI genutzt und wurden mit dem Programm **Paint** von Microsoft Windows erstellt.

Das Icon aus Abbildung 97 wird mit dem folgenden Code gesetzt:

```
1 self.setWindowIcon(QIcon("./vifcon/icons/nemocrys.png"))
```

Bei den Icon wird das Objekt *QIcon* verwendet, welches den Pfad des Bildes bekommt. *QIcon* gehört zu der **PyQt5** Bibliothek. Auch die anderen Icons werden durch *QIcon* erstellt und dann an andere Objekte übergeben. So kann dies z.B. wie folgt aussehen:

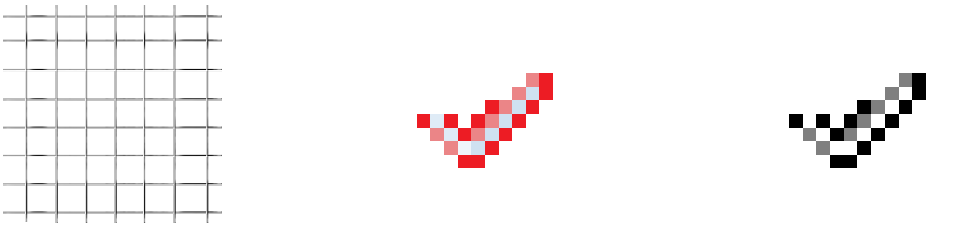
```
1 QPushButton(QIcon(self.icon_1), '')
```

Neben *QPushButton* wird dies auch bei *QAction* (Menü-Knöpfe) genutzt. Die folgenden Icons werden somit für Knöpfe genutzt. Die einzige Ausnahme bilden die Abbildungen 98c und 98b, die über die Funktion *setStyleSheet* des Objektes *QCheckBox* gesetzt werden.

Bei den Icons werden noch die Funktionen:

1. setIconSize (Beispiel: `self.btn_Aus.setIconSize(QSize(50, 50))`),
2. setFlat (Beispiel: `self.btn_Aus.setFlat(True)`) und,
3. setIcon (Beispiel: `self.btn_left.setIcon(QIcon(self.icon_1))`) genutzt.

Die Funktion *setIcon* wird dazu genutzt die Bilder der Knöpfe (Widget, Menü) zu verändern bzw. zu wechseln. Z.B. wird die Abbildung 98a immer mit **p_nichts.png** getoggelt. Die Icons **p_nichts.png** und **unchecked.png** sind unter den Abbildungen nicht zu finden, da diese einfach nur Transparente Bilder sind. Die Gegenstücke sind in Abbildung 98 zu sehen. Bei *setFlat(True)* wird die Umrandung des Knopfes nur dann angezeigt, wenn darauf gedrückt wird. Die letzte Funktion, *setIconSize*, lässt den Nutzer die Geometrie des Bildes einstellen.



a) grid.png **b)** checked.png **c)** checked2.png

Abbildung 98: Plot-Icons (Quelle: eigene Darstellung)

Die Icons aus Abbildung 98 sind für den Plot relevant. Das Icon aus Abbildung 98a kann in der Menü-Leiste (Plot/Gitter) gefunden werden. Der Menü-Knopf toggelt das Gitter im Plot an und aus. Wenn im Plot kein Gitter angezeigt wird, so wird das Icon im Menü angezeigt. Der Nutzer soll somit gesagt bekommen, dass die Betätigung das Gitter einschaltet. Im anderen Fall wird das Leere PNG **p_nichts.png** angezeigt. Bei den Icons aus Abbildung 98b und 98c ist es ähnlich. Auch diese beiden Bilder sind weitestgehend transparent und wechseln sich mit **unchecked.png** ab. Genutzt werden diese beiden für die Checkboxen. In VIFCON ist die rechte Achse rot und die linke schwarz. Aus dem Grund wurden die Häkchen erstellt. Der Hintergrund (in VIFCON eingestellt) hat die selbe Farbe. Das Icon **unchecked.png** und die Häkchen haben die selbe Größe. Dies muss so sein, da sonst Veränderung an der Größe auffallen würden.

a) p_Init_Okay.png **b)** p_Init_nicht_Okay.png

Abbildung 99: Init-Icons (Quelle: eigene Darstellung)

Auch die beiden Icons in Abbildung 99 werden in dem Menü angezeigt. Jedes Gerät erhält so ein Icon. Ist Abbildung 99b aktiv, dann weiß der Nutzer, dass das Gerät nicht initialisiert wurde und die Kommunikation noch nicht wirkt. Die Abbildung 99a zeigt diesbezüglich das Gegenteil. Mit dem Gerät kann aktiv kommuniziert werden.

a) p_stopp.png

b) p_stopp_En.png

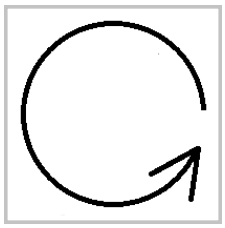
c) p_stopp_all.png

d) p_stopp_all_en.png

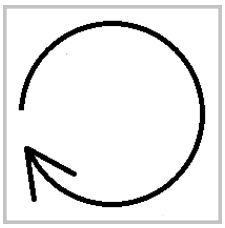
Abbildung 100: Button-Icons 1 - Stopp (Quelle: eigene Darstellung)

Bei den Abbildungen 100 bis 104 handelt es sich um die verschiedenen Knöpfe auf den Geräten bzw. deren VIFCON-Widgets. Die Abbildung 100 zeigt die drei Stopp-Knöpfe die in VIFCON genutzt werden. Abbildung 100a wird dabei nur bei der PI-Achse und den beiden Nemo-Achsen angezeigt. Die Abbildung 100c zeigt den übergeordneten Knopf, der sowohl alle Antriebe als auch alle Generatoren (und Regler) beendet. Jeder Geräte-Tyy (Antrieb, Generator) hat einen. Da es eine Sprach-Auswahl gibt, gibt es die beiden Knöpfe auch in Englisch (Abbildung 100b und 100d).

a) p_start.png



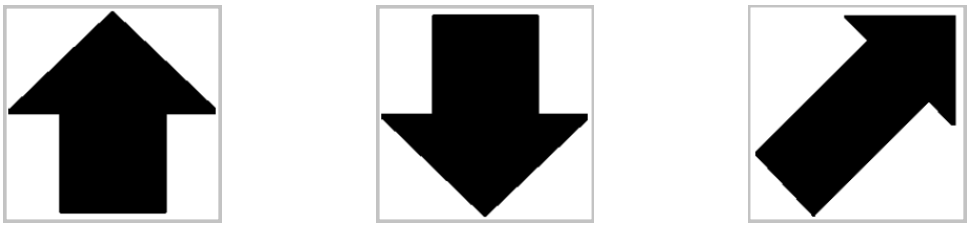
b) p_ccw.png



c) p_cw.png

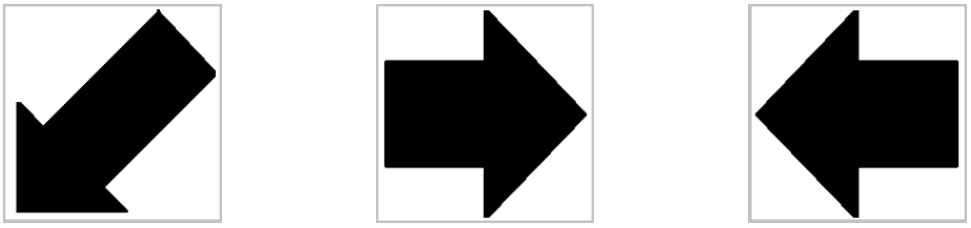
Abbildung 101: Button-Icons 2 - Bewegung 1 (Quelle: eigene Darstellung)

Die Abbildung 101 zeigt die zweite Gruppe der Knöpfe. Dabei sind Abbildung 101b und 101c nur bei dem Nemo-Rotations-Antrieb zu finden und Abbildung 101a nur bei der PI-Achse. Der Knopf für die PI-Achse zeigt den Knopf für die Bewegung zur einer Absoluten Position. Auch hier wird dann der Rechte Bewegungsknopf (bei Wechsel zur Absoluten-Bewegung) mit **p_nichts.png** getoggelt. Bei den Nemo-Knöpfen wird die Bewegungsrichtung angezeigt.



a) p_hoch.png **b)** p_runter.png **c)** p_rein.png

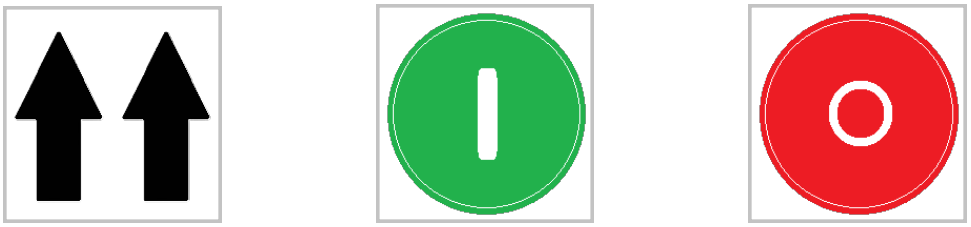
Abbildung 102: Button-Icons 3 - Bewegung 2 (Quelle: eigene Darstellung)



a) p_raus.png **b)** p_rechts.png **c)** p_links.png

Abbildung 103: Button-Icons 4 - Bewegung 3 (Quelle: eigene Darstellung)

Die Abbildungen 102 und 103 zeigen dabei die restlichen Bewegungen. Für den Nemo-Hub-Antrieb werden dabei nur die Abbildungen 102a und 102b genutzt. Die anderen, als auch die, finden bei der PI-Achse ihre Verwendung.



a) p_synchro.png **b)** p_TH_Aus.png **c)** p_TH_Ein.png

Abbildung 104: Button-Icons 5 (Quelle: eigene Darstellung)

Bei Abbildung 104 handelt es sich nochmal um spezieller Knöpfe. Die Abbildung 104a wird auf der Antriebs-Seite genutzt und ist ein übergeordneter. Mit diesem lassen sich ausgewählte Antriebe synchron starten. Die Icons aus Abbildung 104b und 104c sind Teil des Geräte-Widgets des TruHeat-Generators. Beim Design wurde sich an dem Panel (Benutzeroberfläche) des TruHeat-Generators (siehe Abbildung 26) orientiert.

E.10 Geräte-Widgets

In dem Anhang-Kapitel werden die einzelnen Geräte-Widgets gezeigt und beschrieben. Das erste Widget ist das des Eurotherms. Dies ist in Abbildung 105 zu sehen. Bei Eurotherm werden die Ausgangsleistung (Op) und die Temperatur verwendet. Neben dem Senden dieser beiden Werte, werden auch die Istwerte daneben angezeigt. Bei der Temperatur wird neben dem Eingabefeld der Temperatur noch der aktuelle Sollwert mit angezeigt, sollte der z.B. direkt am Eurotherm oder durch ein Rezept geändert werden. Neben diesen Teilen wird die Fehlermeldung unter den Eingabefeldern, neben dem Senden-Knopf angezeigt. Sonst gibt es noch die Rezept-Knöpfe, die Rezept-Auswahl-Combobox und die Checkbox für den Synchronen Rezeptstart.

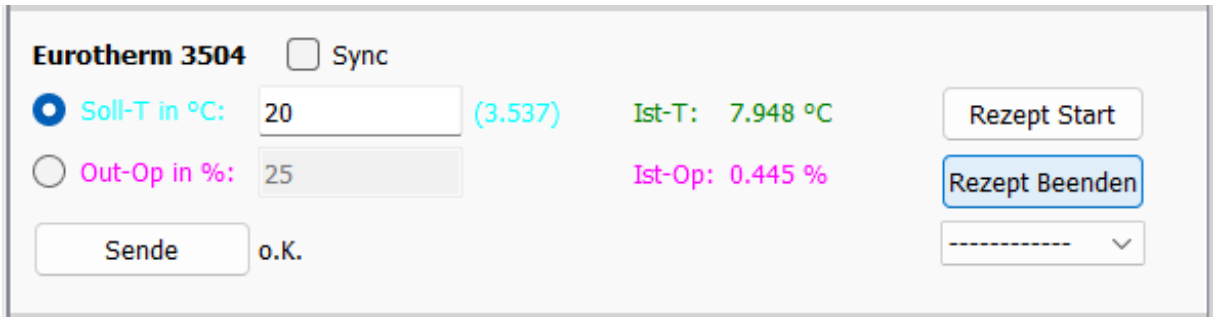


Abbildung 105: Geräte-Widget: Eurotherm (Quelle: eigene Darstellung)

Das zweite Widget, aus Abbildung 106 gehört zu dem TruHeat Generator. Der Aufbau ähnelt dem des Eurotherms. Diesmal gibt es nur mehr Soll- und Istwerte. Durch die Radio-Buttons ist immer nur eine Größe sendbar. Auch bei Eurotherm ist dies so. Zusätzlich hat TruHeat nun noch einen Generator An und Aus Schalter.

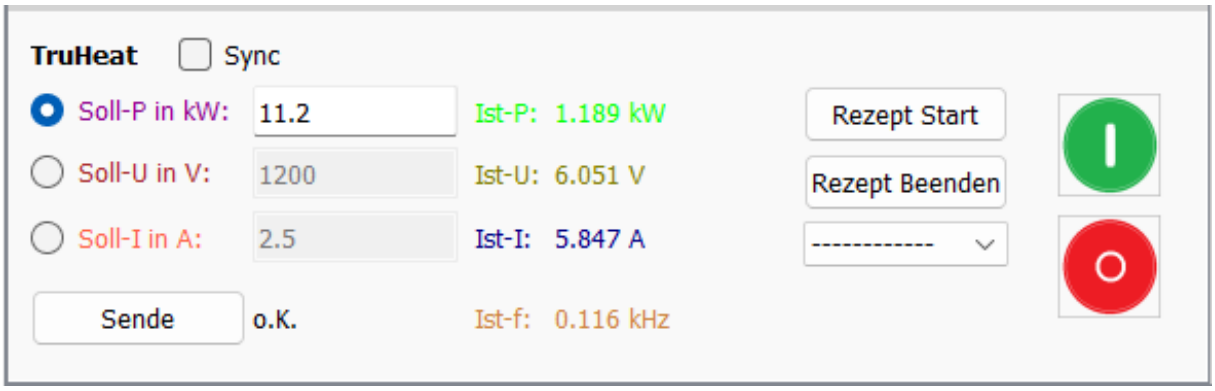


Abbildung 106: Geräte-Widget: TruHeat (Quelle: eigene Darstellung)

Bei der PI-Achse gibt es nun ein paar Änderungen. Hier können nun zwei Werte gleichzeitig gesendet werden, wodurch es auch zwei separate Fehlermeldungen, neben den Eingabefeldern gibt. Bei den Rezept-Knöpfen kommt nun noch ein Define-Home hinzu. Auch die Checkbox für das Synchrone kann neben der Geräte-Bezeichnung gesehen werden. Auch hier dient diese Checkbox dem synchronen Start von Geräten. Weiterhin gibt es noch eine Checkbox für die Nutzung eines Gamepads. Das Widget kann sich auf zwei verschiedene Arten ändern. Einmal durch die Config und zum anderen durch die beiden Radiobuttons. Die Wirkung der Radiobuttons sind in Abbildung 107 und 108 zu sehen. Durch die Angabe der Richtung in der Config werden die Pfeile in Abbildung 107 geändert. So können die Icons aus Abbildung 102 und 103 hier genutzt werden.

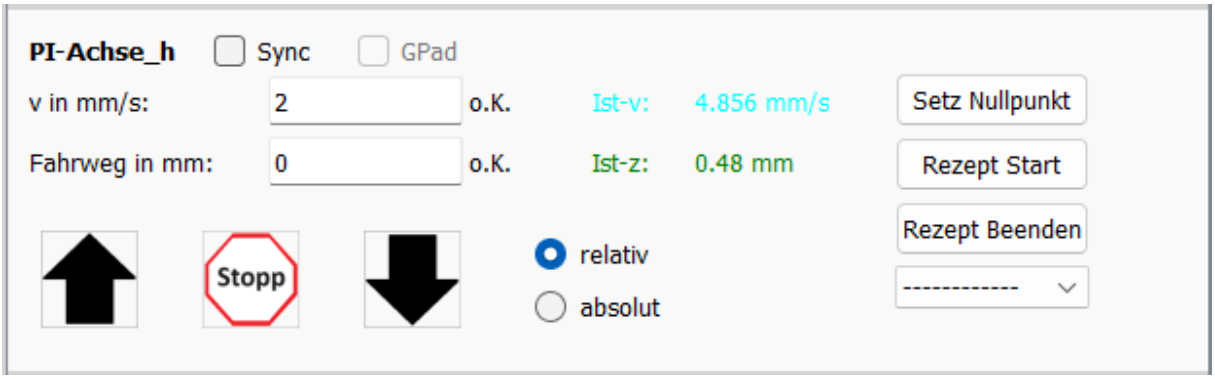


Abbildung 107: Geräte-Widget: PI-Achse Relative Bewegung (Quelle: eigene Darstellung)

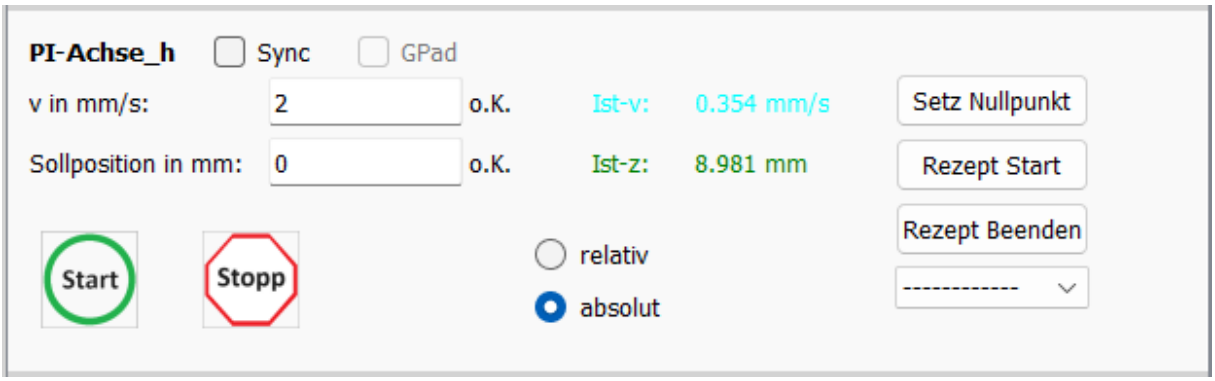


Abbildung 108: Geräte-Widget: PI-Achse Absolute Bewegung (Quelle: eigene Darstellung)

Die Abbildung 109 zeigt das Widget für die Hub-Bewegung der Achsen in der Nemo-1-Anlage. Der Aufbau ähnelt dem der PI-Achse, nur das keine Position gesendet wird und es keine Radio-Buttons braucht. Als Zusatz wird hier unter den Knöpfen der Status der Achse angezeigt. Die Statusmeldungen wurden in Tabelle 18 gezeigt. Auch wird hier ein Unterschied zwischen Simulierten Weg und ausgelesener Weg angezeigt.

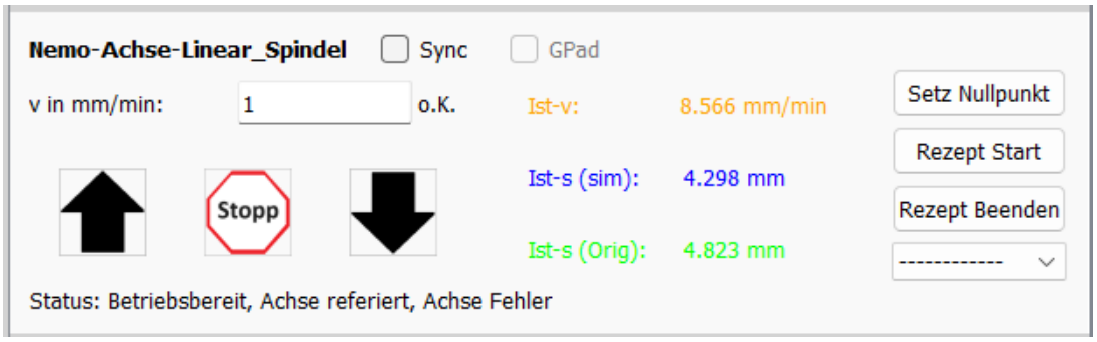


Abbildung 109: Geräte-Widget: Nemo-1-Anlage Hub-Antrieb (Quelle: eigene Darstellung)

Das Widget in Abbildung 110 ist nun die Rotations-Bewegung der Nemo-1-Anlage zu zuordnen. Der Aufbau ist dabei wie beim Hub-Antrieb der Nemo-1-Anlage. Der Unterschied ist hier, das es keinen wahren Winkel gibt und die neue Checkbox „Kont. Rot“, welche eine kontinuierliche Rotation der Achse bewirkt. Bei den Antrieben erfolgt das Senden der Werte durch die Betätigung der Knöpfe.

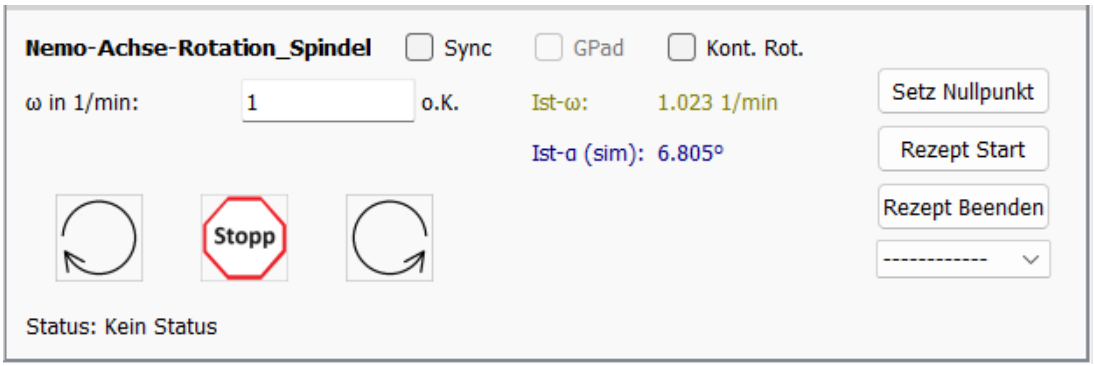


Abbildung 110: Geräte-Widget: Nemo-1-Anlage Rotation-Antrieb (Quelle: eigene Darstellung)

Als letztes gibt es noch die Abbildung 111, welches das Widget für die Nemo-Gase zeigt. In diesem werden nur die Daten aus Tabelle 14 für die Pumpen gezeigt.

Abbildung 111: Geräte-Widget: Nemo-1-Anlage Gase (Quelle: eigene Darstellung)

F Anhang: VIFCON Rezepte

Der Teil des Anhangs soll weitere Punkte zu den Rezepten erläutern. So werden zum einen die Funktionen (Anhang F.1) erläutert und zum anderen wird die Programmierung dieser Funktionen (Anhang F.2) gezeigt.

Um die ausgelagerten Rezepte nutzen zu können, muss der Ordner **rezepte** im Ordner **vifcon** existieren. In dem Ordner befinden sich die Datei **Note.txt** und ein Beispiel Bild (**Beispiel.png**). Die Text-Datei beschreibt die einzelnen Segmente und die Konfiguration dieser.

F.1 Funktionen und Möglichkeiten der Rezept-Funktion

In dem Kapitel sollen die einzelnen Funktionen für die Rezept-Umsetzung erläutert werden. Einige Teile der Funktionen wurde bereits in Abbildung 49 und 50 gezeigt. Die Abbildung 112 zeigt den Aufbau in VIFCON speziell für die Rezepte. Hierbei wird noch ein wichtiger Punkt in VIFCON deutlich. Da VIFCON sowohl für jedes Gerät das „selbe“ Sampler-Objekt, als auch übergeordnete Methoden durch den Controller verwendet, müssen die Methoden/Funktionen in den Geräte-Objekten (Widget, Schnittstelle) den selben Namen besitzen. Die einzelnen Inhalte dieser Methoden können sich aber von Gerät zu Gerät unterschieden. Somit können spätere neue Geräte relativ leicht hinzugefügt werden, da die Struktur der einzelnen Programme nahezu identisch ist. Dementsprechend wurden auch bei dem Rezepten bestimmte Funktionen immer benutzt. Diese Funktionen werden im Folgenden kurz beschrieben.

RezStart:
Bei der *RezStart*-Funktion wird das Rezept gestartet. Aus Abbildung 112 kann gesehen werden, dass diese Funktion zum einen durch das Menü und zum anderen durch das Geräte-Widget gestartet werden kann. Sobald das Rezept gestartet wird, werden die Funktionen *RezKurveAnzeige* und *Rezept_ lesen_ controll* ausgeführt. Beide Funktionen werden noch im Folgenden kurz erläutert. Die Start-Funktion bereitet das Rezept vor und löst die Anzeige im Plot aus. Wenn Fehler auftreten, wie z.B. keine Rezept-Auswahl, dann wird das Rezept nicht gestartet und es gibt eine Meldung im Widget und der Log-Datei. Wenn alles in Ordnung ist, wird hier nun der erste Rezeptschritt über die Dictionaries *write_ task* und *write_ value* gesetzt und der Rezept-Timer *self.RezTimer* gestartet. Als Zeit bekommt diese die Segmentzeit.

RezEnde und Stopp:
Für das Beenden des Rezeptes gibt es ein Verbund von zwei Funktionen. Wichtig hierbei ist zu wissen, dass *Stopp* immer *RezEnde* auslöst, egal um welches Gerät es sich handelt. Dies ist wichtig, da es auch übergeordnete Stopps gibt. Diese müssen die Anlage in einen sicheren Zustand versetzen. Auch die Geräte spezifischen Stopp-Knöpfe müssen die Geräte sicher anhalten. Diese Art des Abschaltens (einzelner Stopp) existiert nur bei den Antrieben, weshalb hier nun spezielle Verbindungen mit den Methoden der Widgets und den Controller-Methoden entstehen. In Abbildung 112 kann dies gesehen werden. Wenn „Rezept Beenden“ oder „Beende Synchron Modus“ (Funktion: *sync_ end_ rezept*) betätigt werden, dann wird bei den Nemo-Achsen (Rotation, Hub) *Stopp* und bei den anderen Geräten direkt *RezEnde* ausgeführt. In den Geräte-Widgets kann die Auswahl der Funktion einfach mit *connect* geregelt werden. Bei den Menü-Controller-Funktionen muss eine If-Anweisung genutzt werden. Eine ähnliche Situation entsteht durch die Funktion *Rezept*. Hier wird bei den Achsen bzw. Antrieben (PI, Nemo) die Funktion *Stopp* ausgeführt, damit sie sicher anhalten und bei den anderen beiden Geräten (Eurotherm, TruHeat) wird *RezEnde* ausgeführt. Zusammenfassend regeln diese beiden Funktionen das Ende des Rezeptes. Somit wird auch der Timer abgestellt.

update_rezept:
Bei der Funktion *update_ rezept* handelt es sich um die dritte und letzte Menü-Funktion. Hierbei wird die Combobox des Geräte-Widgets aktualisiert. Anders als bei den anderen beiden Synchro-Menü-Knöpfen wird hier nicht auf die Checkbox „Sync“ der Geräte geachtet. Die einzige Beachtung wird einem laufenden Rezept gewidmet. In dem Fall wird die Aktualisierung nicht ausgeführt. Die Funktion *update_ rezept* wird durch die Controller-Methode *rezept_ einlesen* aufgerufen.

Rezept:
Bei *Rezept* handelt es sich um die Funktion, die mit dem Timer verbunden ist. Diese sorgt für die Timerzeit, die durch die Segmentzeit der Rezepte bestimmt wird. Mit jedem Schritt wird diese einerseits angepasst und andererseits werden die Sollwerte oder Werte an das Schnittstellen-Objekt übergeben, damit die Geräte ihre Befehle bekommen. Somit werden auch hier die Dictionaries *write_ task* und *write_ value* gesetzt.

RezKurveAnzeige:
Die Funktion *RezKurveAnzeige* wird auch wieder durch zwei Stellen aufgerufen. Zum einen durch *RezStart* und zum anderen durch den Wechsel des Textes in der Combobox. Die Aufgabe der Funktion ist die Erstellung der Rezept-Kurve im Plot. Anhand der Daten aus dem Rezept und der Bearbeitung durch *Rezept_ lesen_ controll* wird diese Kurve erstellt. Aus der genannten Funktion erhält diese auch die Variable *error* (Rückgabewert), durch die ein Fehler angezeigt werden kann. In dem Fall eines Fehlers wird in der Funktion einiges übersprungen und der Wert wird an *RezStart* weitergegeben, wenn der Aufruf von dort kam.

Rezept_lesen_controll:
Rezept_lesen_controll ist die letzte der sechs Funktionen. Wie der Name es schon erwähnt, wird hier das Rezept kontrolliert. So wird z.B. geschaut ob die Limits vom Rezept eingehalten werden. Bei der PI-Achse wird sogar der Weg berechnet, damit diese Achse nicht über das Limit hinausfährt. Neben diesem Umstand werden hier die Segmente des Rezepts angesehen und bearbeitet. In der Funktion werden viele Listen vorbereitet, mit denen dann die Kurve erstellt wird und die folge Schritte gesetzt werden können. Hier wird z.B. das Segment r (Rampe) so umgesetzt, dass die einzelnen Sprünge im Zeitabstand und Höhe berechnet werden.

Insgesamt besitzt VIFCON somit drei Menü-Funktionen in der Menü-Leiste, drei Knöpfe auf dem Geräte-Widgets und in jedem Geräte-Widget-Objekt 6 Funktionen für die Rezepte. Die Verbindungen sind in Abbildung 112 zu sehen. In der Abbildungen sind alle 11 Funktionen wiederzufinden. Wenn an der Verbindung das Wort „Betätigt“ steht, dann zeigt dies zusätzlich noch dass genau diese Funktion mit dem Widget verbunden ist. In der unteren linken Ecke der Abbildung wird ein allgemeiner Ablauf für die Rezepte im Sinne des Geräte-Widget-Objektes gezeigt. In den Objekten werden alle Widgets für das Gerät platziert und die Funktionen zugeordnet. Am Ende der *__init__*-Funktion wird dann der Timer definiert und die Combobox mit einer Funktion versehen. Im Anhang F wird näher auf die Programmierung und somit den Inhalt der gezeigten Methoden eingegangen. Im nachfolgenden wird nun noch auf die Notwendigkeit einiger Funktionen eingegangen.

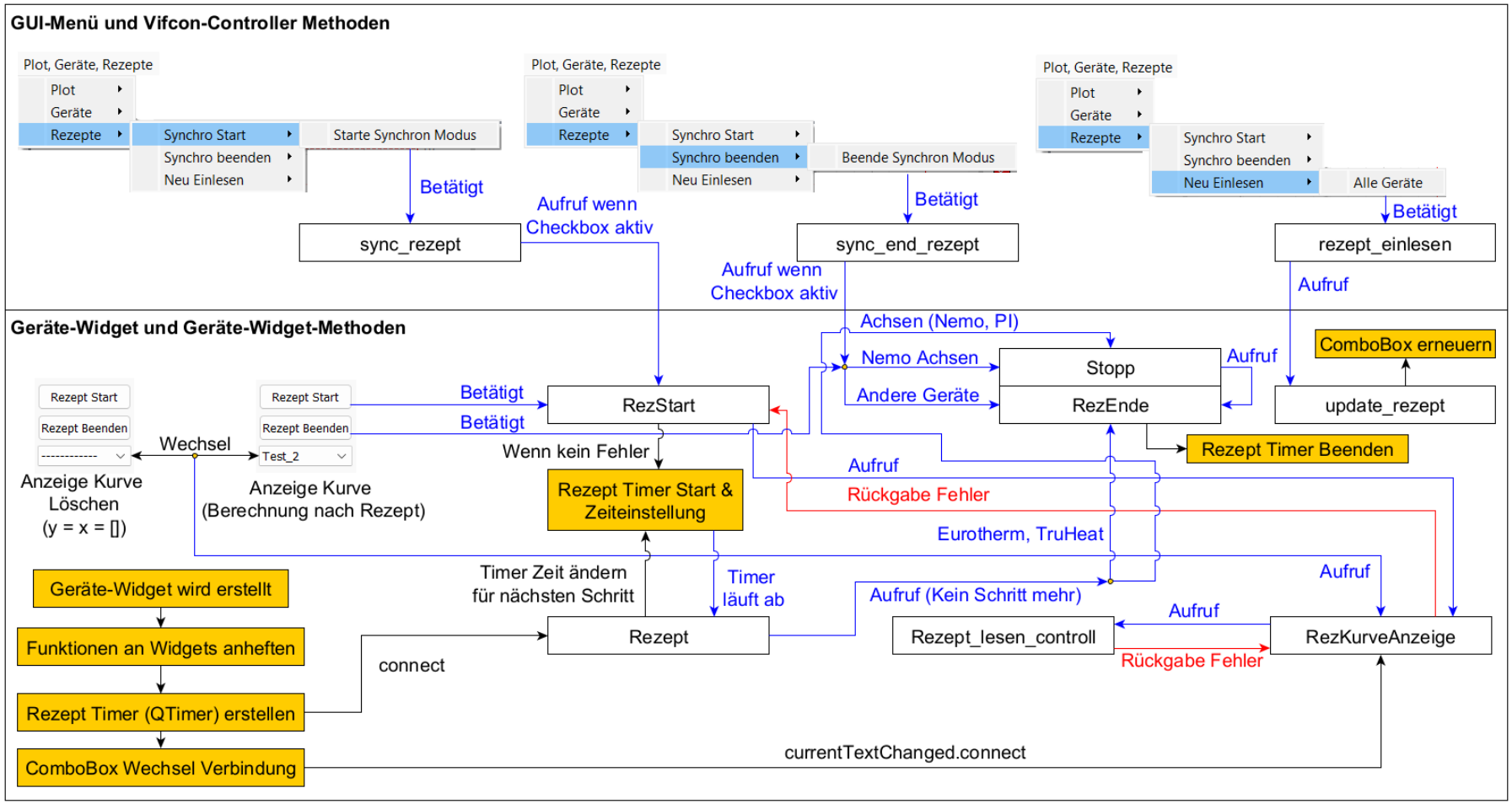


Abbildung 112: Aufbau der Rezepte in VIFCON (Quelle: eigene Darstellung)

Aus den bisherigen Erklärungen konnten die Funktionen:

1. Synchrones Rezept starten,
2. Synchrones Rezept beenden,
3. Anzeige der Kurve vor Start des Rezeptes im Plot und,
4. Aktualisierung der Rezepte im laufenden Betrieb

abgelesen werden. Wenn sich später die GUI in Abbildung 60 angesehen wird, kann bei den Antrieben (rechts) unter dem „Stopp Alle“-Knopf ein Icon gefunden werden, welches aus zwei Pfeilen besteht. Dieser Knopf lässt die Achsen, bei Checkbox-Auswahl, synchron laufen. Im Unterschied zu dem gleich erläuterten, wird hier nur eine Bewegung ausgelöst. Jedes Rezept kann einzeln gestartet werden, was ein gravierendes Problem für die Steuerung erzeugt. Der Nutzer muss hintereinander drei Knöpfe betätigen, was unvermeidlich Verzögerungen auslöst. Somit könnten Probleme wie Kollisionen der Antriebe verursacht werden. Um dies zu verbessern wurde die Möglichkeit des Synchronen Rezept-Startes eingebaut. Dieser ist wie erwähnt in der Menü-Leiste zu finden. Durch die Checkboxen kann der Nutzer festlegen was gleichzeitig starten soll. Weiterhin können somit auch Gruppen zu verschiedenen Zeiten gestartet werden, da laufende Rezepte nicht nochmal gestartet werden. Aus dem Synchronen Start entwickelte sich somit auch das Synchrone Beenden der Rezepte. Anstelle immer alle Rezepte einzeln zu beenden, kann der Anwender nun ein Knopf dafür nutzen. Wichtig ist hierbei zu wissen, dass die Checkbox auch hier wieder die Auslösung bringt. Die Checkbox wird jedoch mit dem Rezept-Start deaktiviert, sodass sie nicht mehr änderbar ist. Dieser Umstand bedeutet, dass alle aktiven Rezepte durch Synchron Ende abgestellt werden! Wie erwähnt befinden sich diese beiden Funktionen in der Menü-Leiste. Dies wurde gemacht um einerseits Platz zu sparen und andererseits wegen dem Grund, dass es sich um Typen unabhängige Funktionen handelt.

Auf die Zeitverzögerung muss nun nochmal kurz eingegangen werden. Im letzten Absatz wurde gesagt, dass die Synchro-Start Funktion aufgrund der Zeitverzögerung durch mehrfache Knopf-Betätigungen erstellt wurde. Auch die jetzige Lösung besitzt Zeitverzögerungen. Diese kommen zum einen durch den Fakt, dass die Funktion *sync_ rezept* eine For-Schleife ausführt. Somit werden die Funktionen der Widgets hintereinander ausgelöst. Diese Verzögerung ist jedoch nicht so groß. Eine größere Verzögerung kommt durch den Mutex-Locker (Kapitel 5.4). Wenn z.B. vier PI-Achsen genutzt werden und diese alle die selbe Schnittstelle haben. So werden zwar alle Geräte gleichzeitig in die *sample*-Funktion gelangen, aber der Mutex-Locker lässt durch die gleiche Schnittstelle immer nur eine PI-Achse den Rest der Funktion ausführen. Die anderen müssen warten bis die vorherige fertig ist. Diese Verzögerungen sind im Plot nicht wirklich zu sehen, da sich diese mehr oder weniger auf das Senden und Lesen konzentrieren. In so einem Fall müssen solche Aspekte akzeptiert werden. Um trotzdem das optimale Ergebnisse zu finden, müssen Delays (Verzögerungen) so optimiert werden, dass die Kommunikation noch immer mit dem Gerät funktioniert. In VIFCON kann es bei dem Lesen und Schreiben von Befehlen zu Delays kommen, da das Gerät auch etwas braucht um die Daten bereitzustellen.

Die Rezepte sollen auch im Plot zu sehen sein. Wenn ein Rezept gestartet wird, wird die Kurve für das Rezept erstellt und im Plot angezeigt. Beim Start wird so das Rezept immer vom Startzeitpunkt geplottet. Somit erhält der Nutzer die Information zur Dauer des Rezepts und weiß wo im Rezept VIFCON gerade ist. Doch was ist wenn etwas mit dem Rezept nicht stimmt. Aus dem Grund soll der Nutzer noch vor dem Start das Rezept sehen. Mit dieser Funktion kann sich der Programmanwender vorab von dem Rezept überzeugen und es im Notfall anpassen. Diese Funktion wird durch *currentTextChanged.connect* gegeben. Wenn die dazugehörige Funktion (*RezKurveAnzeige*) durch die genannte Verbindungsfunktion an die Combobox gehangen wird, dann löst ein Wechsel im Drop-Down-Menü der Combobox die Anzeige des ausgewählten Rezeptes aus. Für diesen Zweck wurden die Rezept-Funktionen etwas angepasst. Um die Anzeige des ersten Rezeptes zu gewährleisten, wurde die erste Zeile mit „----------“ belegt. Somit gibt es einen Default-Wert, bei der keine Kurve angezeigt wird. Durch den Wechsel der Auswahl und beim Rezept-Start, wird die alte Kurve gelöscht und die Neue angezeigt. Wenn auf den Default-Wert gewechselt wird, dann wird die Kurve gelöscht und bei Rezept-Start in dem Zustand eine Fehlermeldung ausgelöst.

Der letzte Aspekt der Rezepte ist die Aktualisierung der Combobox und des Config-Files. In Abbildung 112 gibt es rechts noch ein drittes Rezept-Menü. Dieses heißt „Neu Einlesen“ und funktioniert für alle Geräte gleich. Wenn kein Rezept aktiv für das Gerät läuft, wird die Config-Datei neu eingelesen und protokolliert. Zum Schluss wird dann die Combobox geleert und erneuert. Die Funktion liefert somit eine wichtige Erweiterung. Vor dieser Änderung wären die programmierten Rezepte fest gewesen, sodass jede Änderung ein Neustart erforderte. Mit der Neuerung wird dies nicht mehr benötigt. Somit hat der Nutzer die Möglichkeit die Rezepte im laufenden Betrieb zu schreiben und an VIFCON zu übergeben. Danach können diese dann übe die Knöpfe und Funktionen gestartet werden.

F.2 Programmierung des Rezeptes

Der Aufbau der Rezepte und der Inhalt der Funktionen ist bei den verschiedenen Geräten nahezu identisch. Zwischen den einzelnen Geräten gibt es kleiner Unterschiede wie benutzte Messgröße und Anzahl der möglichen Rezepte. Nur beim Eurotherm-Regler gibt es größere Unterschiede, da dieses mehr Segmentarten hat als die anderen Geräte. Aus dem Grund wird der Rezept-Code am Beispiel des TruHeats gezeigt. Danach werden ein paar Spezielle Sachsen zum Eurotherm Code gesagt. In (Abbildung 112) ist dabei die Funktions-Kommunikation der Rezept-Funktionen zu finden.

Den Start macht die Funktion *RezStart*, die mit dem Knopf zum starten der Rezepte verbunden ist. Sobald ein Rezept gestartet wird, wird die Variable *Rezept_ Aktiv* auf True gesetzt, wodurch ein erneutes Betätigen bzw. ein erneuter Aufruf verhindert wird. Dieser erneute Aufruf kann durch die Synchro-Funktion kommen. Im Folgenden wird hierbei folgendes ausgeführt:

```
1 def RezStart(self):
2     # Teil hier nicht gezeigt
3     # Kurve erstellen:
4     error = self.RezKurveAnzeige()
5     if self.cb_Rezept.currentText() == '----------':
6         self.Fehler_Output(1, self.err_15_str[self.sprache])
7         error = True
8     # Teil hier nicht gezeigt
```

Durch die gesamten Rezept-Funktionen wird die Variable *error* geleitet. Wenn also in *RezKurve-Anzeige*, *Rezept_ lesen_ controll* oder eben *RezStart* ein Fehler ausgelöst wird, wird die Variable *error* auf True gesetzt, wodurch das Rezept nicht gestartet wird und eine Fehlermeldung angezeigt wird. Die Funktion *Fehler_ Output* erstellt diesen Fehler in der GUI, der Log-Datei und wenn gewollt in der Ablauf-Datei. Wenn kein Fehler vorliegt, so werden die folgenden Schritte durchgeführt. Diese sind:

1. das Speichern des Rezeptes und dessen Namens in der Log-Datei,
2. das Senden des ersten Elements (*write_ value['Sollwert']*),
3. die Auswahl der Aufgabe (bei TruHeat Spannung, Strom oder Leistung),
4. das Speeren von GUI-Elementen (z.B. Knöpfe, Checkboxen, etc.) mit *setEnabled(False)* und,
5. den Start des Rezept-Timers mit Übergabe der ersten Segmentzeit.

Bei der Funktion *RezEnde* wird das Rezept-Ende durchgeführt. Dieses kann entweder durch das wirkliche Ende des Rezeptes oder durch Knöpfe ausgelöst werden. In dieser Funktion werden die Sachen von *RezStart* rückgängig gemacht. Neben diesem wird der Timer gestoppt und der Plot mit Autorange wieder angepasst.

Die Funktion *Rezept* ist die Funktion, die mit dem Timer verbunden ist. Für den TruHeat sieht diese wie folgt aus:

```
1  def Rezept(self):
2      self.step += 1
3      if self.step > len(self.time_list) - 1:
4          self.RezEnde()
5      else:
6          self.RezTimer.setInterval(int(abs(self.time_list[self.step]*1000)))
7
8          # Nächstes Element senden:
9          self.write_value['Sollwert'] = self.value_list[self.step]
10
11         if self.RB_choise_Pow.isChecked():
12             self.write_task['Soll-Leistung'] = True
13         elif self.RB_choise_Voltage.isChecked():
14             self.write_task['Soll-Spannung'] = True
15         elif self.RB_choise_Current.isChecked():
16             self.write_task['Soll-Strom'] = True
```

Hier wird die Variable *step* um eins erhöht. Diese Variable sorgt dafür, dass der aktuelle Rezept-Schritt an das Gerät übergeben wird. Das Ende des Rezeptes hängt hierbei von der Länge der Liste *time_ list* und der Variable *step* ab. In dem Code kann nun auch der Aufruf von *RezEnde* gesehen werden. Weiterhin kann gesehen werden, wie der Timer aktualisiert und der nächste Wert gesendet wird.

In der Funktion *Rezept_lesen_controll* werden die Werte des Rezeptes kontrolliert und die Sende-Daten vorbereitet. In der Funktion wird als erstes herausgefunden um welche Größe es sich bei dem Rezept handelt, wodurch die Grenzen der passenden Größe verwendet werden. Danach kommt das Herz-Stück der Rezepte. Wie gesagt wird dies am Beispiel TruHeat gezeigt. Nur bei Eurotherm ist dieser Teil stark anders. Dieser Teil wird als Graph in Abbildung 113 für TruHeat gezeigt. In diesem Code-Teil werden die Listen *time_list* und *value_list* mit Werten belegt. Hierbei ist wichtig das jeder Wert einen Zeitwert erhält. Diese Werte werden dann zum einen für die Rezept-Kurve (Anzeige), für das Senden und die Aktualisierung des Timers verwendet. Neben diesem Umstand werden hier die Werte mit den Grenzen verglichen und geprüft wo das Rezept zu finden ist (Config-Datei oder separate Datei). Der folgende Code zeigt dabei den Teil für das Segment „r“. Die Zeilen 4 bis 7 zeigen die Berechnung der einzelnen Werte. In Kapitel 5.5.2 wurde diese Berechnung gezeigt. Bei dem Sprung (Segment „s“) wird der Sollwert und die Zeit aus der Rezept Zeile nur in die beiden Listen geschrieben.

```
1  def Rezept_lesen_controll(self):
2      # Teil hier nicht gezeigt
3      if werte[2].strip() == 'r':
4          rampen_config_step = float(werte[3])
5          rampen_bereich = value - self.value_list[-1]
6          rampen_value = int(time/rampen_config_step)
7          rampen_step = rampen_bereich/rampen_value
8          #### Berechnung der Listen Elemente:
9          for rampen_n in range(1,rampen_value+1):
10             if not rampen_n == rampen_value:
11                 ak_ramp = self.value_list[-1] + rampen_step
12                 self.value_list.append(round(ak_ramp,3))
13                 self.time_list.append(rampen_config_step)
14             else: # Letzter Wert!
15                 self.value_list.append(value)
16                 self.time_list.append(rampen_config_step)
17     # Teil hier nicht gezeigt
```

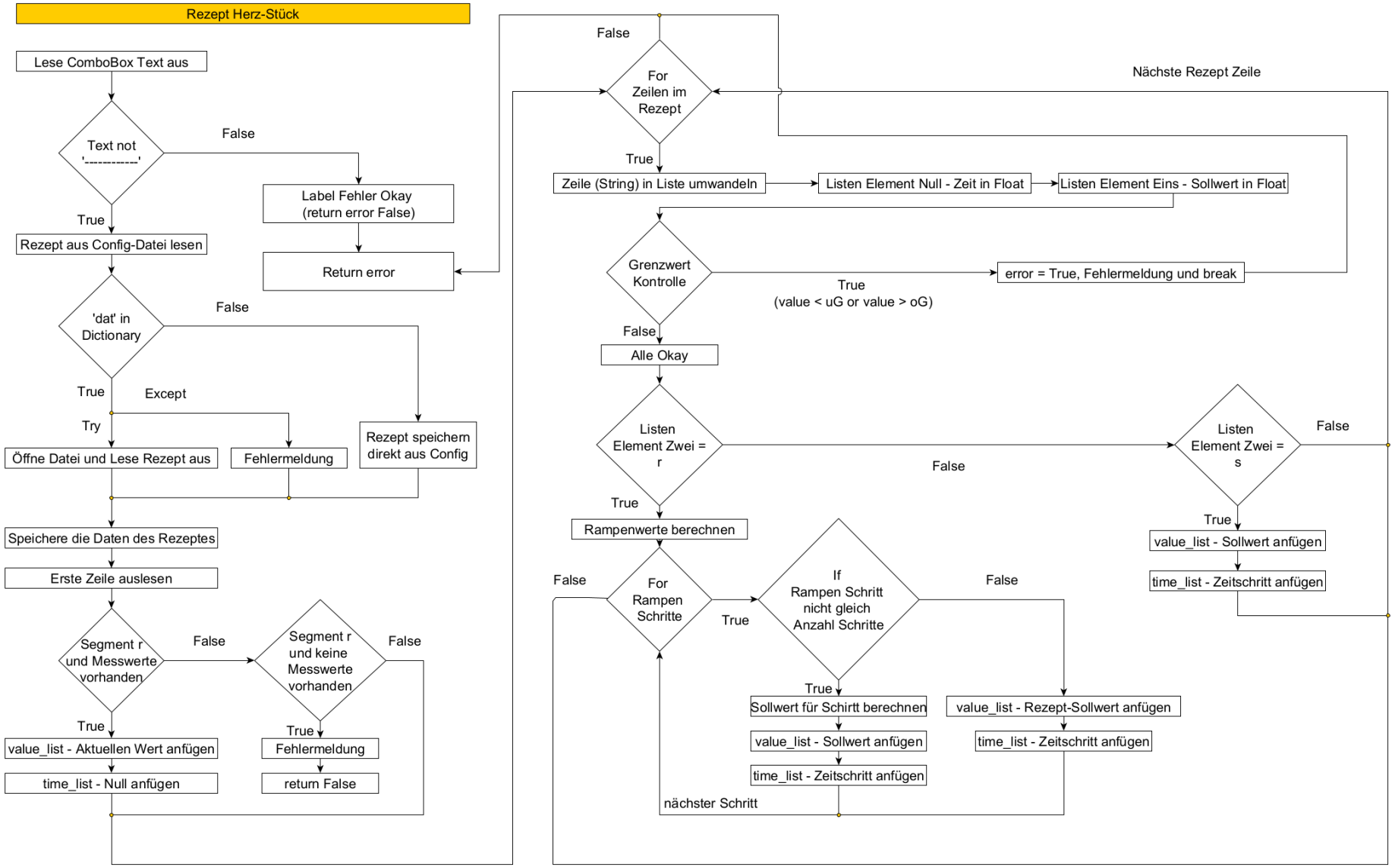


Abbildung 113: Graph zum Herzstück des Rezept-Codes (Quelle: eigene Darstellung)

Die letzte Rezeptfunktion heißt *RezKurveAnzeige*. Mit dem Aufruf dieser Funktion werden die beiden Listen *RezTimeList* und *RezValueList* geleert und somit die Kurve gelöscht. Dies erfolgt mit dem folgenden Code. Da die Rezept-Kurven durch die Config-Datei deaktiviert werden und trotzdem funktionieren sollen, muss die Anzeige in dem Fall verhindert werden. Weiterhin wird bei Aufruf der aktuelle Zeitpunkt ermittelt, sodass die Kurve bei diesem anfangen kann.

```
1 def RezKurveAnzeige(self):
2     # Teil hier nicht gezeigt
3     try: self.curveDict['RezI'].setData(self.RezTimeList, self.RezValueList)
4     except: anzeigeI  = False
5     # Teil hier nicht gezeigt
```

Nach dem dies erledigt ist und kein Fehler anliegt, können die Kurven Punkte erstellt werden. Bei allen Geräten, außer Eurotherm, können die Rezepte nur Sprünge ausführen. Auch die Rampe die hier gezeigt wurde, besteht aus vielen Punkten. Der folgende Code zeigt die Erstellung der Plot-Kurve. Jeder Schritt besteht dadurch aus einer Parallelen zur x-Achse. Die Parallele zur y-Achse wird vom Programm selbst gezogen.

```
1 def RezKurveAnzeige(self):
2     # Teil hier nicht gezeigt
3     # Kurve erstellen:
4     i = 0
5     for n in self.time_list:
6         # Punkt 1:
7         self.RezTimeList.append(ak_time)
8         self.RezValueList.append(self.value_list[i])
9         # Punkt 2:
10        self.RezTimeList.append(ak_time + n)
11        self.RezValueList.append(self.value_list[i])
12        # nächsten Punkt vorbereiten:
13        ak_time = ak_time + n
14        i += 1
15    # Teil hier nicht gezeigt
```

Darauf folgend wird nun nur noch geschaut, um welche Größe es sich handelt. Dies ist z.B. bei TruHeat und Eurotherm speziell, da es dort mehr als eine Größe mit Rezepten gibt. In dem Code wird nun der Skalierungsfaktor und auch die gewollte Anzeige beachtet. Bei z.B. der PI-Achse würde die erste If-Anweisung entfallen, da dort nur die Geschwindigkeit über ein Rezept gesteuert wird.

```
1 def RezKurveAnzeige(self):
2     # Teil hier nicht gezeigt
3     if self.RB_choise_Pow.isChecked():
4         faktor = self.skalFak['Pow']
5         y = [a * faktor for a in self.RezValueList]
6         if anzeigeP:
7             self.curveDict['RezP'].setData(self.RezTimeList, y)
8             self.typ_widget.plot.achse_2.autoRange()
9     # Teil hier nicht gezeigt
```

Wie schon gesagt muss bei dem Code beachtet werden, dass dieser nur den TruHeat Code zeigt. Der Code Unterschied sich von Gerät zu Gerät. Zum Beispiel gibt es bei den Antrieben auch die Betrachtung und Berechnung des Weges, der durch das Rezept abgearbeitet wird. Zuletzt soll der Eurotherm erwähnt werden. Bei diesem sind die ganzen Funktionen umfangreicher, da es mehr Segmentarten gibt. Außerdem gibt es beim Eurotherm die Verbindung zweier Größen in einem Rezept. Bei den Segmenten „op“ und „opr“ müssen somit zwei Kurven erstellt und 4 Limits beachtet werden. Weiterhin muss bei den Eurotherm eigenen Rampen („er“) beachtet werden, dass diese richtig zurückgesetzt und der aktuelle Sollwert an Eurotherm übergeben wird. Somit ändern sich z.B. Abbruch-Bedingungen und auch die Funktion *RezEnde*. Das Herzstück aus Abbildung 113 ist für den Eurotherm noch komplexer, da es nun 5 Segmentarten gibt, an Stelle von zweien. Eurotherm wurde auch um weitere Listen erweitert. Diese sind:

1. value_list_op (OP-Sollwert, für Leistungssprung und Leistungsrampe),
2. Art_list (Speichert die Art des Segments für die anderen Funktionen) und,
3. Steigung (beinhaltet die Steigungen der Segmente)

Somit ändert sich bei Eurotherm das Segment „r“ wie folgt.

```
def Rezept_lesen_controll(self):
    # Teil hier nicht gezeigt
    if werte[2].strip() == 'r':
        rampen_config_step = float(werte[3].replace(',','.'))
        rampen_bereich = value - self.value_list[-1]
        rampen_value = int(time/rampen_config_step)
        rampen_step = rampen_bereich/rampen_value
        #### Berechnung der Listen Elemente:
        for rampen_n in range(1,rampen_value+1):
            if not rampen_n == rampen_value:
                ak_ramp = self.value_list[-1] + rampen_step
                self.value_list.append(round(ak_ramp,3))
                self.time_list.append(rampen_config_step)
            else: # Letzter Wert!
                self.value_list.append(value)
                self.time_list.append(rampen_config_step)
            self.Art_list.append('r')
            self.value_list_op.append(0)
            self.Steigung.append(rampen_config_step)
    # Teil hier nicht gezeigt
```

Im groben und ganzen gibt es nur kleine Unterschiede zum TruHeat bei diesem Segment. Der Unterschied sind die nötigen Erweiterungen, die die einzelnen Geräte brauchen. Sonst sind die Funktionen und der Aufbau dieser Funktionen identisch.

G Anhang: Rezepte für die Experimente

In Kapitel 6 wurden Experimente vorgestellt. Zur Vollständigkeit werden hier die Rezepte gezeigt, wie sie in der Config-Datei aussehen müssten.

G.1 Heiztest an der Nemo-1 Anlage

Das folgende Rezept gehört zu Abbildung 76 (Kapitel 6.2). Die erste Zeile ist der Schlüssel für das Rezept, der auch in der Combobox auf der GUI angezeigt wird. Dabei wird der Nemo-1-Antrieb für die Rotation angesteuert.

```
1      rezept_rezept_TE_Heal_R1:
2          n1: 300 ; 5  ; s
3          n2: 60;   0  ; s
4          n3: 300;  -5 ; s
```

Das folgende Rezept gehört zu der mittleren Abbildung aus Abbildung 75. Dabei wird der Eurotherm-Regler angesteuert.

```
1      rezept_TE_Heal_T1:
2          n1: 1   ; 50  ; s
3          n2: 300 ; 200 ; r ; 5
4          n3: 60  ; 200 ; s
5          n4: 300 ; 20 ; r ; 5
```

Auch das folgende Rezept gilt für den Eurotherm-Regler. Hierbei wird das Rezept aus der linken Abbildung aus Abbildung 75 gezeigt.

```
1      rezept_TE_Heal_T1:
2          n1: 1   ; 100  ; s
3          n2: 300 ; 400 ; r ; 5
4          n3: 60  ; 400 ; s
5          n4: 300 ; 20 ; r ; 5
```

Auch das letzte Rezept gilt für den Eurotherm-Regler. Es zeigt das Rezept der Abbildung 78.

```
1      rezept_TE_Heal_T1:
2          n1: 1   ; 100 ; s
3          n2: 600 ; 800 ; r ; 5
4          n3: 600 ; 800 ; s
5          n4: 43200 ; 20 ; r ; 5
```

G.2 Kristallzüchtung an Nemo-1 Anlage

Alle folgend gezeigten Rezepte, gehören zu dem Experiment aus Kapitel 6.3. Die folgenden drei Rezepte, zeigen die Schritte für das Aufheizen und Ankeimen, die synchron gestartet wurden. Die Umsetzung aller Rezepte erfolgte über die Nutzung der „dat“-Funktion, also dem Auslagern in einer anderen Datei. Hier werden nur die Rezepte gezeigt, wie sie erstellt werden müssen.

Eurotherm:

```
1 n1: 1200 ; 240 ; er ; 0.1825
2 n2: 600 ; 240 ; s
3 n3: 600 ; 234 ; er ; 0.01
4 n4: 1200 ; 234 ; s
```

Nemo-1-Anlage - Antrieb Hub:

```
1 n1: 1200 ; 0 ; s
2 n2: 1080 ; -5 ; s
3 n3: 420 ; 0 ; s
4 n4: 300 ; -2 ; s
5 n5: 600 ; -1 ; s
6 n6: 10 ; 0 ; s
```

Nemo-1-Anlage - Antrieb Rotation Spindel:

```
1 n1: 2700 ; 0 ; s
2 n2: 900 ; 5 ; s
3 n3: 10 ; 0 ; s
```

Nach diesen wurden die vier folgenden Rezepte synchron gestartet. Sie dienen der Züchtung, dem Abziehen und der Abkühlung.

Eurotherm:

```
1 n1: 3000 ; 240 ; er ; 0.002
2 n2: 600 ; 240 ; op ; IST
3 n3: 10 ; 240 ; s
4 n4: 3000 ; 40 ; er ; 0.067
5 n5: 600 ; 20 ; er ; 0.033
6 n6: 300 ; 20 ; s
```

Nemo-1-Anlage - Antrieb Hub:

```
1 n1: 1200 ; 1 ; s
2 n2: 1200 ; 2 ; s
3 n3: 1200 ; 3 ; s
4 n4: 10 ; 30 ; s
5 n5: 30 ; 0 ; s
```

Nemo-1-Anlage - Antrieb Rotation Spindel:

```
1 n1: 3600 ; 2 ; s
2 n2: 10 ; 10 ; s
3 n3: 30 ; 0 ; s
```

Nemo-1-Anlage - Antrieb Rotation Tiegel:

```
n1: 3000 ; 0 ; s
n2: 600 ; -5 ; s
n3: 10 ; 0 ; s
```

H Anhang: Schnittstellen Programmierung in Python

In dem Teil des Anhangs wird mehr zu der Schnittstellen Programmierung in Python erläutert. Dabei wird zum einen die Definition bzw. Instanziierung der Schnittstellen-Objekte (Anhang H.1) und zum anderen die mehrfach Nutzung der selben Schnittstelle (Anhang H.2) beschrieben.

H.1 Schnittstellen Definition in Python

In VIFCON werden zwei verschiedene Schnittstellen verwendet. Aus dem Grund benötigt es nun auch zwei verschiedene Python-Bibliotheken. Für die RS232-Schnittstellen wird die Bibliothek **pySerial** und für das Kommunikationsprotokoll Modbus (dazugehörige Schnittstelle: Ethernet) wird die **pyModbusTCP**-Bibliothek verwendet. Folgend wird die Instanz-Erstellung des Schnittstellen Objektes für beide Schnittstellen-Arten gezeigt. Interessanterweise ist der Aufbau zur Erstellung sehr ähnlich. Genutzt werden die Klassen *Serial* und *ModbusClient*. Bei Modbus bildet der Computer, auf dem VIFCON läuft, den Client (Master) und die Anlage den Server (Slave). In den Code-Zeilen ist das Wort „config“ wieder zu finden, was auf die Config-Datei hinweist. Bei beiden Schnittstellen, kann der Anwender die Schnittstelle über diese Datei konfigurieren. In den Kapiteln 2.3.1.1 bis 2.3.1.3 werden 5 von 6 Größen der RS232-Schnittstelle erläutert. Diese waren Port, Baudrate, Parität, Stoppbits und Datenbytes. Die sechste Größe wäre das sogenannte timeout, welches eine Lese-Zeitverzögerung in Sekunden angibt. Bei der Modbus Schnittstelle werden die Server-IP-Adresse und der Port angegeben. Neben diesem kann bei **pyModbusTCP** auch ein Default Wert (True oder False) gesetzt werden, wodurch die gesendeten und empfangene Daten in der Konsole geschrieben werden. Ein Beispiel dafür wird in Anhang I gezeigt.

```
# RS232:
self.serial = Serial(**config["serial-interface"])

# Modbus:
self.serial = ModbusClient(**config["serial-interface"])
```

Bei den Schnittstellen ist weiterhin sehr wichtig, die Kommunikationsweise der Geräte zu kennen. Dabei spielen z.B. Abschlusszeichen eine große Rolle. Bei der RS232-Schnittstelle, also der **pySerial**-Bibliothek, stehen in VIFCON die Funktionen *write*, *readline* und *read* zu Verfügung. Neben diesen Funktionen müssen auch Funktionen wie *decode* und *encode* beachtet werden. Hierbei ist wichtig, ob dies überhaupt angewendet werden muss. Dieses beiden Funktionen werden in Zusammenhang mit den anderen drei genannten Funktionen verwendet. In Kapitel 3 gibt es für jedes Gerät auch das Kapitel „Umsetzung in Python“. In diesem wurde auf die Nutzung der genannten Funktionen eingegangen. Folgend werden diese Befehle anhand von Beispielen, aus dem VIFCON Code gezeigt:

```
1  # Ordner: devices -> Programm piAchse.py -> Objekt: PIAchse -> Methode: ...
       send_read_command
2  ## Schreiben und Encode:
3  self.serial.write(self.t1+Befehl.encode()+self.t3)
4
5  ## Lesen und Decode:
6  ant = self.serial.readline().decode()
7
8  # Ordner: devices -> Programm truHeat.py -> Objekt: TruHeat -> Methode: read_send
9  ## Lesen:
10 ans = self.serial.read()
```

Die write-Funktion wird in all den drei RS232-Geräten (Eurotherm, TruHeat, PI-Achse) verwendet. Dabei wird beim TruHeat Generator jedes Zeichen einzeln gesendet und gelesen. Auch bei der PI-Achse könnte die gezeigte Zeile in drei Schreib-Befehle zerlegt werden. Die Geräte arbeiten meist mit Abschlusszeichen oder anderen Wegen um das Ende von Nachricht zu finden. Wie bereits erwähnt, können solche Themen aus dem Kapitel 3 oder aus den Geräte-Dokumentationen entnommen werden. Somit wird auch die Nutzung von *encode* und *decode* und auch von *read* oder *readline* deutlich. Der Unterschied hierbei ist, dass *readline* eine ganze Zeile bis zum Newline (einschließlich diesem) ausliest. Mit read wird ein bestimmte Anzahl von Bytes ausgelesen. In dem Code wird die Default Einstellung von eins verwendet. Aus dem Grund wird jeder Aufruf von *read* ein Byte aus dem Gerät lesen. Die **pySerial**-Bibliothek kann somit sehr flexibel auf die verschiedenen Geräte-Anforderungen reagieren, sodass diese Bibliothek sich für diese Zwecke durchgesetzt hat.

Für Modbus wurde sich, wie schon erwähnt, für **pyModbusTCP** entschieden. Diese Bibliothek hat sich gegenüber den anderen Bibliotheken durchgesetzt, da sie einen großen Teil der Kommunikation übernimmt und es bereits Methoden gibt, die das Arbeiten mit den Registern und die Umwandlung der Werte erleichtert. Neben dieser Bibliothek wurden sich z.B. noch **minimalmodbus** [Dokab] und **PyModbus** [Dokad] angesehen. Wie schon erwähnt, hat sich **pyModbusTCP** durchgesetzt, da es eine gute Dokumentation [Dokac] hat, relativ einfach umsetzbar ist und bereits viele Funktionen aufweist. Somit werden für die Nutzung der Register und Coils die Funktionen *read_input_registers*, *write_single_coil* und *write_multiple_registers* (Holding-Register) verwendet. Zur Nutzung dieser müssen die Tabellen 14, 15 und 16 beachtet werden. In der Config-Datei wird dann zwischen den verschiedenen Geräten unterschieden, indem die richtigen Startadressen eines Blockes gesetzt werden. Für die Formatierung und Umwandlung der Daten gibt es die Funktionen *encode_ieee* *decode_ieee* und *word_list_to_long*. Diese Umwandlung und die Nutzung aller Methoden wird in Anhang I.3 gezeigt.

Bei den Schnittstellen muss nun noch das ein oder andere beachtet werden. So ist ein Punkt der seriellen Schnittstellen der, dass immer nur ein Gerät über die Schnittstelle Kommunizieren kann. Zum Beispiel wurde in Kapitel 3.2 erwähnt, dass die PI-Achse nach einer Daisy Chain Verbindung verbunden werden kann. Diese Art Verbindung wurde auch in der Experiment-Skizze (Abbildung 66) aus Kapitel 6.1 gezeigt. Hierbei kann immer nur eine Achse über die Schnittstelle kommunizieren. Somit muss die Instanziierung des Schnittstellenobjekts erweitert und die Kommunikation geblockt werden. Dieses Thema wurde in Kapitel 5.4 aufgegriffen. Neben diesem hat VIFCON auch eine Initialisierungs-Funktion bekommen, wodurch die *wirte* und *read* Funktionen geblockt werden, wenn diese Initialisierung auf False steht. Die Initialisierung hat den Sinn, dass der Port der Schnittstelle zwar aufgebaut wird, aber keine Kommunikation möglich ist, bis der Anwender sein Okay gibt und das Gerät initialisiert. Somit kann ein Neustart des Programms verhindert werden und ein Experiment im laufenden Betrieb erweitert werden. Auch für die Messung und die Aufnahme der Daten wurde dies so umgesetzt, dass VIFCON mit der Initialisierung diese einschaltet. Eine weiterer Punkt ist, dass die Erstellung der Schnittstelle

durch ein Try-Except ausgestattet ist. Sobald das Except auslöst wird, wird das Programm durch den Aufruf von *exit()* beendet. Zuvor wird dies als Warnung und Fehler protokolliert.

Die Schnittstelle ist eins der wesentlichsten und wichtigsten Teile der VIFCON-Steuerung. Durch diese werden die einzelnen Geräte angesteuert und die Kommunikation freigeschaltet. Aus den Beschreibung wird klar, was bei den Schnittstellen zu beachten ist, wie diese aufgebaut werden und was aus den Bibliotheken genutzt wird.

H.2 Mehrfachnutzung einer Schnittstelle

Nach dem die Definition der Schnittstelle in vorherigen Anhang gezeigt wurde, soll es nun darum gehen wie eine Schnittstelle mehrfach genutzt werden kann. Für diese Zwecke muss zum einen die Definition der selben Schnittstelle verhindert werden und dafür gesorgt werden das nicht mehr als ein Thread jeweils auf die Schnittstelle zugreift. Der Code für die Verhinderung des erneuten erstellen einer Schnittstelle ist folgend zusehen (Beispiel PI-Achse) und ist nahezu identisch für alle Geräte. Die Variable *self.serial* zeigt die Schnittstellen-Definition, die in Anhang H.1 gezeigt wird. Diese Definition zeigt den Unterschied zwischen den Geräten (Eurotherm, Pi-Achse, TruHeat - *Serial* und Nemo-1-Anlage - *ModbusClient*). Neben diesem Umstand muss auch im *Controller*-Objekt etwas beachtet werden, dazu später mehr.

```
1  class PIAchse:
2      def __init__(self, sprache, config, com_dict, test, neustart, ...
           add_Ablauf_function, name="PI-Achse", typ = 'Antrieb'):
3          # Teil hier nicht gezeigt
4          if not test:
5              com_ak = ''
6              for com in com_dict:
7                  if com == self.config['serial-interface']['port']:
8                      com_ak = com
9                      break
10             if com_ak == '':
11                 self.serial = Serial(**config["serial-interface"])
12             else:
13                 self.serial = com_dict[com_ak]
14         # Teil hier nicht gezeigt
```

Wenn der gezeigte Code einen Fehler auslöst, wird ein Except ausgelöst, wodurch das Programm beendet wird. Damit wird sichergestellt, dass jedes Gerät zu Beginn funktioniert und die Schnittstelle existiert. Weiterhin kann eine Funktion des Test-Modus gesehen werden. Ist dieser aktiv, wird der Code nicht ausgelöst. Das Dictionary *com_ dict* bekommt jedes Gerät bei der Erstellung der Schnittstellen-Instanz übergeben. In diesem steht als Schlüssel die Port-Bezeichnung und als Wert dazu die Schnittstellen-Instanz. Dieses Dictionary wird in einer For-Schleife bearbeitet, die schaut ob die Geräte-Port-Bezeichnung bereits eine Schnittstellen Instanz hat. Wenn Ja, wird der Port in *com_ ak* gespeichert. In der folgenden If-Else-Anweisung wird dann entschieden ob die Schnittstellen-Instanz des Gerätes eine neue oder eine vorhandene Schnittstellen-Instanz bekommt. In der *Controller*-Instanz Erstellung wird dafür der folgende Code verwendet.

```
1  class Controller(QObject):
2      # Teil hier nicht gezeigt
3      def __init__(self, config, output_dir, test_mode, neustart) -> None:
4          # Teil hier nicht gezeigt
5          ak_com = self.config['devices'][device_name]['serial-interface']['port']
6          if not ak_com in self.com_sammlung:
7              if not self.test_mode:
8                  self.com_sammlung.update({ak_com: device.serial})
9              mutex = QMutex()
10             self.mutexs.update({ak_com: mutex})
11         # Teil hier nicht gezeigt
```

Das Code-Fragment wird nach der Erstellung eines Gerätes (Widget-Instanz und Schnittstellen-Instanz) ausgeführt. VIFCON geht in dem Moment alle Einträge durch, die im Config-Dictionary *devices* stehen. Diese For-Schleife wird in der Abbildung 114 als Graph gezeigt. Für dieses Kapitel ist dabei nur dieses Code-Fragment von Bedeutung. In diesem Code kann nun das Dictionary *com_ sammlung* gefunden werden, welches als *com_ dict* in den Geräte-Instanzen bekannt ist. Immer wenn ein neuer Port auftaucht, wird dieser in diesem Dictionary gespeichert, sodass die Geräte bei Erstellung der Schnittstelle die dazugehörige Instanz von *Serial* und *ModbusClient* finden.

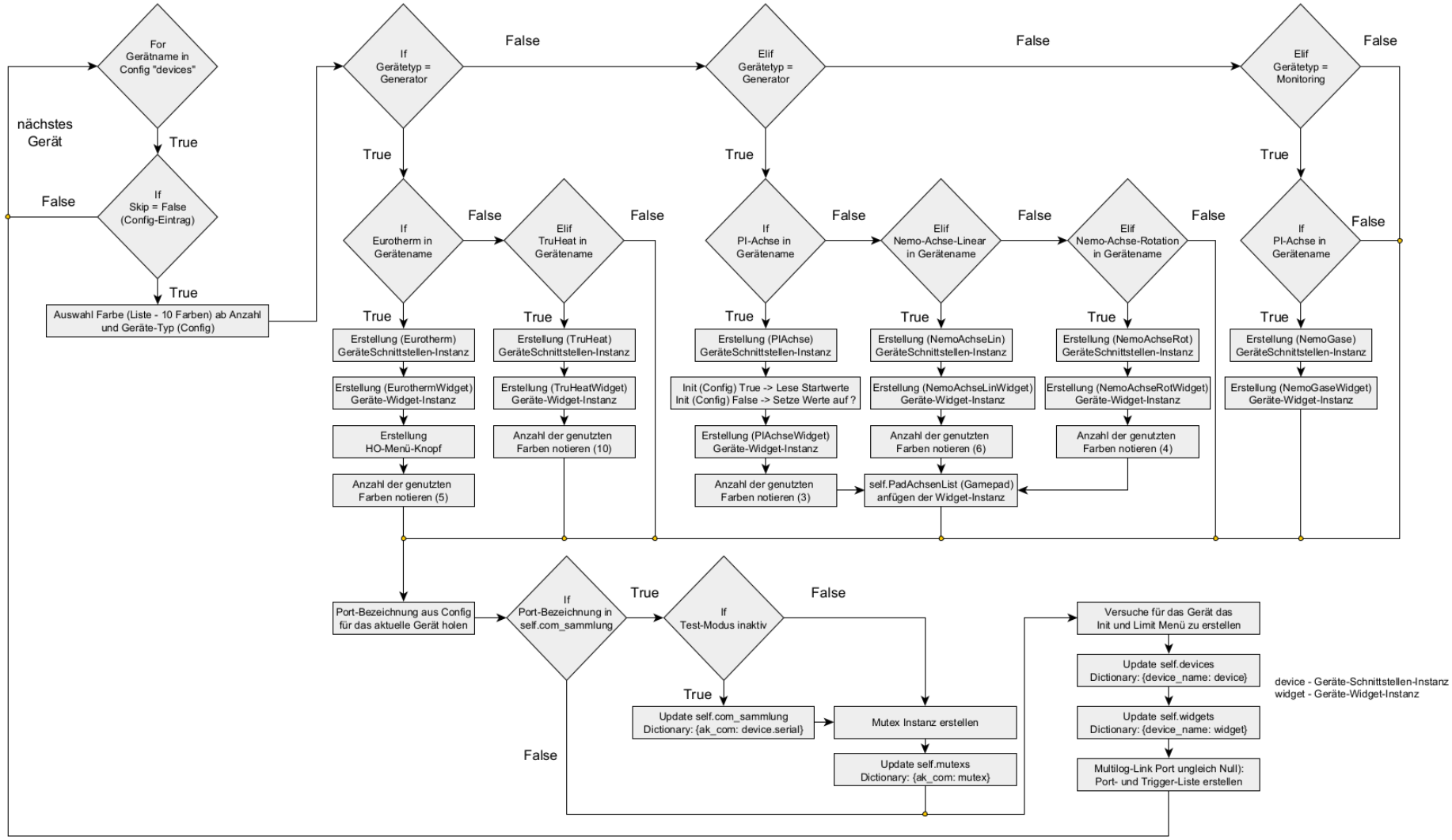


Abbildung 114: VIFCON Code Ausschnitt (vifcon_controller.py - __init__-Funktion des Controller-Objekt) - Graph für die Erstellung der Geräte-Instanzen (Schnittstelle und Widget) (Quelle: eigene Darstellung)

I Anhang: pyModbusTCP - Debug Funktion

In diesem Kapitel sollen die **pyModbusTCP**-Funktionen, die in VIFCON benutzt werden näher beschreiben werden. Hierfür werden Beispiele gezeigt die durch die Debug-Funktion der Bibliothek in der Konsole geschrieben wurden. Nach [Dokac] wird durch dieses Debuggen die Nachricht als auch der Modbus Frame angezeigt, welcher in den eckigen Klammern zu sehen ist. Bei den Funktionen muss der Nutzer nur noch die Adresse und die Anzahl an Registern angeben. Der Aufbau der gesendeten Nachrichten, sowie der Empfangenden kann in Abbildung 16 gesehen werden.

I.1 Beispiel Schreiben/Senden

Dieses Kapitel soll das Schreiben von Werten in die Register zeigen. Hierbei werden in VIFCON das Holding-Register und die Coils verwendet. Das *ModbusClient*-Objekt aus **pyModbusTCP** liefert dafür die beiden Funktionen:

1. write_single_coil() und,
2. write_multiple_registers().

Wenn der Dokumentation [Dokac] von **pyModbusTCP** gefolgt wird, dann finden sich die Funktionscodes der verschiedenen Modbus Objekte. Das Schreiben der Coils hat den Funktionscode 5 und das beschreiben der Holding-Register den Funktionscode 16. Bei dieser Funktion werden mehrere Register beschrieben. In Tabelle 2 finden sich die Funktionscodes auch wieder. Bei beiden Funktionen muss als erstes die jeweilige Register- oder Bit-Adresse und dann der zuschreibende Wert oder die zuschreibenden Werte angegeben werden. Die angegebene Adresse ist die Start-Adresse. Bei mehreren Werten, werden die Register ab dieser Adresse befüllt.

Coil:

In dem Beispiel wird die Bit-Adresse 20 bearbeitet. Durch das Setzen dieser Adresse auf True wird sich die Spindel rechtsum drehen (Rotation). Dies kann in Tabelle 16 nachgelesen werden.

```
1  Befehl:
2  modbus.write_single_coil(20, True)
3
4  Senden/Empfangen:
5  Tx
6  [6A 47 00 00 00 06 01] 05 00 14 FF 00
7  Rx
8  [6A 47 00 00 00 06 01] 05 00 14 FF 00
9
10 Antwort:
11 True
```

Bei der Anfrage (Tx) und der Antwort (Rx) ist das nach den eckigen Klammern die Nachricht. Das in den eckigen Klammern beschreibt den Frame des Modbus. Wenn der Abbildung 16 gefolgt wird so sind die Anfrage und die Antwort bei dem Schrieben von Coils identisch. Sie bestehen aus:

1. dem Funktionscode (1-Byte),
2. der Adresse (2-Byte) und,
3. dem Daten-Wort (2-Byte).

Somit ergeben sich jeweils 5 Bytes. Der Funktionscode ist bei diesem Objekt die 5. Die 05_{16} entspricht der 5_{10}. Dies stimmt somit. Danach folgt die Adresse. Die Adresse die Beschrieben wurde ist die 20_{10}. Dies entspricht in Hexadezimal der 0014. Das Datenwort beinhaltet den Wert $FF00_{16}$.

Bei den Coils kann das Datenwort nur $FF00_{16}$ oder 0000_{16} sein. Mit dem ersten wird ausgesagt, dass der Coil Aktiv ist, also auf True steht. Das zweite sagt wiederum aus, dass der Coil inaktiv und somit auf False steht. (Quelle: [Modb])

Somit kann das gesendete in der Anfrage und der Antwort wieder gefunden werden. Die Python-Bibliothek gibt bei den Schreib-Befehlen lediglich ein True zurück, wenn das Schreiben funktioniert hat!

Holding-Register:

In dem Beispiel wird die Sollgeschwindigkeit Spindel Hub beschrieben. Nach Tabelle 15 hat dieser Wert die Registeradresse 4 und ist 2 Register groß. Somit müssen zwei Werte gesendet werden. In dem Beispiel wird die Zahl 1,5 mm/min gesendet. Wie die Umwandlung funktioniert wird in Kapitel I.3 näher erläutert.

```
1  Befehl:
2  modbus.write_multiple_registers(4, [16320, 0])
3
4  Senden/Empfangen:
5  Tx
6  [AF FD 00 00 00 0B 01] 10 00 04 00 02 04 3F C0 00 00
7  Rx
8  [AF FD 00 00 00 06 01] 10 00 04 00 02
9
10 Antwort:
11 True
```

Die Anfrage beim Holding-Register bzw. Multiple Registers schreiben beinhaltet:

1. den Funktionscode (1-Byte),
2. der Adresse (2-Byte),
3. der Anzahl der Register (2-Byte),
4. der Anzahl an Bytes (1 Byte) und
5. den Datenwörtern.

Die Antwort beinhaltet nur die ersten drei Teile und besteht somit aus 5 Byte, während die Anfrage aus 6 festen Bytes besteht und danach die Daten folgen. Hierbei können Daten von Byte 6 bis Byte 2N+5 nach Abbildung 16 beschreiben werden. Das N steht dabei für die Anzahl der Register.

Der Funktionscode für dieses Beispiel ist die 16_{10}. Die 10_{16} entspricht dieser, womit der Funktionscode stimmt. Die Adresse die in dem Beispiel gesendet wird ist die 4_{10} (Integer). Diese ist folgend als 0004_{16} zu finden. Die Anzahl der Register wird nicht direkt durch die Funktion übergeben. In der Anfrage steht 0002_{16} für die Anzahl. Wenn sich die übergebene Liste angesehen wird, so kann gesehen werden, dass diese nur 2 Elemente hat. Somit werden 2 Register beschrieben. Diese drei Werte sind auch in der Antwort richtig widergespiegelt. Bei der Anfrage wird danach dann die Anzahl der Bytes angegeben. Da 2 Register beschreiben werden und jedes

Register 16 Bit, also 2 Byte, groß ist, wird die Anzahl bei 4 liegen, was diese auch tut. Als letztes folgen die Daten. Die 1,5 mm/min werden zu der Hexadezimalzahl 3FC00000. Hierbei ist zu beachten, dass in dem ersten Register das Höherwertige Wort ($16320_{10} \hat{=} 3FC0_{16}$) liegt und in dem zweiten das niederwertige Wort. Dieser Umstand hängt von der Programmierung der SPS ab. Dazu wird in Kapitel I.3 noch näheres erläutert.

Wie bei den Coils gibt auch diese Funktion (aus **pyModbusTCP**) ein True zurück, wenn das Schreiben gelungen ist.

I.2 Beispiel Lesen

Dieses Kapitel soll das Lesen von Werten aus den Registern zeigen. Hierbei wird in VIFCON nur das Input-Register verwendet. Das *ModbusClient*-Objekt aus **pyModbusTCP** liefert dafür die Funktion *read_input_registers*.

Wenn der Dokumentation [Dokac] von **pyModbusTCP** gefolgt wird, dann finden sich die Funktionscodes der verschiedenen Modbus Objekte. Das Lesen der Input-Register hat den Funktionscode 4. In Tabelle 2 finden sich die Funktionscodes auch wieder. Das erste was diese Funktionen bekommen ist wieder die Adresse der Register. Danach folgt ein Integer, der die Anzahl der zu lesenden Register angibt. Dies kann bei dem ersten Beispiel gesehen werden.

Im Folgenden wird der Aufbau der Anfrage und Antwort gezeigt. Somit besteht die Anfrage aus:

1. dem Funktionscode (1-Byte),
2. der Adresse (2-Byte) und,
3. der Anzahl der Register (2-Byte).

Die Antwort besteht aus:

1. dem Funktionscode (1-Byte),
2. der Anzahl der Bytes (1-Byte) und,
3. den Datenwörtern.

Bei den Datenwörtern werden die Bytes 2 bis Byte 2N+1, wobei N wieder die Anzahl der Bytes ist (hier aus der Anfrage). Folgend wird ein Beispiel für das Auslesen der Input-Register gezeigt. In dem Beispiel werden die 7 Gleitpunkt-Werte (außer Adresse 18) aus Tabelle 14 der Register 2 bis 15 ausgelesen. Die 15 ist in der Tabelle nicht zu sehen, da dieses Register zu dem Register 14 gehört. In Register 14 und 15 steht der Wert für den Druck der PP22 Pumpe.

```
1  Befehl:
2  modbus.read_input_registers(2,14)
3
4  Senden/Empfangen:
5  Tx
6  [55 C7 00 00 00 06 01] 04 00 02 00 0E
7  Rx
8  [55 C7 00 00 00 1F 01] 04 1C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...
       44 74 87 89 00 00 00 00 44 7C 2D 8C
9
10 Antwort:
11 [0, 0, 0, 0, 0, 0, 0, 0, 17524, 34697, 0, 0, 17532, 11660]
```

Bei dem Input-Register ist der richtige Funktionscode in beiden Nachrichten zu finden. Der Funktionscode ist hier 4. Die Startadresse ist hierbei die 2 gewesen, welche in der Anfrage gefunden wird. Im Python-Befehl sollen 14 Register, ab der Adresse 2 ausgelesen werden. Beide Zahlen sind in der Anfrage zu finden ($0E_{16} \hat{=} 14_{10}$).

Als Byteanzahl wird hier die Zahl $1C_{16}$ angegeben, was der Dezimalzahl 28 entspricht. Danach folgen 28 Datenbytes. Das erste Byte befindet sich beim Byte 2, das letzte nach der Aussage von oben bei Byte 29 ($2 * 14 + 1$).

In den oben gezeigten Beispiel wurde auch die Antwort der Python Funktion gezeigt. Bei der Lese-Funktion bekommt das Programm eine Liste von Integer-Werten zurück. In dem folgenden Kapitel I.3 werden dann diese empfangenden Werte in die richtigen Werte umgewandelt und ihre Interpretation gezeigt.

Auch die Holding-Register können ausgelesen werden. In VIFCON ist dies nicht nötig, da die geschriebenen Sollgeschwindigkeiten auch in die Input-Register geschrieben werden, durch die SPS. Somit kann VIFCON aus diesen Registern die Sollgeschwindigkeit bekommen. Ein weiterer Vorteil wird so gegeben. Die Sollgeschwindigkeit kann somit durch VIFCON und die Nemo-1-Anlage geändert werden. Beide Stellen bekommen die Änderung der anderen mit.

I.3 Daten Interpretation

In drei der vier Beispielen wurden Werte umgewandelt. Hierbei wird nun das Objekt *utils* der **pyModbusTCP** Bibliothek genutzt. Mit diesem Objekt lassen sich die Integer interpretieren und umwandeln. In dem Fall von VIFCON müssen Gleitpunkt-Variablen (in Python Float, in der SPS auch als REAL bezeichnet) und die Reihenfolge der Bytes beachtet werden. Für diesen Zweck werden die Funktionen/Methoden:

1. encode_ieee(),
2. decode_ieee() und,
3. word_list_to_long() verwendet.

Im Folgenden werden diese drei Funktionen erläutert und anhand der Beispiele aus Kapitel I.1 und I.2 dargestellt.

Beispiel 1 - Holding-Register beschreiben:

In Kapitel I.1 wurde ein Beispiel zum Beschreiben 2 Register des Holding-Register gezeigt. Dabei wurde die Zahl 1,5 mm/min als Liste von Integer-Werten (Liste: [16320, 0]) dargestellt. Im Folgenden soll die Umwandlung beschrieben werden.

```
1 write_value = 1.5
2 float_to_int = utils.encode_ieee(write_value)   print: 1069547520
3 int_to_hex = hex(float_to_int)[2:]              print: 3fc00000
4 hex_HB = int_to_hex[0:4]                        print: 3fc0
5 hex_LB = int_to_hex[4:]                         print: 0000
6 int_HB = int(hex_HB,16)                         print: 16320
7 int_LB = int(hex_LB,16)                         print: 0
```

Bei der dazugehörigen Funktion (siehe Beispiel Holding Kapitel I.1) werden die Registerwerte als Integer übergeben. Weiterhin ist die 1,5 vom Typ Float, was nicht einfach an die SPS übergeben werden kann. Die SPS erwartet in den Registern zwar eine Gleitkommazahl, doch diese wird aus

zwei Integer-Werten interpretiert. In dem Beispiel kommt somit die 16320 in das Register der Adresse 2 und die 0 in das Register der Adresse 3.

Mit der Funktion *encode_ieee* wird eine Python Float Variable in einen 32-Bit Integer im Standard der IEEE. Das IEEE steht für „**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers“. Durch diese Organisation werden Standards festgelegt. (Quelle: [Iee])

Dieser Integer Zahl wird nun in eine Hexadezimal-Zahl umgewandelt und bearbeitet. Die Bearbeitung umfasst das entfernen des Präfixes (hier 0x) und das darauf folgende Teilen der beiden Wörter. Da die Register 16 Bit enthalten, muss beachtet werden, dass bestimmte Werte 32 Bit, also 2 Register, benötigen. In dem Fall muss die Reihenfolge der Wörter beachtet werden. Die Hexadezimalzahl 3FC00000 besteht aus 4 Bytes, also 2 Wörtern (in SPS WORD). Bei der Darstellung in Hexadezimal entspricht eine Ziffer einer Anzahl von 4 Bits. In der SPS ist die Interpretation so eingestellt, dass das $3FC0_{16}$ als höherwertiges Wort und die 0000_{16} als niederwertige Wort angenommen werden. Als letztes werden die beiden Hexadezimalzahlen noch in Integer umgewandelt und dann wie im Beispiel gezeigt versendet.

Beispiel 2 - Holding-Register auslesen:

Um die Umwandlung besser zu beschreiben, wird hier das Beispiel 1 nun umgedreht. In Kapitel I.2 wurde erwähnt, dass das Auslesen der Holding-Register nicht benötigt wird. In einer älteren Version war dies jedoch noch beinhaltet. Wie gesagt soll dies hier nur der Interpretation der Daten dienen.

```
1 lese_value = [16320, 0]
2 Bits_List_32 = utils.word_list_to_long(lese_value, big_endian=True, ...
      long_long=False)
3 int_to_float = utils.decode_ieee(Bits_List_32[0])
4
5 utils.word_list_to_long --> print: [1069547520]
6 utils.decode_ieee       --> print: 1.5
```

Wie in den Beispielen gezeigt, bekommt das Programm von der SPS in dem Fall eine Liste zurück. Zunächst kann gesehen werden, dass die Bibliothek **pyModbusTCP** somit die Werte als Integer verlangt. Weiterhin können die beiden Ausgaben der Umwandlung, auch in Beispiel 1, gesehen werden. Somit stimmt die Umwandlung der Werte.

Die Funktion *word_list_to_long* wird genutzt um die Werte umzuwandeln die 2 oder 4 Register einnehmen. Mit dem Argument *long_long* wird diese Entscheidung getroffen. Bei False werden 2 Register verbunden und bei True 4 (64 Bit = 8 Byte = 4 Wörter). Das andere Argument *big_endian* legt die Reihenfolge der Wörter fest. Wenn dieses Argument auf True steht so wird das erste Wort als höherwertiges angesehen. In dem Beispiel der Input-Register (Beispiel 3) wird dies deutlicher.

Beispiel 3 - Input-Register auslesen:

Die Umwandlung dieser Register ist wie in Beispiel 2 gezeigt.

```
1  lese_value = [0, 0, 0, 0, 0, 0, 0, 0, 17524, 34697, 0, 0, 17532, 11660]
2  Bits_List_32 = utils.word_list_to_long(lese_value, big_endian=True, ...
       long_long=False)
3  value_list = []
4  i = 1
5  for word in Bits_List_32:
6      int_to_float = utils.decode_ieee(word)
7      value_list.append(round(int_to_float, 3))
8      i += 1
9
10 Bits_List_32 --> [0,     0,   0,   0, 1148487561,   0, 1148988812]
11 value_list   --> [0.0, 0.0, 0.0, 0.0,    978.118, 0.0,   1008.712]
```

In VIFCON wird diese For-Schleife verwendet um die Integer im IEEE Standard wieder in Float-Zahlen darzustellen. Somit wird die Antwort der SPS, durch die Funktion *word_list_to_long* halbiert. Nun muss jedoch aufgepasst werden. Wenn sich Tabelle 14 angesehen wird, so kann erkannt werden, dass nicht alle Werte 2 Register besitzen. Die Statusmeldungen bestehen z.B. nur aus einem Register. In dem Fall kann die Funktion *word_list_to_long* nicht angewendet werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACK	**Ack**nowledged, acknowledgement (Deutsch: Bestätigt)
API	**A**pplication **P**rogramming **I**nterface
AS	**A**blauf **S**prache
AWL	**A**n**w**eisungs**l**iste
COM	**C**omponent **O**bject **M**odel
CTS	**C**lear **T**o **S**end
CZ	**Cz**ochralski-Verfahren
DBD	**D**aten **B**austein **D**ouble Word
DBW	**D**aten **B**austein **W**ord
DBX	**D**aten **B**austein Bit
DCD	**D**ata **C**arrier **D**etect
DEE	**D**aten**e**n**d**e**i**nrichtung
DSR	**D**ata **S**et **R**eady
DTR	**D**ata **T**erminal **R**eady
EVA	**E**ingabe-**V**erarbeitung-**A**usgabe
FBS	**F**unktions**b**austein-**S**prache
FUP	**Fu**nktions**p**lan
FZ	**F**loat-**Z**one-Verfahren
GND	**G**rou**nd**
GUI	graphische Umgebungen (engl. **G**raphical **U**ser **I**nterface)
IKZ	Leibniz **I**nstitut für **K**ristall**z**üchtung
IP	**I**nternet **P**rotocol
ISO	**I**nternational **O**rganization for **S**tandardization
KOP	**Ko**ntakt**p**lan
LSB	**L**east **S**ignificant **Bi**t (niedrigstwertige Bit)
MFC	**M**ass **F**low **C**ontroller (Massendurchflussregler)
MSB	**M**ost **S**ignificant **Bi**t (höchstwertige Bit)
Mutex	**mut**ual **ex**clusion (Deutsch: gegenseitiger Ausschluss, wechselseitiger Ausschluss)
MVC	**M**odel **V**iew **C**ontroller
NAK	**N**egative **Ac**k**n**owledgement, Not Acknowledged (Deutsch: nicht bestätigt)
NEMOCRYS	**Ne**xt Generation Multiphysical **Mo**dels for **Crys**tal Growth Processes
Op oder OP	**O**perating **P**oint
OSI	**O**pen **S**ystems **I**nterconnection
PID	**P**roportional-**I**ntegral-**D**erivative
PLC	**P**rogrammable **L**ogic **C**ontroller
POU	**P**rogram **O**rganisation **U**nits
RI	**R**ing **I**ndicator
RTS	**R**equest **T**o **S**end
RXD	**R**eceive **D**ata
SPS	**S**peicher**p**rogrammierbare **S**teuerung
ST	**S**trukturierter **T**ext
TCP	**T**ransmission **C**ontrol **P**rotocol
TXD	**T**ransmit **D**ata
VIFCON	**Vi**sual **F**urnace **Con**trol

Danksagung

Ich bedanke mich bei meinen Betreuern: Prof. Dr.-Ing. Steffen Borchers-Tigasson und Dr. Kaspars Dadzis für hilfreiche Diskussionen und wertvolle Hinweise zur Arbeit.

Einen besonderen Dank möchte ich an meine Kollegen im IKZ der Modellexperimente-Gruppe (Sektion: Fundamentale Beschreibung) Dr. Kaspars Dadzis, Arved Enders-Seidlitz, Iason Tsiapkinis, Dr. Sepehr Foroushani und Felix Oesterle richten. Speziell danke ich Arved Enders-Seidlitz für die Erläuterungen zu Multilog im Sinne der Funktion und des Codes; Felix Oesterle für die Mithilfe bei der Umsetzung der VIFCON-Multilog Verbindung; Sepehr Foroushani für die Erläuterung der Nemo-1-Anlage und Kaspars Dadzis und Iason Tsiapkinis für die Ausführung von Experimenten und Versuchen mit der VIFCON-Steuerung, um diese auszutesten und weiterzuentwickeln.

Auch danke ich dieser Gruppe für die Bereitstellung der Geräte und der Unterstützung bei der Nutzung der verschiedenen Anlagen.

Des Weiteren möchte ich auch dem restlichen Personal des IKZ für die freundliche Behandlung und die tatkräftige Unterstützung bei Anfragen danken. Besonderen Dank möchte ich dabei an Max Scheffler, Stefan Pueschel und Adrian Wagner für die Vorstellung und Erläuterung der drei Anlagen am IKZ richten. Dazu möchte ich auch den Mitarbeitern von der Firma Auteam, besonders Bernd Eberhardt, für die Erläuterung und den Test der Modbus-Schnittstelle zur Nemo-1-Anlage danken.

Ich möchte weiterhin Karin Funke danken für das Kontrolllesen in Hinsicht auf Rechtschreibung und Grammatik, sowie auch noch einmal Dr. Kaspars Dadzis für die Hinweise zur Korrektur und Überarbeitung.

Eidesstattliche Erklärung

Hiermit erkläre ich durch meine Unterschrift, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe erstellt und andere als die angegebenen Quellen und Hilfsmittel nicht verwendet habe.

Alle Texte, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Publikationen entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form - auch auszugsweise - noch keiner anderen Prüfungsbehörde vorgelegen und wurde bisher nicht veröffentlicht.

Berlin, den 03.06.2024
Vincent Funke

This research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (project NEMOCRYS, grant agreement No 851768).